Adapting Communications in Cloud Games

Richard Ewelle Ewelle

THÈSE
Pour obtenir le grade de
Docteur

Délivré par **L'UNIVERSITE DE MONTPELLIER**

Préparée au sein de l'école doctoraleI2S
Et de l'unité de recherche LIRMM

Spécialité :**INFORMATIQUE**

Présentée par **Richard EWELLE EWELLE**

ADAPTER LES COMMUNICATIONS DES JEUX DANS LE CLOUD

Date de soutenance : 28/08/2015 devant le jury composé de

Mr. Stefano CERRI,Pr,Université de Montpellier	Directeur de thèse
Mr. Claude Frasson, Pr,University of Montréal	Rapporteur
Mr. Abderrafiaa KOUKAM, Pr,Université de Technologie de Belfort-Montbéliard" (UTBM),	Rapporteur
Mr. Abdelkader GOUAICH, MCf,Université Montpellier,	Co-encadrant de thèse
Mr. Laurent LEFEVRE,HDR,Inria, École normale supérieure de Lyon	Examinateur
Mr. Chouki TIBERMACINE, MCf,Université de Montpellier	Examinateur

Académie de Montpellier

Université Montpellier II

Sciences et Techniques du Languedoc

Ph.D Thesis

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier pour
obtenir le diplôme de doctorat

Spécialité	:	**Informatique**
Formation Doctorale	:	**Informatique**
École Doctorale	:	**Information, Structures, Systèmes**

Adapting Communications in Cloud Games

par

Richard EWELLE EWELLE

Version du August 28, 2015

A la mémoire de mon papa 'Papson'

Contents

Remerciements

Une thèse est une route que l'on ne parcours pas seul. Pour cela avant de vous présenter mon travail, je tiens à remercier tous ceux qui m'ont aidé à la réalisation de ce travail.

Mes premiers remerciements vont à mes directeurs de thèse, le Professeur Stefano A. Cerri et le Docteur Abdelkader GOUAÏCH, pour leurs conseils et encouragements exprimés tout le long de mon parcours de thèse.

Je tiens également à remercier tous les membres de l'équipe SMILE du LIRMM, pour leur amitié, leur soutien, leur accueil dans le groupe et les collaborations scientifiques.

Je ne saurais terminer sans remercier le seigneur mon Dieu sans qui rien de ceci ne serait possible, les membres de ma grande famille au Cameroun pour leurs encouragements, leurs prières et leur présence malgré la distance.

Chapter 1

Introduction

Contents

Cloud computing sector has received considerable attention from global and local Information Technology (IT) stakeholders, national governments, and international agencies. For example, IBM[1] has established cloud computing centres in China, India, Vietnam, Brazil, and South Korea. Other global IT majors such as Microsoft[2], VMware[3], Salesforce[4], Dell[5], and Parallels[6] are actively searching for opportunities around the world.

1. http://www.ibm.com/cloud-computing/
2. http://www.microsoft.com/enterprise/microsoftcloud/
3. http://www.vmware.com/cloud-computing/
4. https://www.salesforce.com/
5. http://www.dell.com/learn/us/en/555/dell-cloud-computing
6. http://sp.parallels.com

Many definitions of cloud computing exist and there seems to be no consensus on what a cloud is. But, generally speaking, cloud computing could be understood as a hardware and software infrastructure able to provide *services* at any traditional IT layer : software, platform, and infrastructure, with minimal deployment time and reduced costs. The National Institute of Standard and Technology defines cloud computing as *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [Mell and Grance, 2011]. In software provisioning for example, a high-performance web application can be easily deployed in a remotely virtualized server running on a distributed hardware infrastructure. Charges are considered only for resources actually used, while sharing the cloud infrastructure with other users.

It is difficult to talk about cloud computing without referring to the grid computing paradigm because of their similarities and the fact that cloud computing originates from grid computing. As a matter of fact, cloud computing can conceptually be viewed as a kind of grid computing with respect to the definition of grid computing. Grid computing pioneers Ian Foster and Carl Kesselman, define a grid as *"a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service"* [Foster and Kesselman, 1999]. This definition shows the similitude between grid and cloud computing in terms of hardware and software infrastructure providing an inexpensive access to high-end computing resources for better quality of service. Giving a detailed comparison of these notions is out of the scope of this thesis but for further information about their similarities, refer to [Foster et al., 2008].

Video games increasingly gain attention from the entertainment industry and scientific community. Beginning with basic games, they quickly evolved into rich, interactive, animated and intricate virtual environments approaching movies in terms of visual excitement. This popularity, led the adoption of video games in other training/teaching disciplines and gave birth to what is now called *serious games*. Serious games refer to using video games for serious purposes. They enable video games to become more than just entertainment tools, but real assets to help increase the effectiveness of training and teaching. Moving these game applications to the cloud will increase their accessibility and therefore their popularity. This is referred to as *cloud gaming*. In fact, cloud gaming is a type of online gaming that allows direct and on-demand streaming of games onto a receiving device.

This chapter introduces the context of this research which is building video games with cloud gaming paradigm. More precisely this thesis is about delivering player's quality of experience (QoE) by putting in place an intelligent client-server communications in cloud gaming platforms. The end result is an adaptive communications scheme that deals with network constraints while maintaining an acceptable user experience.

After exposing the motivations behind the use of cloud computing for video games, we analyse and discuss the state of art of cloud gaming systems; the research question is then stated and followed by an overall presentation of our proposition and contributions.

1.1 Cloud computing and video games

The potential of cloud computing is vast, and current cloud computing application sectors include academics, health-care, business and entertainment.

Why cloud computing

Depending on the sector, cloud computing adoption is motivated by many factors. Here are some examples of institutions that embraced cloud computing:

— **In scientific research**, for some universities, high-end computing power availability through cloud computing for research purposes is welcomed. For instance, researchers at the Medical College of Wisconsin Biotechnology and Bioengineering Center in Milwaukee are making protein research more affordable to scientists worldwide [Susa, 2009]. They do this by renting processing time on Google's powerful cloud-based servers. Since with cloud computing there is neither maintenance, nor setup overhead, researchers are able to reduce the cost of their studies and analyze their data in greater depth than it was previously attainable.

— **In healthcare**, as a good example of cloud computing utility, Max Healthcare, a large hospital chain in India, moved its Health Information System (HIS) to a private cloud-computing system provisioned by Dell. This makes it easier for Max Healthcare to treat and follow up with patients from any of its locations across the country. As Max Healthcare adds more hospitals to their network, the cloud deployment gives a near plug-and-play capability for information technology deployment [Groen, 2012].

In May 2009, Guang Dong Hospital of Traditional Chinese Medicine has implemented a suite of healthcare data-sharing and analytics technologies, known as Clinical and Health Record Analytics and Sharing (CHAS). Hospitals use CHAS to share electronic medical records (EMRs), incorporating this data across the hospital network [Kshetri, 2010].

Cloud computing adoption for health purpose have also generated a lot of interest in the research community in the past few years. For instance [Doukas et al., 2010] and [Rolim et al., 2010] , propose a telemedecine service that automates patient's information workflow from data collection to information processing and delivery in healthcare institutions using cloud computing.

— **In business and entertainment**, just like in education and healthcare, information technologies implementation in business organizations is a high resource-consuming process. Using cloud computing in business has been an effective way to deal with this issue. In media and entertainment industry, using cloud computing, media companies are able to develop new and better ways to quickly and efficiently deliver content to fine-grained targeted consumers while potentially reducing costs. Companies such as Amazon, Apple, Google and Netflix are big players for delivering entertainment via the cloud. Some of the most used cloud applications nowadays include Google apps, Youtube, Instagram, Spotify, Netflix, Facebook, Twitter, LinkedIn.

To summarize, it is clear that cloud computing can be beneficial for many industries. Among others, the entertainment industry is particularly well suited for cloud computing. In fact, cloud gaming operators exploit cloud computing accessibility and ubiquity features to deliver game content around the globe. Cloud gaming enables video games to be played with various types of client devices and to be accessed as easily as movies and music through the Internet.

Cloud gaming

Cloud gaming, also referred to as on-demand gaming [Nvidia, 2014], is a type of online gaming that is similar to video on demand. It allows a direct and on-demand streaming of video games using ”thin” clients. In the architecture point of view, cloud gaming is a client-server system, where the actual game is stored and run on operator’s data center (server) and graphics or game states are directly streamed to client devices. Clients game states are therefore synchronized with the server, in order to replicate the game as it is executed on the server. Massive computation and storage resources of data centres enable users to shift their workload to remote servers. Consequently, with the supply from remote servers, thin clients can be more convenient and also more powerful, than traditional thick clients without the server’s supply.

Cloud gaming nowadays is a reality and companies such as Onlive[7], G-Cluster[8], StreamMyGame[9], Gaikai[10] and T5-Labs[11] are already offering commercial cloud gaming services.

This new sector is also seen as a serious competitor for traditional game market. This has prompted video game majors such as Sega, Ubisoft, Epic Games, Atari, Warner Bros, Disney Interactive studio to establish partnerships with Onlive to distribute their games [Onlive, 2014].

1.2 Cloud gaming advantages

Besides computing power and storage volume, five main reasons can motivate usage of cloud gaming:

— **Flexibility:** cloud gaming takes advantage of cloud computing flexibility to adjust resources allocation based on dynamic demands, in a manner which is transparent to players.

— **Ubiquity:** games are available at anytime from almost any device including thin clients such as smart phones and tablets. With broadband internet connections, it is even possible to access games from literally everywhere in the world.

— **Cost :** for game studios, cloud computing offers a cost and energy efficient centralization of gaming infrastructures. In fact, game studios are able to reduce their operating cost by delegating the responsibility of operating and maintaining game infrastructures to cloud gaming providers. For players, having to buy an expensive PC or console to play good quality video games is no longer a necessity. Cheap thin clients such as smartphones and tablets are already very popular, and can be used to access these same games via cloud gaming.

7. http://www.onlive.com/
8. http://www.g-cluster.com/
9. http://streammygame.com/
10. https://www.gaikai.com/
11. http://www.t5labs.com/

— **Security compliance and privacy:** player's game data (sensitive of not) are collected in a single point of control. This centralized approach reduces hacking and "cheating" possibilities since only the server holds the game logic. Players are able to access these data from everywhere and at any time. In health oriented serious games for instance, patient's clinical information and training curves are strictly confidential, that is the reason why it is paramount to centralize these data and host them on certified operator's premises.

— **Set up overheads and compatibility issues:** trying new games sometimes requires to install new softwares and deal with hardware incompatibility between the game and player's device. This can be a time consuming and dissuasive task that can even prevent players from purchasing new games.

  This problem is also present in most institutions adopting video games as a pedagogical tool in a large scale. For instance, we encountered this problem in MoJOS project. MoJOS was a serious gaming project, started in 2009, with the objective of creating a health-oriented game engine, with a focus on arm rehabilitation after a stroke [MojOS, 2009]. During the course of this project, we deployed several rehabilitation games in hospitals such as the hospital Lapeyronie in Montpellier and the hospital of Grau du roi. The installation and the configuration of the whole serious gaming system was a very time consuming task. Some therapists were even reluctant to use this technology because of this setup overhead and the time it took to start a training session. In addition, most healthcare institutions do not have the finances to hire new personnel dedicated to setting up these games for patients.

  With cloud gaming, video games are hosted and upgraded at the server level, rather than on each individual client device. There is no need to constantly upgrade clients computers and no compatibility issues when trying new games.

— **A new economic model:** with cloud gaming, video games are available online on subscription. Players can start playing games right after the subscription, without having to wait for games to be shipped or go to a physical store to purchase games.

Some of these features are exhibited in the onlive cloud gaming solution as shown in figure 1.1.

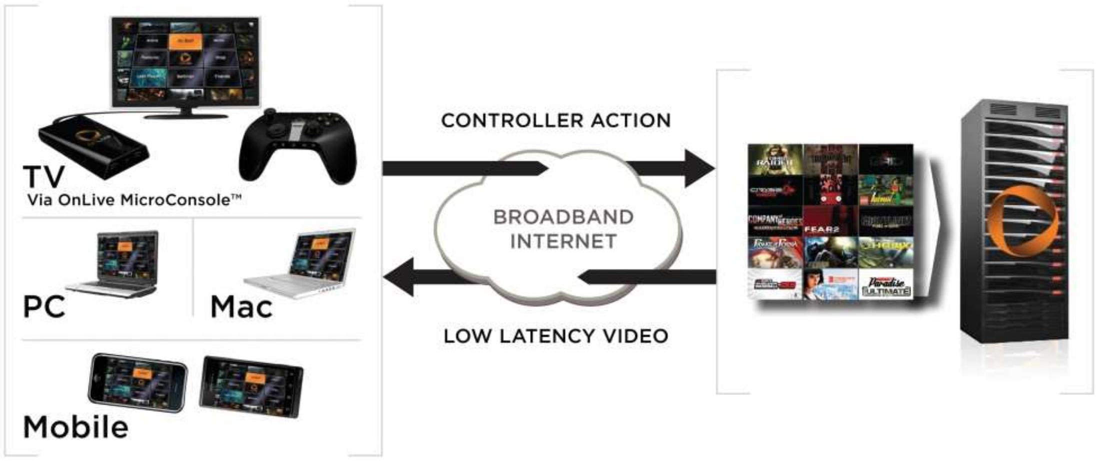


Figure 1.1 – Onlive cloud gaming solution [Onlive, 2010]

1.3 Cloud gaming limitations

As promising as it is, cloud gaming is also facing many challenges that, if not well resolved, may prevent its adoption. The most important are related to quality of experience perceived by players and data security.

1.3.1 Quality of experience (QoE)

In various formal and informal definitions, player experience has been framed as a *"subjective measure of the degree to which a system meets the target user's tacit and explicit expectations for experience"* [Beauregard and Corriveau, 2007]. In video games, player experience is a broad concept that determines player's overall perception of a game. It includes many subjective factors influencing the degree of satisfaction of the player with a game. Some of these factors are: his/her feelings, his/her understanding of the game rules, difficulty of the game, interactions, aesthetics, storytelling etc.

Quality of experience, can be considered as the objective component of player experience. It represents measurable technical aspects of game quality and has a direct relationship with the traditional Quality of Service (QoS). In fact, in video games, it is proven that QoS metrics such as video frame rate and response time have a direct influence on player's perceived QoE [Wu et al., 2009]. Moreover factors such as the amount of game entities in the virtual world and the resulting processing load have also been correlated to the user QoE [Miller et al., 2014] . These QoS metrics are easy to measure and have consistent thresholds beyond which player's QoE is degraded [Miller et al., 2014].

In centralized networked games, as far as objective game quality is concerned, system response time plays a significant role. It can be defined by the time needed to detect a user-initiated input event, to send it to the server, to process it, to send the updated game state to the client and to render the updated game state on the output device. This definition shows that, along side with processing power, available bandwidth for inputs and updates transmission can have a considerable impact on response time therefore on player's QoE.

As a matter of fact, there is a strong correlation between network conditions, game's responsiveness and video quality. Low bandwidth capacity and a drastic presence of limiting network factors such as delay, jitter or packet loss, can cause important lag in the player-game interaction and eventually decrease the smoothness of the game video. That is one of the reasons why, depending on requested video quality, cloud gaming providers always set a minimum bandwidth requirement to play a game. For instance, Onlive [Onlive, 2010] requires a minimum bandwidth of 2 Mbps for all their games, which are delivered only in HDTV 720p format. Unlike Onlive, StreamMyGame supports game streaming in a variety of resolutions, from 240p to 1080p, requiring an Internet connection between 256 Kbps(240p) and 30 Mbps(1080p).

We can see with these examples that, cloud gaming services have very strong requirements in terms of network resources. In fact in order to ensure the delivery of QoS for bulk data transfer generated by cloud games, a certain amount of network resources should be available on the receiving end.

The bandwidth related issues of cloud gaming can therefore be summarized as follows:

— Thin clients with low bandwidth capabilities as well as people located in areas with limited network resources, can not take full advantage of cloud gaming at its current state. Another example is people subscribed to a 3G/4G networks with limited data connection per month can not afford playing games with bulk data transmissions. An interesting alternative will be, letting clients with low bandwidth access these games, and even better, allowing clients with different network capabilities to play the same game with approximately the same QoE.

— In current cloud gaming systems, a client with unstable network resources, will see its game session stopped if its network throughput drastically dropped bellow the required minimum. These network conditions fluctuations can happen in case of congested network due to the presence of concurrent network traffic for example. Cloud gaming platforms therefore need to take into account network conditions changes, while minimizing game disruption which is very detrimental to player experience.

The objective of this thesis is then to make it possible for devices with low and/or unstable network capabilities, to run cloud games with an acceptable QoE.

1.3.2 Data security

The recurrent dilemma presented by cloud computing for security professionals is to be able to embrace its benefits while maintaining security controls over sensitive data and assets. According to Vormetric [Vormetric, 2012], some of the most important data security challenges of cloud include:

— There is a crucial need to protect confidential data. These sensitive data are generally protected by the law. Therefore cloud providers have to put in place the adequate security (confidentiality, integrity, availability) policy in order to ensure that sensitive data are kept safe. In practice, a cloud provider may have control and visibility into your data, and this constitutes a violation of the law on personal data. This is the main reason why, most healthcare institutions hesitate when it comes to moving their systems in the cloud.

— In order to reduce costs, most cloud service providers share physical resources and services between multiple consumers and client organizations (tenants). In this scenario, a tenant can deliberately or inadvertently interfere with the security of the other tenants. To prevent this to happen, there is a necessity to apply a logical segregation and other controls between tenants.

— Data mobility worldwide is regulated by the law. There are often legal issues between cloud service providers and governments, relative to government rules such as the EU Data Privacy Directives and the CNIL in France, with regards to sensitive data. Cloud providers must have the required certifications and respect the regulation in place, in order to store or process sensitive data.

— Lack of standards about how cloud service providers securely recycle disk space and erase existing data. Meaning that your data may still be accessible even after ”deletion” from operator’s data base.

All these challenges make it difficult for cloud service providers to deliver their services with adequate level of security. All over the world, there are regulations over sensitive data like medical data to make the service providers (hosting or processing these data) accountable for any security flaw. In this thesis, we will not focus on security concerns, but it is worth noting that any public cloud gaming service needs to follow the guidelines of the regulations in place concerning sensitive data. This includes meeting the security challenges caused by the cloud. These challenges are even more present in serious gaming.

1.4 Problem statement

Our study focuses on the interaction between video games and cloud platform, more precisely on the efficiency of the game data synchronization mechanism between client and server for QoE support. Even though the motivations for this thesis was in serious gaming, our approach and contributions have to be generic enough to be applied to all game categories. We aim at building a distributed game engine for cloud games, and application programming interfaces (APIs) with pre-defined network-based functions that enable an efficient execution of video games in the cloud.

In cloud gaming, bandwidth issues represent an important challenge as explained in section 1.3.1. Hence in the context of health oriented serious games for example, facilities with lossy and/or unstable network access, will not be able to train their patients via these traditional cloud gaming services. This is the reason why, in order to preserve game accessibility and ubiquity, we need to: (i) weaken these network constraints; (ii) minimize the effects of network conditions fluctuations on player's QoE; (iii) ensure the transmission of the game content.

Thus the general question guiding this thesis is:

– How to build cloud games that are accessible on devices with limited and/or unstable network resources, while maintaining an acceptable player's QoE?

In order to respond to this question, we have identified the following research challenges: (i) a game model aware of game's communications needs; (ii) game's communications adaptation to network conditions.

Game model

In fact, in order to be able to decrease game's network resources constraints and respond to network fluctuations, game systems have to be able to control the amount of network communications generated by games.

We consider a game as a collection of scenes populated with game objects. These objects represent physical game entities that players see on the screen. They are the only game elements that are synchronized between client and server, as they are sufficient to reproduce game scenes in a client-server game architecture. The first step in attempting to control game's network communications is therefore to find a model representing the game communications profile in terms of game objects and network resources. Actually, game objects can have different importance in game scene, therefore they may be subjected to different network requirements vis-a-vis state updates transmission. Thus the communications adaptation process needs to know this abstract representation of the game in order to deliver appropriate QoS to each game object.

Communications adaptation

In shared networks, individual throughput of each user can change at any moment. In addition, the "best-effort" nature of Internet, makes it prone to delay, latency and jitter. Each of these network factors can contribute to response time's prolongation. In online games, a prolonged response time creates lags in player's interaction with the game and consequently results to a bad player's QoE. As a matter of fact, network delay, a cause of lag in player-game interaction is considered as "fun killer" in online games in general. We therefore need a system capable of: (i) fostering player experience by controlling game's communications in low bandwidth networks in order to provide QoE support against lag in cloud games; (ii) adapting game's communications to current network conditions in unstable network setups.

1.5 Contributions

The contributions of this thesis are summarized as follow:

Game models

During the course of this thesis, we presented two adaptive game models enabling us to express different sensitivity to network lag, thus different communications needs among game objects, depending on their importance in the game scene. These models are then used to adjust objects' importance as the game evolves. They are based on principles of agents organizations and gameplay components.

With agent organization, the focus is made on objects' functional role in the game scene. Then a mapping is made between roles and importance values. In that way, a change in object's role changes the object's importance. Our organizational model clusters game objects with the same role in groups and the network resources attributed to game objects are proportional to their role in the game organization.

With gameplay components (introduced in [Francillette et al., 2012]), complex game mechanics are decomposed in simple components also known as OCR loops. These loops are then combined hierarchically to form a gameplay components tree representing the game. Each node of the tree is assigned an importance value that changes as the tree evolves (nodes are added or removed) and each game object is associated with a node. Each time a gameplay component is added or removed from the tree, and even when it simply ends, the importance values of the rest of the tree are recalculated. This enables nodes' importance values to adapt to the changes in the game tree.

Adapting communications for cloud games

The main contribution of this thesis consists of a game's communications adaptation to maintain player's QoE, inspired by level of detail principles [Wissner et al., 2010]. The idea is to monitor current network conditions, and changes in game objects' importance in the game scene, and use our adaptation model to adjust game's communications accordingly. The game models presented above are used to manage changes in objects' importance.

This communications adjustment is performed in two steps: the first step is reorganizing the game model, in case of changes in game objects' importance; and the second is selecting appropriate communications levels for game objects. A communications level identifies the amount of network resources attributed to the game objects (determined by synchronization rate). This adaptation can affect game communications in two ways: (i) it decreases game communications when network conditions are low, by replacing objects' current communications levels by levels with lower synchronization rate; (ii) it increases game communications in case of favorable network conditions. This bidirectional adaptation approach results in the maintenance of an acceptable QoE in case of network resources shortage and the improvement of the QoE in case of better network throughput.

We validate our approach using prototype games in a controlled environment and assess user's QoE in pilot experiments. Results show that the proposed adaptation framework provides a significant QoE enhancements, compared to a solution without adaptation.

1.6 Structure of the thesis

The rest of the document is organized as follows:

— Chapter 2 presents the backgrounds of the thesis by emphasizing on main concepts such as video games, cloud gaming, quality of experience and game adaptation.

— Chapter 3 presents the state of the art in cloud gaming and the attempts to maintain acceptable player's QoE in constrained networks by adapting game communications to the network conditions. This study enables us to point out the weaknesses of the existing approaches and therefore support the need to propose a new one.

— Chapter 4 introduces the general framework used for adaptation, as well as the core concepts on top of which our adaptation framework is built. This includes level of detail in 3D graphics, agents organization and gameplay components.

— Chapter 5 presents our game's communications adaptation approach, inspired by the level of detail technique. Current network conditions and game objects' importance are used as inputs for adaptation.

— Chapter 6 presents prototype implementations and experimental evaluations of our proposition. We describe the pilot experiments carried out and show the results. We conclude the chapter with a general discussion on the QoE enhancement provided by our approach on lossy and congested networks.

— Chapter 7 draws the conclusions about the proposed approach and discusses some future directions.

CHAPTER 2

Background

Contents

This chapter presents the background of our work. First, we discuss in section 2.1 and 2.2, the definitions of video games and present core concepts such as online games and update/render game loop. Section 2.3 presents cloud gaming technologies such as video streaming and game objects replication. Section 2.4 introduces the concept of quality of experience (QoE) in games and discusses lag and its effects in online games. Finally, adaptation is introduced in section 2.5.

2.1 Video games

Different definitions of game can be found in literature. These definitions attempt to capture different principal aspects of games: some focus purely on game as an activity with characteristics such as uncertainty and unproductivity, others focus on rules governing the game, and while others emphasize on game outcomes.

One commonly cited definition in this domain, is given by R. Caillois in 1961 [Caillois, 1961]. R. Caillois identifies a game as an activity and emphasizes the voluntary and unproductive aspects of game as well as game rules. He defines a game as *"an activity which is essentially: free (voluntary), separate (in time and space), uncertain, unproductive, governed by rules, make-believe"*.

Katie Salen and Eric Zimmerman consider a game as *"a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome"* [Salen and Zimmerman, 2003]

According to these definitions, a game can be viewed as a regulated activity with an entertaining purpose in which players can act. Objectives in games are expressed as a formal system of quantifiable goals.

Juul Jesper in 2003 emphasizes the player-game interaction, and puts players' actions and feelings as part of the outcome equation [Juul, 2003]. He describes a game as *"a rule-based formal system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort in order to influence the outcome, the player feels attached to the outcome, and the consequences of the activity are optional and negotiable."*

Games require also a set of artefacts, known as game media, to be played (e.g. miniatures, a ball, cards, a board and pieces, or a computer). However, the given definitions above emphasise on the separation between the game as an abstract structure of actions and as an artefact.

As a matter of fact, there is no single game medium, but rather a number of different game media to play the same game. Furthermore, we know that many games actually move between media. For example card games are played using playing cards or on computers. The computer just happens to be one of the various existing game media.

This leads us to the definition of a video game which is simply a game that uses a computer as medium [Hawes, 2004]. This definition of video game as computer-controlled game is also shared by Zyda [Zyda, 2005]: *"A video game is a mental context, played with a computer in accordance with specific rules that uses entertainment"*.

So before presenting video game's concepts such as the update/render game loop, we should first describe the structure of a game independently from the technologies.

Game structure

A game, as any system, has a specific structure with well defined components. Game engineering process starts by identifying and specifying these components explicitly to produce a game design document.

The most important components of a game structure can be summarized as follow:

— **Action:** Players interact with game environment using a finite set of actions. In video games for example, player's actions are gathered through a controller device such as a keyboard, a mouse or a console controller, connected to the game platform.

These low level inputs are translated to high level actions and interpreted to change the game state according to dynamics rules.

— **State:** Every game has an explicit state represented with a specific formalism. A state fully describes the game process at any time. In the game of chess for example, game state is represented by squares on the chessboard and their contents. Actions therefore allow players to navigate through the space of possible states. This navigation is constrained by game's rules.

— **Rules:** Rules in games are constitutive rules [Chauvier, 2007]: they regulate and identify the game. Rules generally determine turn order, rights and responsibilities of players, goals, rewards and penalties as well as game's dynamics. Any alteration of game's rules changes the identity of the game. For instance, baseball can be played with "real" baseballs or with wiffleballs. So changing an element of the game medium does not necessarily change the game. However, if players decide to play with only three bases, they are arguably playing a different game.

— **Goals:** Goal characterizes particular states of the game where players are rewarded (success state) or punished (failure state). Players have to mobilize all means allowed to achieve game goals by reaching success state while avoiding failure states. Goals have to be achievable; as soon as they become unattainable with certainty, the game ends. The journey towards the achievement of a goal is usually referred to as a quest. A game can have multiple quests with different goals.

— **Challenge:** Challenge represents a set of factors that can delay the player in her/his quest and can ultimately contribute to his failure. Among these factors we can name:

1. player's skills and aptitudes,
2. other players or game entities which goals conflict with the player's goals,
3. unpredictability, the dynamic and the non-deterministic nature of the game.

Game update/render loop

In a game engine, game loop can be considered as a repeating chain of events that creates the game dynamics. The first step of a game loop is to update the game's state according to players' inputs; this is followed by rendering a view of the game's state. This entire cycle of update and rendering is repeated a number of times per second. The number of updates per second defines the update speed while the number of rendering per second defines the FPS (frame per second).

A generic game loop is presented in figure 2.1. The game starts with an initialization stage to create initial objects and load necessary assets. Once the initialization stage is completed, the game loop is repeated until the player quits the game. When a terminal state is reached, the end stage is executed to free used resources and quit the game environment properly.

A detailed description of each stage is given by [Whitaker, 2014]:

— **Initialization stage:** It is the entry point of the game, where all required pre-game setups need to be performed. This may includes loading content, preparing input devices, and initializing game state. This stage is only executed once.

— **Update stage:** The update stage calculates the game's state at the current time. The common tasks done in this stage are: (i) gathering of player's actions through

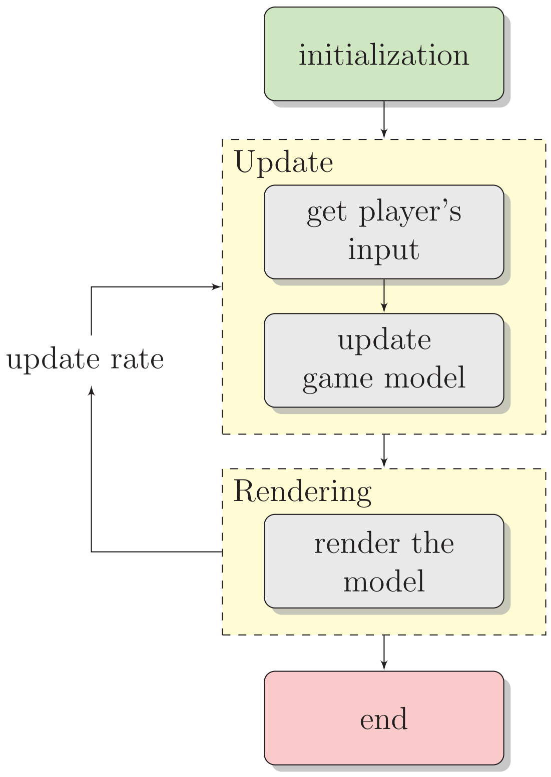


Figure 2.1 – Game loop.

inputs devices; (ii) updating object's state within the game; (iii) executing game logic and running the scripts controlling the behaviors of the Non Player Characters (NPCs)(in the video games jargon, this is referred to as AI). The running frequency of the update stage is generally referred to as the *update rate*, which is usually around 20 times per second.

— **Rendering stage:** The rendering stage draws a snapshot of the entire game model and renders it on a video output. The rendering frequency is defined as *Frame Per Second* (FPS). A common concern around rendering is the ability to anticipate this frequency so that, the game does not appear choppy and its smoothness is maintained. 20 FPS is usually the minimum, and it relates to the eye perception frequency limit.

— **End stage:** The end stage cleans up everything that is no longer needed. This usually corresponds to disposing art assets (image, font, etc.) that were loaded by the graphical processing unit (GPU). After the cleaning up, the game exits. This stage is also executed once.

2.2 Online video games

Also known as networked games, online video games are video games played through a network. Online gaming supports gaming over different kinds of networks (e.g: Local Area Networks (LANs), Internet), and enables players to connect to multiplayer games as well as monoplayer games.

Depending on realtime requirements of their gameplays, multi-player video games often follow two synchronization patterns: asynchronous gameplay and synchronous gameplay. From these patterns result different network resources requirements.

In an asynchronous gameplay, players are not required to be online simultaneously. In fact, their actions can be performed asynchronously at different times. Still, game state is updated coherently and rules ensure that the rate of actions per player is fairly balanced.

Most online social games are asynchronous games where the player can interact with a virtual environment without necessarily having other players connected and interacting at the same time. FarmVille[1] is an example of such a model.

Unlike asynchronous gameplay, synchronous gameplay enables multiple players to be online and to interact with the game world simultaneously. Synchronous multiplayer games are the most greedy in processing and network resources, thus they are more likely to be impacted by low and unstable network conditions. Because of these considerations, online games with synchronous gameplay are the one that will benefit the most from an efficient resources distribution for better performance.

Depending on the game and the level of control given to the players, different categories of architectures are often utilized to build online games. client-server is the most common. Here a server is responsible of running game logics and communicating updates to clients. A client sends inputs to the server, receives game state updates from the server and updates the game accordingly. This configuration can involve multiple clients connecting to a single, central server which is required to have an important processing speed and a high bandwidth capacity to process all clients' inputs.

In an online game with the client-server architecture, several configurations are possible here. The server can be either a broadcast server (simply broadcasts players' actions from one client to others) or a centralized server(receiving players' actions and running the game logic). The inconvenient of the broadcast server is that, it offers "cheating" opportunities. In fact, with a broadcast server, a client runs and controls the game logic locally. The player can therefore modify the game states in a way that gives him/her an advantage over other players. A centralized game logic, where only the server knows the actual state of the game, is a solution to most cheating problems. With this setup, the client and the server have a representation of the game model. The client side periodically synchronizes its local game model with the server side remote game model which is the central one. This is done by receiving update messages about state changes in the central game model.

As illustrated in figure 2.2, differences between single-system game loop of figure 2.1 and this client-server system with centralized game logic are:

1. **On the client side:** Two more tasks are added to the update stage: (i) client gathers inputs from the controller device, and sends them to the server; (ii) client receives update messages from the server, and then updates its local game model.
2. **On the server side:** There is no rendering stage. In fact the server only runs the game logic which receives players' inputs, updates the central game model accordingly and then sends update messages to the client. All the update code (objects' updates, Non Player Characters (NPCs)' updates, collision detection, etc.) is executed here.

If update messages from the server do not arrive at a fast enough rate, game data on the client game model will be outdated and therefore rendering will not reflect the

1. Farmville is a farming social network game developed by zynga, allowing players to cultivate their farms by plowing, planting and harvesting crops and trees. https://zynga.com/games/farmville

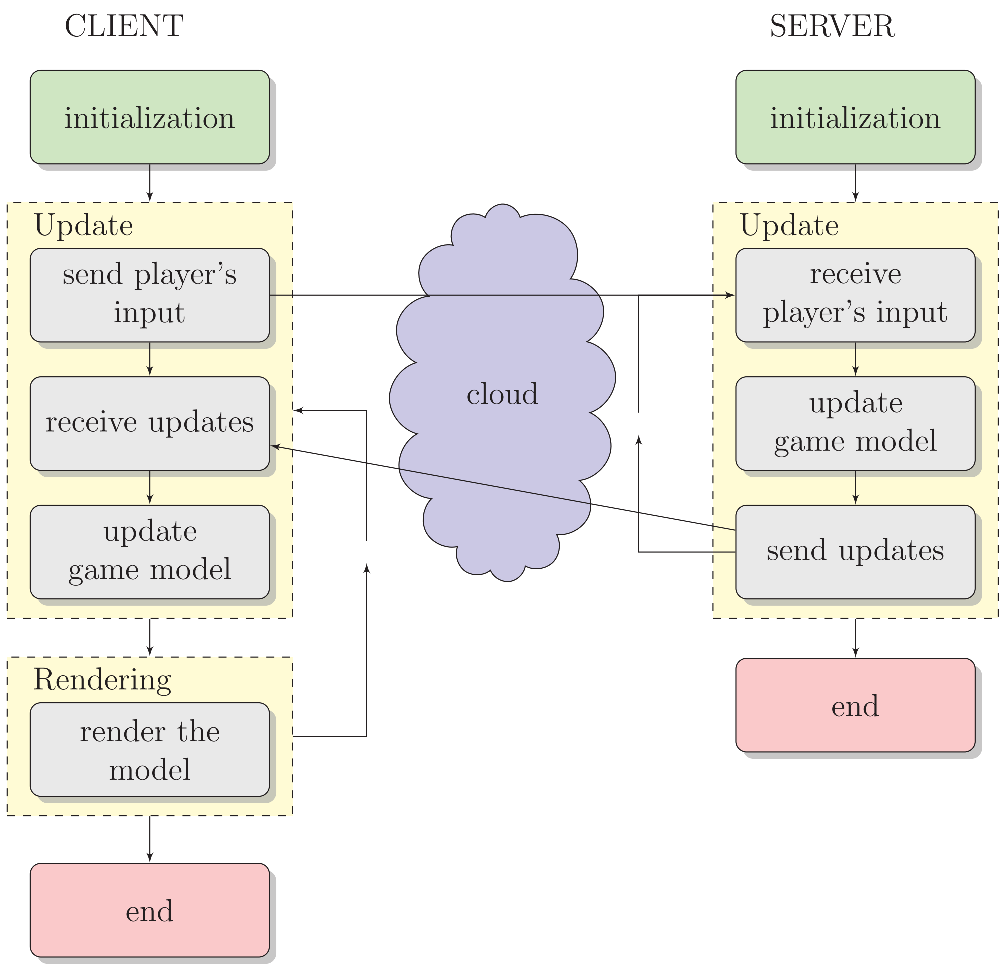


Figure 2.2 – Client-server game loop

current state of the game as it is on the server. These game loops show the effect that can have network latency and server's update frequency on the overall end-to-end lag. This can decrease the perceived quality of the game, since updates and inputs have to cross the network.

2.3 Cloud gaming

Cloud gaming can be defined as a client-server gaming paradigm where the game logic is placed on the server's side and managed as a cloud service. Another strong characteristic of the cloud gaming paradigm is the ability to use low-end or thin devices to play high quality games. Depending on how game's workload is divided between cloud servers and clients, four main trends can be defined:

1. **3D graphics streaming:** In 3D graphics streaming approach [Eisert and Fechteler, 2008], [Jurgelionis et al., 2009], the server executes the game logic, intercepts graphics commands, compresses these commands and streams them to clients. The client renders the game scene by executing the received commands using its own graphics chip. Client's graphics chip must be not only compatible with streamed graphics commands but also powerful enough to render game scenes in high quality and at real time. Since rendering is done on client devices, this approach is less suitable to

resource-constrained devices, such as mobile devices and set-top boxes for television sets.

2. **Video streaming:** In this approach the server renders all graphics, compresses resulting videos and streams them to clients [Holthe et al., 2009; De Winter et al., 2006]. Clients only decode and display the video stream. This approach relieves clients from computationally intensive graphics rendering and is ideal for thin clients on resource-constrained devices. Computational load is now paid with network load since transmitting videos is more expensive than transmitting just graphic commands.

3. **File streaming:** File streaming approach [cloudtweaks, 2013] makes it possible to download game content in form of fragments. The actual game is executed on user's game device. A small part of the game, usually less than 5% of the total game size, is downloaded initially so that players can start playing quickly. The rest of the game content is downloaded while playing. File streaming uses advanced methods of progressive downloading, data compression and prediction algorithms to provide steady motions and smooth visuals.

   File streaming has low bandwidth requirement to operate and is less sensitive to problems associated with video streaming such as lag and the need of high network resources. File streaming solutions for cloud gaming such as Kalydo [2] enable to cache the downloaded content on user's device, so that no download will be needed the next time the user plays the game.

4. **Objects replication:** With game objects replication or state streaming approach [Ferretti et al., 2007], the game is executed on the server. Game objects (game model) states are collected and streamed to clients. Clients update their local game model and render the resulting game scene using their graphics chip. This approach shares advantages and disadvantages of the 3D graphic streaming approach since clients are responsible for graphic rendering. Game objects replication is mostly used on Massively Multiplayer Online Games (MMOGs) [Cronin et al., 2002], [Ferretti and Roccetti, 2005] systems to diminish the delivery time of game events, while maintaining full consistency of game state.

These approaches have advantages and disadvantages, but depending on the context of the game and the resources of the user device, one of them can be preferred to another. 3D graphics streaming and video streaming follow the same principles. However, gain obtained by the 3D graphics streaming in network resources is compensated by GPU resources since rendering is done on client device.

File streaming is a good solution for users with lower bandwidth devices. But with the use of prediction algorithms, and client-side rendering, it results into a heavy client (CPU and GPU) running the game which is not ideal for resource-constrained devices. ”Cheating” can also be an issue with file streaming, since for the next time the game is played, no download is made and all game logic is executed on client device.

In this thesis we are only going to focus on video streaming and objects replication approaches. Many reasons motivate this choice: their wide spread adoption in the cloud gaming industry; they do not imply cheating and they keep their clients thin by offloading all computations to the server. These approaches are presented in more details in following sections.

2. http://kalydo.com/

2.3.1 The video streaming approach

Most of industrial cloud gaming systems are based on video streaming [Onlive, 2010], [G-Cluster, 2012], [StreamMyGame, 2007], [T5-Labs, 2007]. In this configuration, only the server maintains a representation of the game model. The actual game is stored, executed and rendered on the server. clients simply gather inputs from users and send them to the server. clients receive video streams from the server and display them to their screen. When launched client contacts the server to establish a connection and start the game. As explained in [D. Barièevicæ et al., 2011], there are usually two connections for each client: one connection to send the user's controller inputs to the server and another connection to receive video stream from the server. Figure 2.3 shows this game loop.

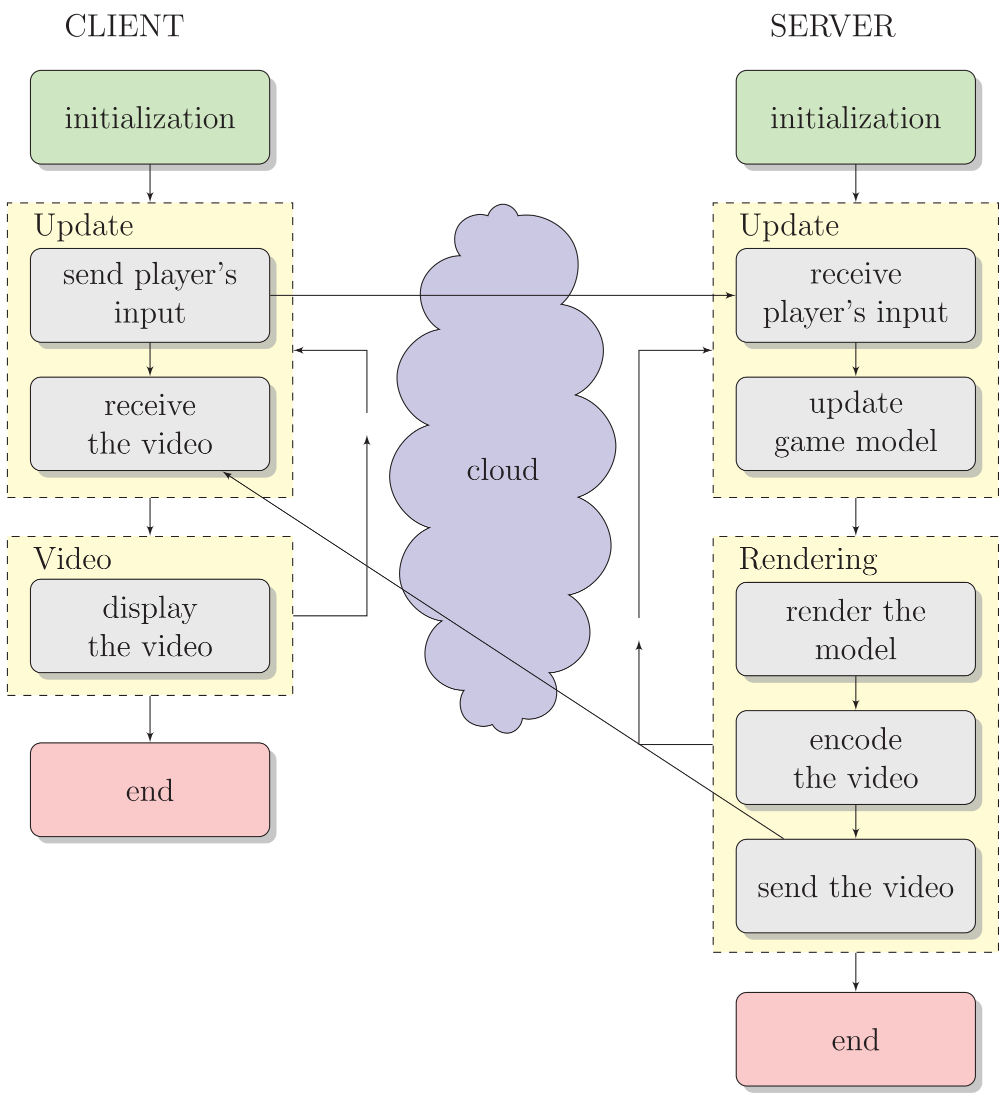


Figure 2.3 – Cloud game loop: Video streaming

The client game loop runs two main stages: update stage and video stage. On the update stage, the client polls for input events from the user. When an event is received it is written into a packet and sent to the server. The client then receives packets from the server, containing the video stream.

On the video stage, when a complete frame has been decoded, it is displayed in the user's screen.

A typical server game loop also runs two main stages. An update stage that receives inputs from the client and passes those inputs to the game model. A rendering stage that captures frames from the game, encodes them into a video, and then stream the video output to the client.

2.3.2 The game objects replication approach

Game objects replication approach uses the same game loop as the client-server architecture presented in figure 2.2. Both client and server have a representation of the game model and synchronization packets are sent repeatedly. The difference between a client-server with the cloud gaming paradigm and a classic client-server setup are summarized as follow:

— **Client's update stage:** The idea of cloud gaming is to allow thin clients to run complex games by limiting the amount of computation required on client devices. So the client's update stage should do as less computation as possible before handing the game model for rendering. Therefore, there is no client-side prediction involved here.

— **Server's update frequency:** The frequency at which the server sends game updates has to be fast enough to be greater than the client's rendering frame rate. Otherwise the client will notice a lag in the responsiveness of the game, in the motions of game characters and in the smoothness of the game in general.

  For example, for a game running at 60 FPS, normal client server games can have the server sending an update every 200 ms (5 Hz without lag problems. This is possible because the client uses techniques such as interpolation, extrapolation and client-side prediction to predict game states. With the cloud gaming paradigm, there is no prediction. The client simply updates the objects' states with the received packets and proceeds to the rendering stage.

  So the server will need to send update at a rate closer to the client's frame rate(something greater or close to 60 Hz, or one update every 16,66 ms). We can immediately see that the commonly used server update frequency for classic client-server games (5 Hz) is not appropriate for a lag free game using cloud gaming paradigm. A higher frequency and therefore more bandwidth are required on server side for an acceptable QoE.

Both video streaming and objects replication have benefits when it comes to delivering a good game experience in online games:

— With video streaming, games are accessible everywhere on different types of devices. Even devices with low CPU capabilities since clients only plays a video stream. No extra client side processing is required. The inconvenient is network bandwidth needed to receive the video stream.

— With objects replication, games are also accessible from everywhere, but client devices have to be powerful enough to do the processing required to render video displays using their own GPU. The price paid in computational power is rewarded by some gain in bandwidth requirements. In fact replicating game objects on clients requires much less network resources than streaming video frames.

In this thesis we aim at using the object replication approach in the cloud gaming paradigm by assuming no extra client-side processing. This way, clients with low processing power can run games using less bandwidth than with traditional video streaming games.

2.4 Quality of experience

User experience is generally referred to when evaluating the quality of the interaction of a user with an interactive system. In fact the ISO definition refers to user experience as the result of user-system interaction that can be influenced by the system, the user and the context of use [ISO, 2010].

Many concepts are used in the game community to refer to player experience. Developers often describe a game and gameplay with words such as fun, feeling, scary, atmosphere, feel, immersion, presence, satisfaction, having a good time etc. All these words refer to subjective characteristics and, just like player experience, are not simple components that can mechanically be built into the game.

A commonly cited user experience characterization is the one introduced by Mihaly Csikszentmihaly in 1991: flow zone. Flow zone is a psychological state between anxiety and boredom, where the user experiences a feeling of optimal focus and engagement on an activity with great level on enjoyment [Csikszentmihalyi, 1991]. Some of the major elements of flow are: challenge, direct and immediate feedback, concentration on a task, sense of control, loss of self-consciousness and altered sense of time. Even though flow refers to a subjective psychological state, most of its elements can be objectively evaluated and correlated to perceived quality of experience.

Quality of experience provides an assessment of human expectations, feelings, perceptions, cognition and satisfaction with respect to a particular product, service or application [Crespi et al., 2011]. User's QoE of a service can evoke a wide range of emotions and attitudes. These emotions and perceptions of human experience make it very challenging to measure and analyze QoE factors with precision and accuracy. In general, there is a direct correlation between the user's QoE and the user's experience. In video games for example, player's QoE is often considered as a more objective equivalent of player experience. Many studies have tried to objectively characterize player's QoE [Chen and El Zarki, 2011], [Verdejo et al., 2010], but the individualistic nature of some aspects of human experience makes a completely objective evaluation approach to QoE nearly unachievable. This is the reason why most evaluation models of QoE [Chen et al., 2009], [Chang et al., 2010] have resorted to the subjective evaluation methodology called the MOS (Mean Opinion Score) rating test [Rec, 1996], using players' opinion rating to assess players' QoE.

Properties defining QoE can be multidimensional. For example, QoE for VoIp conversations includes criteria for sound quality such as voice loudness, noise levels, etc. In online gaming, the main quality of experience properties that are found in the research community [Chen et al., 2009; Ida et al., 2010; Chen and El Zarki, 2011] are:

— **interactivity** refers to the ability of the game to interact with the player, allowing a two way flow of information between them.

— **responsiveness** refers to the time taken for the system to respond to an event. This property represents the player's perception of the game process.

— **consistency or fairness** refers to the degree of difference in the presentation of the virtual world among all players for a multiplayer game.

— **smoothness** refers to how smooth the game visuals are. To what extent motions in the game are smooth and steady. The number of FPS can play an important role in determining the smoothness of a game.

— **precision** refers to the degree of accuracy required to complete an action successfully.

Fairness or consistency represents a shared property owned by all players in the game, whereas interactivity, responsiveness, smoothness and precision are personal properties related to each player.

Among those properties, the ones reported most frequently are no doubt interactivity and responsiveness, and both can be heavily influenced by network parameters such as network delay and packet loss. In fact, a lag perceived in the interactivity and the responsiveness of the game as well as the smoothness for an online game can be caused by various sources. The most recurrent of them will be disclosed in next section.

2.4.1 Lag in online games

In networked games, there is a misconception about terms such as delay, latency and lag. In general, *"delays"* are due to hold-ups in data being sent (e.g., packet processing, queuing due to network congestion, or retransmission of data due to packet loss). On the other hand, *"latency"* basically means "inherent delay" (e.i., delay caused by underlying technology or network being used). It represents physical attributes of the network medium. For example the latency of a Wide Area Network (WAN) is always greater than the latency in a LAN. The overall transmission delay is therefore a combination of the network delay and the underlying network latency.

"Lag" is used to characterize a perceived unexpected delay. For example, player's input transmission delay can be extended because of built-in latency of the network itself and as a result the player can experience a lag in his/her interaction with the game.

In general, lag is strongly correlated to the time it takes to run the game loop. Two main types of lag are most reported in interactive applications such as games: update lag and input lag.

The main difference between both lags is that, the update lag is not directly related to an action from the player, whereas the input lag is mostly associated with an action performed by the player. This differentiation is supported by Matthias Wloka in virtual reality in general [Wloka, 1995]. He defines input lag as: *"the time between when a user performs an action and when the application displays the result of that action"*. Input lag is also known as the end-to-end lag (see figure 2.4). It hence influences game's responsiveness and interactivity, which (as we saw in the previous section) are very important for player's QoE and enjoyment of the game.

The update lag is more about the delays in the execution of the game logic. It concerns the server's update phase where actions of game objects are executed and game model is updated. It is also referred to as the *spectator lag*, and can also be found in simulations, since there is no user-system interaction.

As human beings, we are extremely sensitive to lag. For instance, depending on the task and the surrounding environment, a lag of as little as 100 $\mathtt{ms}$ can degrade human performance. In the same token, a lag exceeding 300 $\mathtt{ms}$, causes the human to start to dissociate his/her movements from displayed effects, thus destroying any flow state [Held and Durlach, 1991].

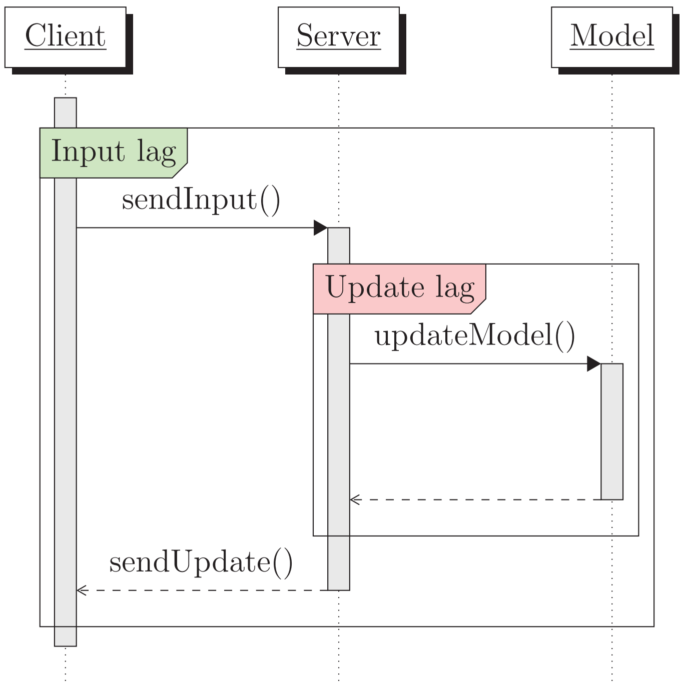


Figure 2.4 – Input lag and update lag

The example of figure 2.5 illustrates input lag as it applies to targeting and shooting at another character with an instant-hit weapon in an online game.

Lets imagine a scenario where a player with a weapon is trying to shoot at a game character (Xaero in the figure). At the same time, s(he) is experiencing 200 ms of delay. At this moment of the game, because of lag, the character will be seen at position (1) on player's screen instead of position (2) where it actually is on the server. If the player presses the attack button, his/her command with the button pressed will reach the server when the character is at position (3). Consequently, the player will miss it. To be able to hit the character, the player will have to target the position (3) even though s(he) is seeing it at the position (3).

With this example, we can see how a lag can affect the real-time nature of a game. The responsiveness of the game is degraded and players with no precise control of the game, will perceive their experiences more negatively.

2.4.2 Causes of lags

To understand the lag experienced by end-users, it is important to determine the response time of network games. The interactive response time is defined as the elapsed time between when an input of the player is captured by the system and when the result of this trigger can be perceived by the player. [Choy et al., 2012] formulated the interactive response time T of a cloud game. This overall delay includes several types of delays:

Formulation:

$$T = T_{client} + T_{network} + T_{server}$$

— T_{client} is the playout delay, which refers to time spent by the client to send controller inputs, receive game updates and display the resulting scene on the screen. Client's

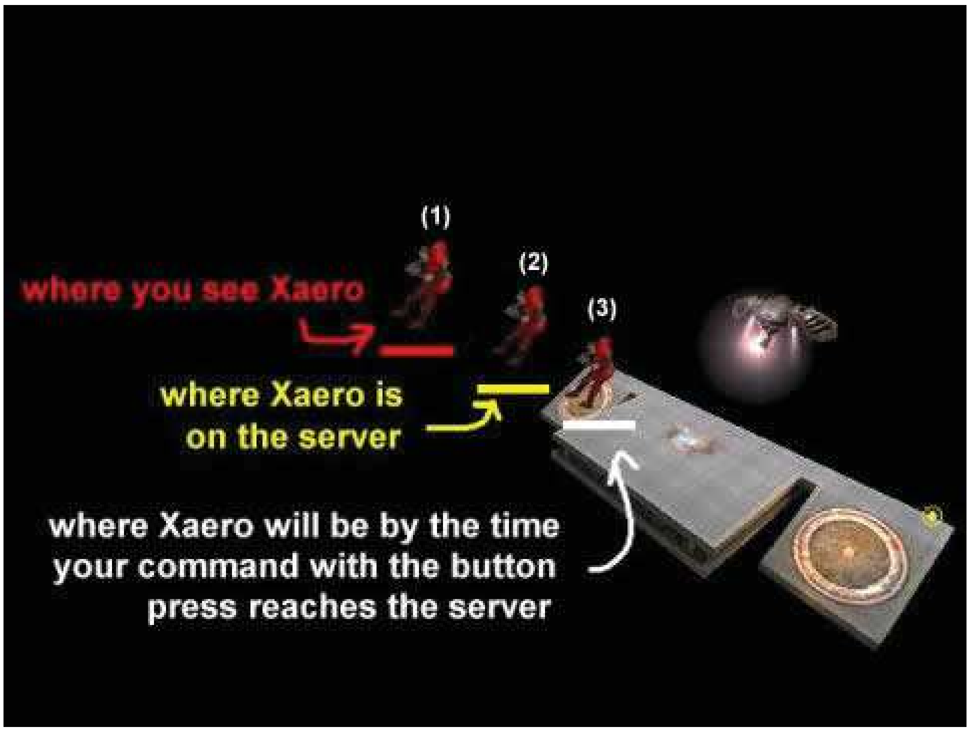

(1)
(2)
(3)
where you see Xaero
where Xaero is on the server
where Xaero will be by the time your command with the button press reaches the server

Figure 2.5 – The Effects of lag on targeting [Unlagged, 2002]

hardware is responsible of $\mathsf{T}_{\mathsf{client}}$; it is the notorious ”built-in 50 ms lag”. This delay can be the consequence of bursty graphics (Central Processing Unit (CPU)/Graphic Processing Unit (GPU) Interaction), bursty application (CPU/Memory), hard-drive accesses, memory management/swapping and background apps/tasks.

— $\mathsf{T}_{\mathsf{network}}$ is communications delay referring to the time it takes for data transmission from the client to the server and vice versa. Internet Service Providers (ISP) and data centers of cloud providers are responsible for this delay. Communications delay is affected by poor client bandwidth, long routes (hops), high latency, packet loss, faulty equipment, ISP congestion, backbone congestion and bursty datagrams.

— $\mathsf{T}_{\mathsf{server}}$ is the processing delay which refers to time spent by the server to process incoming information from the client, to update the game model and to transmit the game state back to the client. This delay is affected by a number of computational parameters: input processing (validation of user actions), per-user calculations, output processing (per-object message passing), server network bandwidth and back-end interactions (e.g Database).

[Beverly, 2009] illustrates the distribution of these main causes of lag. According to this study, playout delay, communications delay and processing delay are in average respectively responsible for 35%, 30% and 35% of the overall lag experienced by players.

These are average values that change from one game to another. In addition, studies on traditional gaming systems have found that different styles of games tolerate different thresholds for maximum latency before the QoE begins to degrade. [Claypool and Claypool, 2006] introduces a novel categorization of the effects of latency on different player actions based on two salient action properties: the *precision* required to complete the action and the *deadline* by which the action must be completed. Using this categorization, M. Claypool et al. classified games into two interaction models:

— The *Avatar* model, where player controls a single character, and interacts with the game world through that character. Generally, the character exists at a particular location in the virtual world and can only influence its local vicinity. First person shooter (FPS) games, role-playing games (RPGs), action games, sports games and racing games are all examples of game genres that have an Avatar interaction model. These games have actions with higher precision and tight deadlines, making them sensitive to even modest latencies (e.g, first person shooter games, latency around 100 ms).

— The *Omnipresent* model, where player has the possibility to view and interact with different aspects of the game world simultaneously. The player is said to be omnipresent controlling the entire set of resources under his/her control. The player's actions, thus have a more global influence than do actions in an Avatar model. Real-time strategy games (RTS), and construction and simulation games (CMS) are examples of game genres with Omnipresent interaction model. These games have actions with lower precision and loose deadlines. Thus are nearly impervious to typical Internet latencies (e.g, strategy games, latency around 200 ms).

2.4.3 Attempts to fix the lag

Both players and game developers are trying to find solutions for the lag damaging online gaming experience. Figure 2.6 illustrates different attempts to limit delay and its effects on QoE.

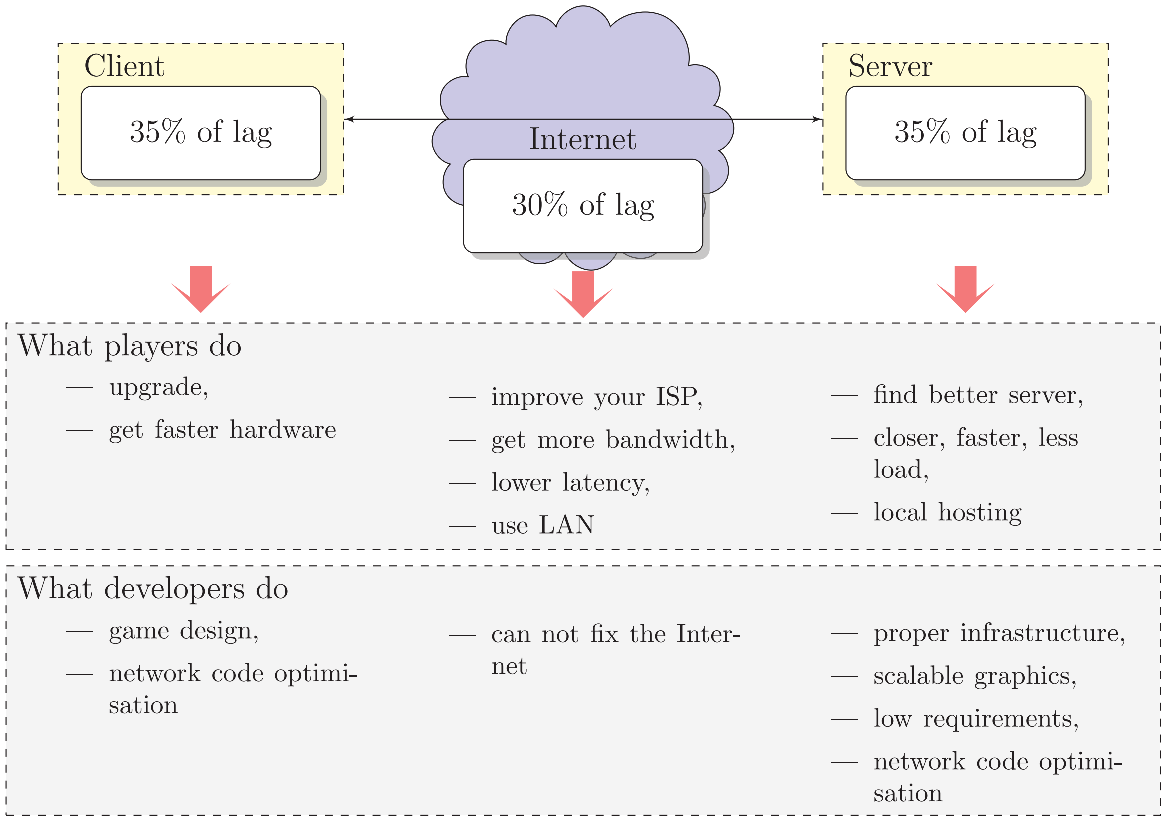


Figure 2.6 – Fixing lag

1. **Players:** since they can only control their devices and the connection they can have, the two obvious solutions for them are:
   - *For the playout delay:* players can simply upgrade their devices by acquiring faster hardwares (CPU and GPU).
   - *For the network delay:* players can acquire more bandwidth from their ISPs for lower latency or host the server on a LAN for less latency.
   - *For the server delay:* players must find better game servers. These are servers that are faster, have less load and are closer to their locations to minimize the network latency. The servers may simply choose to drop clients with high latencies in order to avoid having to deal with the resulting problems. They can also decide to deploy their server locally on a LAN.
2. **Game developers:** they have no control over the internet, nor the client hardware, but they have a range of techniques that they use to optimize their games against lag issues. They can use:
   - *Game design:* here two design principles are possible; structure the server system to avoid lag and latency (restrict number of players par game server, use asynchronous gameplay, partitioning/sharding[3], use lots of servers) or hide the latency by using animation and prediction.
   - *Network code and protocol:* the network code can implement some bandwidth conservation techniques and some delay compensation techniques such as dead reckoning. The description of these techniques will be presented in next section.
   - *Server infrastructure:* choose the ”best” server infrastructure possible to host the game.
   - *Scalable graphic system with low system requirements:* MMOGs such as World of Warcraft tolerate very high latencies (500ms+) and have very low system requirements because of their Last Gen 3D graphics [Dean and Kim, 1999].

Networking techniques

To optimize the network traffic, several state of the art techniques have been developed empirically and are implemented in most networking game engines. They can be divided into two groups: bandwidth conservation and delay compensation techniques.

Bandwidth conservation

This category gathers techniques that minimize bandwidth usage and cope gracefully when bandwidth is constrained. Most known techniques are:

1. **Encoding and compression:** Compression means reducing message's size by finding a more compact way to represent it. Making messages smaller enables to speed up transmissions by reducing bandwidth requirements [An et al., 2012]. In video streaming for example, the performance of video encoding and compression algorithm is very important in reducing transmission delay. The most used standard

3. The partitioning and sharding techniques exploit the geography of the game world, decomposing the game into different areas, each of which can be mapped to a hosting server.

of video compression in game systems is H.264/MPEG-4[4]. H.264 specifies how compressed video should be decoded; it does not specify how it should be encoded, thus allowing encoder vendors to be innovative. There are many H.264 compliant video encoders, with different features and strengths. One commonly used H.264 video encoder is X.264[5]. It is a free award winning video encoder implementing the H.264 standard.

2. **Aggregation**: Aggregation is a technique used to reduce the overhead associated with each new transmission. Since messages sent by games are often small, before being transmitted, they are packed and merged in larger ones thus reducing overhead of sending multiple small messages rather than one coarse-grained message.

Delay compensation

The delay compensation method, also known as *dead reckoning* aims to reduce lags by compensating delays. In fact, as explained in [Pantel and Wolf, 2002], delays are not reduced directly, but their impacts on QoE can be limited.

In general, dead reckoning calculates a position obtained by measuring or deducing displacements from a known starting point in accordance with the motion of an object [Judd and Levi, 1996]. This method is used in many domains such as navigation, interactive simulations, military applications and online games.

In online gaming, dead reckoning refers to a method used to predict the state of a game object (position, velocity, acceleration, etc...) using known states. Attempting to limit the bandwidth required by the game, this scheme utilizes a reduced frequency in sending state updates while compensating the lack of information with prediction techniques [Valve, 2009]. Obviously, predicted movements and actions are not always trustful and correction of the eventual errors is paramount to maintain a consistent game state. These eventual restoring actions can further impact player's QoE.

On the client-side, prediction primitives (interpolation and extrapolation) and animation are commonly used by the dead reckoning technique to hide the delay. Here is a brief description of each of these notions:

1. **Interpolation**: interpolation is a method of constructing new data points between a range of discrete set of known data points. In online games, clients render three or more frames per update message. Interpolation systems prevent unsteady and jumpy motions caused by network delay, by buffering server updates and playing them back in increments, with smoothly interpolated gaps between.

   It can also protect against malfunctions or irregularities caused by packet loss for example [Valve, 2004].

2. **Extrapolation**: By contrast with interpolation, extrapolation is the process of estimating a value of a variable using its last observed values. It is therefore subject to greater uncertainty. In online gaming, extrapolation attempts to estimate a future game state. As soon as a packet from the server is received, the position of an object is updated to the new position. Until next update is received, positions are extrapolated based on past positions. When a new update is received, extrapolated positions may be corrected.

4. http://tools.ietf.org/html/rfc3984
5. http://www.videolan.org/developers/x264.html

3. **Client-side prediction**: It is similar to extrapolation, in that, client tries to guess the state of an object owned by the server. In case of simple extrapolation, however, the client does not have the benefit of user inputs. In fact, client-side prediction is always conducted in response to user control inputs. It uses the inputs of the user to make a better guess about the location of the user-controlled object. Thus, the client does not just send the user input to the server and waits for an updated game state in return. In parallel with this, the client also predicts the game state locally, and gives the user feedbacks without awaiting an updated game state from the server. Basically, the way this prediction works is that, the client performs the same simulation that the server will eventually perform on that user input [GarageGames, 2009].

4. **Animation**: This technique makes it possible to reduce the lag perception using animations until receiving server's updates. For example, in most MMOs, when an "action" button is pressed to perform an action, a message is sent to the server for the action, and an animation is played to gain some time until receiving the server's acknowledgement.

2.5 Adaptation

The term adaptation can be found in different domains with different semantics. But in software systems adaptation can be defined as a process by which a system is able *"to adapt itself efficiently and fast to changed demands"* [Andresen and Gronau, 2005]. Among the common perspectives of adaptation and adaptive systems, some definitions refer to adaptive systems as systems with the ability to change their own characteristics automatically according to the user's needs [Oppermann, 1994].

Other definitions are more general and do not necessarily associate system adaptation with user's needs or preferences. In automatic systems for example, adaptation is considered as a key feature in developing self-managing, self-configuring, and self-regulating systems. As a matter of fact, [Dobson et al., 2006] decomposes the system adaptation in 4 verbs that sum up the whole adaptation process: (i) *"collect"*: information is collected from a variety of sources including user needs, network sensors, user context, application requirements etc. (ii) *"analyze"*: collected information are analysed and transformed to a model used as a basis for adaptation decisions. (iii) *"decide"*: decisions are taken using techniques such as risk analysis, hypothesis generation, etc. (iv) *"act"*: the system acts upon the decisions and changes its behaviour accordingly. Depending on the system, these changes can range anywhere from interface presentation, to core system behaviours.

2.5.1 Adaptation in video games

In video games, adaptation is often defined as the ability to dynamically change game objects' behaviors in accordance with changes in game information. Games generally use adaptation to provide a better experience to players. This can happen through game difficulty adjustment, game agents behaviors adaptation, game context adaptation, unpredictability and challenge, personalized gameplay, etc.

Lopes and Bidarra propose a steering method to guide the adaptation algorithm in deciding what, when and how to adapt the game [Lopes and Bidarra, 2011]:

— *"the what"* is the reason or the information that starts the need for adaptation. It is also referred to as adaptation input. In difficulty adjustment for example, the adaptation first recognizes that consecutive failures are a sign of high difficulty. The difficulty level of the game must then be decreased [Hocine, 2013].

— *"the when"* identifies when adaptation decisions are made. This can be online (during the game session) or offline (before or after the game session). As example of online game adaptation, the adjustment of game characters' behaviours for better usage of scarce resources such as computation power is introduced by Mahdi et al. [Mahdi et al., 2013].

— *"the how"* determines actions taken by the game as response to the recognized information. This depends on the game and the game elements affected by the adaptation. In the game difficulty example the result will be a generation of a less difficult game level, whereas in the behaviour adjustment example game characters will adopt simpler behaviours (with less computational requirements).

2.5.2 Communications adaptation

Nowadays, with the arrival of cloud gaming, real time game traffic over unstable networks such as wireless and mobile networks is growing at a fast pace. However, inadequacy between networks' capacities and amount of real time communications generated by games are implying the need for a more dynamic and adaptive communications model.

As a matter fact, communications adaptation is not new to the online application world. In autonomous systems and wireless networks for instance, the main objective of communications adaptation is generally to avoid network congestion as much as possible [El Masri et al., 2011], [Dobson et al., 2006]. In this model, adaptation inputs are collected through network sensors and are used to estimate network congestion and trigger appropriate adaptation actions.

The same model is applied to networked games. Most game libraries offer game developer services to adapt their games to network conditions. For example, Torque Network Library (TNL) [GarageGames, 2009] implements primitives to monitor the amount of bandwidth being used and enables the developer to build an adaptation strategy by offering virtual methods specifically for reacting to network resources changes. An adaptive flow control enables TNL to limit the bandwidth by adapting to the number of packets currently in the network.

Limiting the amount of communication generated by the game, enforces developers to use compensation techniques such as dead reckoning to replicate game objects' behaviours. So the adaptation here is tailored to ensure an efficient distribution of a limited resource. This implicitly brings forth the necessity of prioritization or scheduling among game objects needing network communications for state updates. At the decision stage, the adaptation process will therefore use these priorities to determine what game objects need synchronization the most. The objective being that all objects do indeed meet their communications requirements.

There are two common ways to assign priorities to game objects: static priority and dynamic priorities. There are advantages and disadvantages to both paradigms, so that one is more appropriate for certain games while the other is more appropriate for different games:

— **The static priority:** game objects' priorities are set before the game session by the developer and do not change during the game. The (pre-game) static priority assignment is best for games with simple game mechanics where game objects' behaviors or role do not change during the game. Some of the main game networking engines using static priority are, Raknet [JenkinsSoftware, 2011] and Unreal engine [EpicGames, 2012].

— **The dynamic priority:** in games with constantly changing game mechanics where for example an object might be very important at a moment and becomes quite unimportant a minute later, the idea here is to calculate the priorities during the game session. The goal is to dynamically adapt to the progression in the game. Examples of networking engines supporting dynamic priority include Zoidcom [Rüppel, 2011] and TNL [GarageGames, 2009].

2.6 Conclusion

In this chapter, we presented the pre-requisites needed to understand the rest of this document. We began by defining essential concepts such as video games, online video games. We showed the difference between two main cloud gaming approaches, namely video streaming and objects replication and then showed how the quality of experience is characterized in online video games in general. We presented the lag as the principal problem of QoE in online games and ended by defining the concept of adaptation and presenting different adaptation mechanisms in network communications for online games.

CHAPTER 3

Adapting Communications in Cloud Games

Contents

Two main approaches are currently used in cloud gaming technologies: video streaming and objects replication. In both approaches, there is usually a fixed synchronisation rate, and this parameter is set manually and sometimes is ad-hoc, therefore requiring considerable development costs. Other characteristics of these approaches include:

— With video streaming, there are high bandwidth requirements but with the advantage of ubiquity and using thin clients. Therefore there is a need for better alternatives to support low and dynamic network conditions for better player's QoE.

— With objects replication, games do not require as much bandwidth as in video streaming. But there is also a need to minimize network resources usage and to dynamically adapt to changing network parameters such as capacity limits, delay and packet loss.

As seen in previous chapters, network latency is an important problem to deal with in the context of cloud gaming given that cloud games require more bandwidth than classical online games. Consequently, dealing with QoE issues caused by network conditions is a key problem and appropriate solutions need to be found.

To deal with this problem, two general approaches can be defined:

— lowering bandwidth requirements of cloud games. Thus, players with low bandwidth networks can benefit from an acceptable QoE.

— being aware of the network conditions to adapt the game itself. In case of network congestion for example, when the game is aware of network conditions, some elements of the game structure can be adapted to consume less network resources while maintaining an acceptable game experience.

The objective of this chapter is to present a survey of adaptation approaches used in cloud games to deal with network constraints in order to maintain acceptable QoE. We start by stating the inclusion criteria; then the analysis framework is presented. We continue by presenting the analysis of selected works followed by an overall discussion.

3.1 Inclusion criteria

This survey involves studies found in the domain of game development and interactive applications in general. We focused on scientific literature. Various keywords have been used for search including: QoE, communications adaptation, game lag, bandwidth control, online game, networking, game engine, game libraries, game middlewares, video streaming and cloud gaming. The publication period was set from 2008 to 2014.

From this pool of studies, we have selected those highlighting efficient communications adaptation in client-server video games with cloud gaming paradigm.

We are interested in game middlewares with generic models and patterns. It is important to state that, the lack of openness of commercial systems makes it difficult to find valuable information and scientific publications. Hence we resorted to a broader scope of information, including domains such as video streaming which is predominant in the cloud gaming research community. The sources of this survey come from academic papers as well as studies and information published online on game developers' websites, wiki and blogs.

3.2 Analysis framework

As any control process, the communications adaptation receives some inputs and produces control outputs. Inputs are variables defined for the player, the network or the game itself. The output can affect game elements or communications parameters. In the following we present the analysis criteria considered in this survey: adaptation inputs, affected elements, message scheduling and models.

Adaptation inputs

With this criterion, we are interested in finding what information is collected to trigger the adaptation process. In communications adaptation, two categories of inputs are often found:

— *Network inputs*: the probed information here is related to the network layer. Examples of adaptation inputs are: bandwidth, packet loss rate, latency and delay. Any change (exceeding a specified threshold) in these parameters usually initiates the adaptation process.

— *Game inputs*: these are game data used by the adaptation process. For instance, a commonly used information in object replication is the priority assigned to each game object. In this case, when the priority of an object is changed this is reflected by changing the amount of communications resources attributed to that object.

Affected elements

After analyzing collected inputs, the adaptation process makes decisions. Depending on the cloud gaming approach, different elements can be affected by these decisions. This criterion lists all elements potentially affected by adaptation decisions.

For video streaming the adaptation often adjusts game's frame rate and/or stream bit rate. This is how a fluctuation in the network conditions alters the resulting game traffic. For object replication, the adaptation usually acts on the rate control and/or on the flow control.

Rate control is the process of managing data transmission rate between two nodes to prevent a fast sender from overwhelming a slow receiver. This rate control can be applied at two levels: (i) at object level for games supporting objects replication with specific synchronization rate for each object *"object rate control"*; (ii) at network level where all game objects are synchronized at the same rate *"network rate control"*.

Concerning flow control, the objective is to adapt the number of packets currently in the network to the available bandwidth, in order to limit the risks of congestion. This is accomplished using *a congestion window*.

Depending on the streamed data and the nature of the game, different cloud gaming approaches have different requirements in terms of bandwidth. These data are used to replicate game sounds and the visual game scene. In this thesis, we are focusing on the replication of the visual game scene, while leaving game audio data replication for future works. Identifying the minimum bandwidth necessary for the player to play the game using the adaptation approaches, is critical in determining to which extent each approach is greedy in network resources.

Message scheduling

Different synchronization messages have different impacts on lag and different relevance levels for a particular player. For this reason, several game networking engines implement scheduling protocols to prioritize messages delivery.

Latency-sensitive messages for example, need to arrive before all other information despite the possibility of delay and packet loss. One way of dealing with this scenario is to associate each message with a priority value, thus a message with high priority (latency-sensitive) is sent before, and more often than a message with low priority (latency-tolerant).

To complement the priority of game messages, most game networking middlewares provide different levels of reliability to different game messages. In general, highly reliable, totally ordered messaging provides high fidelity. Multiple levels for ordering and delivery are possible using Transmission Control Protocol(TCP) [1] or/and User Datagram Protocol(UDP) [2].

In some multiplayer game scenario, messages generated are relevant only for a small fraction of players. Therefore, implementing an area-of-interest scheme for filtering messages, as well as a multi-cast protocol, could be beneficial. With Interest Management technique [Morse et al., 2000], every packet is scheduled for transmission with the nodes that really need to receive it and, consequently, both the traffic and processing burden at each node are reduced. In addition to priority and reliability, we will also look at interest management in our analysis of the state of the art.

Models

Adaptation methods present in cloud gaming platforms follow different models or patterns. In addition, the way the game is structured in terms of game entities, and the way those entities interact and progress for the accomplishment of the game objective, can be organized and modeled for fast prototyping and reuse.

Hence we are interested in finding whether the gaming platforms enable any of these three types of models:

— *Adaptation model*: depending on elements affected by the adaptation algorithm, you can have either a rate-based adaptation (bit rate, frame rate, synchronization rate) or a flow-based adaptation.

— *Game model*: some objects replication game engines provide developers with a game structure, generic enough to be reused in other games.

— *Priority model*: refers to any model for automatic priority adjustment in the game system with the objects replication approach.

3.3 Related work

This section presents each selected work of the state of the art.

[Rüppel, 2011]

Created in 2002 by Jörg Rüppel, Zoidcom is a network library designed for action games. Zoidcom provides features for automatic replication and synchronization of game objects over a network connection. With the latest release in 2011, Zoidcom is freely available for non commercial use.

Zoidcom is used in several game projects such as *My World*[3] and *Ethereal*[4].

Ethereal, is a team-based online-multiplayer shooter game that can handle more than 20 players [Agar, 2012] (see screenshots in figure 3.1)

For message scheduling, Zoidcom supports object level priority and priorities are set manually. Priorities allow objects with different characteristics to have different synchronization rates. Zoidcom provides rate control functionalities using a dynamic bandwidth

1. http://www.rfc-base.org/rfc-793.html
2. http://www.rfc-base.org/rfc-768.html
3. http://www.sam-ko.com/en/projects/33-my-world-is-now-a-open-source-project
4. https://www.youtube.com/watch?v=455UnqQEqBU

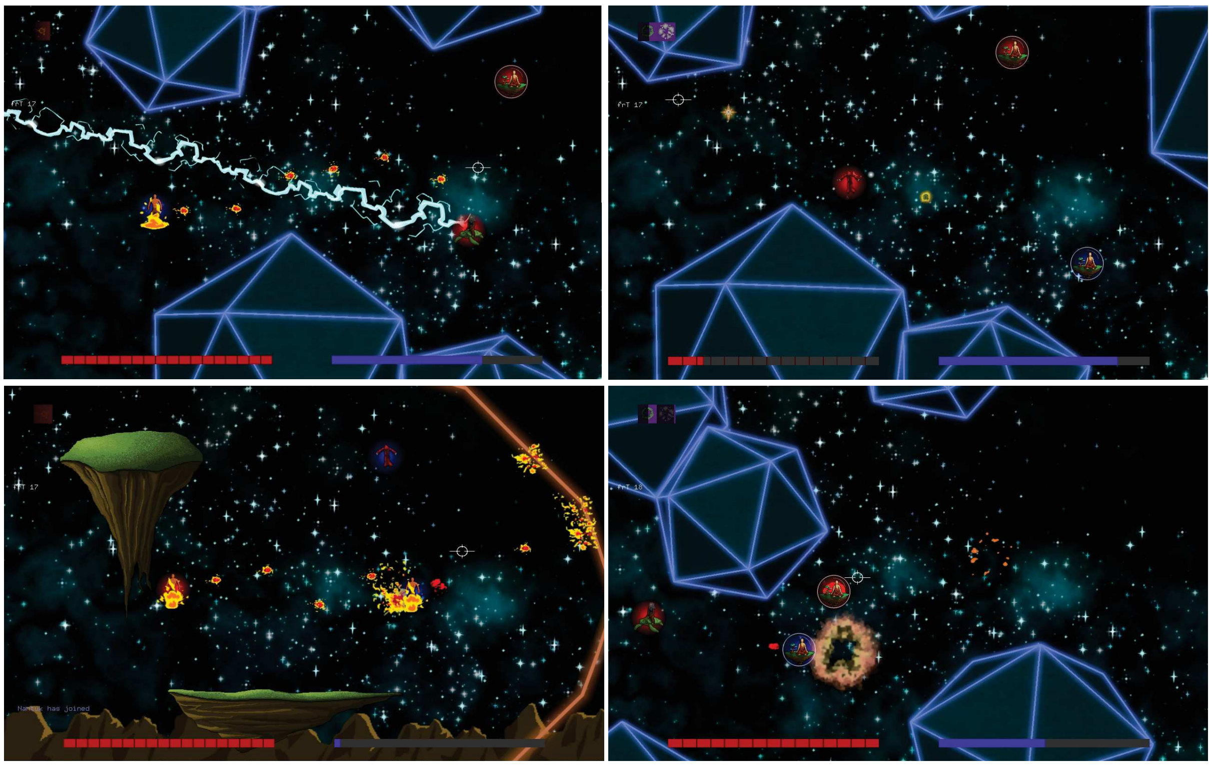


Figure 3.1 – Screenshots of Ethereal: 2D multiplayer shooter game [Mogwai, 2012]

limitation and distribution, as well as a control of minimum and maximum synchronization frequency for each data item. Object dependencies are used to define the order in which the update messages are transmitted.

To avoid lags, Zoidcom incorporates latency reduction techniques such as dead reckoning, custom and automatic interpolation between state updates as well as extrapolation methods. Field of view is used as an interest management criterion, to adjust object relevance per client. Zoidcom also implements object subscription groups for different, distinct game areas for example. It uses different reliability levels for different communications types such as file transfer, session discovery, connection statistics and direct communications.

Concerning adaptation, Zoidcom implements functionalities that enable the traffic generated by the game to adapt to the available bandwidth.

In fact with ping[5] measurements, Zoidcom is able to accurately monitor available network resources and adjust the game's traffic accordingly.

Zoidcom implements both rate-based and flow-based adaptation. For the rate-based adaptation, each object's rate adapts to the current network conditions, varying within the ranges specified by the developer. As regards to the flow-based adaptation, the sending mechanism sends individual object updates from highest to lowest priority until bandwidth limits are reached – at which point low priority unreliable messages are dropped, and low-priority reliable messages must wait.

5. Ping is one of the most widely used active network performance measurement tool. It measures the reaction time of a connection by sending an Internet Control Message Protocol (ICMP) echo request packet and reporting the time it takes for the sender to receive the ICMP echo reply packet for its request. Ping is measured in milliseconds

Zoidcom is very generic, and it does not provide a specific structural model that developers can use to build their games.

In addition, with Zoidcom there is no explicit reusable model for priorities adjustments, thus the developer has to implement priority adjustment on his/her own if (s)he needs it.

[JenkinsSoftware, 2011]

RakNet is a high-performance network library designed for games. Like Zoidcom, this middleware provides features for efficient objects replication in online games with the objective of reducing lags. RakNet supports many commercial platforms including the Playstation 3, Xbox 360, PC, iPhone, and Android. RakNet has been integrated as a network layer in many popular middleware engines such as Unity [6], Steamworks [7] and is used by leading studios such as Foundation 9 [8] and Sony Online Entertainment [9] [IGN, 2011]. It is also used in several commercial multiplayer projects such as: The Slant Six Games's multi-platform project [IGN, 2011] and *Drakensang Online* [Gamasutra, 2011] by Bigpoint [10].

Network communications is provided by two interfaces: a UDP interface and a TCP interface with 10 reliability levels. Raknet's approach allows developers to handle their own priority scheduling and supports four different priority levels to choose from: IMMEDIATE_PRIORITY, HIGH_PRIORITY, MEDIUM_PRIORITY and LOW_PRIORITY. Packets with each of these priorities are sent approximately two times more often than packets with the priority below it.

Raknet does not state how packets with different priorities are processed by the application. Instead, it provides several abstract methods that can optionally be implemented by the game developer to define how the application processes messages of different priorities.

Raknet can monitor available bandwidth and uses object level rate control to adapt to current network conditions.

Just like Zoidcom, Raknet's object rates vary within an interval specified by the developer. In addition, Raknet implements an adaptive flow control using a window size to limit bandwidth. In fact a congestion window is used similarly to the TCP flow control, to monitor the amount of packets in the network link. The window size is used as an indication of how many unreliable packets must be dropped(high drop probability). Raknet drops all unreliable messages in case of congestion (when the window is full), to make room for higher priority reliable messages (with low drop probability).

Raknet does not require any specific structural model for the game and since priorities are static there is no incentive for priority adjustment during game sessions.

[Salzman, 2014]

Created by Lee Salzman in 2002, for Cube game engine [11], ENet is an objects replication system with a claimed simple, flexible and consistent C/C++ API. It is used by

6. http://unity3d.com/
7. http://www.steampowered.com/
8. http://www.f9e.com/
9. https://www.soe.com/
10. http://www.bigpoint.com/
11. http://cubeengine.com/

games such as the commercial SilentWings flight simulator [12] The SilentWings flight simulator enables players to practice their flight techniques; play online with other pilots in virtual skies, replay real-flight GPS logs (see figure 3.2). With up to 32 players online on a selection of several available servers, SilentWings enables also inter-player communications via online chat. It also features, advanced physics-based algorithms for accurate simulation of aircraft movements even on poor internet connections [SilentWings, 2006].

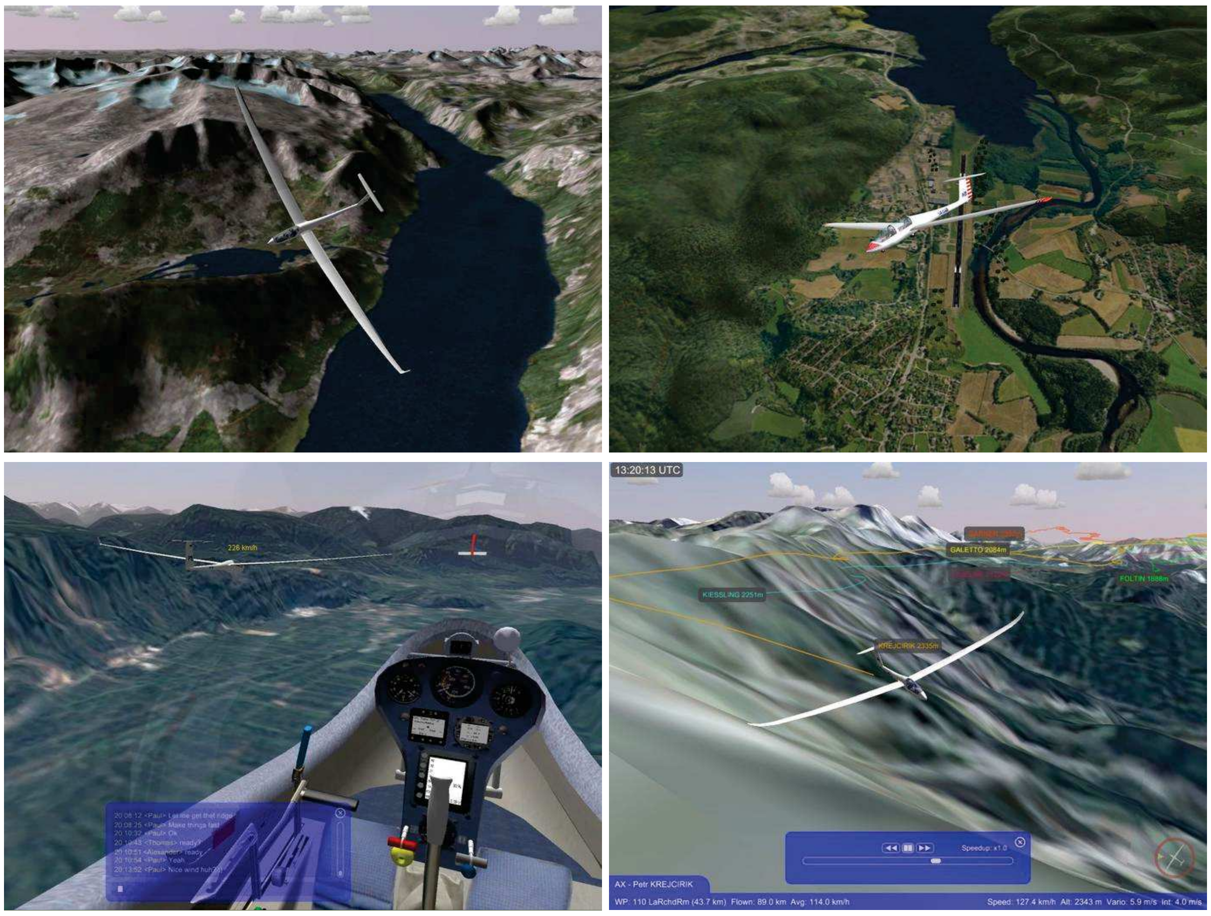


Figure 3.2 – Screenshots of SilentWings flight simulator gameplay [SilentWings, 2006]

Enet is distributed as an open source library and provides both reliable and unreliable transmission of messages. Each message is assigned a flag, with any combination of the following flags: ENET_PACKET_FLAG_RELIABLE, ENET_PACKET_FLAG_UNSEQUENCED, ENET_PACKET_FLAG_NO_ALLOCATE. Each flag represents a different reliability level.

In terms of message scheduling, Enet uses the policy ”reliable messages first”, since reliable messages are more lag sensitive. Indeed, the reliability flag of a message determines its level of priority and reliable messages have priority over unreliable messages.

Concerning adaptation, Enet implements a probability driven model for adaptive flow control. With this model, in case of congestion, unreliable messages are held back to make room for higher priority reliable messages. In fact, Enet also uses a congestion window. The drop probability of unreliable packets increases as the window size grows. When bandwidth is sufficient, all messages are sent; as available bandwidth decreases, the probability of dropping unreliable messages increases until an equilibrium state with

12. http://www.silentwings.no/

a stable window size is reached. Like the previous middlewares, the adaptation decision is made on a static parameter assigned manually: the flag. Therefore no priority adjustment feature is provided. As previous lightweight libraries, Enet is developed to be coupled with a game engine. Hence it does not provide a specific model to structure game elements such as scenes, characters and objects.

[GarageGames, 2009]

Torque Network Library (TNL) is a cross-platform C++ networking API designed for online games.

This library has been used in many commercial projects and won many awards. The Torque 3D[13] game engine uses TNL to manage bandwidth, delay and packet loss in multiplayer networked games [GarageGames, 2007]. Derived from the network code of the first person shooter games by GarageGames[14], Tribes and Tribes 2 [Gamespot, 2001], the library was released in 2004 and the latest version 1.5.0 dates from January 2009.

Tribes 2 is a team-based game designed primarily for online multiplayers with up to 128 players [MobyGames, 2001]. The game may be played from both first and third person perspectives. The player must achieve various objectives specified in the scenario, ranging from attack-and-hold objective, to assaults on enemy fortresses, taking enemy flags, and more. Weapons vary from the sniper laser rifle to chain guns, to grenade launchers, to the massive mortar (see screenshoots in figure 3.3).

Using UDP, TNL allows at least 5 levels of reliability implemented at message level: guaranteed ordered, guaranteed, unguaranteed, current state and quickest delivery.

TNL implements a very fine-grained prioritization scheme for synchronizing objects so that important objects are updated with a greater frequency. Priorities are numerically assigned and can be automatically adjusted. TNL also provides a multitude of scheduling policies such as the ”quickest delivery” which gives a message the highest priority.

TNL does not provide built-in primitives for latency hiding strategies such as interpolation, extrapolation, client side prediction or dead reckoning. Thus making the network code less heavy for thin clients. TNL provides a mechanism for computing the average round-trip time of a connection from which these connection latency strategies can be implemented.

With respect to adaptation, TNL implements primitives to monitor the amount of bandwidth being used. It also enables developers to build customized adaptation strategies by providing virtual methods specifically for reacting to fluctuations in network resources.

Similarly to Zoidcom and Racknet, TNL allows developers to setup a rate control policy that maintains specified minimum and maximum synchronization frequencies.

Along side with this rate control, TNL also provides a receiver-driven adaptive flow control to limit the amount of packets sent by adapting to the number of packets currently in the network. For that, TNL uses a window size. When packets are received, it increases the window size and when packets are lost, the send window is decreased [Dyck et al., 2007]. A packet is therefore sent only if there is enough room to send another packet.

As a standalone library, TNL does not require a specific game model. Concerning adaptation model, TNL provides a generic model through virtual classes and methods, alongside with the receiver-driven control, to adapt to changes in network conditions.

13. http://www.garagegames.com/products/torque-3d
14. http://www.garagegames.com/

Survey Hanakush Lowlands.

Survey Hanakush Lowlands.

Run over freestanding Health Patches to repair your armor.
You picked up a repair patch.
You picked up a repair patch.
Monkeykiller suffers a serious hosing from Lone Wolf's chaingun.

Investigate tower at waypoint.

Repair Kit Used.

Survey Hanakush Lowlands.

Figure 3.3 – Screenshots of Tribes 2 gameplay [Gamespot, 2001]

The priorities are adjustable, but just like Zoidcom, TNL does not provide a model for priority adjustment.

[EpicGames, 2012]

The Unreal Engine is a C++ game engine used to develop "Unreal" by Epic Games[15] (some screenshots of this game are shown in figure 3.4) Unreal is a first-person shooter game with a multiplayer mode with more than ten maps [MobyGames, 1998].

The unreal engine has been used for all games developed by the Epic studio. It has also been used by other studios such as Acony Games with the MMOFPS *Hedone*[16], KTX Software with the third Person Action Horror *The Haunted: Hells Reach*[17] and Gearbox Software with the first-person shooter *Brothers in Arms: Hell's Highway*[18].

All networking service has been implemented an available since the version 3 released in 2006. The networking architecture of Unreal Engine 3 (UE3) uses a custom multiplayer client-server model. In this model, the server is still authoritative over game states; however, clients maintains a subset of game states locally using prediction heuristics. This is done by executing the same game code as the server, on approximately the same

15. http://www.epicgames.com/
16. http://www.gamespot.com/hedone/
17. http://www.hells-reach.com/
18. http://brothersinarmsgame.uk.ubi.com/hellshighway/

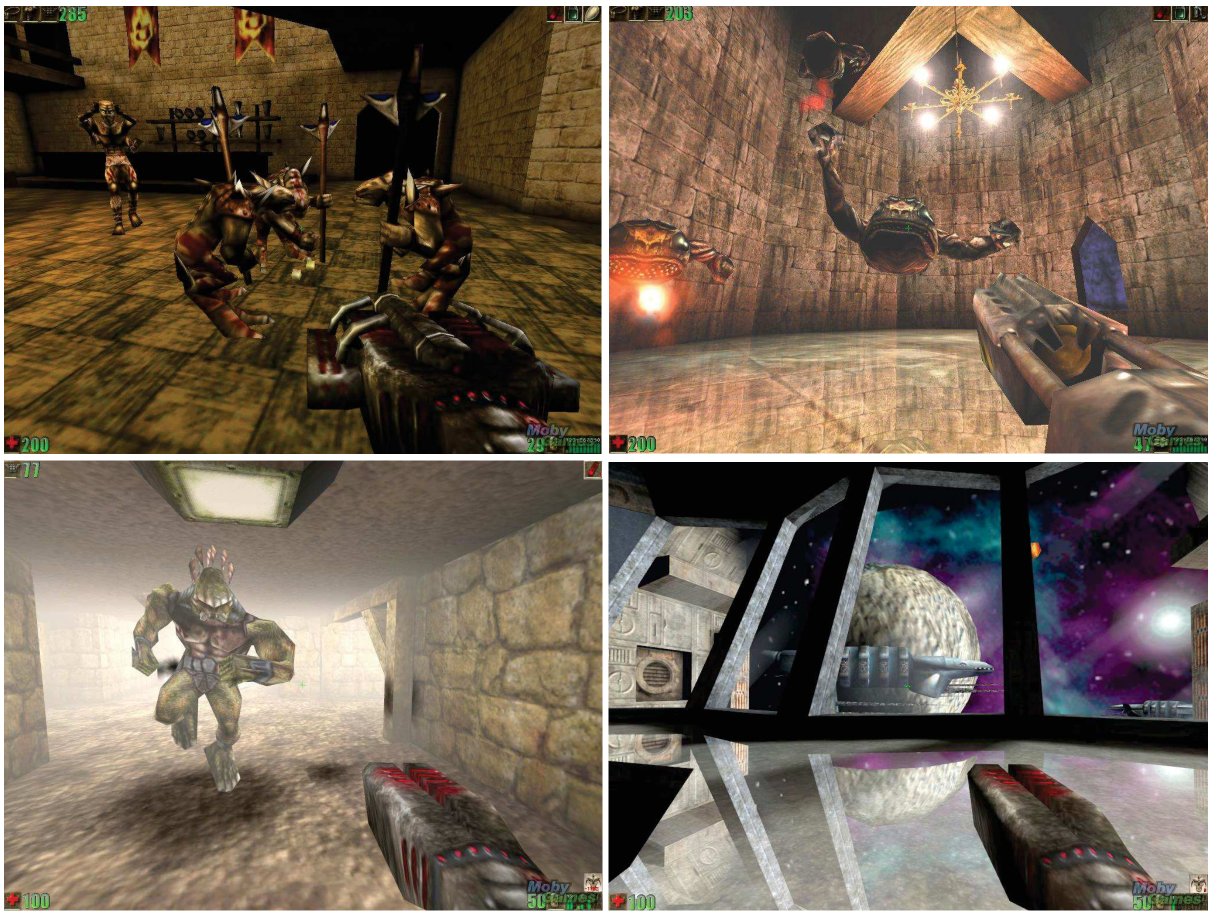


Figure 3.4 – Screenshots of Unreal gameplay [MobyGames, 1998]

data, thus minimizing the amount of data that must be exchanged between the two machines [EpicGames, 2012].

Unreal divides a game into a set of *levels*, which in turn contain a set of *actors*. An actor is a game object capable of independently moving around in a level and interacting with other actors in that level. The *game state* of a level refers to the complete set of all actors that exist in that level and the current values of their attributes. Every actor has a Role and a RemoteRole property, with different values on the server and the client. These variables describe how much control local and remote machines, have over the actor. For example every actor on a server has a Role set to *ROLE_Authority*; meaning that the server has authoritative control over the actor.

With Unreal, message scheduling and communications adaptation are built on top of this game structure. An Unreal level can be very large and interest management is applied for bandwidth optimization. The set of actors that a server deems are visible to, or capable of affecting a client are considered the relevant set of actors for that client. The sever only informs clients about actors in their relevant set.

Unreal uses a load-balancing technique that prioritizes all actors and gives each one a fair share of the bandwidth based on how important it is to the gameplay. These priorities (floating point variable) are assigned by game developer. An actor with a priority of 2.0 will be updated exactly twice as frequently as an actor with priority 1.0.

The network code is based on three primitive low-level replication operations for communicating information about game state between the server and clients: actor replication, variable replication and function call replication. Each of these operations can be processed reliably or unreliably using UDP transport protocol.

For adaptation, Unreal uses available bandwidth as input and actors' synchronization frequencies are adjusted according to actor priority. With this rate-based adaptation: actors with a high priority are usually preferred when the bandwidth is limited. Because these priorities are purely static and cannot be changed dynamically.

[Onlive, 2010]

Onlive is a commercial cloud gaming platform. It is based on video streaming for on-demand game service. Game publishers such as Sega[19], Ubisoft[20], Epic Games[21], Warner Bros.[22], Disney Interactive Studios[23] and others have partnered with OnLive.

Data and graphics are rendered at Onlive remote servers. These servers just take player's inputs and stream real-time video of games back to the player using H.264 video compression standard.

A study of Onlive performance has been conducted by Mark Claypool et al. with the goal of understanding traffic characteristics of Onlive [Claypool et al., 2014]. The experimental testbed measured network conditions and frame rates. The authors analysed the gathered network statistics and other performance data with the following conclusions:

— OnLive requires a considerable internet bandwidth. Indeed, games have high downstream bit rates, about 5 Mbps with 1000 byte packets, with much more moderate upstream bit rates of about 100 Kbps with 150 byte packets.

— Onlive does adapt bit rates to capacity limits (which is a static characteristic of the network defined by bandwidth capacity and receiver's capabilities), but does not adapt bit rates to loss, latency, nor to delay. This means that fluctuations in network conditions due to congestion are not taken into account by Onlive's adaptation scheme.

— OnLive frame rates adapt to both capacity limits and loss, but not latency nor delay. In fact, Onlive streams video games in different frame rates depending on the bandwidth capacity.

Onlive recommends a 5 Mbps internet connection to achieve what they consider as "good performance" and the minimum bandwidth required is 2 Mbps.

Onlive uses the scalability of the H.264 codec to fine tune the video compression rate in order to adapt the resulting bit rate to the network conditions. This adaptation model is known as the *scalable encoder* model. It reduces processing costs since the raw video is encoded once and adapted by exploiting the scalability features of the encoder.

This frame rate adaptation follows the transcoding model. The transcoding-based approach in video streaming adapts the video content to match a specific bit rate by transcoding the raw content. This is usually done by regulating frame rate and/or resolution.

19. http://www.sega.co.uk/
20. http://www.ubi.com/
21. http://www.epicgames.com/
22. http://www.wbgames.com/
23. http://www.disneyinteractive.com/

[Tizon et al., 2011]

[Tizon et al., 2011] presents a framework for remote rendering of 3D and 2D interactive content required for both gaming and professional applications called Kusanagi. The architecture of the platform uses a ”lobby server” to act as an interface between users and applications. The system optimizes the encoding and streaming method to improve adaptability to available bandwidth. Two kinds of data are streamed: users' inputs from client to server and audio/video data from server to client.

The lobby server is responsible of achieving an acceptable QoE for user by implementing resources management functionalities: latency measurement and adaptation.

In this framework, latency is used as input for the adaptation. The authors start by distinguishing two components of the latency: the intrinsic delay of the network infrastructure and the congestion related latency. They assume that congestion based latency is due to an over-estimation of video bit rate compared with user's available bandwidth. The idea here is to measure this latency and adapt the video bit rate in order to reduce the delay and consequently the game lag.

Similarly to Onlive, Kusanagi uses the scalable encoder approach. This approach uses the video encoder's features to change the quantization step Q_p in the frame compression process to adapt the resulting bit rate. Quantization in image processing is a lossy compression technique achieved when the number of discrete symbols in a given image is reduced. The quantization step determines the amount of compressed symbols for a single quantum value [Richharya et al., 2014]. With color quantization for example, reducing the number of colors required to represent an image makes it possible to reduce its size.

This bidirectional adaptation, is based on empirical data by using repeated round trip time (RTT)[24] measurement to compute the variability of the latency and set a threshold for the accepted latency. Thus when the latency grows above this threshold, the quantization step increases and when the latency is bellow the threshold, the quantization step decreases.

[De Cicco and Mascolo, 2010]

Akamai Technologies is an Internet content delivery network also offering high definition video distribution using adaptive video streaming. Akamai's streaming service adapts the content bit rate given current available bandwidth. This makes it possible for Akamai to provide acceptable QoE [De Cicco and Mascolo, 2010] even in presence of abrupt changes of available bandwidth. This adaptation is based on the *stream-switch* approach consisting of having multiple video levels with different bit rates to choose from. As a matter of fact, Akamai's system encodes any video at five different bit rates, producing five video levels stored at the server. The client computes the available bandwidth and sends a feedback signal to the server, that then selects the right video at the bit rate matching the available bandwidth. The values of video level bit rate $l(t)$ are in a set of available video levels $L = l_0, ..., l_4$ at any given time t. H.264 codec is used and video levels are encoded at 30 frames per second.

The client issues various commands (via a separated TCP connection) to capture and send network conditions. The most frequently used are:

24. RTT measures the latency between the client and the server. It is the time it takes to send a packet and to receive an acknowledgement of that packet.

— throttle: is the throttle percentage $T(t)$ used to identify when the server is sending the video at a rate that is greater than the video level encoding rate $l(t)$.

— rtt-test is periodically issued to ask the server to send data in greedy mode in order to actively estimate the end-to-end bandwidth.

— lvl1 is a string made of 12 feedback variables including received video frame rate, estimated bandwidth, current level identifier and current level bit rate.

These information are used by the server to select the appropriate video level for the user.

[De Cicco and Mascolo, 2010] evaluated Akamai's video streaming platform ability to adapt the video level in three different bandwidth variations scenario: (i) a step-like change of the bottleneck capacity with the available bandwidth changing from 500 kbps to 4000 kbps; (ii) a square-wave varying bottleneck capacity with available bandwidth shaped as a square-wave function with a period of 200 ms; (iii) a concurrent greedy TCP flow. The results show that, Akamai uses only video level to adapt the video source to the available bandwidth, kipping a constant video frame rate of 30 fps. When a sudden increase of the available bandwidth occurs, the matching of the new bandwidth is made in roughly 150 s. In case of sudden bandwidth decrease, short interruption of the video playback occur before adaptation is achieved.

[Prangl et al., 2008]

[Prangl et al., 2008] proposes a TCP-based mechanism for increasing the consumer's media experience under unstable network conditions. This approach is based on video content adaptation to fit the actual network bandwidth continuously monitored by the sender. The objective here is to enable clients with insufficient bandwidth capacity to consume the original video. The video quality can be gracefully degraded in order to meet the given resource constraints.

This adaptation can be achieved in two ways:

— A request is initially performed by the client to explicitly demand an adequate content quality. A *transcoding-based* adaptation is used here. In fact the video content is adapted by changing the resolution of each frame, and modifying the number of frames per second. The way it works is that, a HTTP URL is dynamically created by the client according to the device capabilities and user preferences, and it is used to formulate the transcoding request. For example, the URL *http://mediaserver.com:8080/starwars.avi?vc=h264&s=320x200&fr=30*, is a transcoding request for downscaling the video to 320x200 pixels, reducing the frame rate to 30 fps and encoding it as an H.264 video.

— A dynamic adaptation process adjusts the video content to dynamically changing network conditions. The scalability of the encoder is exploited here. The video is changed in a signal-to-noise ratio domain, by modifying the encoder's quantization step for frame compression, resulting into reduced bit rate. Three variables are measured to fuel this process: the available bandwidth *bw*, the actual delivery bit rate *dbr* enforced by the client and the actual video stream bit rate *br*. There is a delivery module between the client and the adaptation process, capturing the delivery bit rate every second. In general rule, the *br* and the *dbr* are adjusted to match the current *bw*.

The proposed adaptation approach was evaluated by its authors, with a network link limited at 6 Mbps and a bottleneck created with two simultaneous streams with a bit rate of 3.8 Mbps each. At t=0s the player 1 requested the video stream, and at t=40s the player 2 requested the same video. The result shows that, the adaptation leads to a smoother playback at the client, compare to the normal, non adaptive approach.

3.4 Discussion

The following aspects have been identified when adapting communications in game: adaptation inputs, elements affected by adaptation decisions, message scheduling and adaptation model. In this section, we present a discussion for each surveyed networking system regarding these four aspects and table 3.1 summarises this analysis.

In game communications adaptation, inputs triggering the adaptation process are usually related to network conditions or/and to the game itself. For objects replication approach, both network parameters and game parameters are used. Here available bandwidth, object priority and message reliability are the most used parameters (Zoidcom, Raknet, Enet, TNL and Unreal engine 3) For video streaming, since the structural content of a game is not modified, only network conditions are monitored. For example, Onlive adapts to bandwidth capacity and the packet loss; Akamai and [Prangl et al., 2008] use available bandwidth while Kusanagi is the only system adapting to congestion based latency.

Adaptation decisions can affect a wide rage of elements. Since the goal of adaptation is to adjust the amount of communications to current network conditions, rate and flow control are very useful. All surveyed systems implement rate or/and flow control primitives. The way rate control is performed in video streaming systems differs from objects replication systems.

As regards to objects replication, the adaptation systems affects the objects' synchronization rates or/and the network flow. Objects' synchronization rates are adjusted in all the surveyed systems except for Enet. Zoidcom, TNL, Enet and Raknet provide flow control features using a communications window to regulate the amount of messages currently in the network.

Video streaming systems adaptation consists in adjusting the downstream bit rate or/and the encoding frame rate. Here, unlike with objects replication, no different rates are associated to different game objects. Actually, the entire game state is sent in a form of a single video frame. Therefore the adaptation does not reorganize individual objects' synchronization, but instead adjusts the global bit rate and/or frame rate to network conditions. From the four video streaming systems, only Onlive is capable of adapting both its frame rate and bit rate to the network limits. Akamai and Kusanagi only provide bit rate adaptation, while [Prangl et al., 2008] provides only frame rate adaptation.

In terms of bandwidth requirements, it is clear that video streaming systems use much more bandwidth than their objects replication counterparts.

For video streaming, the required downstream bit rate is around 2-5 Mbps. With objects replication the required bandwidth depends on game content. In fact the number of objects in the game and their synchronization rates determine bandwidth usage. Therefore there is no fixed minimum bandwidth requirements for the surveyed middlewares. But to have an idea of this value we looked at some of the most bandwidth greedy games such as Battlefield 3[25] and found that they use about 42 MB/hour (0.031 Mbps) [Simmonds, 2011]. Thus video streaming approaches are quite expensive in terms of bandwidth usage, and objects replication with cloud gaming paradigm could be an alternative to video streaming.

In message scheduling, we found that numerically assigned priority is a common practice. In fact developers are able to assign a static priority to each replicated object in most of middlewares (Zoidcom, Raknet, TNL, and Unreal engine). TNL and Zoidcom also support automated or adjustable priority policies. Other middleware specific prioritization approaches can also be used. For Enet reliable messages are sent first and TNL supports ”quickest delivery” strategy.

Using priority as input for adaptation, we can have either an adaptive object rate control with static priorities or an adaptive object rate control with adjustable priorities.

Static priority adaptation works well in games where objects' importance does not change during the game session. But in case of gameplay with changing objects' importance, a dynamic priority adaptation would be preferred.

Among the reviewed middlewares only TNL and Zoidcom support both static and dynamic priorities in object rate adaptation. Raknet and Unreal engine implement only the adaptive object rate control with static priority.

This lack of built-in adjustable priorities makes Raknet and Unreal engine, not genuinely suitable for games where objects do not always have the same role or importance in the game scene. Developers therefore have to implement this feature on their own.

Techniques such as interest management can be used to reduce both network load and computational cost. Interest management are not always available in networking middlewares. From our study, only two middlewares propose interest management primitives: Zoidcom and Unreal engine. Meaning that, for other middlewares, developers have to implement interest management on their own. In video streaming, there is nothing to prioritize, since the whole game state is captured in a single video frame. Interest management is implicit in video streaming because the user only receives the video frame corresponding to his/her vicinity in the game scene.

Rate-based adaptation in video streaming follows three patterns: (i) the *transcoding-based* approach, used by Onlive and [Prangl et al., 2008]; (ii) the *scalable encoder* approach, applied by Onlive, Kusanagi and [Prangl et al., 2008]; (iii) the *stream-switching* approach, utilized by Akamai. Video streaming services do not access structural content of the game, therefore there is no game model proposed, nor priority adjustment model.

With object replication, two adaptation models are often used: the rate-based adaptation and the flow-based adaptation. Except for Enet, all studied middlewares incorporate rate-based adaptation. For flow-based adaptation, an adaptive window size is used to control the network flow. This flow control mechanism can be implemented following two approaches: the receiver-driven approach used by TNL, and the probability-based approach used by Zoidcom , Enet and Raknet.

25. http://www.battlefield.com/fr/battlefield3/

	Adaptation inputs	Affected elements	Message scheduling	Models
Zoidcom *[Rüppel, 2011]*	available bandwidth, object priority	objects replication approach, object rate	assigned and adjustable priority, multiple levels, UDP , interest management	probability-based and rate-based adaptation, no game model, no priority model
Raknet *[JenkinsSoftware, 2011]*	available bandwidth, object priority	objects replication approach, object rate	assigned and static priority, multiple levels, mixed	probability-based and rate-based adaptation, no game model, no priority model
Enet *[Salzman, 2014]*	available bandwidth, message reliability	objects replication approach, flow control	reliable messages first priority, multiple levels, UDP	probability-based adaptation, no game model, no priority model
TNL *[GarageGames, 2009]*	available bandwidth, object priority	objects replication approach, object and network rate, flow control	assigned and automated priority, quickest delivery, multiple levels, UDP	receiver-driven and rate-based adaptation, no game model, no priority model
Unreal Engine *[EpicGames, 2012]*	available bandwidth, object priority	objects replication approach, object rate	assigned and static priority, multiple levels, UDP , interest management	rate-based adaptation, level-actor-role game model, no priority model
Onlive *[Onlive, 2010]*	bandwidth capacity, packet loss	video streaming approach, frame rate and bit rate, 2 Mbps minimum	no priority, implicit interest management	transcoding-based and scalable encoder adaptation, no game model, no priority model
Kusanagi *[Tizon et al., 2011]*	latency	video streaming approach, bit rate: quantization step, 2.5 Mbps	no priority, implicit interest management	scalable encoder adaptation, no game model, no priority model
Akamai *[De Cicco and Mascolo, 2010]*	available bandwidth	video streaming approach, bit rate, from 0.5 Mbps to 4Mbps	no priority, implicit interest management	stream switching adaptation, no game model, no priority model
[Prangl et al., 2008]	available bandwidth	video streaming approach, frame rate and bit rate, 3.8 Mbps	no priority, implicit interest management	transcoding-based and scalable encoder adaptation, no game model, no priority model

Table 3.1 – Analysis of the communications adaptation approaches in cloud games

Except for TNL and Unreal, most of these middlewares are low level libraries, designed to be very generic and therefore do not provide a specific model upon which the developer can build his/her game. Only Unreal networking architecture defines a game structure to be used by replication and adaptation schemes. This game model structures the game in terms of levels and actors and tells whether objects are autonomous or not, to identify replicated objects. It does not take into account the changes in object's requirements in terms of communications resources.

None of the reviewed works have equipped game developer with a model for online (during the game session) priority adjustment. For games with changing objects' synchronization needs, it is preferable to have an adaptation process powered by a dynamic game model. The game model should therefore be able to consider the dynamic nature of game objects roles and importance in the game, determining their communications needs.

Figures 3.5 and 3.6 picture some essential points of communications adaptation in surveyed systems. They do not attempt to classify these systems, but rather show a visual map of the elements affected by the adaptation, as well as the adaptation models present in these systems.

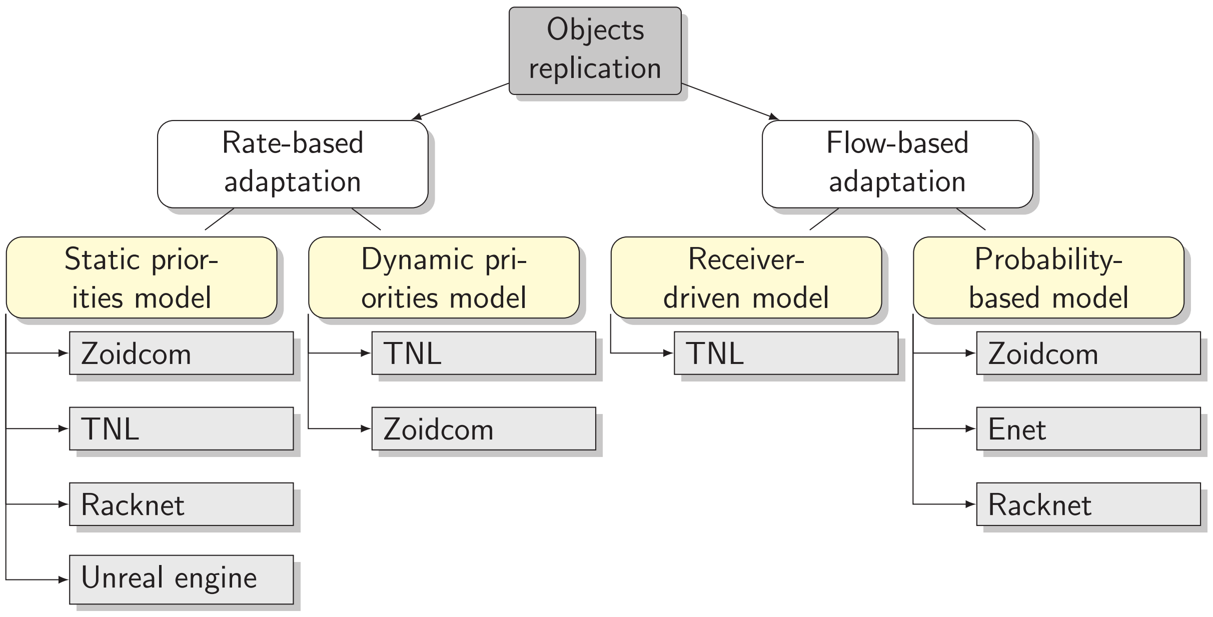


Figure 3.5 – Visual map for communications adaptation in objects replication systems

3.5 Conclusion

This chapter presented the state of the art in communications adaptation approaches in cloud games. Four main criteria captured our attention: adaptation inputs, elements affected by the adaptation, message scheduling, adaptation model, and priority adjustment.

The surveyed works propose approaches with the objective of enhancing the player's QoE by improving the responsiveness of the game and the smoothness of the game visuals. This is achieved by making a better use of the available bandwidth between client and server.

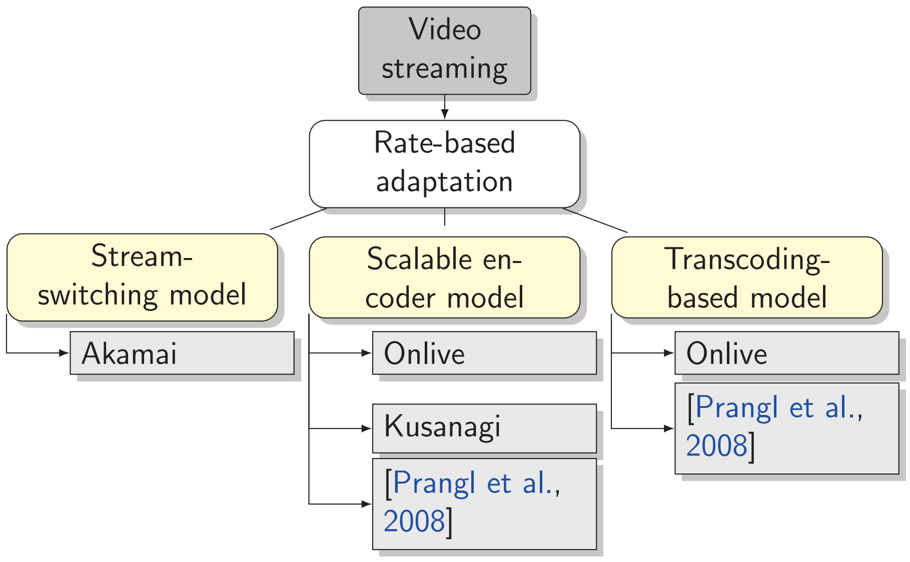


Figure 3.6 – Visual map for communications adaptation in video streaming systems

Adaptation to current network conditions is supported by most middlewares. But in adaptive synchronization rate control policies of most of the surveyed middlewares, the change in the rate of an object is driven by a static priority. This rigid priority policy can be very inefficient in a dynamic game environment where the importance of an object can fluctuate at any moment and several times during the game session. Thus an interesting approach is to take into account the variability in the object priority and the current capacity of the network. Only two middlewares proposed a dynamic priority adaptation: Zoidcom and TNL. However there is no model for priority adjustment and this has to be implemented from scratch by developers. What is advisable is to have a generic model for dynamic priority adjustment for the adaptation process. In addition, except for Unreal engine, the studied platforms do not provide the developers with a specific game model to fine tune their adaptation techniques. Unreal's game model does not capture the variability of the objects' synchronization needs.

Chapter 4

Framework Overview and Background

Contents

Our proposed framework for adaptive QoE support in cloud games is introduced in this chapter. This framework adapts synchronisation schemes between server and clients to maintain an acceptable QoE when network resources are limited. The approach is based on three main concepts: level of detail, agents organizations, and gameplay components.

This chapter is organised as follow: first, an overview of the framework is presented followed by a presentation of level of detail technique in 3D graphics. We then introduce agents and organizations, and the gameplay components model for game design.

4.1 Framework overview

In classical online game with object replication, game objects of a scene are synchronized repeatedly. Since each message consumes a certain amount of network resources, the synchronization rate determines the amount of consumed bandwidth.

Our goal in this thesis is to make this synchronization mechanism between client and server more effective in terms of QoE metrics when network conditions become limited.

Synchronization needs are different for different game objects. Some objects need high synchronization rate while a low synchronization rate can be sufficient for others. For instance, objects living in the background of a scene need less synchronization than objects interacting with the player. We therefore need an efficient message passing protocol that takes into account these differences to create priorities among update messages.

To tackle this problem, we suggest an adaptive system composed of different modules. These modules are responsible for probing changes in network conditions and game objects' importance within the scene and then adapting objects priorities accordingly.

Before presenting the adaptation framework, it is important to introduce the used adaptation model. We use an approach called *level of detail(LOD)*, first introduced by James Clark [Clark, 1976]. In 3D graphics, this approach is meant to manage processing load when representing a 3D object by modulating its complexity(number of polygon) according to its distance to camera. We will introduce this technique in details in section 4.2. In our context of game objects synchronization, we use level of detail to manage network load by regulating the amount of bandwidth consumed by each game object according to its importance in the game scene and the network conditions. Figure 4.1 illustrates this adaptation model.

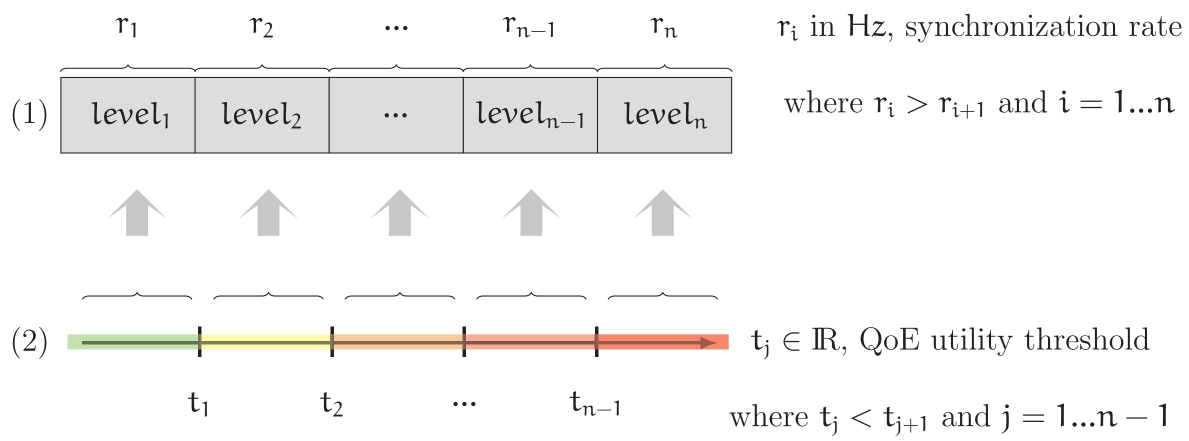

r_1
r_2
...
r_{n-1}
r_n
r_i in Hz, synchronization rate
(1)
level_1
level_2
...
level_{n-1}
level_n
where $r_i > r_{i+1}$ and $i = 1...n$
(2)
$t_j \in \mathbb{R}$, QoE utility threshold
t_1
t_2
...
t_{n-1}
where $t_j < t_{j+1}$ and $j = 1...n-1$

Figure 4.1 – Adaptation model

As shown in this figure, level of detail enables us to provide different synchronization rates (r_i) for each game object through different communications levels (level_i). A communications level is an objects container, that determines the synchronization rate of its objects. In fact each game object is assigned to a communications level configured with a fixed synchronization rate. The selection of the communications level for a game object is done using a composite metric that we called *"QoE utility"*. This value is computed for each game object using its importance and network conditions. This way, the selected communications level and consequently the synchronization rate of each object depend on its importance in the game scene and network conditions. We provided a set of QoE utility thresholds (t_j) in order to determine the communications level corresponding to the calculated object's QoE utility. With the example of figure 4.1, given n communications levels ($level_1$ to $level_n$) and $n-1$ QoE utility thresholds (t_1 to t_{n-1}), the level selection for game objects is the following (see the mapping between (2) and (1) in figure 4.1):

— the $level_1$ is used when the QoE utility is less or equal to t_1,

— the $level_2$ is used when the QoE utility is between t_1 and t_2,

— the $level_n$ is used when the QoE utility is greater than t_{n-1}.

This mapping is then made for each object each time there is a change in its QoE utility value.

To complement figure 4.1, figure 4.2 summarizes the steps of the overall communications adaptation mechanism. They are the following:

— Current conditions of the network link between client and server are continuously collected by a network module on the server system. Changes in objects' importance in the game scene are also monitored to serve as complementary input for the adaptation (see (1) in figure 4.2).

— QoE is evaluated using the QoE utility value, in order to decide whether to trigger the adaptation process or not (see (3) in figure 4.2). A drastic change in the current QoE utility value (see (2) in figure 4.2), materialized by a change in objects' importance, and/or a radical fluctuation in network conditions, automatically initiates the adaptation process.

— The adaptation process updates LoD settings (see (4) in figure 4.2) using the new network conditions and objects' importance. LoD settings consist of all parameters necessary for the LoD technique to work, namely communications levels, synchronization rates and game objects.

— The game then uses these new LoD settings to configure the network module for game objects synchronization between client and server.

In the remaining of this document we refer to level as a communications level in the LoD system.

As shown in figure 4.2 the adaptation decision is made by evaluating the QoE utility value, which is a function of network conditions and objects' importance. Figure 4.3 shows the big picture around all the adaptation inputs and the approaches used to manage those inputs.

In order to manage objects importance, we studied two different approaches: the first using an agents' organization model and the second using gameplay components. For network conditions we used a combination of the network delay and packet loss to compute a composite network metric.

Objects' importance

Different game objects have different behaviors and take different functions according to their role in the game. The combination of these parameters can determine the importance of some objects over others for a player in accomplishing his/her goal. In a shooting game for example, the player will pay more attention to all the objects that

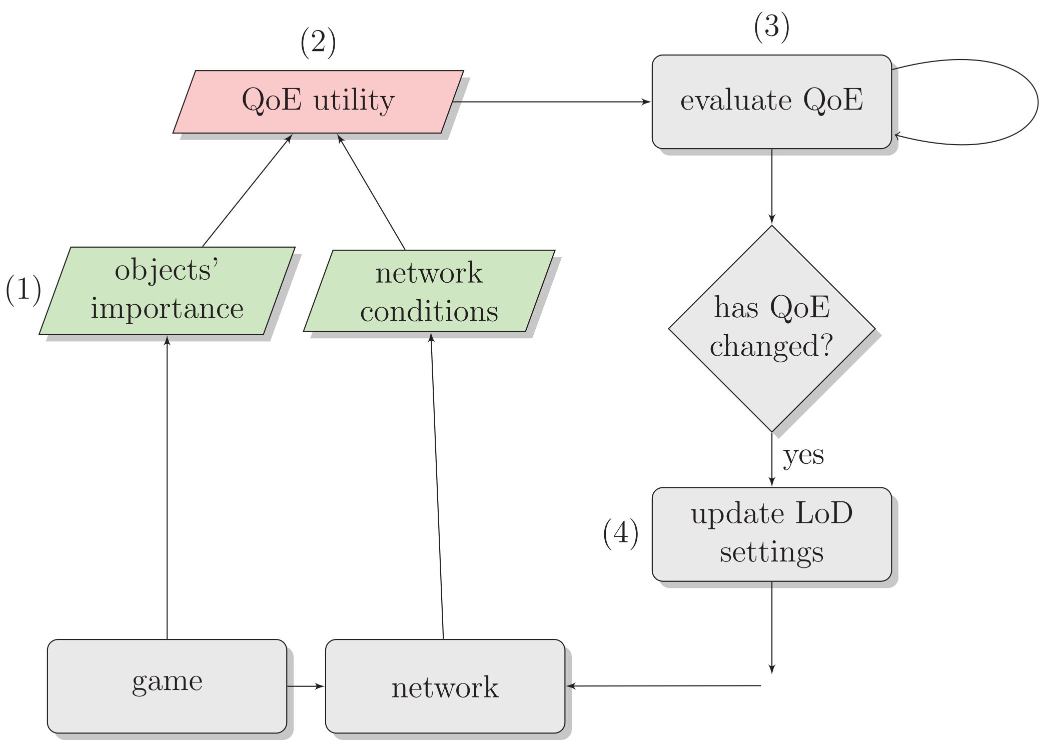


Figure 4.2 – General framework

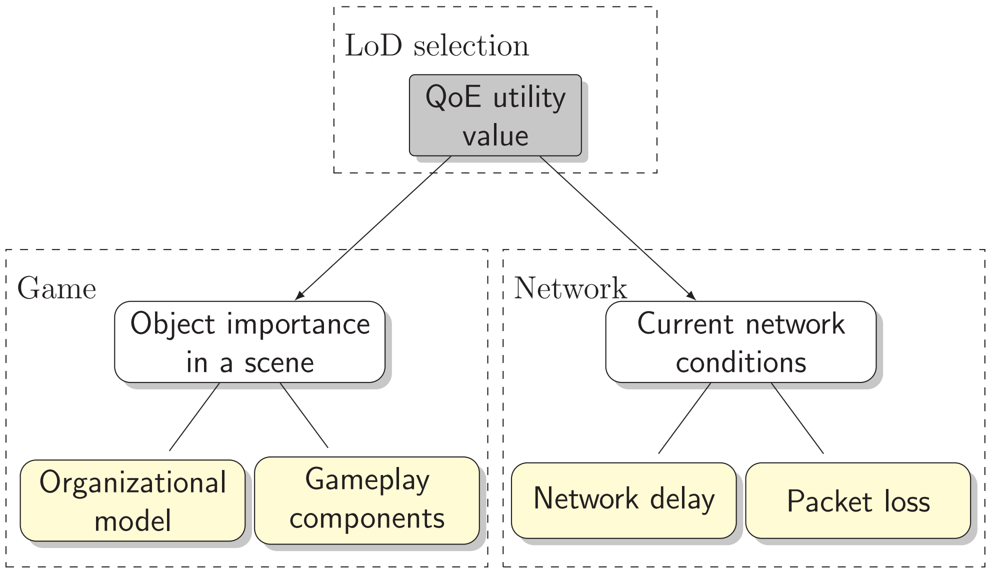


Figure 4.3 – Adaptation inputs

(s)he should shoot at, compare to simple background or decoration objects in the game scene. The importance of an object for the player therefore depends on the game scenario. Since player's focus is on the most important objects, any lag affecting these objects will impact the player's QoE more severely than a lag on less important objects. Thus there is a direct correlation between object's importance in the game scene and its sensitivity vis-a-vis of unfavorable network conditions creating the lag.

In our model, object's importance is used to organize game objects in "importance sets". This way any change in network conditions and any adaptation decision will affect all objects in a set in the same way.

As shown in figure 4.2, for a game with dynamically changing objects' importance, each time a change occur, the communications adaptation process is triggered so that importance sets are updated. To manage changes in objects' importance, we used two different models: (i) the first one is based on agents organization and focuses on objects' functional role in the game rather than directly focusing on their importance (Agent, Group, Role [Ferber et al., 2003]); (ii) the second one uses the semantics and the structure of the game to determine game objects' importance as the game progresses (gameplay components [Francillette et al., 2012]).

Network conditions

As shown in section 2.4, QoE properties such as responsiveness and visual smoothness are highly influenced by network capacity in online games in general. This explains why there is a strong correlation between the perceived quality of a game by a player and available bandwidth on the network. Most of the time these network resources are very limited especially in case of lossy wired or wireless connections. In fact, the bandwidth capacity of a network link limits its available bandwidth. For high traffic applications such as cloud games, this shortage of resources can create network congestion, which severely degrades player's QoE. The first step in attempting to address this problem, is to be able to accurately assess the current conditions of client-server network link.

This same notion is used by TCP in its congestion and flow control mechanism. Since the whole point of flow control is to avoid flooding the connection and increasing round trip time (RTT), it makes sense that one of the most important metrics as to whether or not we are flooding our connection is the RTT itself. For this reason, to avoid network congestion by adapting the amount of communications of the game, we need a way to measure the RTT of our connection.

[Comer Douglas, 2000] defines the RTT as the time it takes for a signal to be sent plus the time it takes for an acknowledgment of that signal to be received (see figure 4.4).

Following this definition, the following steps can be used to estimate the RTT:

— For each packet we send, we add an entry to a queue containing the sequence number of the packet and the time it was sent.

— Each time we receive an ack, we look up this entry and note the difference in local time between the time we receive the ack, and the time we sent the packet. This is the RTT time for that packet.

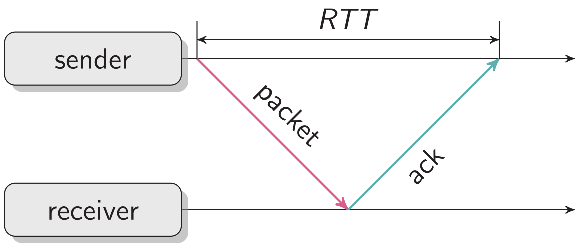


Figure 4.4 – RTT sequence diagram

The RTT value is then used as network delay by the communications adaptation process to determine the communications profile that is appropriate for current network situation.

The reliability provided by protocols such as TCP is not always needed for all applications. With synchronous video games for instance, it is more important to deliver recent state updates quickly than to ensure that stale packets are delivered. This is the reason why we also study the use of UDP as transfer protocol. In fact most game networking middlewares ([Rüppel, 2011], [Salzman, 2014] and [GarageGames, 2009]) use UDP for game object synchronization.

With UDP, busy or congested network are implicitly signaled by packet loss. As a matter of fact, Internet Protocol allows for routers to simply drop packets if the router or a network segment is too busy to deliver the data in a timely fashion. This is the reason why our adaptation approach also has to take into account, packet loss, as a metric for network conditions.

Our computation of the packet loss in a client-server connection is straightforward. We send every 50 ms what we call ”monitoring packets” to the client, and count out the responses we received from the client. The ratio between the responses received and the monitoring packets sent gives us the percentage of packet loss in the client-server link, which is used as input for the adaptation.

For the evaluation of our adaptation approach, we built two prototype games and experimented them on two versions of the adaptation system. One using solely packet loss as network conditions and the other combining network delay and packet loss to compute a composite network metric.

In the following sections, we are going to introduce the concepts of level of detail, agents and organization as well as the AGR model, and then gameplay components. More detailed description about how these notions are used in our adaptation system, will be presented in the next chapter.

4.2 Level of Detail

LoD is a technique that attempts to address the tradeoff in computer graphics between complexity and performance. In fact despite the progress in graphics hardware, the complexity of 3D models seems to grow faster than the ability of hardware to render them. This problem of limited rendering resources is well known to video game and game developers can therefore benefit from regulating the level of detail of their 3D objects.

The fundamentals

The LoD technique has been widely used in 3D graphics and simulations [Döllner and Buchholz, 2005], [Chen et al., 2006]. The basic purpose of the technique is to modulate the complexity of a 3D object representation according to the distance from which it is viewed (or any other criterion) [Luebke et al., 2002]. As introduced by James Clark [Clark, 1976], this technique is meant to manage the processing load on graphics pipeline while delivering an acceptable quality of images. Generally, when an object in the scene is far from the camera, the amount of details that can be seen on it is greatly reduced. However, the same number of polygons will be used to render the object, even though the details will not be noticed. The LoD technique suggests to reduce the number of polygons rendered for an object as its distance from camera increases, resulting in great gain in processing.

Figure 4.5 presents four different representations of one 3D object with changing number of polygons. The farther an object is from the camera, the less details its representation will have.

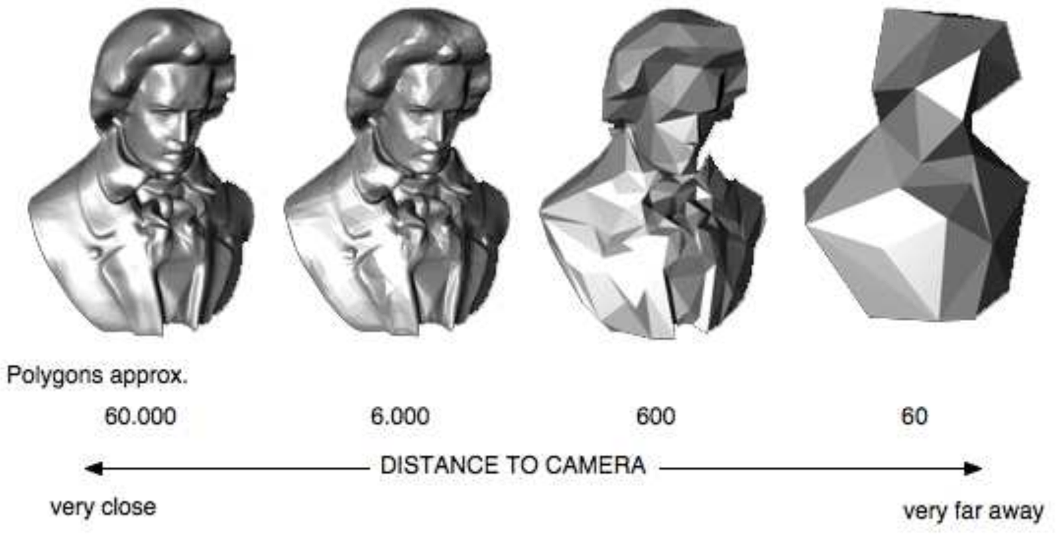


Figure 4.5 – Four different representations by changing the number of polygons [Luebke et al., 2002]

Geometric datasets are usually too large in data size and complex (in terms of time and computational resource demands), so their rendering can become a tedious and time consuming process. LoD approach suggests different representations of a 3D object model by varying in the details and geometrical complexity. The geometrical complexity of an object is determined by the number of polygons used for its representation. The more complex an object is, the more time consuming its rendering will be.

The main idea of LoD is simple: when rendering, use a less detailed representation for small and distant portions of the scene. This representation consists of a selection from several versions of objects in the scene, each version less detailed and faster to render than the one before.

At run time, the LoD technique selects the appropriate representation for each object, based on certain metrics. The distance to the camera and size are the must used in 3D graphics rendering.

LoD approaches

Depending on the smoothness of the transition between LoD representations and the precision of the represented object, two main algorithm families are used for managing level of detail: the discrete and the continuous LOD.

1. **Discrete LoD** is the traditional approach introduced by Clark in 1976 without modification. It is the most used by 3D graphic applications nowadays and creates multiple versions of every object (called levels), each at a different level of detail,

during off-line pre-process. This simplification therefore typically reduces detail across the object [Chen et al., 2006]. The result is discrete number of detail levels for each object. The appropriate version of the object is selected at run-time in accordance with size, distance, etc. The advantage of discrete LoD is the decoupling of simplification or levels creation, with rendering. This separation makes this model simple to program, and the simplification can take as long as necessary to generate LoDs while the run-time rendering simply needs to choose which LoD to render for each object. A negative point of discrete LoD is that, sometimes there is no smooth transition between LoD levels.

2. **Continuous LoD**: Rather than creating LoDs during preprocessing stage, the simplification creates a data structure of the polygon mesh of the 3D object being rendered. A polygon mesh is a collection of vertices, edges and faces that defines the shape of an object. The desired level of detail is then extracted from this structure at run-time by continuously evaluating the ”mesh” function of the structure with some heuristics (usually distance) [Döllner and Buchholz, 2005]. One advantage of this approach is smoothness, since the LoD of each object is specified exactly rather than selected from pre-created options.

LoD technique has been utilized for efficient usage of scarce resources in fields other than 3D graphics. In the artificial intelligence of video games for example, [Mahdi, 2013] applied the discrete LoD method to coordinate the distribution of computational resources to game agents in order to provide an acceptable QoE to players. This is done by providing different levels of behaviors to game agents. Different behaviors have different computational needs according to their complexities. When the FPS of the game drops, less complicated behaviors are adopted by the agent, and when the FPS increases, slightly more complicated behaviors are selected.

Following these steps, the application of this technique for communications adaptation is the ability to have different synchronization rates for each game object, and select one at a particular time based on certain criteria. This way, only important objects will get the maximum amount of network resources while others get less. This hierarchical resource distribution is done through communications levels. A communications level is an objects container. With its configured synchronization rate, it determines the amount of network resources attributed to the game objects it contains. An object can move from one level to another, in order to adapt to a change in network conditions and/or when its importance in the game changes.

4.3 Agents and organization

[Castelfranchi, 1995] describes a multi-agent system (MAS) with four main concepts which are: Agent, Environment, Interaction and Organization (AEIO). It defines agents as the autonomous goal-directed entities that populate MAS. Agents evolve in an environment, perceive this environment and act in this environment. They have a set of resources in this environment and interact with other agents in order to achieve their design goals. The flexibility of agents' actions enables them to be either reactive, proactive or social. Interaction is defined by [Ferber, 1999] as the set of mechanisms used by agents either to share knowledge or to coordinate joint activities. Organization enables a society of agents to cooperate effectively and makes it behave as a coherent whole by reducing the lack of predictability.

In video games, agents are often used to model the behavior of Non-player characters (NPCs). As a matter of fact, these game entities have many parallels with agents in MAS. A NPC can be autonomous: it can operate without direct intervention of the player and have a control over its actions. NPCs can also exhibit social abilities by interacting with other game entities to achieve a common objective. They can also act and react to changes in the game world. Many works in literature have coupled MAS with game engines in order to control characters in game environments. Some of the motivations behind this are:

- Reducing complexity and enabling modularity and reuse.
- Increasing effectiveness in entity's behaviors in respects with human behavior [Alvarez-Napagao et al., 2011].
- Enabling a reactive style of controlling and coordinating game entities [Gemrot et al., 2009], [Briot et al., 2009].
- Adapting the game to the player or opponent skill level [Bakkes et al., 2009], [Westra et al., 2011].

In this thesis, we are going to use agents not only to control the behavior of game objects or game entities, but also to coordinate the whole structure of the game model.

Agents organization

Agents can be used in video game implementations as a means to reduce complexity and provide autonomous game entities. But without some centralized coordination it becomes difficult to follow the intended general behavior and the storyline of the game, since each autonomous agent locally controls its state and behavior. This is probably one of the reasons why agents frameworks such as [Hadad and Rosenfeld, 2011], [Westra et al., 2011] and [Alvarez-Napagao et al., 2011] have been modeled based on organizational theory to coordinate game agents using social structures.

What is an organization

An organization is a set of entities forming an interdependent unit, oriented towards common objectives. According to [Bedeian and Zammuto, 1991], organizations are (on-purpose) structured so that it is possible to systematically divide complex tasks among multiple units to achieve a collective purpose. [Wooldridge et al., 2000] emphasizes on the importance of roles in an organization by defining an organization as *"a collection of roles, that stand in certain relationships to one another, and that take part in systematic institutionalized patterns of interactions with other roles"*.

These definitions of organization are used within MAS. In fact, alongside with interaction, organization is an essential concept of MAS. MAS are considered as *"societies of agents"*, meaning a set of agents that interact together to coordinate their behaviour and often cooperate to achieve some collective goals.

[Ferber et al., 2003] conceptualizes an agent organization as a unit with the following main features:

- Individuals: an organization is made of agents (individuals) that manifest a behaviour
- Partitions: the overall organization may be partitioned into groups (partitions or teams) that may overlap

— Function: agent's behaviors (roles) are functionally related to the overall organization activity

— Agents are engaged into dynamic relationship which can be classified through a taxonomy of roles, tasks or protocols, which suggests a kind of supra-individuality.

Now that we know what an organization is, let us see some examples of how this structural concept is used in video games and simulations.

Agents organization frameworks in video games and simulations

Recently a particular interest has been given to the use of organizational concepts within MAS in video games and simulations. Concepts of 'organizations', 'groups', 'communities', 'roles', 'functions' are often utilized. For instance, [Hadad and Rosenfeld, 2011] suggests to use abstraction hierarchies in order to succinctly model teamwork. The objective here was to control characters in a teamwork scenario, so that characters or agents can effectively address teamwork in dynamic environments.

With the goal of making training applications and games suitable for trainees with different skill levels, [Westra et al., 2011] proposes the use of learning agents for adaptation. Agent organizations coordinate agents and allow adaptation in very complex scenarios. The proposed framework makes sure that both the storyline and the right difficulty level for the trainee are preserved.

In the framework Alive, [Alvarez-Napagao et al., 2011] proposes to deliver the illusion of "intelligence" in the non-player characters' behaviour. It provides methodology and tools to model gaming scenarios using social structures based on organization, as well as theoretical methods to control NPC's behavior. Alive is developed as a framework coupled with game engines allowing developers to think in terms of why-what-how when defining the decision-making actions for NPCs.

Based on agents organizations' concepts listed in previous subsection, [Ferber et al., 2003], proposes a model to express organizational structure in MAS: Agent-Group-Role(AGR) model. AGR model is based on three structurally connected primitives: Agent, Group and Role.

The AGR model

AGR model's primitives are defined as follow:

Agent: an agent is an active, communicating entity playing roles within groups. An agent may hold multiple roles, and may be member of several groups. In the "cheeseboard" diagram of figure 4.6, agents are represented as skittles that stand on the board and sometimes go through the board when they belong to several groups. A, B, C, D, E, F, H and J are agents of the organization.

Group: a group is a set of agents sharing some common characteristics. A group is used as a context for a pattern of activities, and is used for partitioning organizations by assembling them under a collection. In the example of figure 4.6, a group is represented as an oval that looks like a board. There are three groups: the group G_1 formed by the agents A, B, C and D, the group G_2 formed by the agents F and J, and finally the group G_3 formed by the agents D, E,F and H. The example also shows how an agent can belong to more than one group. D belongs to G_1 and G_3, while F belongs to G_2 and G_3. In addition, two agents may communicate if and only if they belong to the same group.

Role: a role represents a functional position of an agent in a group. An agent may play one or several roles in a group, and several agents can play one role. In our example of figure 4.6, a role is represented as a hexagon and a line links this hexagon to agents. Agent F for example has three different roles: roles R_4 and R_5 in group G_3 and role R_6 in group G_2. A role may be played by several agents as shown in the example with agents A, B and C having role R_1 in group G_1.

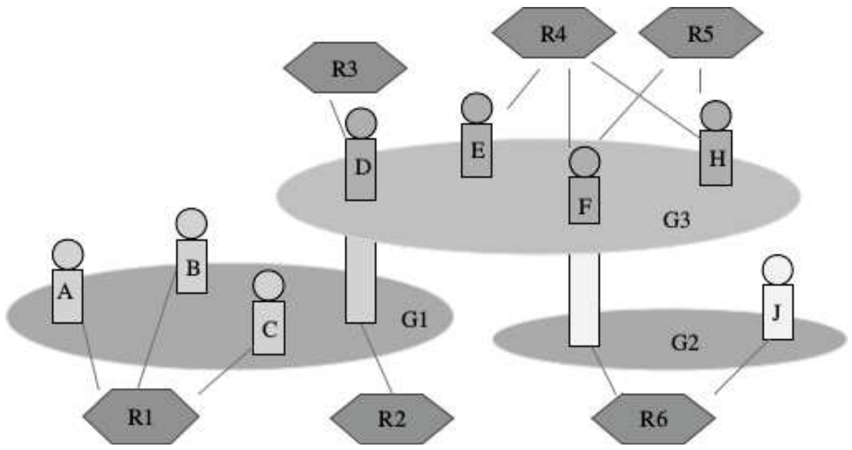


Figure 4.6 – Basic primitives of AGR in the "cheeseboard" notation [Ferber et al., 2003]

The summary of this organization is given by the equation below.

$$\mathsf{Organization}\begin{cases} \mathsf{G}_1 & \text{where roles} = \{\mathsf{R}_1, \mathsf{R}_2\}; \quad \text{agents} = \{\text{A, B, C, D}\} \\ \mathsf{G}_2 & \text{where roles} = \{\mathsf{R}_6\}; \quad \text{agents} = \{\text{F, J}\} \\ \mathsf{G}_3 & \text{where roles} = \{\mathsf{R}_3, \mathsf{R}_4, \mathsf{R}_5\}; \quad \text{agents} = \{\text{D, E, F, H}\} \end{cases} \tag{4.1}$$

As example of systems using the AGR model, we can cite MadKit[1]. MadKit is a modular and scalable multiagent platform written in Java for designing and simulating Multi-Agent Systems.

One of the main advantages of AGR is its simplicity. Its simply partitions the agents society in groups then in roles, so that each agent has at least one role in at least one group. The fact that AGR provides multiple groups and roles, makes it easier to define importance and priorities of different game objects based on their role in the game.

In our proposition, to model the variability of object importance, we use an organization model inspired by the AGR model. As a matter of fact, our organization model assembles game objects in groups of objects with similar role in the game. The role of a game object is used to determine its relative importance in the game. Thus each role in the organization is associated with an importance value. Meaning that most important objects get more valuable roles, and a change in object's role denotes a change in its importance for synchronization.

Before any further description, it is essential to emphasize the difference between an agent and an object. Ferber considers agents as a subset of objects with particular abilities. Both agents and objects have states and behaviors, but here are some differences [Ferber, 1999]:

1. http://www.madkit.org/

— An agent controls its state and its behavior, whereas an object only controls its state.

— Agents interactions are wider and more diverse than methods calls between objects.

— In a MAS, several flows of control are possible (reactive, proactive, deliberative, social, ...) whereas an objects system has only one.

In our model, we also differentiate game objects from agents controlling them. In fact, a game object represents a physical artefact on the game scene. These artefacts can be graphics or audio elements. In other words, all that a player sees or hears when playing a game are game objects, not to be confused with game agents controlling them. This differentiation enables us not only to build a game engine that can execute game mechanics independently from the aesthetic aspects, but also to facilitate client-server objects replication, since only physical objects are pertinent for synchronization.

Our organization will therefore focus on game objects instead of game agents and the role of an object in an organization is the role of the agent controlling it.

4.4 Gameplay components

Rational game design is a method that aims to streamline and rationalize the design of game levels by keeping things simple but significant, in order to create a good and meaningful player experience. It advocates to introduce game mechanics in an orderly and easily digestible fashion, and preserve the learning and difficulty curves of a game. A game mechanic being a challenge based on a specific input and skill. For instance, game designers at Ubisoft[2] consider rational game design as a valuable tool for producing entertaining and thoroughly satisfying player experiences. They implement it in the development process of most of their games [McEntee, 2012].

Rational game design has inspired researchers to introduce design methods for hierarchical gameplay creation. Many definitions of gameplay can be found in the literature, but in most of the cases, a gameplay is considered as a pattern defined through rules, challenges and actions the player is permitted to take to address those challenges [Adams, 2013],[Salen and Zimmerman, 2004]. In 2011, [Albinet, 2011] presents a model of structuring the different gameplay elements of a game using gameplay loops. The main idea here is to decompose large and complex gameplay scenarios into small and simple gameplay loops.

This game design approach is referred to as *Objective Challenge Rewards(OCR)* level design. An OCR loop is defined by these three primitives: (i) *objective* is the game state or the goal that the player needs to achieve to succeed; (ii) *challenge* consist of basically all the obstacles that the player needs to face and overcome to achieve his/her end goal; (iii) *reward* is the recompense that the player receives for achieving the objective.

OCR loops can be hierarchically composed to create different levels from simple micro OCR loops to more complex OCR loops. In addition, OCR loops can also be used to carry out gameplay related metrics adjustments, such as objects' importance in the game.

[Francillette et al., 2012] proposes a formal language to model these OCR loops as gameplay components (GC) and use them in game design. A gameplay component defines player's goal as objective, the various obstacles faced in accomplishing that goal as challenges and the payoff of the accomplishment as reward. The reward can be either

2. https://www.ubisoft.com

explicit (such as gaining score points or items), or implicit (any game content adding fun or excitement for example).

The lifecycle of a gameplay component is given by the state machine of figure 4.7. Hereafter the description of this lifecycle:

— The component starts with the initialization where, it defines and initializes the game elements needed for the loop.

— The evaluation state is where the component checks the objective of the loop. Three scenarios are possible: (i) the objective can be achieved, in this case the component moves to the success state; (ii) the objective can become unachievable, in this case the component goes to the failure state; (iii) the objective can remain in progress, meaning it is not failed nor achieved. In this case the component remains in progress and stays in the same state.

— From the success state, a reward is attributed to the player and the component moves to the end state.

— From the failure state, a penalty is attributed to the player and the component moves to the end state.

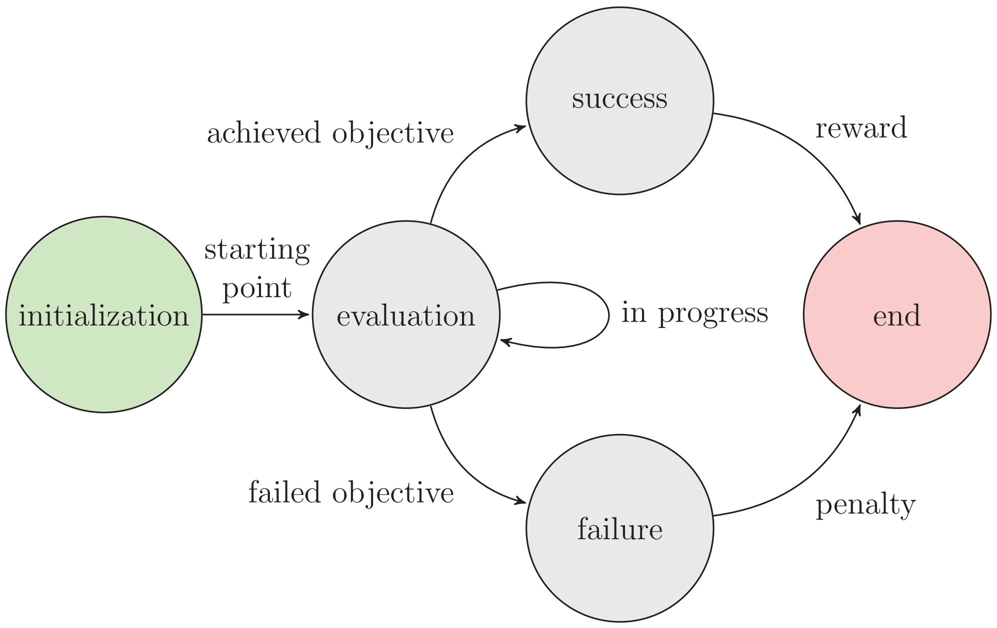


Figure 4.7 – Gameplay component's lifecycle

Using gameplay components, elementary building blocks of the game are set in a game scene and are called *atomic gameplay components*. These are GCs that cannot be decomposed to sub GCs. The reward determines whether a GC can be decomposed or not, because each sub GC has to have a reward. For example in a shooting game, the action of eliminating an object A (eliminate(A)), can be decomposed into the actions of aiming, and then shooting. But because the aiming part does not give any reward, the whole eliminating action only has one reward, which is the shooting reward. Therefore the GC eliminate(A), is an atomic GC since it can not be decomposed in sub GCs with distinct rewards.

Atomic GCs are provided by game designer at an early design phase, and the combination of these components is performed in later phases to create a GC tree representing the game. An example of GCs is a game provided with three atomic GCs. The game scene could be a composite GC combining the three atomic GCs using a set of operators.

Gameplay components operators

GC operators enable to combine existing GCs to create more complex GCs. To illustrate the use of operators let's take a basic shooting game as example. The game designer provided two atomic GCs: eliminate(object) to eliminate the object and timer(x) to initiate a count down of x seconds. Let's say we want to build a game scene with the objective to eliminate object1 and object2, while avoiding to be eliminated during a period of 30 seconds. To do this we must introduce the operators (and) $\wedge$ and (first) F that are simply composed of two components running simultaneously; and the operator (not) $\neg$ that is the negation of the component. The meanings of these operators are given in next paragraph. The resulting GC tree will be:

$$\begin{tikzcd}
 & & & \wedge \arrow[dll] \arrow[drr] & & & \\
 & \mathsf{F} \arrow[dl] \arrow[dr] & & & & \wedge \arrow[dl] \arrow[dr] & \\
\neg\ \mathsf{eliminate(player)} & & \mathsf{timer(30)} & & \mathsf{eliminate(object1)} & & \mathsf{eliminate(object2)}
\end{tikzcd}$$

In their work, Francillette et al. introduced ten different GC operators that can be used to design a game. During the course of this thesis, we had the occasion to put the gameplay components model into practice, in an industrial project called village[3]. In this project, we co-developed six game prototypes using GCs. From that experience, eight GC operators emerged as the most used. Here are these GC operators and their semantics:

— **"Parallel and" ($\wedge$):** all sub-components are running simultaneously. When at least one sub-component fails, the overall component fails. If all sub-components succeed then the overall component succeeds.

— **"Parallel or" ($\vee$):** all sub-components are running simultaneously. When at least one sub-component succeeds, the overall component succeeds. If all sub-components fail, then the overall component fails.

— **"Sequential and" ($\overrightarrow{\wedge}$):** the evaluation of sub-components is done sequentially from left to right. When a sub-component succeeds, the next sub-component is activated and evaluated and so on. If any of the sub-components fails, the overall component is considered failed and if all sub-components succeed then the overall component succeeds.

— **"Sequential or" ($\overrightarrow{\vee}$):** like the sequential and, the sub-components are activated and evaluated from left to right in a sequential manner. If at least one sub-component succeeds, then the overall component succeeds, whereas if all sub-components fail, then the overall component fails.

— **"First" (F):** all sub-components are running simultaneously. If the first sub-component to end succeeds, then the overall component succeeds and if it fails,

3. http://www.projet-village.fr/

then the parent component also fails. Thus the evaluation of the parent is equal to the evaluation of the first child to end.

— **"Ignore" (I):** it is an unary operator, defining a component for which the result must not be considered in its parent's evaluation. Therefore its success or failure does not directly affect other components. It is often used to express optional objectives in the game.

— **"Continuation" ($\mathsf{C}_{\mathsf{f} \in \mathsf{N}}$):** it is an unary operator that allows to continue with another GC when the current GC has ended. The new GC is generated by a continuation function. Schematically we can represent it with sub-components (see figure below), but it is important to know that, only one sub-component is actually in the GC tree at a given time. With the example below, only the first sub-component is in the GC at the initialization state (the one with a bold arrow), the rest will be inserted in the tree as game continues.

$$\begin{tikzcd}
 & \mathsf{C}_{\mathsf{f} \in \mathsf{N}} \arrow[dl] \arrow[d, dashed] \arrow[dr, dashed] & \\
\text{eliminate(object1)} & \text{eliminate(object2)} & \text{eliminate(object3)}
\end{tikzcd}$$

— **"Not" (¬):** it is an unary operator that simply inverts the statement of its component. If the component succeeds, the expression returns failure, and if the component fails the expression returns success.

In our adaptation approach, we use gameplay components to capture changes in objects' importance. As a matter of fact, gameplay components are suited for complex and dynamic games, with lots of quests with different objectives and different challenges. In these types of games, an object can go from being very important at one time, to become less important at another time. Gameplay components and their operators enable our adaptation to dynamically take in to account these types of game scenarios.

4.5 Conclusion

This chapter presented the building blocks of our adaptation approach. In fact, our approach is based on key principles of level of detail technique, agents organization, and gameplay components. Level of detail is the approach used to provide objects with appropriate amount of communications resources according to their importance in the organization and network conditions. With agents organization, game objects are represented in a virtual society of objects. The gameplay components model enables to design game scenario in terms of OCR loops and can be used to dynamically adjust the importance of game objects as the game session progresses. The next chapter will deal with the details of the adaptation process using each of these concepts.

Chapter 5

Communication Adaptation Process

Contents

We started the previous chapter by presenting an overview of our adaptation framework. In this chapter, we are going to present the entire adaptation mechanism in detail. We start with a presentation of our network based LoD approach for adapting game communications. The focus is then made on associating this LoD technique with models for object importance adjustment, and on the combination of fluctuating network conditions and changing objects' importance for game communications adaptation. In this context, we present organizational LoD and gameplay components based LoD.

5.1 Adapting game communications

This section presents our network based LoD aiming to adapt game communications to network conditions. The idea is to bring network conditions and objects' characteristics together, and use them simultaneously as a composite criterion for communications level selection in the LoD system.

5.1.1 The network based LoD approach

Our adaptation approach uses LoD principles to manage network resources distribution to game entities. With the discrete approach, LoD enables us to define multiple levels on which game objects can rely for state synchronization. Therefore instead of utilizing the commonly used "camera distance" as metric for level selection, object's importance and network conditions are used. This enables important game objects to use more network resources than objects with less importance.

Regarding network conditions, an object will see its network resources diminished as network congestion grows. This is done simply by moving it from its current communications level to a communications level with lower synchronization resources. The amount of network resources is determined by synchronization rate in Hz which represents the number of synchronization messages sent per second for each object.

The most important aspect of LoD management is to decide when to switch to a lower or higher level. In fact, the basic observation that we use less resources for a less important object or when network is congested seems straightforward. But how less important should an object be or how high should be network congestion before switching to a lower level? Implementing an LoD requires a data structure to store different levels for each object, and a list of thresholds to indicate when each level should be used. Each level is configured with a synchronization rate indicating the update frequency of the objects in that level. Given such a structure, a simple pointer switch suffices to select the most appropriate level for an object importance and/or for a particular network condition. The thresholds can therefore be objects' importance, a network related metric or a combination of both.

To keep the level selection simple, we use a composite metric reflecting both object's importance and network conditions. We called it *"QoE utility"*. A QoE utility (qoe_util) is a real value, indicating the factual importance of an object in the game hierarchy with the current network settings. It determines the adequate communications level and consequently the amount of synchronization resources deserved by each game object. The rational guiding this communications level selection is presented in the next paragraph.

Communications level selection

Given n communications levels (level_1 to $\mathsf{level}_\mathsf{n}$) and $\mathsf{n}-1$ thresholds ($\mathsf{threshold}_1$ to $\mathsf{threshold}_{\mathsf{n}-1}$) to specify QoE utilities at which to switch between communications levels, the appropriate level to use for an object, given a QoE utility q is computed as:

$$\mathsf{level}(q) = \begin{cases} \mathsf{level}_1 & \text{where } q \leq \mathsf{threshold}_1 \\ \mathsf{level}_i & \text{where } \mathsf{threshold}_{i-1} < q \leq \mathsf{threshold}_i \quad \text{for } 1 \leq i < n-1 \\ \mathsf{level}_n & \text{where } q > \mathsf{threshold}_{n-1} \end{cases}$$

For example, lets say we have QoE utilities in the range [0,1] with a QoE utility threshold array of ($\frac{1}{4}$, $\frac{2}{4}$, $\frac{3}{4}$) and four communications levels: optimal level, enhanced level, medium level and degraded level, respectively configured with synchronization rates r_1, r_2, r_3 and r_4. The level selection of this example is shown in figure 5.1. Following the formula above, optimal level is selected when the QoE utility value is less or equal to $\frac{1}{4}$ (a), enhanced level is selected when QoE utility value is between $\frac{1}{4}$ and $\frac{2}{4}$ (b), medium level is selected when the QoE utility value is between $\frac{2}{4}$ and $\frac{3}{4}$ (c), and degraded level is selected for all QoE utility values over $\frac{3}{4}$ (d).

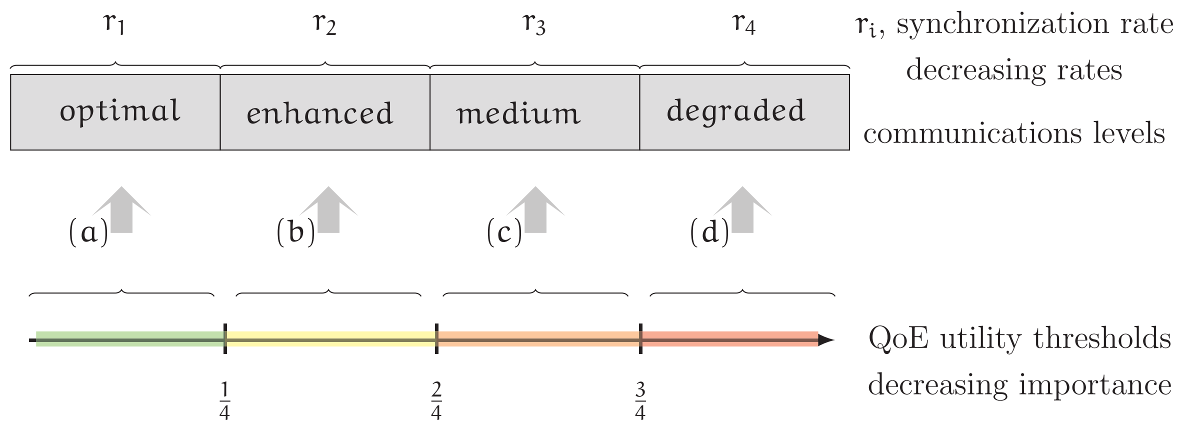


Figure 5.1 – Example of level of detail selection

Throughout the game, the system evaluates changes in game objects' QoE utilities and decides whether to start the adaptation process or not. To do this, it generates a QoE utility value for each game object. The newly generated QoE utility is then used to choose a new communications level for each object by using the QoE utility thresholds of the levels. Then a comparison between the current level (the level the object is currently mapped with), and this newly found level of each object is made (see Algorithm 1). Three situations are possible:

— **The new level is equal to the current level:** In this case, current network conditions are in concordance with current game's traffic. The available network resources plainly support the communications generated by the game. The current level is appropriate and there is no need to change it.

— **The new level is lower than the current level in synchronization rate:** This situation indicates that network conditions have degraded (This can be materialized by packet loss and/or network delays or by a decrease in object importance). Hence there is a need to accommodate the network resources distribution. In fact, lossy network parameters negatively impact the perceived quality of the game. In order to cope with this problem, we need to tighten the communications generated by the game. We do this by moving the game object to the new inferior level.

— **The new level is greater than the current level in synchronization rate:** Here the new network situation is very favorable or there is an increase in object importance. This signifies that, there is still unused network resources available for the game. For a better game responsiveness and a smooth video playout, game's communications profile has to be adjusted. This is accomplished by moving the game object to the new superior level.

```
Algorithm updateLoD;
Data: objects, the list of game objects with their current importance and levels;
current_network, the current network conditions
begin
    /* Loop for all objects in the game scene                        */
    foreach object in objects do
        /* Compute the new level of the object                       */
        qoe_util ← getQoEutil(current_network, object.getImportance());
        new_level ← generateLevel(qoe_util);
        if new_level ≠ object.getLevel() then
            /* Move from the current level to new_level              */
            object.getLevel().remove(object);
            new_level.add(object);
        end
    end
end
```

Algorithm 1: LoD framework update

The estimation of network conditions is very important for this adaptation scheme to be efficient. Next paragraph details formulas used to compute the metrics representing network link quality.

Network conditions

Network conditions are continuously probed to compute QoE utiliy and adapt game communications accordingly. During the course of this thesis, we have considered two network metrics: packet loss and network delay. We could use both criteria separately or compute a composite network metric using the criteria together.

For the composite network metric, we use the weighted mean of network delay and packet loss. A weighted mean is often used when we wish to average a number of values by attaching more importance to some numbers than to others [Kozak, 2008]. This is done by assigning different weights to the observations. This enables us to give different contributions to network delay and packet loss for the composite network metric.

A formal definition of weighted mean is: given a set of non-empty set of data $X_1,...,X_n$, each with non-negative weights $W_1,...,W_n$ the weighted mean is given by the formula:

$$mean = \frac{\sum_{i=1}^{n} W_i \times X_i}{\sum_{i=1}^{n} W_i} \tag{5.1}$$

The weights enable us to compromise between network delay and packet loss. They represent the degree of influence each data has to the overall network metric. Hence we define weights W_{delay} and W_{loss} so that

$$W_{\mathsf{delay}} + W_{\mathsf{loss}} = 1$$

Where W_{delay} and W_{loss} are allocated weights for network delay and packet loss respectively.

It is important to state that for weighted mean to be faithfully representative of its various data, it is recommended for all data to be expressed in a normalized form [Etzkorn, 2011]. However, whether normalized or not, weights associated with each data can scale appropriately to adjust for the disparity in data sizes. But with normalized data, the weights will reflect meaningful relative activity between each data and the degree to which each data influences the objective function. For example, if we want to have equal participation of network delay and packet loss in computing the network metric, we will simply set the weights to $\frac{1}{2}$ with normalized values of network delay and packet loss. But if the values are not normalized it is more complicated to find adequate weights to reach that equilibrium.

The resulting network metric is given by the formula:

$$\mathsf{network} = W_{\mathsf{delay}} \times \mathsf{delay} + W_{\mathsf{loss}} \times \mathsf{loss} \tag{5.2}$$

Where delay and loss are normalized values of current network delay and packet loss respectively.

For a better understanding of our approach, we will use only network delay as network conditions when describing the adaptation steps, knowing that the same adaptation principles apply when using the composite network metric or packet loss only.

5.1.2 QoE utility

We now know how to determine the network metric for the adaptation, let's see how it is used in order to compute QoE utility.

A communications level is configured with a synchronization rate and in order to determine the adequate level for an object given its importance and current network conditions, the LoD system uses object's QoE utility.

We describe object's qoe_util as an abstract measurement unit for the notion of relative priority regarding communications resources for weighting objects communications requirements. Object's qoe_util is calculated at runtime using a combination of current network conditions and object's importance. This way the object's QoE utility is proportional to the network load at running time. This qoe_util is therefore represented as a function of object's importance and network conditions:

$$\mathsf{qoe\_util} = \mathsf{function}(\mathsf{importance}, \mathsf{network})$$

This generic notion of qoe_util is a value that defines the relative importance of an object given the current network conditions. As the importance of an object depends on game scenario and its role in the game, this notion can be exploited in many ways.

The qoe_util formula is also used to compute composite thresholds using both network parameters and object importance thresholds. These qoe_util thresholds are then used for communications level selection.

To compute qoe_util we have used the weighted mean (see formula 5.1) of network metric and object importance metric. In our context of composite metrics, weights enable us to compromise between network parameters and object importance. They represent the degree of influence each metric has to the QoE utility. Hence we define weights W_{net} and W_{imp} so that

$$W_{net} + W_{imp} = 1$$

Where W_{net} and W_{imp} are allocated weights for network conditions and object importance respectively .

For this computation, we used normalized values of objects' importance and network conditions. The resulting object qoe_util is given by the formula:

$$qoe\_util = W_{net} \times metric_{net} + W_{imp} \times metric_{imp} \tag{5.3}$$

Where $metric_{net}$ and $metric_{imp}$ are normalized values of current network conditions and object importance respectively.

To get these normalized values, we convert raw values of network parameters and objects importance by bringing them into a common range. Also called feature scaling [wik], this data normalization can be generalized to restrict a set of values between any arbitrary points a and b using the formula:

$$normalized_X = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}}$$

Where X_{min} and X_{max} are respectively minimum and maximum values of variable X. For simplicity, we chose the range [0,1]. We will then have:

$$normalized_X = \frac{(X - X_{min})}{X_{max} - X_{min}}$$

The normalized metric for object's importance, then becomes:

$$metric_{imp} = \frac{(imp - imp_{min})}{imp_{max} - imp_{min}} \tag{5.4}$$

Where imp represents object's importance.

The same formula applies for normalized network metric.

$$metric_{net} = \frac{(net - net_{min})}{net_{max} - net_{min}} \tag{5.5}$$

Where net represents network parameter.

5.1.3 Theoretical properties

To further understand the principles guiding this communications level selection, we formulated some properties using the following notation: we denote by $i(o)|_{net}$ the importance value of object o given a network situation net; and by $r(o)|_{net}$ the synchronization rate of object o resulting from associating that object to appropriate communications level in the LoD system, given a network conditions net. Here are the properties describing this level of detail approach at any given time:

1. **Synchronization rate:** Given two objects of different importance and stable network conditions, the most important one (with the smallest important value) is

always associated to a communications level with higher synchronization rate than the less important one:

$$\begin{aligned}&\forall o_1, o_2 : \mathsf{Object}, \forall net : \mathsf{Network},\\ &i(o_1)|_{net} < i(o_2)|_{net} \Leftrightarrow r(o_1)|_{net} \geq r(o_2)|_{net}\end{aligned} \tag{5.6}$$

Demonstration: The synchronization rate of an object is determined by its communications level, which in turn is selected by comparing its qoe_util with the qoe_util thresholds.

$$\begin{aligned}&\forall o : \mathsf{Object}, \forall net : \mathsf{Network},\\ &qoe\_util(o) = w_{imp} \times i(o)|_{net} + w_{net} \times net,\\ &\forall o_1, o_2 : \mathsf{Object},\\ &i(o_1)|_{net} < i(o_2)|_{net} \Leftrightarrow (w_{imp} \times i(o_1)|_{net} + w_{net} \times net) < (w_{imp} \times i(o_2)|_{net} + w_{net} \times net),\\ &i(o_1)|_{net} < i(o_2)|_{net} \Leftrightarrow qoe\_util(o_1) < qoe\_util(o_2),\end{aligned}$$

According to the qoe_util to synchronization level mapping rule,

$$\begin{aligned}&qoe\_util(o_1) < qoe\_util(o_2) \Leftrightarrow r(o_1)|_{net} \geq r(o_2)|_{net},\\ &then \quad i(o_1)|_{net} < i(o_2)|_{net} \Leftrightarrow r(o_1)|_{net} \geq r(o_2)|_{net}\end{aligned}$$

2. **Network fluctuations:** The previous property (5.6) holds when network conditions change from net to $net + d$:

$$\begin{aligned}&\forall o_1, o_2 : \mathsf{Object}, \forall net : \mathsf{Network}, \forall d : \mathbb{R},\\ &if \quad (i(o_1)|_{net} > i(o_2)|_{net} \Leftrightarrow r(o_1)|_{net} \geq r(o_2)|_{net})\\ &then \quad (i(o_1)|_{net+d} > i(o_2)|_{net+d} \Leftrightarrow r(o_1)|_{net+d} \geq r(o_2)|_{net+d})\end{aligned} \tag{5.7}$$

Where d quantifies the experienced change in network conditions.

3. **End-to-end transmissions:** Given two objects of different importance and stable network conditions, the most important one is always guaranteed to achieve more successful end-to-end packet transmissions than the less important one in a given period t. In other words, more important objects are guaranteed to update their states more frequently. We denote by $E(o)_t$ the mathematical expectation of the object o in the mean number of successful packets transmissions during period t.

$$\begin{aligned}&\forall o_1, o_2 : \mathsf{Object}, \forall t : \mathsf{Period},\\ &i(o_1) < i(o_2) \Leftrightarrow E(o_1)_t \geq E(o_2)_t\end{aligned} \tag{5.8}$$

Demonstration:

Given network conditions, some synchronization messages may not arrive at the destination. Let X represent the number of successful end-to-end packets transmissions during a period t. $P(X)$ represents the probability of having X successful transmissions during a period t. The mathematical expectation of an object o is given by the formula:

$$E(o)_t = \sum_{i=1}^{n} X_i \times P(X = X_i)$$

Where n is the maximum number of successful transmissions for the object o during the period t. Given a synchronization rate of $r(o)$, $n = r(o) \times t$.
Therefore,

$$\begin{aligned}
&\forall o_1, o_2 : \text{Object}, \forall t : \text{Period},\\
&i(o_1) < i(o_2) \Leftrightarrow r(o_1) \geq r(o_2),\\
&r(o_1) \geq r(o_2) \Leftrightarrow \textstyle\sum_{i=1}^{r(o_1)\times t} X_i \times P(X = X_i) \geq \sum_{i=1}^{r(o_2)\times t} X_i \times P(X = X_i),\\
&\\
&\mathit{then} \quad i(o_1) < i(o_2) \Leftrightarrow E(o_1)_t \geq E(o_2)_t
\end{aligned}$$

4. **Hunger probability:** Given two objects of different importance and stable network conditions, the most important one is less likely to experience a hunger concerning network resources than the less important one in a given period t.
We denote by $H(o)_t$ the probability of object o to experience a hunger, meaning the probability of having 0 end-to-end packet transmission during period t.

$$\begin{aligned}
&\forall o_1, o_2 : \text{Object}, \forall t : \text{Period},\\
&i(o_1) < i(o_2) \Leftrightarrow H(o_1)_t < H(o_2)_t
\end{aligned} \tag{5.9}$$

Demonstration:
The probability of an object o to have at least 1 end-to-end transmission during a period t is: $\sum_{i=1}^{r(o)\times t} P(X = X_i)$ therefore,

$$H(o)_t = 1 - \sum_{i=1}^{r(o)\times t} P(X = X_i)$$

So,

$$\begin{aligned}
&\forall o_1, o_2 : \text{Object}, \forall t : \text{Period},\\
&i(o_1) < i(o_2) \Leftrightarrow r(o_1) \geq r(o_2),\\
&r(o_1) \geq r(o_2) \Leftrightarrow \textstyle\sum_{i=1}^{r(o_1)\times t} P(X = X_i) \geq \sum_{i=1}^{r(o_2)\times t} P(X = X_i),\\
&r(o_1) \geq r(o_2) \Leftrightarrow -\textstyle\sum_{i=1}^{r(o_1)\times t} P(X = X_i) < -\sum_{i=1}^{r(o_2)\times t} P(X = X_i),\\
&r(o_1) \geq r(o_2) \Leftrightarrow 1-\textstyle\sum_{i=1}^{r(o_1)\times t} P(X = X_i) < 1-\sum_{i=1}^{r(o_2)\times t} P(X = X_i),\\
&\\
&\mathit{then} \quad i(o_1) < i(o_2) \Leftrightarrow H(o_1)_t < H(o_2)_t
\end{aligned}$$

As a brief summary we can say that, our proposition theoretically guarantees some important properties in network resources distribution. Levels with higher synchronization rates are attributed in priority to most important game objects and this priority is maintained in case of fluctuations in the network. In average, most important game objects achieve more successful end-to-end transmissions and have a smaller probability of hunger than less important objects.

Depending on how objects importance are managed, we utilized this LoD approach in two ways during the course of this thesis: (i) with agents organizations, where objects' importance are mapped with their roles in the game organization. Here a change in object's role ushers a change in object's importance (see next section); (ii) with gameplay components, enabling us to dynamically adjust object importance according to game scenario(see section 5.3).

5.2 Organizational level of detail

Different objects play different functional roles in a game, and we assume that these roles are correlated to levels of sensitivity of these objects to QoE factors. In this model, objects' roles are used as indicators of objects importance in a way that, there is a direct mapping from an object's role to its importance. Hence we used organizations to group all game objects with the same role, since they have the same sensitivity to QoE factors. Instead of focusing on objects' importance, the focus is made here on their roles. A change of role for an object therefore steers a change of its importance. With an organization, instead of individually selecting communications levels of each object, we are going to select the levels for all objects in a particular group at once. The importance of a group is the importance of its objects which is determined with their role. The specificity of this organization model is presented in the following section.

5.2.1 The organization model

We follow the main principles of AGR model to structure our game entities. AGR model is a very simple organization framework, that just provides a way for partitioning system entities through the concepts of group and role. Another advantage of AGR architecture is that, it does not describe the "how", but only specifies the "what" by describing organizational structures made of groups and roles. The semantic behind the notion of role is left open so that it can be used in different scenarios. In our context of game, a role can determine the relative importance of an object for the game. With multiple roles in the game, multiple levels of importance and priorities can therefore be defined among game entities. The role of a game entity can change at any moment, and as a result, its importance also changes.

As we explained in section 4.3, our framework synchronizes objects' artifacts and not agents, and the importance of an object is the importance of the agent controlling it. Therefore instead of focusing on game agents, our organization assembles game objects in groups according to their importance.

Still, compare to the original AGR, following are some simplifications made on our organization model:

— **Role:** AGR defines roles within the scope of a group, hence entities with different roles can be in the same group. In our model, roles are defined within the scope of the game and are used to partition objects in groups of objects with the same role. Hence, all game objects in a group have the same role.

— **Group:** AGR enables agents to belong to multiple groups simultaneously. This is not possible in our model. In fact our model identifies a group with the functional importance of its objects (role) and an object can not have two levels of importance in the game at the same time.

To facilitate understanding and to formalize the structural aspects of our organization model, we use a notation introduced by AGR to formalize the semantics behind their model. Using this notation, we denote by $\mathtt{plays}(x, r)$ the statement that the object x plays the role r. We also denote by $\mathtt{member}(x, g)$ the statement that an object x is a member of a group g. g_r denotes the statement that g is a group of objects playing the role r. Here are the axioms describing our organization at any given time:

— Every object is member of one and only one group:

$\forall x : \mathsf{Object}, \exists! g : \mathsf{Group}, \mathsf{member}(x, g)$

— Every object plays one and only one role in the game:
$\forall x : \mathsf{Object}, \exists! r : \mathsf{Role}, \mathsf{plays}(x, r)$

— An object is a member of the group of objects playing the same role:
$\forall x : \mathsf{Object}, \forall r : \mathsf{Role}, \mathsf{plays}(x, r) \Leftrightarrow \exists! g_r : \mathsf{Group}, \mathsf{member}(x, g_r)$

— All objects in the same group play the same role in the game:
$\forall g : \mathsf{Group}, \forall x : \mathsf{Object}, \mathsf{member}(x, g) \Leftrightarrow \exists! r : \mathsf{Role}, \mathsf{plays}(x, r)$

5.2.2 LoD and organization

This combination of the LoD technique with agents organization is called *organizational level of detail* [Mahdi, 2013]. In fact, we are not the first to think about coupling LoD and agents organization to manage an efficient distribution of scarce resources. For instance, Mahdi Ghulam, in his PhD thesis defended in 2013, used organizational LoD to deliver a QoE support, for maintaining an acceptable frame rate in video games. For that, he used LoD to adapt the amount of CPU allocated to game agents depending on their importance in the game organization. The evaluation conducted showed that, organizational LoD enables significant QoE gains.

We are interested in using LoD to adapt the amount of network resources allocated to game objects taking to account their role in the virtual society. We created several communications levels, each configured with a synchronization rate for state updates of all the groups of objects belonging to that level.

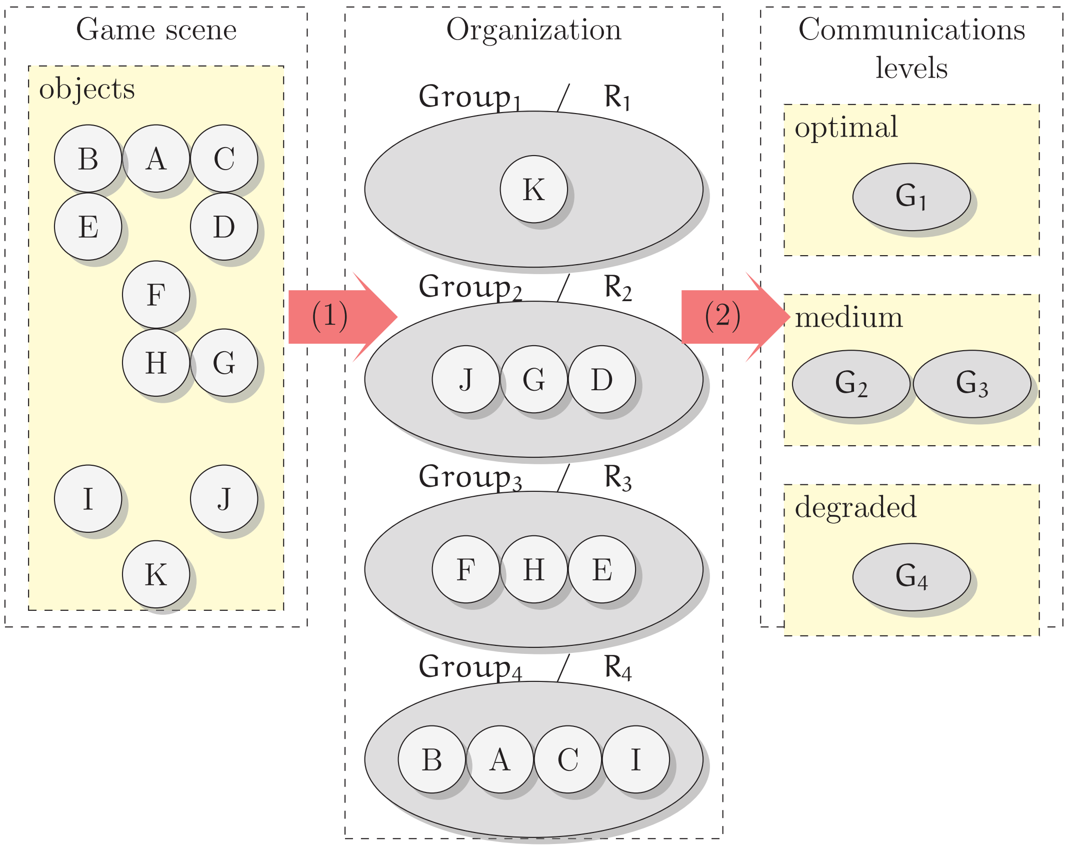


Figure 5.2 – Organizational level of detail

The example of figure 5.2 illustrates how game objects of a game scene are associated to functional groups of the organization, according to their role (1). The groups are then associated to appropriate communications levels (2) in the LoD system. Organization and communications levels of this example can be summarized as follow:

$$\text{Organization}\begin{cases} G_1 & \text{where objects} = \{K\} \\ G_2 & \text{where objects} = \{J, G, D\} \\ G_3 & \text{where objects} = \{F, H, E\} \\ G_4 & \text{where objects} = \{B, A, C, I\} \end{cases}$$

$$\text{levels}\begin{cases} \text{optimal} & \text{groups} = \{G_1\} \\ \text{medium} & \text{groups} = \{G_2, G_3\} \\ \text{degraded} & \text{groups} = \{G_4\} \end{cases}$$

When network resources are sufficient, all objects are assigned to the optimal level, and are synchronized at the highest rate in the LoD system. When there is a shortage of resources, the groups of objects are redistributed to degraded levels, according to their functional role. The network communications module then uses the communications levels to synchronize objects' states.

5.2.3 Example

To illustrate this LoD and organization association, here is a simplified version of a shooting game, "My Duck Hunt", developed to evaluate our adaptation approach with organizations [Ewelle et al., 2013]. Figure 5.3 shows a screen shot of the game.

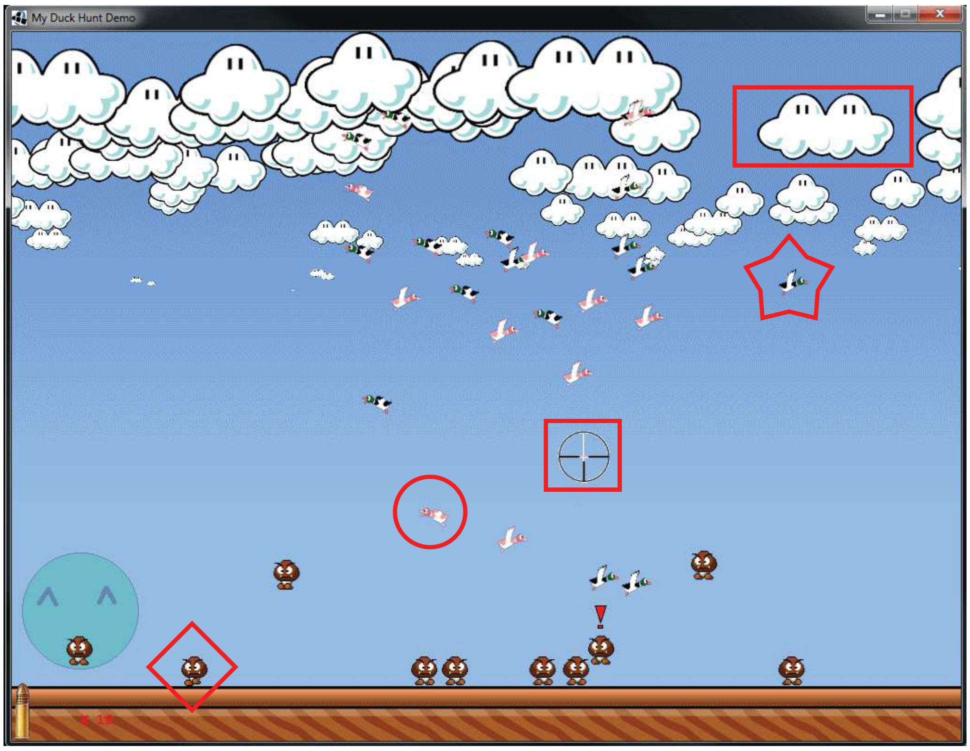


Figure 5.3 – Screenshot of the game "My Duck Hunt"

Five kinds of game objects evolve in the game scene:

— Player controlled reticle, which is the round game object, highlighted with a square in the screenshot. The player should point the reticle on a target, and shoot to eliminate the target.

— Ducks are the flying game objects, highlighted with a star.

— Flamingos are the flying game objects, highlighted with a circle.

— Gombas are the game objects on the floor, highlighted with a diamond.

— Clouds are the game objects, highlighted with a rectangle.

The game is divided in 5 stages or quests and the player has to achieve the following goals:

— Eliminate as many ducks as possible. For each duck eliminated, the player gains points.

— Do not eliminate flamingos.

— Protect flamingos from gombas by eliminating gombas. Each flamingo eliminated results in points loss.

From this description and the functional roles of these objects, the following organization can be formed:

$$Organization \begin{cases} G_1 & \text{where role} = \{player\}; \quad \text{objects} = \{\text{reticle}\} \\ G_2 & \text{where role} = \{target\}; \quad \text{objects} = \{\text{ducks}\} \\ G_3 & \text{where role} = \{protect\}; \quad \text{objects} = \{\text{flamingos}\} \\ G_4 & \text{where role} = \{enemy\}; \quad \text{objects} = \{\text{gombas}\} \\ G_5 & \text{where role} = \{decoration\}; \quad \text{objects} = \{\text{clouds}\} \end{cases}$$

In order of importance, player's reticle comes first, target objects come second, followed by protects objects, then enemy objects and finally decoration objects. Network resources distribution will then prioritize objects based on this order.

The LoD system is the following: (i) object importance variable is an integer value, ranging from 0 to 10. 0 being the highest importance and 10 being the lowest importance. For this example, we will use an object importance threshold array of (2, 5, 7); (ii) the network metric here is the network delay specified in ms and ranges from 0 to 400 ms. For this example we will use the network delay threshold array of (100, 160, 220); (iii) four communications levels: optimal, enhanced, medium and degraded.

QoE utility thresholds are computed using both network and object importance thresholds. This computation is done in 3 steps:

— Normalized importance metric: Using importance ranging from 0 to 10 and the importance thresholds of (2,5,7), the normalized threshold array for level selection using object importance will become $(\frac{2}{10}, \frac{5}{10}, \frac{7}{10})$.

— Normalized network metric: With the range [0, 400] and the network thresholds of (100,160,220), the normalized threshold array for level selection becomes $(\frac{5}{20}, \frac{8}{20}, \frac{11}{20})$.

— QoE ulitily: After calculations, given equal weights of $\frac{1}{2}$ for both metrics, the composite threshold array of the overall level selection in the example will be $(\frac{9}{40}, \frac{18}{40}, \frac{25}{40})$.

$$(\frac{9}{40}, \frac{18}{40}, \frac{25}{40}) \Leftrightarrow (t_1 = 0.22, t_2 = 0.45, t_3 = 0.62)$$

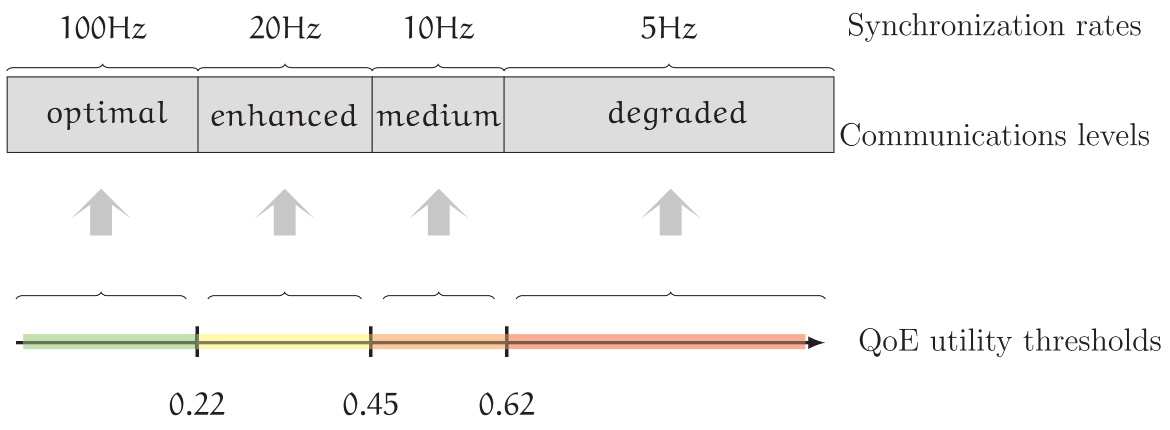


Figure 5.4 – Example of LoD selection

With these settings, appropriate communications levels for the composite metrics are selected as shown in figure 5.4.

For simplicity, in this example, objects' roles are static, thus the only input that changes is the network delay. But provided a game with changing objects' roles, the same principles apply. Let's consider the progression of figure 5.5:

— The game starts with a network delay of 10 ms.

— After 30 seconds the network delay increases to 190 ms.

— After 90 seconds the network delay decreases to 80 ms.

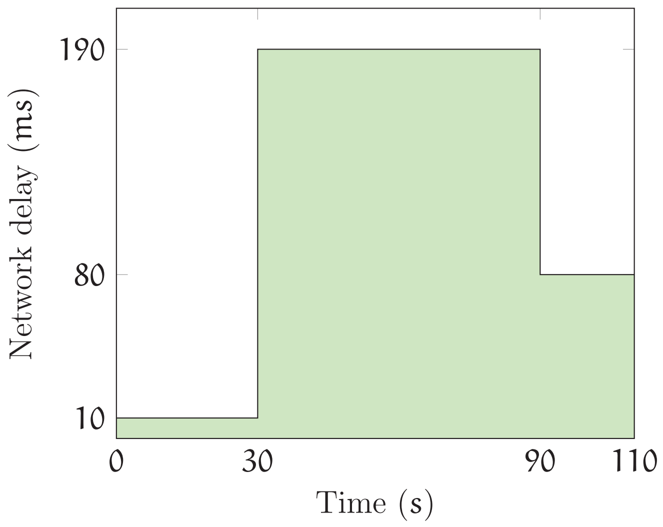


Figure 5.5 – Example of network delay progression

The objective of this example is not to present the calculation details of object's qoe_util, but rather it focuses on communications level changes for group of objects in response to network condition fluctuations.

— **At the starting point:** the network delay of 10 ms is considered stable, and a resulting level distribution could be the following:

$$levels \begin{cases} optimal & \text{where groups } =\{G_1,\, G_2,\, G_3,\, G_4,\, G_5\} \\ enhanced & \text{where groups } =\{\} \\ medium & \text{where groups } =\{\} \\ degraded & \text{where groups } =\{\} \end{cases}$$

In this situation, the normalized qoe_utils of all groups are below the threshold of the optimal level, thus they will all be assigned to the optimal level, which is normal given the ”favourable” network conditions.

— **After 30 seconds:** the delay goes to 190 ms. The qoe_util of each group is recalculated and the groups are reassigned to appropriate levels. LoD settings could then be the following:

$$levels \begin{cases} optimal & \text{where groups } =\{G_1,\, G_2\} \\ enhanced & \text{where groups } =\{G_3\} \\ medium & \text{where groups } =\{\ G_4\} \\ degraded & \text{where groups } =\{G_5\} \end{cases}$$

Reticle and ducks stay on the optimal level, while other game objects are reallocated to gracefully degraded levels. Clouds are the game objects with the lowest importance, they are therefore assigned to the lowest level with a synchronization rate of 5Hz. This level association verifies our property 5.6 about the coherence of network resources distribution according to objects' importance. We can already see that, all game objects are not affected by network degradation in the same way, unlike in a situation without LoD.

— **After 90 seconds:** the delay reverts back to 80 ms. With these network conditions, levels are reorganized and the resulting settings could be:

$$levels \begin{cases} optimal & \text{where groups } =\{G_1,\, G_2\} \\ enhanced & \text{where groups } =\{G_3,\, G_4\} \\ medium & \text{where groups } =\{\ G_5\} \\ degraded & \text{where groups } =\{\} \end{cases}$$

Player and targets objects remain on the optimal level, the enemy group is moved up to the enhanced level, while decoration game objects move from the lowest level to the medium level. This verifies our property 5.7 about synchronization rate distribution when network conditions change. The most important objects still get a larger portion of the available network resources.

5.3 GC based level of detail

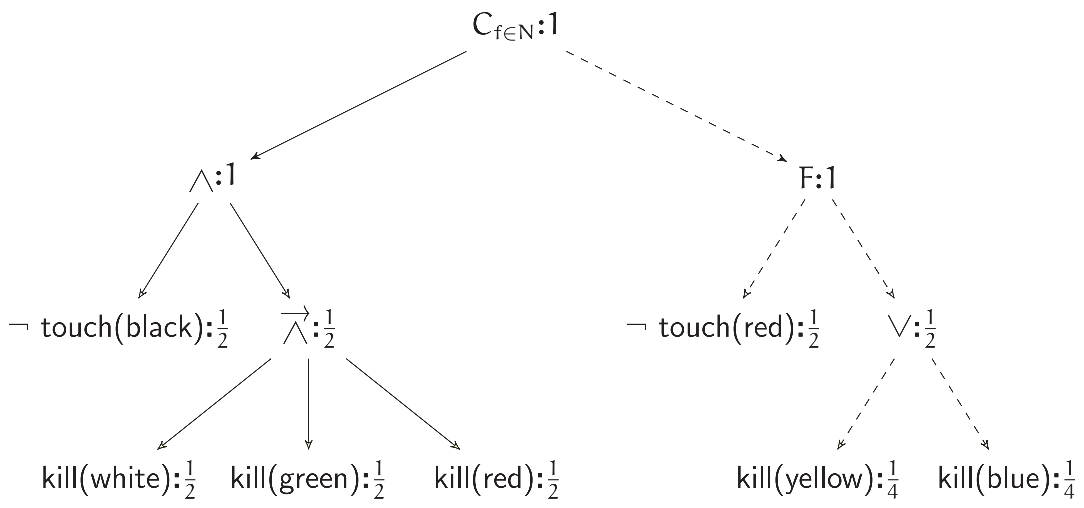
Section 5.2 presented how we use agents organization to manage objects importance in the LoD system. In this section, we present a LoD system using the semantic of gameplay components to determine the importance of game objects. The overall LoD system does not change. We still have n communications levels with different synchronization rates, $n-1$ QoE utility thresholds and game objects are distributed to communications levels according to their QoE utility values. The only thing that changes here is the way to provide objects with importance values that are then used to compute QoE utility values. Assuming that objects' importance change as the game evolves, our objective here is to use GCs to enable objects' importance adjustment during game sessions. We do this by associating an *importance value* to each node of the GC tree so that a change in the game state is reflected by a change in this *"importance tree"*.

As explained in section 4.4, when using gameplay components, a game is designed as a set of atomic GCs combined together by GC operators. A game is therefore represented as a GC tree. It is important to have in mind that, GCs represent the different game mechanics around the concepts of objective, challenge and reward. In a game scene, there are always some objects that do not participate in any of these game concepts. Background objects are an example. In general, background objects are not related to a specific game objective, nor add any challenge to the game, nor constitute a particular reward to player's actions. As a result, these objects are not represented in a game's GC tree.

Since our adaptation approach consists of associating each object in the game with an importance value, we need to consider all objects. This is the reason why we assemble game objects in two types: those that are present in a GC, we call them *"GC objects"*; those that are not present in any of the GCs and therefore are not in the GC tree. We call them *"Non-GC objects"*. As a basic rule, all Non-GC objects will be attributed a default static importance value, whereas GC objects will see their importance values fluctuate throughout the game session according to the game tree. This process is done in two phases: (i) an inceptive phase where an initial importance value is attributed to each node of the tree; (ii) a propagation phase where the importance values are updated.

5.3.1 Initial importance distribution

The importance value is a real number between 0 and 1, where 0 represents the lowest importance and 1 represent the highest importance. The root of the GC tree has an importance value of 1. This value is then distributed in a recursive manner to the rest of the GC tree. For example, figure 5.6 illustrates how importance values are distributed in the initialization phase of the GC tree.



$C_{f\in N}$:1
$\wedge$:1
F:1
$\neg$ touch(black):$\frac{1}{2}$
$\overrightarrow{\wedge}$:$\frac{1}{2}$
$\neg$ touch(red):$\frac{1}{2}$
$\vee$:$\frac{1}{2}$
kill(white):$\frac{1}{2}$
kill(green):$\frac{1}{2}$
kill(red):$\frac{1}{2}$
kill(yellow):$\frac{1}{4}$
kill(blue):$\frac{1}{4}$

Figure 5.6 – Example of a GC tree with importance values at initialization

This importance distribution algorithm starts with the root node of the GC tree. The root is attributed the global importance of 1 and this importance is recursively distributed to the rest of the tree following the algorithm 2. The rule is that the importance of each parallel composite GC is distributed to its children:

$$\sum \mathsf{child}_{\mathsf{importance}} = \mathsf{parent}_{\mathsf{importance}}$$

The unary and sequential composite GCs have the same importance as their children, since only one child is active at a given time.

```
Algorithm distributeImportance;
Data: imp, the node's importance;
node, the GC node
begin
    /* Set the importance for the current node                                */
    node.setImportance(imp);
    children ← node.getChildren();
    if children ≠ null then
        /* Depending on the operator distribute importance to children
           nodes                                                              */
        switch node.operator do
            /* Parallel nary operators                                        */
            case $\wedge$, $\vee$, $\mathsf{F}$
                foreach child in children do
                    /* Distribute the importance according to child's
                       weight                                                 */
                    childImp ← child.getWeight() × imp;
                    distributeImportance(childImp, child);
                end
            end
            /* Sequential and unary operators                                 */
            case $\overrightarrow{\wedge}$, $\overrightarrow{\vee}$, I, $\mathsf{C}_{\mathsf{f}\in\mathsf{N}}$, $\neg$
                /* Distribute the importance to the next or the only
                   child                                                      */
                child ← children.getNext();
                distributeImportance(imp, child);
            end
        endsw
    end
end
```

Algorithm 2: Importance distribution at initialization

By default, we decided that all children of a node have the same importance value. Therefore the same percentage of their parent's importance (weight). But it is important to state that, depending on the game scenario, some children can require more focus than others, and therefore should have a bigger share of the parent's importance. For example, let's consider the GC tree of figure 5.7.

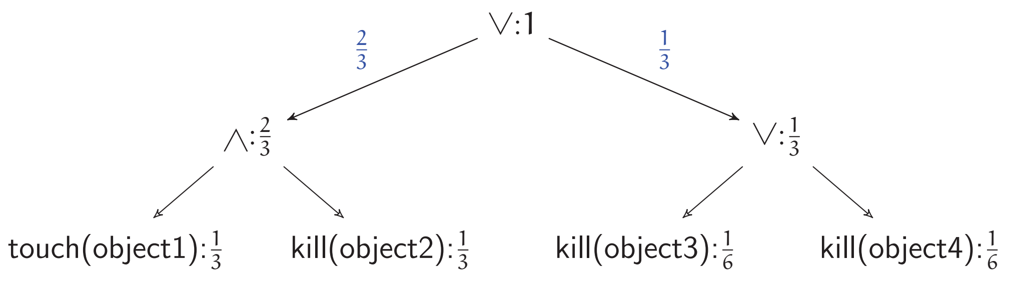

$\vee$:1
$\frac{2}{3}$
$\frac{1}{3}$
$\wedge$:$\frac{2}{3}$
$\vee$:$\frac{1}{3}$
touch(object1):$\frac{1}{3}$
kill(object2):$\frac{1}{3}$
kill(object3):$\frac{1}{6}$
kill(object4):$\frac{1}{6}$

Figure 5.7 – Example of a GC node with different importance weights on children

With the default settings, the $\wedge$ node and the $\vee$ node will have the same importance of $\frac{1}{2}$. But we know that the evaluation of both operators is not the same. With $\wedge$, the player must achieve both sub-objectives (touch(object1) and kill(object2)) to succeed. Even though this can be very subjective, one can argue that, the player needs to be two times more focused than, with $\vee$ operator where (s)he only has to kill only one of the objects and thus has two chances to succeed. A possible distribution can be the one proposed, assigning an importance of $\frac{2}{3}$ to the $\wedge$ operator and an importance of $\frac{1}{3}$ to the $\vee$ operator.

In another situation, it could be that killing the object3 or the object4 is more challenging and more rewarding than touching the object1 and killing the object2. Thus with this setup, it could be advisable to have a bigger importance on the $\vee$ operator.

These examples show that, importance values really depend on how the game designer qualifies the level of challenge and/or reward for the GCs as well as the influence of the GCs on the QoE.

To deal with these game dependent scenarios, we gave game designers the possibility to manually set an importance weight of each child (w_i). The weights are then used to distribute parent's importance value to children. These weights are real numbers between 0 and 1, and they are set in the way that their sum equals to 1.

$$\mathtt{childImportance}_i = w_i \times \mathtt{parent}_{\mathtt{importance}} \quad \text{where } \sum w_i = 1$$

In the scenario above the weights of node $\wedge$:$\frac{2}{3}$ and node $\vee$:$\frac{1}{3}$, are attributed manually by the game designer. But by default an equal distribution is applied and all weights are equal to:

$$w_i = \frac{1}{\mathtt{number\_of\_children}}.$$

In figure 5.7, the importance weights are the numbers in the middle of the arrows. In the remaining of this document, GC tree's arrows without weights signify an equal distribution of parent's importance to children.

5.3.2 Updating importance

At the evaluation state, each active GC checks the return value of its evaluation function. As we saw in figure 4.7, three cases are possible: *success*, *failure* and *in progress*. The GC tree has to adapt to any of these situations and adjust the nodes' importance value accordingly (see the algorithm 3):

— **In case of success**, all sub-nodes' importance are set to default. The GC calls a reward script and the game objects manipulated by this script have the same importance value as the GC.

— **In case of failure**, all sub-nodes' importance are set to default. The GC calls a penalty script and the game objects manipulated by this script have the same importance value as the GC.

— **In case of GC in progress**, the evaluation function is used to ensure that any progress toward the objective is taken into account in nodes' importance adjustment. As example, let's say you have a GC sub-tree as the one below. At initialization state, its importance is $\frac{1}{2}$ and all the sub-components have the same importance of $\frac{1}{6}$.

$$\begin{tikzcd}
& \lor:\frac{1}{2} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{kill(object1)}:\frac{1}{6} & \text{kill(object2)}:\frac{1}{6} & \text{kill(object3)}:\frac{1}{6}
\end{tikzcd}$$

The evaluation function periodically checks the return value of each of the sub-components. If the first sub-node (kill(object1)) fails, its parent still has 2 sub-nodes to achieve its objective, thus its evaluation function will return in progress. But it has to adjust the importance of its sub-components because, the first sub-objective can no longer be achieved, the focus will now be on the other sub-components. The way we deal with this situation is by redistributing the whole importance, only to the remaining, unended sub-components. With this *propagation* phenomenon, the termination of one sub-component, causes the change of importance values of its remaining siblings. The objects involved in the ended GC now become Non-GC objects, and therefore their importance is set to the default value. The resulting sub-tree is the one below where the ended GC is identified by the dashed arrow and the remaining GCs with the bold arrows.

$$\begin{tikzcd}
& \lor:\frac{1}{2} \arrow[dl, dashed] \arrow[d] \arrow[dr] & \\
\text{kill(object1):default} & \textbf{kill(object2):}\frac{1}{4} & \textbf{kill(object3):}\frac{1}{4}
\end{tikzcd}$$

The objective of the propagation algorithm is to adjust the importance of siblings sub-nodes in reaction to the termination of a sub-node. The way these importance values are propagated differs from one operator to another:

1. **"Parallel and" ($\land$) operator:** With $\land$ operator, if a child ends with a success, it becomes less important and set its importance to the default value. The focus in now on the other children to which the ended child's importance is redistributed. In case of child failure, the parent fails as well (see the algorithm 4).
2. **"Parallel or" ($\lor$) operator:** In case of child success, $\lor$ operator succeeds as well and calls the reward script. When a sub-component fails, the operator continues to evaluate the remaining sub-components. Therefore the ended sub-component becomes less important and associated to the default importance value. The ended

```
Algorithm updateImportance;
Data: node, the GC node;
default, the default importance value for the Non-GC objects
begin
    children ← node.getChildren();
    /* Depending on the evaluation function                              */
    if evaluation(node) = success then
        /* All the sub-nodes are set to default                          */
        foreach subnode in children do
            distributeImportance(default, subnode);
        end
    else
        if evaluation(node) = failure then
            /* All the sub-nodes are set to default                      */
            foreach subnode in children do
                distributeImportance(default, subnode);
            end
        else
            /* Check if any sub-component has terminated                 */
            if children.lenght() > 0 then
                foreach child in children do
                    /* Propagate the importance change in the siblings   */
                    if evaluation(child) = success $\vee$ evaluation(child) = failure then
                        switch node.operator do
                            case $\wedge$
                                propagateParallelAnd(child, node);
                            end
                            case $\vee$
                                propagateParallelOr(child, node);
                            end
                            case $\overrightarrow{\wedge}$
                                propagateSequentialAnd(child, node);
                            end
                            case $\overrightarrow{\vee}$
                                propagateSequentialOr(child, node);
                            end
                            case $C_{f \in N}$
                                propagateContituation(child, node);
                            end
                        endsw
                    end
                end
            end
        end
    end
end
```

Algorithm 3: Importance update at evaluation state

```
Algorithm propagateParallelAnd;
Data: child, the ended child node;
parent, the parent GC node
begin
    if evaluation(child) = success then
        children ← parent.getChildren();
        children.remove(child);
        /* Distribute the ended child's importance equally */
        foreach subnode in children do
            extraImp ← child.getImportance()/children.lenght();
            newImp ← extraImp + subnode.getImportance();
            distributeImportance(newImp, subnode);
        end
        /* Set the ended child's importance to default */
        child.setImportance(default);
    end
end
```

Algorithm 4: Importance propagation of the ”parallel and” operator

sub-component's importance is redistributed to the remaining sub-components (see the algorithm 5).

```
Algorithm propagateParallelOr;
Data: child, the ended child node;
parent, the parent GC node
begin
    children ← parent.getChildren();
    if evaluation(child) = failure then
        children.remove(child);
        /* Distribute the ended child's importance equally */
        foreach subnode in children do
            extraImp ← child.getImportance()/children.lenght();
            newImp ← extraImp + subnode.getImportance();
            distributeImportance(newImp, subnode);
        end
        /* Set the ended child's importance to default */
        child.setImportance(default);
    end
end
```

Algorithm 5: Importance propagation of the ”parallel or” operator

3. **”Sequential and” ($\overrightarrow{\wedge}$) operator:** The only difference in propagating $\overrightarrow{\wedge}$ in comparison with the $\wedge$ is that, instead of redistributing parent's importance to all remaining children, with $\overrightarrow{\wedge}$ only the next child in the operator gets all the parent's importance. This happens because, with the sequential operators, only one child is active at a given time (see the algorithm 6).

```
Algorithm propagateSequentialAnd;
Data: child, the ended child node;
parent, the parent GC node
begin
    if evaluation(child) = success then
        children ← parent.getChildren();
        children.remove(child);
        /* The importance is assigned to the next sub-node              */
        subnode ← children.getNext();
        if subnode ≠ null then
            distributeImportance(parent.getImportance(), subnode);
        end
        /* Set the ended child importance to default                     */
        child.setImportance(default);
    end
end
```

Algorithm 6: Importance propagation of the "sequential and" operator

4. **"Sequential or" ($\overrightarrow{\vee}$) operator:** Like with $\overrightarrow{\wedge}$, $\overrightarrow{\vee}$ differs from $\vee$ by only giving all the parent's importance to the next child (see the algorithm 7).

```
Algorithm propagateSequentialOr;
Data: child, the ended child node;
parent, the parent GC node
begin
    if evaluation(child) = failure then
        children ← parent.getChildren();
        children.remove(child);
        /* The importance is assigned to the next sub-node              */
        subnode ← children.getNext();
        if subnode ≠ null then
            distributeImportance(parent.getImportance(), subnode);
        end
        /* Set the ended child importance to default                     */
        child.setImportance(default);
    end
end
```

Algorithm 7: Importance propagation of the "sequential or" operator

5. **"Continuation" ($\mathsf{C}_{f \in N}$) operator:** With $\mathsf{C}_{f \in N}$ operator, no matter the status of the ended child, the parent's importance is passed to the newly generated child in a sequential way. The importance of the ended child is set to the default value (see the algorithm 8).

Operators such as "ignore" (I), "negation" ($\neg$) and "First" (F) end when a sub-component ends. Hence there is no incentive to propagate the importance. All sub-nodes' importance values are set to default and the reward or penalty script is invoked.

```
Algorithm propagateContinuation;
Data: child, the ended child node;
parent, the parent GC node
begin
    children ← parent.getChildren();
    children.remove(child);
    /* The importance is assigned to the next GC generated by the
       continuation function                                          */
    newGC ← continuationFunction.getNext();
    if newGC ≠ null then
        children.add(newGC);
        distributeImportance(parent.getImportance(), newGC);
    end
    /* Set the ended child importance to default                      */
    child.setImportance(default);
end
```

Algorithm 8: Importance propagation of the ”continuation” operator

5.3.3 LoD and GCs

This subsection highlights the overall LoD system configuration and necessary transformations made on GCs generated importance values for QoE utility computations.

Figure 5.8, presents an example of game objects mapping to adequate communications levels using GCs. As with agents organization, game objects are clustered in sets of objects with the same importance value (1) called *”importance sets”*. These sets are then associated to the appropriate communications level (2).

Since importance values are dynamic and are updated during game session, objects can move from one set to another and sets can be created, and deleted at any moment. Non-GC objects, do not influence the GC tree, they are therefore considered secondary and clustered in a set called ”default”. Further decomposition within the default set can be made, but for our experiment, we consider all Non-GC objects as part of one default set. And as we saw in importance propagation algorithms, ended GCs' objects become Non-GC objects and are affected to the default set if they still exist in the game scene. In case of object removal from the game scene, the objects are simply removed from the LoD system.

The game designer also has the possibility to explicitly attribute the default importance to a GC-object (s)he deems less important. The default set is the set of objects with the lowest importance in the organization, hence its importance value is the lowest. In our model, the importance value for the default set is automatically generated at tree initialization and it is half of the lowest generated importance.

$$\mathit{default}_{\mathrm{imp}} = \frac{1}{2} \times (\mathit{lowest}_{\mathrm{imp}}).$$

This importance value can be manually modified by the game designer.

With GCs, we saw that generated importance values are in interval [0,1], where more important objects have higher importance values. Since thresholds for LoD selection

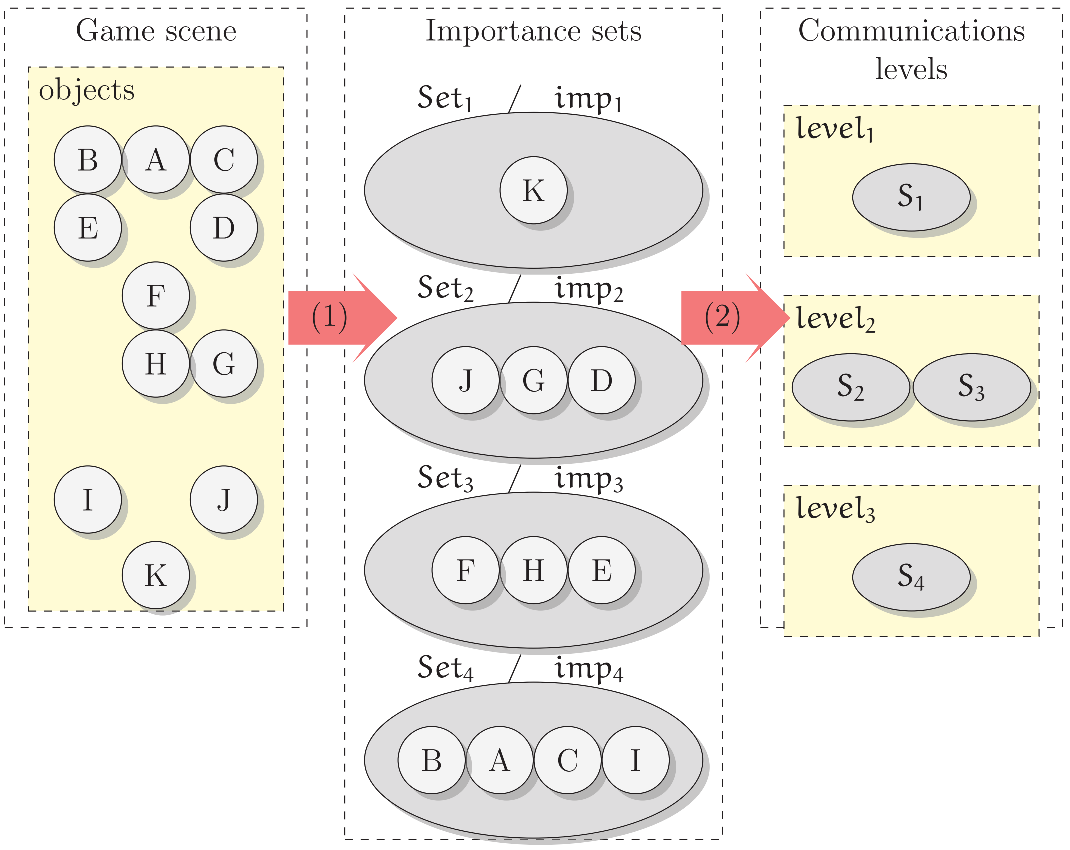


Figure 5.8 – GC based level of detail

are given from the most important objects to the less important objects, we will have decreasing importance thresholds ($\mathrm{threshold}_i > \mathrm{threshold}_{i+1}$) using GC. Thus, to be used with the qoe_util formula 5.3, a little transformation of importance values is needed. In fact, for the level selection to work, we should have a threshold array of the same size as the network threshold array ($n-1$, where n is the number of levels), and with increasing values ($\mathrm{threshold}_i < \mathrm{threshold}_{i+1}$). The array is defined, so that the smallest value refers to the objects with the highest importance and the biggest value corresponds to the lowest importance, as for the example in section 5.2.3, we had $(2, 5, 7)$. We therefore need to transform the importance values generated by GCs, in order to have increasing values as object importance decreases while keeping the same proportion and disparity between values.

We will use a simple transformation (t_r) that keeps the disparity between importance values but inverses their order in the [0,1] interval. A property for additive inverse state that, for any real numbers a, b and c:

$$\text{if} \quad a < b < c \quad \text{then} \quad -c \leq -b \leq -a \quad \text{thus} \quad 1-c \leq 1-b \leq 1-a$$

$$t_r(x) = 1 - x \quad \text{where} \quad x \in [0, 1]$$

Thus we use the transformation $t_r(x)$ where x is an importance value in the interval [0, 1]. A threshold array of $(\frac{9}{45}, \frac{6}{45}, \frac{2}{45})$ for example is transformed to $(\frac{36}{45}, \frac{39}{45}, \frac{43}{45})$.

After this transformation, the values are already in the range [0,1], thus there is no need for normalization.

5.3.4 Example

We are now able to adjust objects' importance in accordance with the changes in the gameplay. In this example we aim at illustrating step by step, how level of detail selection is made using generated importance sets in conjunction with network conditions.

We will simulate the overall adaptation process using the game "My Duck Hunt" presented in the section 5.2.3. The overall game GC tree is given in figure 5.9. The game is designed in sequential stages ($C_{f \in N}$). For this illustration example, each stage introduces 4 ducks, 2 flamingos and 3 gombas in the game scene. The atomic components provided here are: (i) eliminate (object), enables the player to eliminate ducks and gombas; this component is also used by gombas to eliminate flamingos; (ii) countdown, determines the duration of a stage. According to game rules, a stage can end for 3 reasons:

— all ducks are eliminated,

— the count down has ended,

— all flamingos are eliminated.

The elimination of gombas can not end a stage; it is optional and simply enables the player to protect flamingos and gain some score points. Let's suppose that the game designer provided different importance weights for children of the components stage(F) and protection($\wedge$) as seen in figure 5.9.

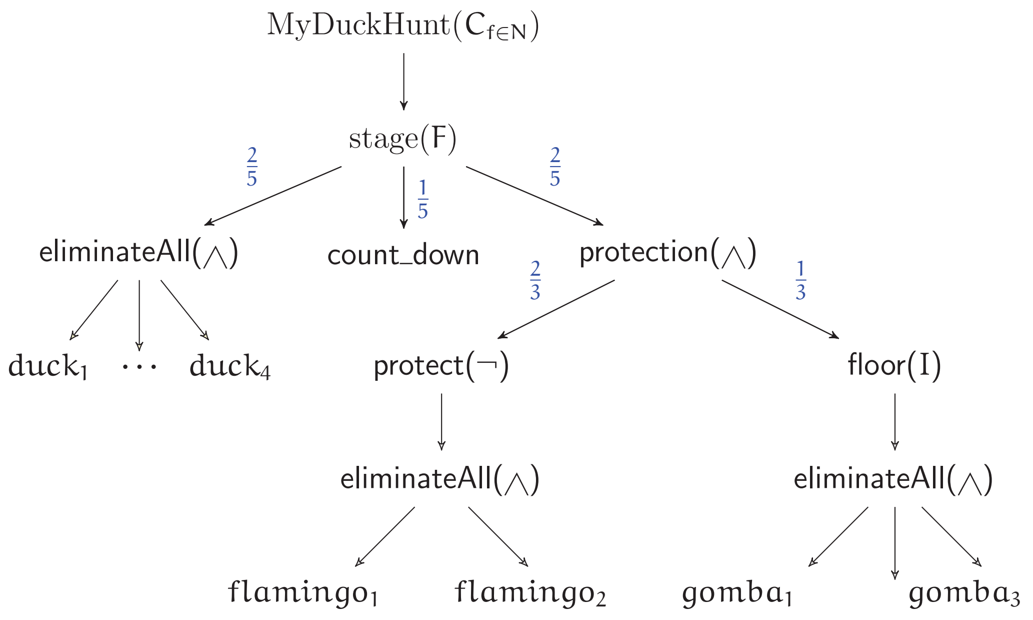


Figure 5.9 – GC tree for the game My Duck Hunt

For this example, we will use the same communications levels as for the example of subsection 5.2.3. Thus we will have four communications levels: optimal, enhanced, medium and degraded, with respectively 100Hz, 20Hz, 10Hz, 5Hz as synchronization rates. To compute qoe_util threshold array, we need threshold arrays for both network conditions and object importance.

For network delay, we will use the same array as in the example of section 5.2.3: $(100, 160, 220)$ in range $[0, 400]$. But once normalized in interval $[0, 1]$ as show in formula 5.5, this array becomes $(\frac{5}{20}, \frac{8}{20}, \frac{11}{20})$.

For object importance, the threshold array of $(\frac{1}{5}, \frac{2}{15}, \frac{2}{45})$ (which is derived from importance values of the game's GC tree at the initialization, see figure 5.12) is considered. After applying the transformation $t_r(x) = 1 - x$ in the interval $[0, 1]$ as explained in the subsection 5.3.3, object importance array becomes $(\frac{36}{45}, \frac{39}{45}, \frac{43}{45})$.

From these threshold arrays the composite threshold for the overall level selection will be:

$$(\frac{189}{360}, \frac{228}{360}, \frac{271}{360}) \Leftrightarrow (t_1 = 0.52, t_2 = 0.63, t_3 = 0.75)$$

The communications level selection semantic is shown in figure 5.10.

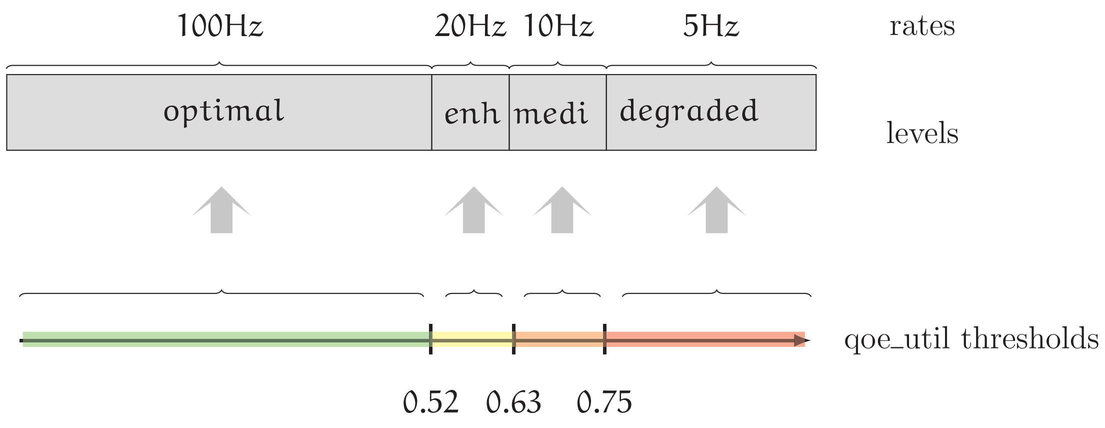


Figure 5.10 – Example of LoD selection

Following the scenario of figure 5.11, the game's progression steps are:

— the game starts with a network delay of 10 ms,

— after 30 seconds, network delay increases to 190 ms,

— after 50 seconds, eliminate(duck_4) ends with a success.

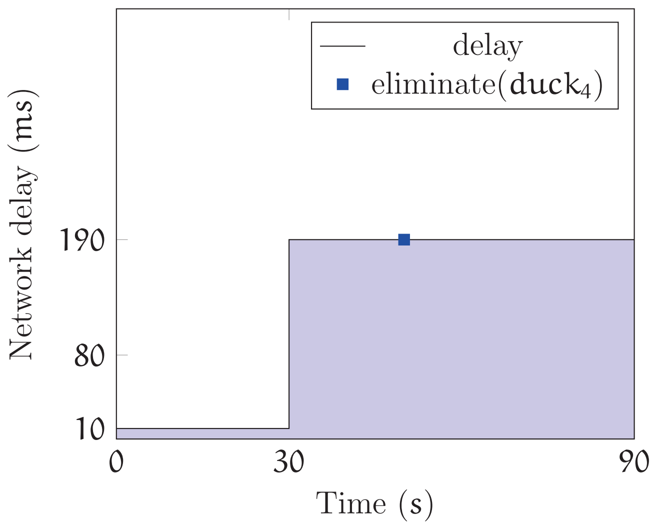


Figure 5.11 – Example of game scenario progression

At game initialization with a delay of 10 $\mathtt{ms}$

At game initialization, the importance distribution algorithm initializes each GC with an appropriate importance value according to different weights as shown in figure 5.12. With the given importance values, five different importance sets can emerge. The partitioning of the equation 5.10 shows these various sets ordered by importance.

$$\mathsf{Sets}\begin{cases} S_1 & \text{where imp} = \frac{1}{5}; \quad \text{objects} = \{\mathtt{count\_down}, \mathtt{reticle}\} \\ S_2 & \text{where imp} = \frac{2}{15}; \quad \text{objects} = \{\mathtt{flamingo}_1, \mathtt{flamingo}_2\} \\ S_3 & \text{where imp} = \frac{1}{10}; \quad \text{objects} = \{\mathtt{duck}_1, \dots, \mathtt{duck}_4\} \\ S_4 & \text{where imp} = \frac{2}{45}; \quad \text{objects} = \{\mathtt{gomba}_1, \dots, \mathtt{gomba}_3\} \\ S_5 & \text{where imp} = \text{default}; \quad \text{objects} = \{\mathtt{clouds}\} \end{cases} \tag{5.10}$$

The set S_5 with an importance of $\mathtt{default}$ always contains Non-GC objects, at this stage, the clouds. The lowest generated importance is $\frac{2}{45}$ thus the default importance is $\frac{1}{45}$.

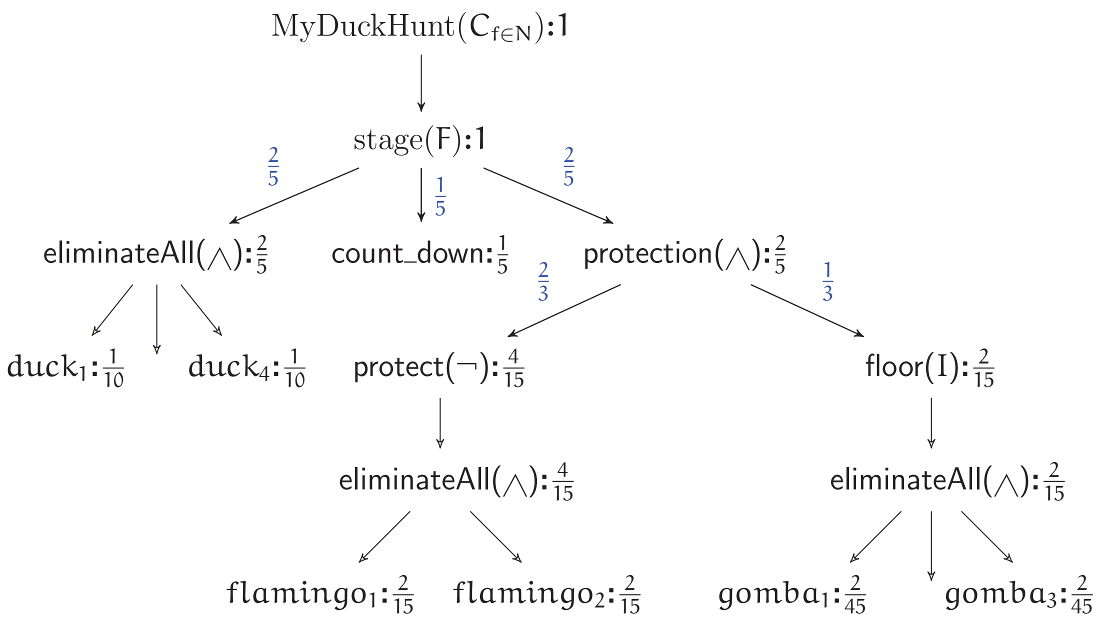


Figure 5.12 – Initial importance distribution

Let's now compute the qoe_util of each set and then use it to determine the appropriate communications level. With a delay of 10 $\mathtt{ms}$, we have $\mathtt{metric}_{\mathrm{net}} = \frac{1}{40}$.

- For set S_1, with an importance of $\frac{1}{5}$, the computed $\mathtt{metric}_{\mathrm{imp}} = \frac{4}{5}$. The resulting $\mathtt{qoe\_util} = \frac{33}{80} = 0.41$, which is less than t_1. Therefore the selected level is the **optimal level**.
- For set S_2, with an importance of $\frac{2}{15}$, the computed $\mathtt{metric}_{\mathrm{imp}} = \frac{13}{15}$. The resulting $\mathtt{qoe\_util} = \frac{107}{240} = 0.44$, which is less than t_1. Therefore the selected level is the **optimal level**.
- For set S_3, with an importance of $\frac{1}{10}$, the computed $\mathtt{metric}_{\mathrm{imp}} = \frac{9}{10}$. The resulting $\mathtt{qoe\_util} = \frac{37}{80} = 0.46$, which is less than t_1. Therefore the selected level is the **optimal level**.

— For set S_4, with an importance of $\frac{2}{45}$, the computed $\mathtt{metric}_{imp} = \frac{43}{45}$. The resulting $\mathtt{qoe\_util} = \frac{353}{720} = 0.49$, which is less than t_1. Therefore the selected level is the **optimal level**.

— For set S_5, with an importance of $\frac{1}{45}$, the computed $\mathtt{metric}_{imp} = \frac{44}{45}$. The resulting $\mathtt{qoe\_util} = \frac{361}{720} = 0.50$, which is less than t_1. Therefore the selected level is the **optimal level**.

The results of these calculations are given below, where q represents the $\mathtt{qoe\_util}$:

$$\begin{cases} S_1 & q_1 = 0.41, & level = optimal \\ S_2 & q_2 = 0.44, & level = optimal \\ S_3 & q_3 = 0.46, & level = optimal \\ S_4 & q_4 = 0.49, & level = optimal \\ S_5 & q_5 = 0.50, & level = optimal \end{cases}$$

Figure 5.13 shows the obtained level selection at this point.

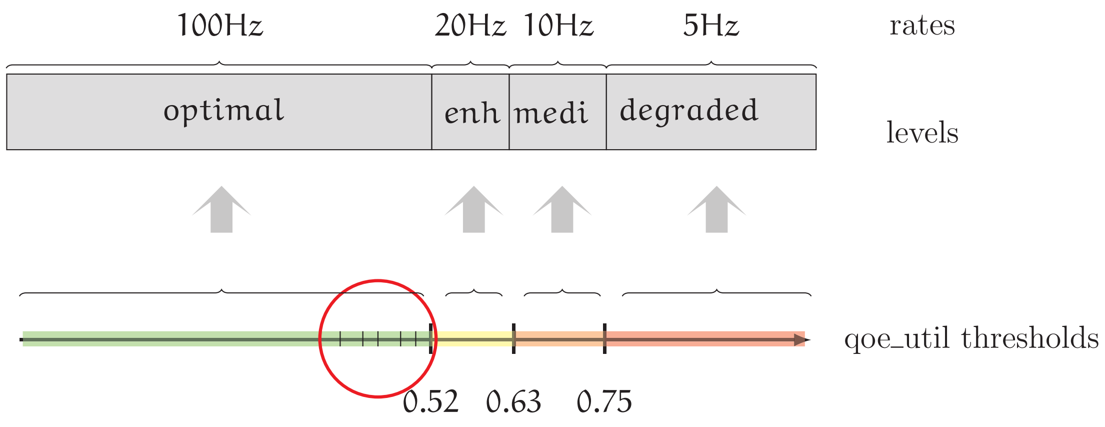


Figure 5.13 – LoD selection at initialization

We can see that, only one level is occupied and all objects have the biggest share of the communications resources(see the circled interval on the qoe_util axis). Even the Non-GC objects of the default set are in a communications level with high synchronization rate.

After 30s, the delay goes to 190 ms

At this point, the sets have not changed, we calculate a new network metric using 190 ms. We have $\mathtt{metric}_{net} = \frac{19}{40}$.

The results of qoe_util calculations with these new settings are given below:

$$\begin{cases} S_1 & q_1 = 0.62, & level = enhanced \\ S_2 & q_2 = 0.67, & level = medium \\ S_3 & q_3 = 0.68, & level = meduim \\ S_4 & q_4 = 0.71, & level = meduim \\ S_5 & q_5 = 0.72, & level = meduim \end{cases}$$

Figure 5.14 shows the obtained level selection at this point.

This configuration shows how the LoD reacts to bad network conditions. We can notice that our approach guarantees that, important objects have more synchronization

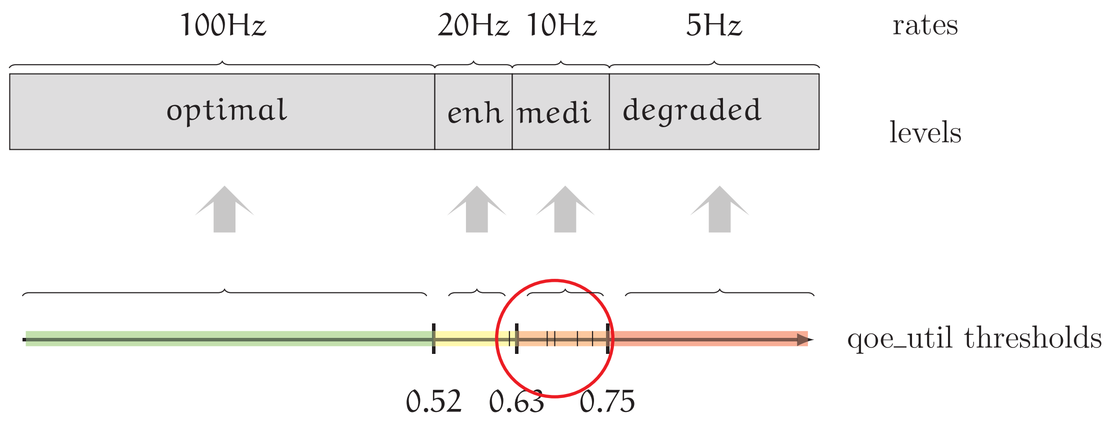


Figure 5.14 – LoD selection when the delay goes to 190 ms

resources(see property 5.6). Unlike our approach, a naive solution without LoD will have all the game objects synchronizing their states at the highest possible rate despite the network condition. They will evenly compete for network resources, therefore will be impacted the same way by any network congestion. Whereas with the LoD adaptation, less important objects are assigned to communications levels with gracefully lowered synchronization rates. This demonstrates the validity of our property 5.7 about adaptability to network resources fluctuations.

After 50s, eliminate(duck_4) ends with a success

At this point, the network delay has not changed. When a GC ends, its importance goes back to the default value. The GC tree importance is shown in figure 5.15. In this example, eliminated objects are removed from the game scene. Therefore there is no need to set their importance to default, nor to transfer them to the default set. The remaining siblings (with dashed arrows in figure 5.15) ducks_1, duck_2 and duck_3 benefit from this and see their importance go from $\frac{1}{10}$ to $\frac{2}{15}$. They are therefore moved to the S_2, and the S_3 with an importance of $\frac{1}{10}$ is dissolved. The resulting sets are the ones below.

$$\text{Sets}\begin{cases} S_1 & \text{where imp} = \frac{1}{5}; \quad \text{objects} = \{\mathsf{count\_down}, \mathsf{reticle}\} \\ S_2 & \text{where imp} = \frac{2}{15}; \quad \text{objects} = \{\mathsf{flamingo}_1, \mathsf{flamingo}_2, \mathsf{duck}_1, \dots, \mathsf{duck}_3\} \\ S_3 & \text{where imp} = \frac{2}{45}; \quad \text{objects} = \{\mathsf{gomba}_1, \dots, \mathsf{gomba}_3\} \\ S_4 & \text{where imp} = \frac{1}{45}; \quad \text{objects} = \{\mathsf{clouds}\} \end{cases}$$

The network metric has not changed, $\mathsf{metric}_{\mathsf{net}} = \frac{19}{40}$.
The results of qoe_util calculations with these new settings are given below:

$$\begin{cases} S_1 & q_1 = 0.62, & \mathsf{level} = \mathsf{enhanced} \\ S_2 & q_2 = 0.67, & \mathsf{level} = \mathsf{medium} \\ S_3 & q_3 = 0.71, & \mathsf{level} = \mathsf{medium} \\ S_4 & q_4 = 0.72, & \mathsf{level} = \mathsf{meduim} \end{cases}$$

Figure 5.16 shows the obtained level selection at this point.

This levels-objects mapping shows that, as the game progresses the importance values of game objects change and as a result their associations with communications levels

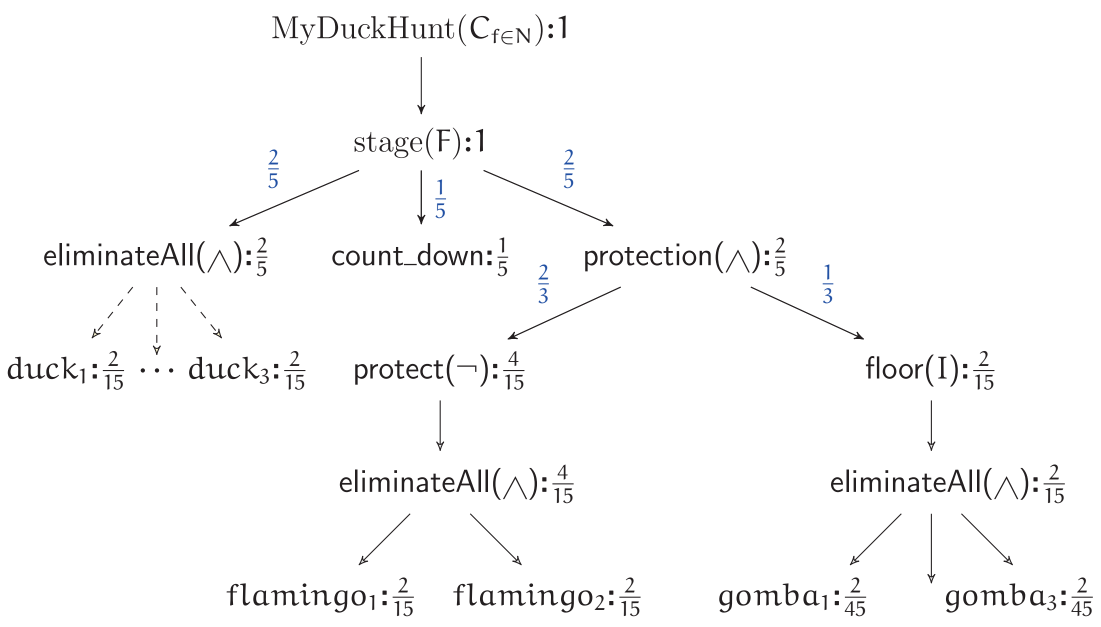


Figure 5.15 – GC tree after eliminate($\mathtt{duck}_4$) succeeded

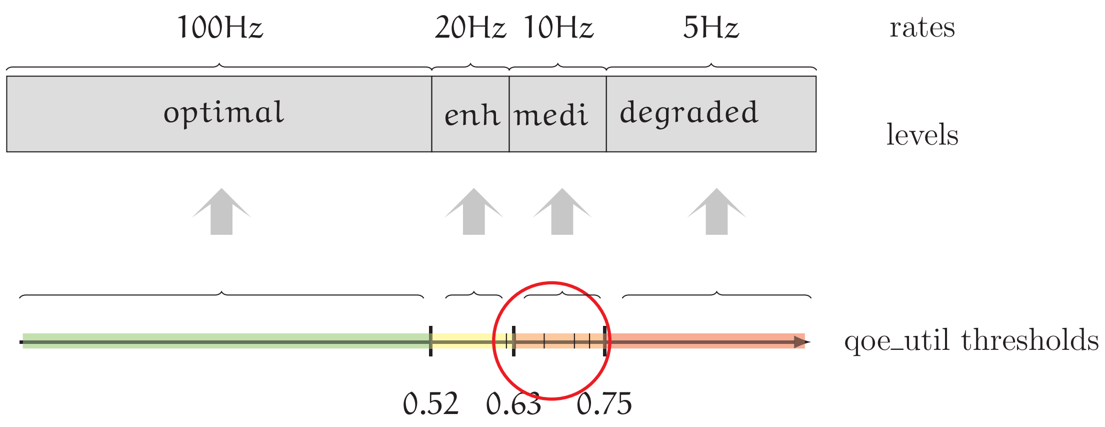


Figure 5.16 – LoD selection when eliminate($\mathtt{duck}_4$) ends with a success

change. We can see that our invariant axioms 5.6 about objects importance and their network resources is always respected even when objects importance change.

As a summary for this example, we can see how objects importance are dynamically recalculated and objects are assigned to appropriate sets as the game progresses. As the importance sets change, the members of the communications levels also change. This verifies our axioms 5.6 and 5.7. It is important to know that the efficiency of this adaptation approach highly depends on the LoD setup: the synchronization rates and the threshold arrays for both network conditions and object importance must therefore be set carefully.

5.4 Conclusion

This chapter presented our LoD based communications adaptation approach for cloud games. The game's communications is dynamically adapted to the network conditions as well as the importance of the game objects as the game scenario progresses. To model the structure of the game and changes in objects importance two approaches are used: agents organizations and gameplay components to model the structure of a game. With agent organizations, we exploited the roles of game objects in the organization to determine their importance. Therefore, a change in object's role steers a change in its importance. Using gameplay components, the game is modeled as an OCR tree, that changes as the game progresses. From this OCR tree, we create an importance tree by associating an importance value to each node of the tree. A change in the game state is therefore reflected by a change in the importance tree.

In the next chapter, we will present the game prototypes and the pilot experiments carried out to evaluate the benefits of our adaptation scheme.

CHAPTER 6

Experimental Evaluation

Contents

This chapter presents the implementation and experimental evaluation of our adaptation approach. In the implementation part, we present the game architecture of the game engines used to develop our games. We then present two games and their designs, followed by the design of the game communication adaptation. For the evaluation, we describe pilot experiments carried out to assess the benefits of our proposition. We conclude the chapter with a general discussion on QoE maintenance provided by our approach on lossy and congested networks.

6.1 Prototype

Prototypes developed over the course of this thesis were implemented in JAVA, using game libraries such as Slick2D[1] and libGDX[2]. We present the overall game architecture and the final implementation of each game prototype.

6.1.1 Overall game architecture

At the highest level, the basic architectural pattern is a simple client-server model with the cloud gaming paradigm. This configuration involves multiple clients connecting to a single central server. As shown in figure 6.1, the used architecture consists of four major components: client, connection server, game server, and persistence server.

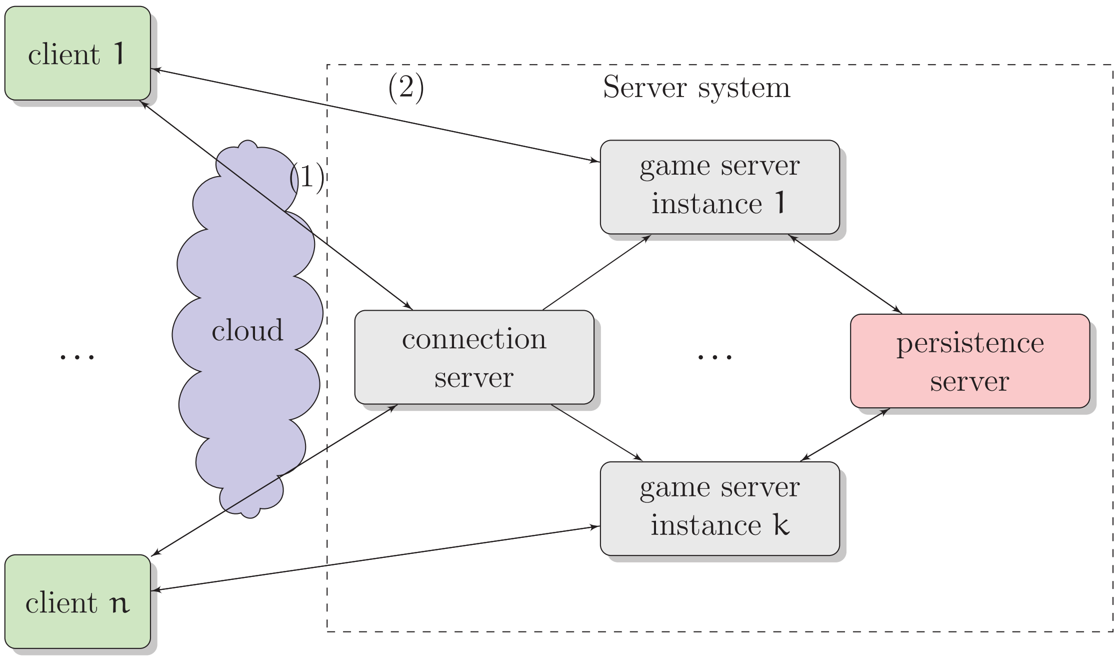


Figure 6.1 – The overall game architecture

On the frontend, we have a game client, representing the game itself. It interacts with a game server system by sending controller inputs and receiving game state updates through the network using standard Internet communications protocols (TCP and UDP).

On the backend, we have:

— **Connection server:** clients connect to the game via a connection service (see the link (1) in figure 6.1). It is in charge of authenticating clients (if necessary) and creating new game server instances (also known as shards) to host new game sessions. For multiplayer game sessions, the connection server redirects clients joining an existing game session to the appropriate game server instance. Once the connection is established, clients directly communicate with their respective game server instances (see the link (2) in figure 6.1).

— **Game server instance:** it is an instance of game server created for a specific game session. It contains the central game model and runs the game logic as explained

1. http://slick.ninjacave.com/
2. http://libgdx.badlogicgames.com/

in the section 2.2 of the chapter 2. It is responsible for receiving players' inputs, updating the game model and sending game state updates to clients.

— **Persistence server:** it holds all information that need to be persisted. Mainly user's information and credentials, as well as game statistics and some game events.

We adopted this same architecture in the Village project using the RedHat's[3] cloud application platform OpenShift[4], to deploy and test the game prototypes we co-developed. MongoDB[5] was used as persistence server. For the evaluation of our adaptation technique, all the components of the server system were hosted on a single machine. So we had a server machine and a client machine connected to a LAN.

6.1.2 Game model

Our focus in this thesis is adapting state synchronization of all game entities in the vicinity area of the player. This view area is based on the game model. We chose a core game model, where a game is divided into a set of scenes containing game entities (see figure 6.2). The number of scenes is variable and depends on the game scenario. A scene is a part of a play in a single location in the game, in which a particular action or activity occurs. It defines player's view of the virtual environement with game entities.

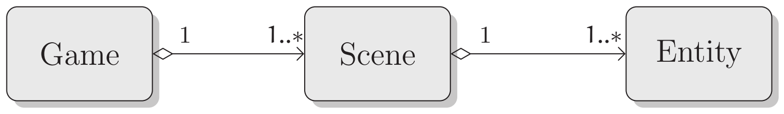


Figure 6.2 – Core game model

With this game model, the synchronization module will simply periodically cycle through the list of entities in the current game scene, and send their state updates to clients.

In the course of this thesis, we have co-developed two game engines using this game model as a basis for game structure: the *Agent Game Development Engine* (AGDE) and the *Game Agent Mechanics Engine* (GAME).

AGDE

AGDE [SMILE, 2012] is a game engine developed by the SMILE research team at LIRMM, with the aim of using artificial agents for fast 2D game prototyping. It is a Java based framework, developed using the game library Slick2D. Slick2D provides a set of tools and utilities wrapped around LWJGL and OpenGL bindings. Slick2D includes support for images, animations, particles, sounds, music and more utilities for game development. The game structure of AGDE is illustrated in figure 6.3, and it is the following:

— Structure: it is the container class of all the structure of a game, it holds a table of game levels. A particular level is considered active.

— Level: it represents a game level and can contain multiple game scenes, with only one active scene.

— Scene: it represents a game scene and belongs to only one game level.

3. http://www.redhat.com/en
4. https://www.openshift.com/
5. http://www.mongodb.org/

— Layer: it represents a layer in a game scene. A game scene can have one or more layers, but only one is considered active.

— Entity: it is a game agent with a state that can be updated and a behavior that can be controlled. Entities have a set of scripts and actions that enable them to control their states and behaviors. Scripts play a major part in this framework, because they implement the logic and control flows of game entities. An entity's functional role in the game is assigned through the script associated to the entity. An entity can also have one or multiple graphic and audio representations that can be rendered. In our context of objects replication, only entities with physical representation are synchronized: they represent the replicated game objects.

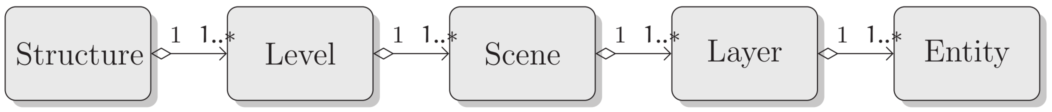


Figure 6.3 – AGDE's game structure

On the implementation side, AGDE is based on the concept of "Activity". An activity represents every game activity with state management. As shown in the class diagram of figure 6.4, each activity has a life cycle and predefined methods to call upon activity creation and destruction, insertion and suppression, activation and deactivation from game tree (See the interface ActivityLifeCycle in figure 6.4). As its name suggests, StructureBase class is the base class for all game structure elements. Elements such as Structure, Level and Scene, inherit from the StructureBase class. Each of these elements therefore has a special entity controlling its behavior, namely game director, level director and scene director respectively. Only entities and scripts have an update cycle, while only entities, levels and scene have rendering capabilities.

We used AGDE to develop the shooting game called "My Duck Hunt", for the evaluation of our communications adaptation technique with organization.

GAME

GAME [SMILE, 2013] is a Java framework for fast game mechanics implementation and multi-agent behavior. Also developed by the SMILE research team at LIRMM, core modules of the engine are designed so that the engine can build and run game mechanics independently from aesthetic aspects. This characteristic enables GAME to be plugged into a variety of game graphics libraries (back-end engines) for graphics rendering. LibGDX is the back-end game library provided by default with GAME.

Libgdx allows developers to write his/her code once and deploy it to multiple platforms without modification using the GWT technology. It currently supports Windows, Linux, Mac OS X, Android, iOS and HTML5 as target platforms. Just like Slick2D, Libgdx ties in many third party libraries such as LWJGL and OpenGL to provide its functionalities. Its powerful set of APIs helps developers with common game tasks such as rendering sprites and texts, building user interfaces, playing back sound effects and music streams. It also features linear algebra and trigonometry calculations, parsing JSON and XML, etc [LibGDX, 2013].

The game structure of GAME is illustrated in figure 6.5, and it is the following:

— Game: a game is simply a sequence of scenes. At game creation, an initial agent called "game director" is created and associated with the initial script, responsible

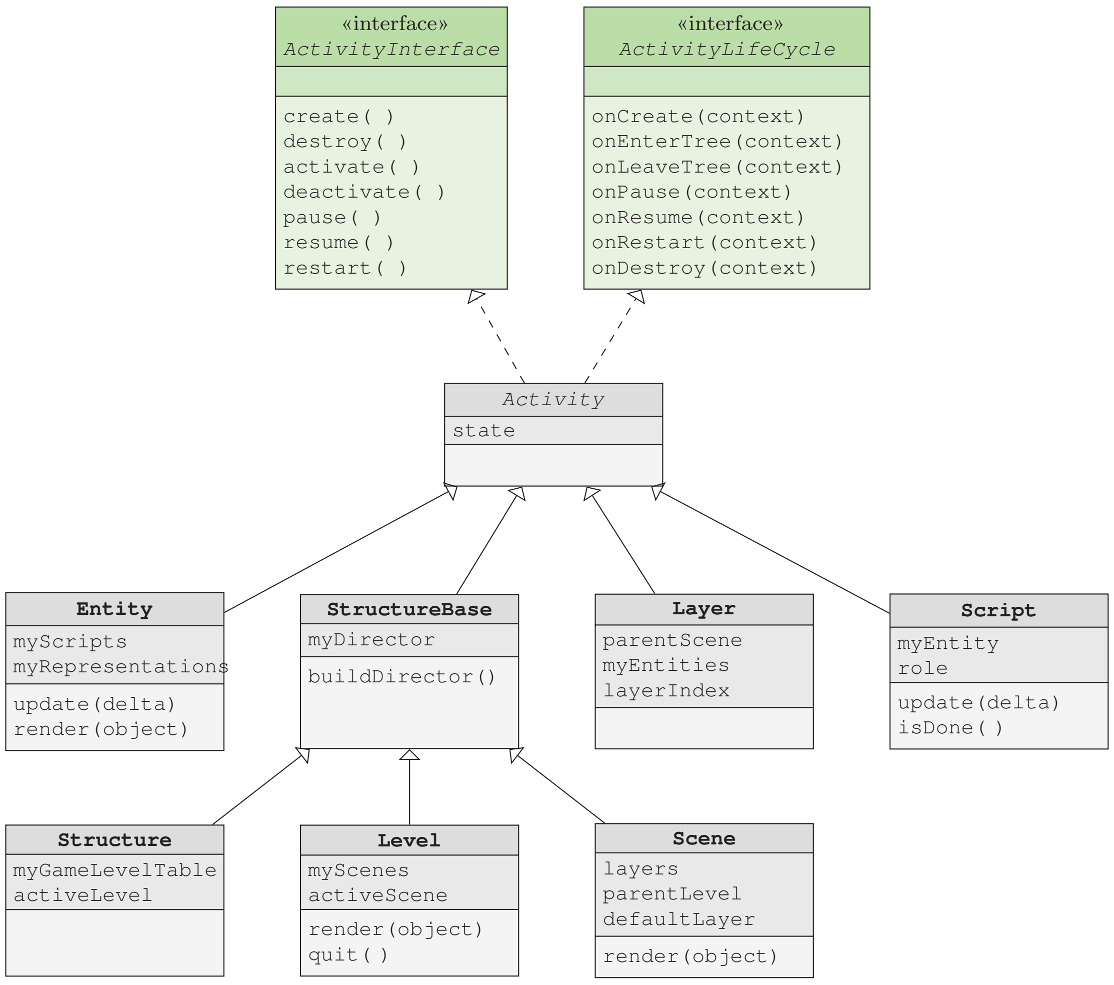


Figure 6.4 – AGDE's core inheritance class diagram

for loading the first game scene. As for AGDE, only one scene is active at a given point in time.

— Scene: it represents a game scene, and contains a sequence of game agents in the scene and an artifact container used to store artifacts of the scene. A scene is inserted in the game tree, by a script, owned by a game agent.

— Agent: it is a game agent responsible for implementing the game mechanics and the game objects' behaviors. It can be a proactive agent or only a reactive agent. It maintains a list of scripts that enable it to interact with its environment. It can perceive changes in its environment and maintain a list of events received over the update cycles. An agent can control one or multiple game artifacts. The importance of an agent in the game determines the importance of its artifacts.

— Artifact: it represents an object of the back-end engine, used as a buffer between GAME scripts and back-end objects. An artifact is associated with a "Content" object, that can be used either by the scripts or the back-end engine as a placeholder for graphic and audio assets. For our objects replication, game artifacts are the only game objects synchronized between clients and server.

On the implementation point of view, the class diagram of figure 6.6 illustrates the core concepts of the engine. All game elements are based on the concept of "GameObject". A GameObject represents an identified game object. All these game elements are capable

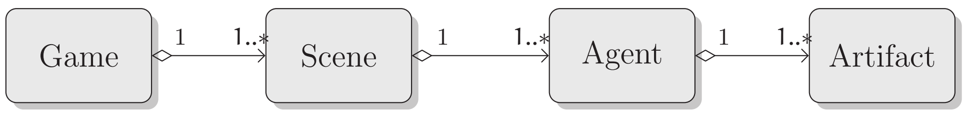


Figure 6.5 – GAME's game structure

of updating their states, while only artifacts have rendering capabilities and only agents can receive perception events from the game.

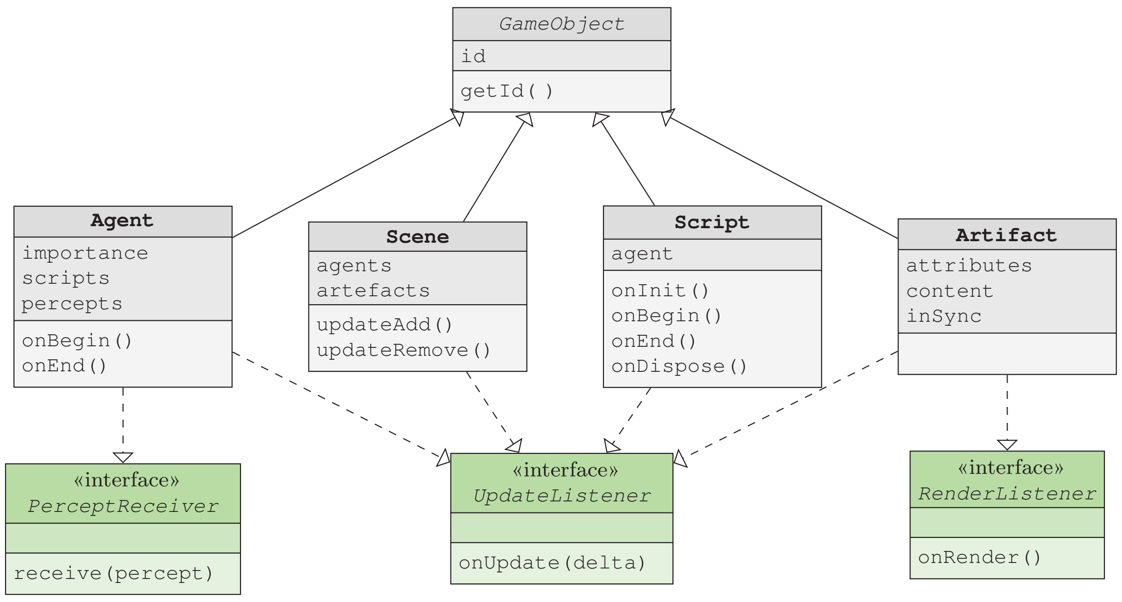


Figure 6.6 – GAME's core inheritance class diagram

We used GAME to develop the game called ”Crazy Runner”, for the evaluation of our communications adaptation with gameplay components.

6.1.3 ”My Duck Hunt” game

As introduced in 5.2.3, My Duck Hunt is a single player shooting video game we developed for evaluation purposes. In this game, player has to eliminate or protect entities moving at different paces and directions in the game scene. The basic game mechanic for the player is to move its reticle to reach target objects and try to eliminate them in order to gather the most points. As illustrated in the screenshoot of figure 6.7. The game has five types of entities:

— Player controlled reticle, which is the round entities, highlighted with a red square in the screenshot.

— Ducks are flying entities, highlighted with a star.

— Flamingos are flying entities, highlighted with a circle.

— Gombas are entities, highlighted with a diamond.

— Clouds are entities, highlighted with a rectangle.

The game is structured in stages or quests containing each a certain amount of these game objects. To go to the next stage, the player has to eliminate all ducks while protecting flamingos from gombas. Each stage has a timer, and when the timer expires,

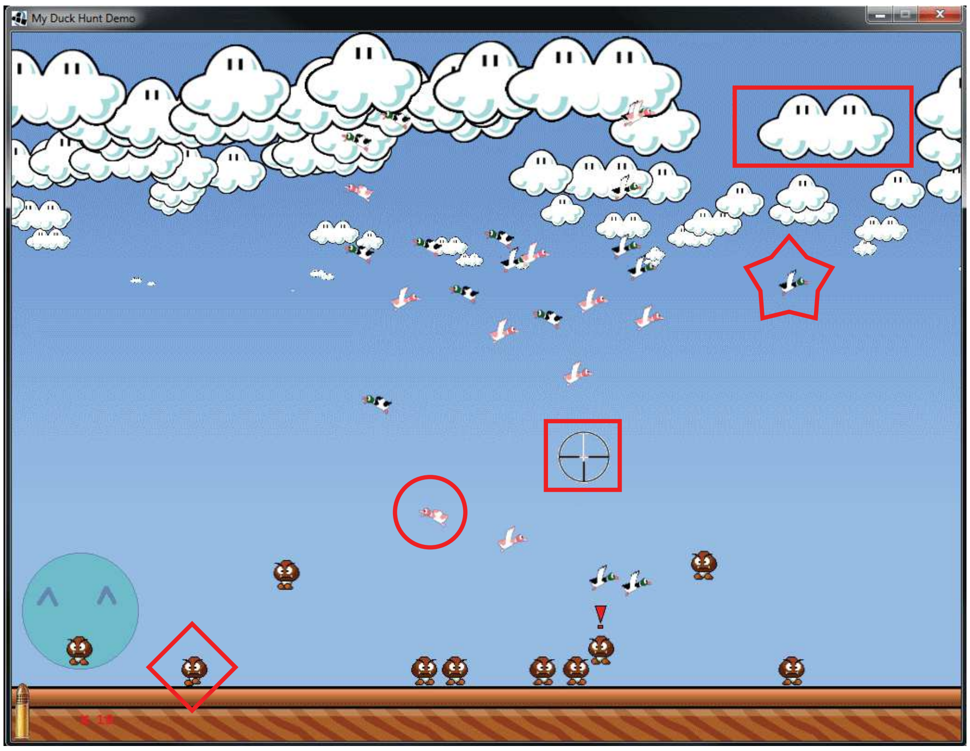


Figure 6.7 – "My Duck Hunt"'s game objects

regardless of the game state, the current stage ends and the next stage is automatically loaded.

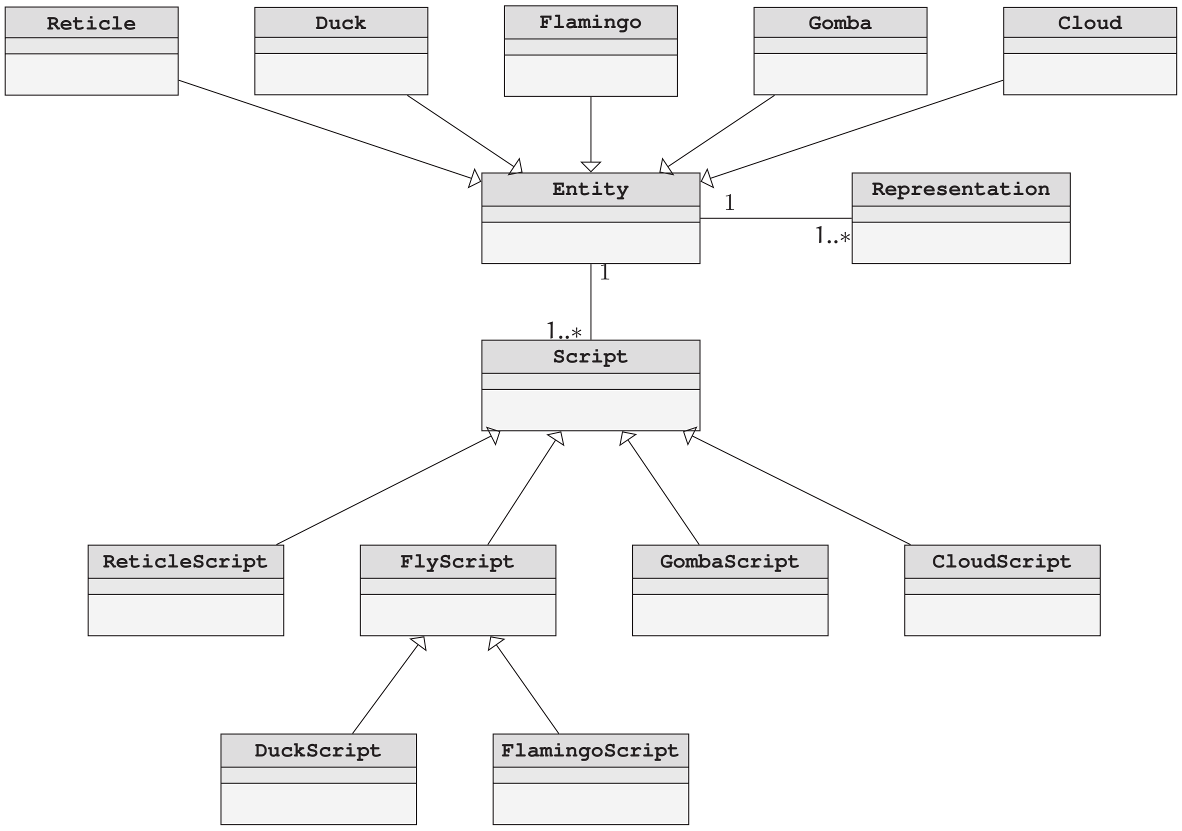


Figure 6.8 – A simplified class diagram of "My duck hunt"'s game objects

Figure 6.8 shows a simplified class diagram of the game. Each game object is an entity associated with a graphical representation and a script. For example, a duck is a kind of flying object, which graphical representation is an animation (denoted by the star in the screenshot of figure 6.7). The script Duckscript associated with this type of objects, enables them to carry a set of actions and move around the game screen at a specified speed. Through ReticleScript, the player is able to control the reticle object using the mouse, in order to aim and shoot at its targets. GombaScript, enables gombas to move and also to detect and attack flamingos.

The organization used for the adaptation is the one below.

$$\mathsf{Org}\begin{cases} \mathsf{G}_1 & \text{where role} = \{\mathsf{player}\}; \quad \text{objects} = \{\text{reticle}\} \\ \mathsf{G}_2 & \text{where role} = \{\mathsf{target}\}; \quad \text{objects} = \{\text{ducks}\} \\ \mathsf{G}_3 & \text{where role} = \{\mathsf{protect}\}; \quad \text{objects} = \{\text{flamingos}\} \\ \mathsf{G}_4 & \text{where role} = \{\mathsf{enemy}\}; \quad \text{objects} = \{\text{gombas}\} \\ \mathsf{G}_5 & \text{where role} = \{\mathsf{decoration}\}; \quad \text{objects} = \{\text{clouds}\} \end{cases} \tag{6.1}$$

6.1.4 "Crazy runner" game

Crazy-Runner is a memory and card game asking players to rapidly identify a card among a set of cards. A screenshot of this game is shown in figure 6.9. The player has a protege, that is a game character continuously running in the game scene with the aim of reaching a finish line. We call him the "runner" (highlighted with the diamond in the screenshot). Player's goal is to assist his protege by making him achieve the best race time possible. Along the way the runner faces some obstacles in form of game objects, and any collision with an obstacle slows him down. Among those obstacles, you can have grenades (highlighted with the circle in the screenshot), turtles (highlighted with the square in the screenshot) and birds (highlighted with the rectangle in the screenshot). The game scene also contains background objects such as trees, benches and boxes.

The player has the ability to eliminate turtles moving on the ground floor and flying birds simply by shooting at them using his reticle(highlighted with the star in the screenshot). As it refers to the grenade, the player should respond to a memory assignment in order to eliminate the grenade. The way it works is, the name of a card is first announced. After a few moments, three cards are presented in the game screen (as shown in the screenshot). The player must quickly select the previously announced card among presented cards. If (s)he chooses the wrong card or if (s)he takes too long to respond, the runner collides with the grenade and loses speed. If (s)he chooses the right card, all the obstacles are eliminated and a new wave of obstacles is generated for the next assignment.

We implemented this game using gameplay components to handle changing objects' importance. Figure 6.10 shows the GC tree of Crazy Runner. Three atomic GC are provided:

— select enables the player to select a card among presented cards.

— reach enables the runner to reach a given position. This GC can be used to represent the fact of reaching the finish line ($\mathsf{finish}_{\mathsf{line}}$), and also the fact of reaching a grenade posted in the runner's path $\mathsf{grenade}$.

— $\mathsf{eliminate}$ enables the player to protect the runner by shooting at birds and turtles attempting to slow the runner down.

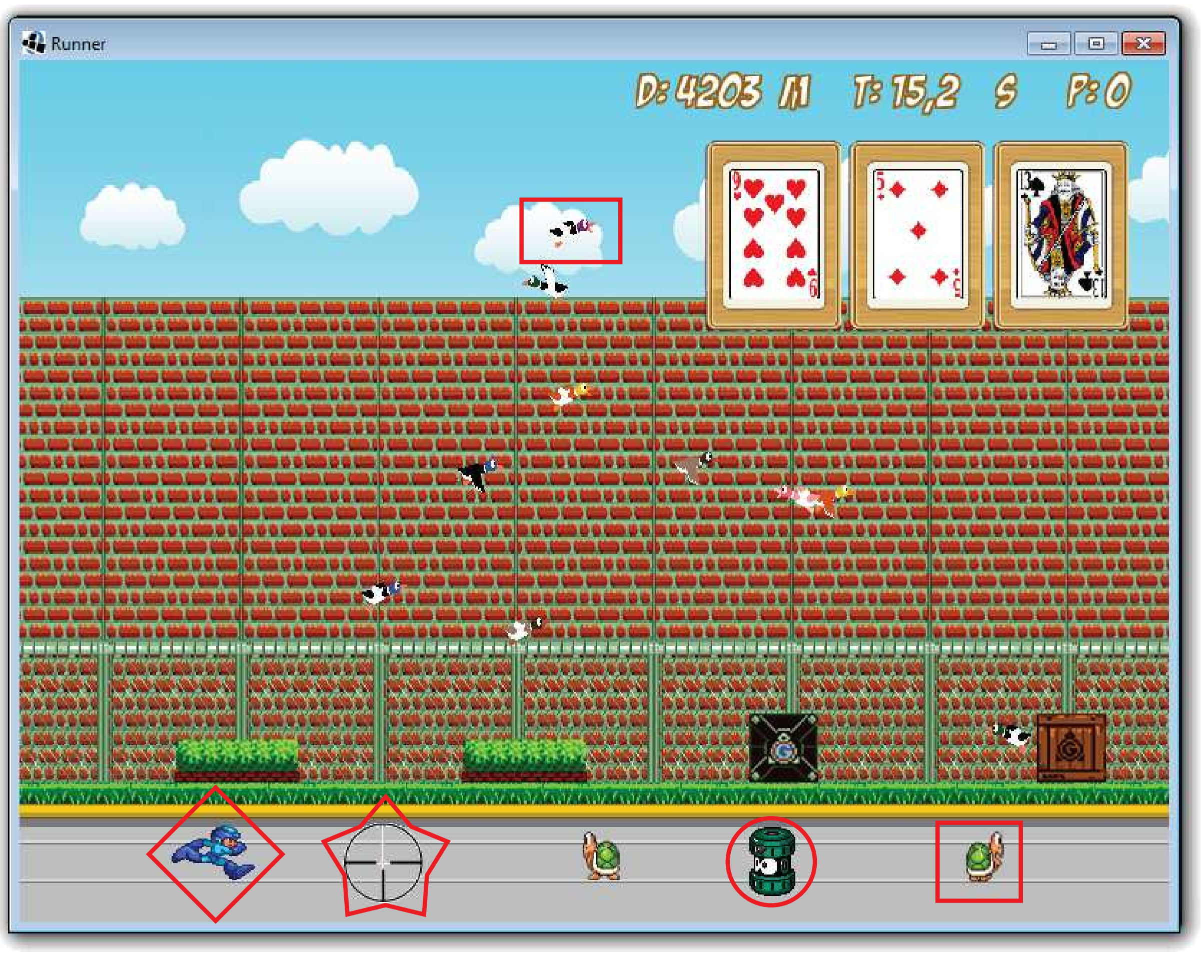


Figure 6.9 – "Crazy runner"'s game objects

As composite GC, we have `step` marks a step in the ultimate goal of reaching the finish line. Thus to reach the finish line, the runner has to go through `n` steps. In each game step, the player has to respond to `m` card assignments materialized by the composite GC find. `Find` represents a card assignment and has three components: `k` obstacles to eliminate provided by the GC `eliminate`; a grenade moving toward the runner provided by the GC `grenade` and 3 cards to select from presented in the top right corner of the scene and provided by the GC `select`.

Figure 6.11 presents a simplified class diagram of game objects of Crazy Runner game. Each game artifact, is manipulated by an agent through scripts. We therefore have an agent for player's reticle, runner, grenades and other obstacles such as turtles and birds. This game was part of the 6 serious games we co-developed in the "Village" project to explore game mechanics that are suitable for cognitive stimulation [Village, 2013].

6.1.5 Game communications adaptation

The adaptation scheme focuses on adapting network resources distribution according to game objects' importance and network conditions. Figure 6.12 shows the class diagram of importance sets and communications levels. The rational behind this diagram is that: objects are assembled in sets according to their importance in the game, and each set of objects belongs to a specific communications level at a given point in time. The

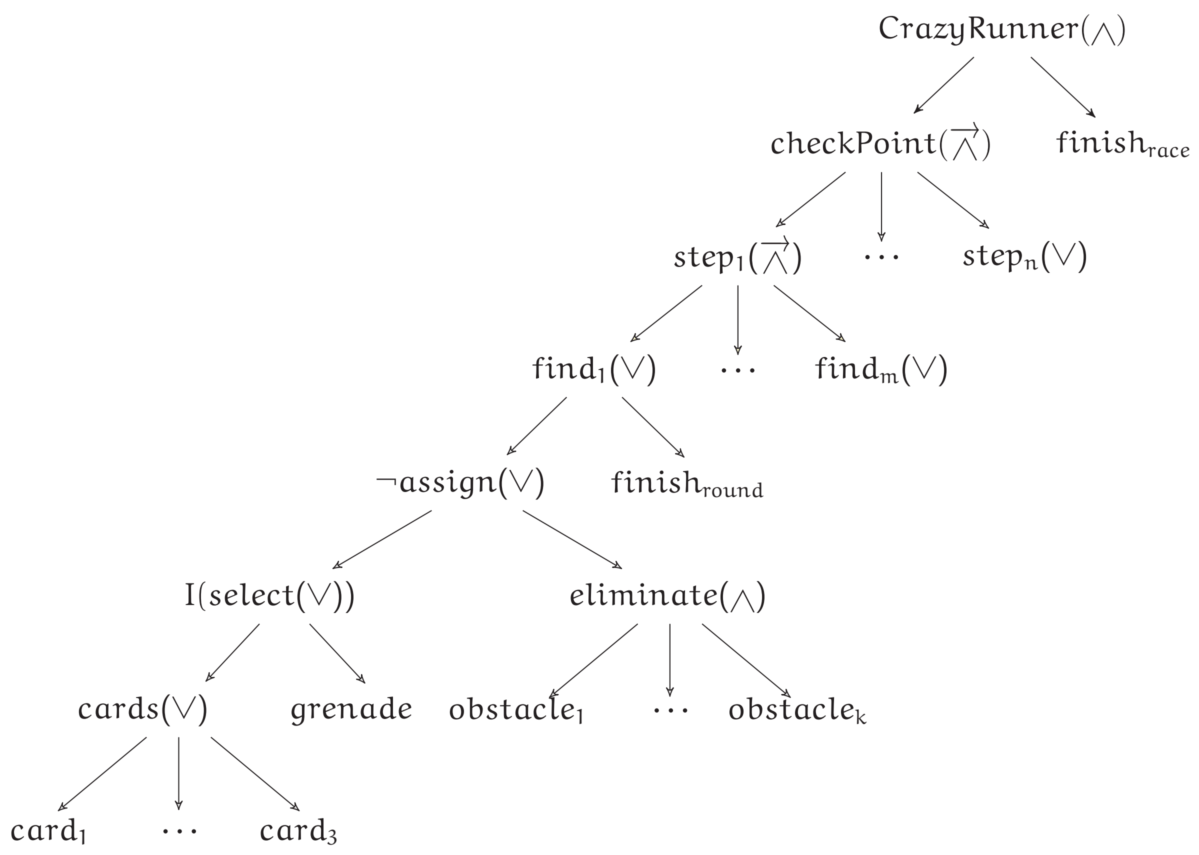


Figure 6.10 – GC tree of the game ”Crazy Runner”

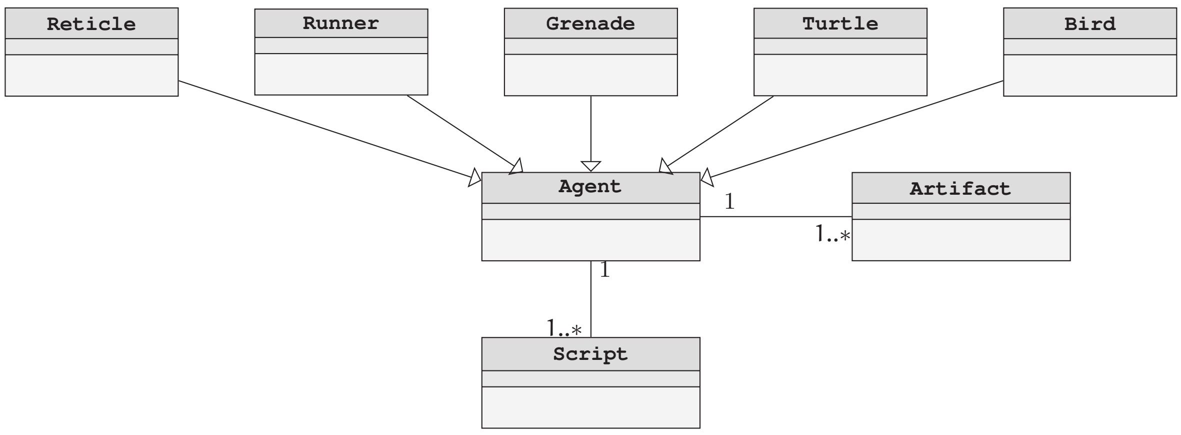


Figure 6.11 – A simplified class diagram of Crazy Runner’s game objects

selection of the appropriate communications level for a set is made by comparing objects’ QoE utilities with QoE utility thresholds of the LoD system. As showed in previous chapter, these QoE utilities are computed and updated during game session using objects’ importance and current network conditions.

Once a level is selected for an object, object’s states is synchronized at a rate determined by the level.

Of course, there is no limit in the number of objects, sets and levels, but for the prototypes used in the experimental evaluation, we defined four communications levels sorted by synchronization resources:

— Optimal level: provides objects with the highest communications resources. In this level the synchronization rate of an object is 100Hz.

— Enhanced level: provides objects with relatively high communications resources but lesser than the optimal level. rate = 20Hz.

— Medium level: provides objects with average network resources. rate = 10Hz.

— Degraded level: provides objects with the lowest communications resources. rate = 5Hz.

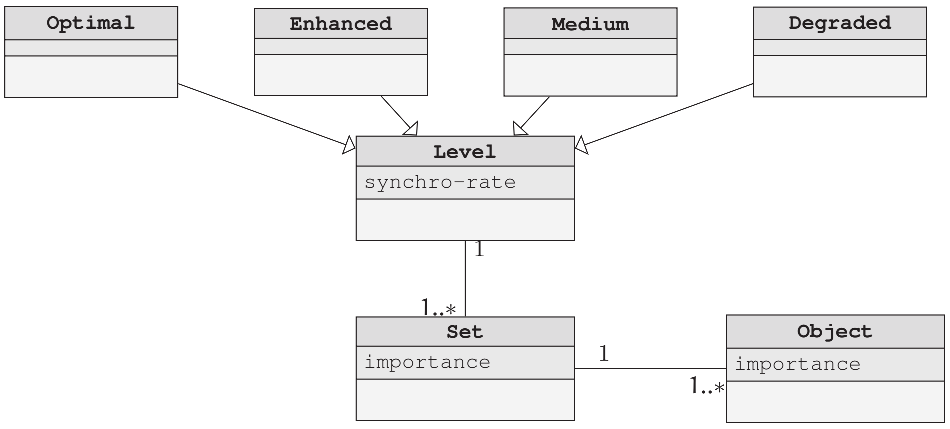


Figure 6.12 – A class diagram of the communications levels

Experiments conducted with the games ”My Duck Hunt” and ”Crazy runner” use this adaptation mechanism and are presented in next section.

6.2 Experimentation

The objective of these experiments is to evaluate the impact of our LoD based adaptation in cloud gaming on player’s QoE. In order to determine this impact, we assess and compare players’ QoE during game sessions with and without the proposed approach. Due to the subjective nature of human experience, it is very difficult to objectively evaluate player’s QoE. Most works in the literature [Chen et al., 2009], [Chang et al., 2010] use the subjective methodology called the MOS (Mean Opinion Score) rating test, to obtain player’s view of game quality. In this thesis we used MOS measurements to evaluate our proposition. Using MOS, players are required to give each game session a rate using the rating scheme of table 6.1:

MOS	**Quality**	**Impairment**
5	Excellent	Imperceptible
4	Good	Perceptible but not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

Table 6.1 – MOS rating scheme

We used the paired Student's t-test to determine whether the two game versions are significantly different from each other in terms of players' QoE. This test is also referred to as *"repeated measures" t-test.* Student t-test is a statistical test which is widely used to compare the mean of two groups of samples. In fact, t-test is suitable for any statistical hypothesis test in which test statistics follow a Student's t-distribution. Student's t-distribution is a family of continuous probability distributions that arise when estimating the mean of a normally distributed population in situations where the sample size is small [Hurst, 1995]. This definition fits with the population and the observations of our experiment as well as what we are trying to demonstrate with this statistical study. This explains why we chose this test.

6.2.1 Organization based adaptation

This subsection presents the experimental evaluation of the adaptation using organization to manage objects' importance.

Participants

The test was conducted with 9 participants aged between 21 and 30 years with an average age of 25.33. The distribution of participants, based on their playing frequencies is given in table 6.2. Only one participant reported that he does not play video games. Most participants play games at least once per week.

Never	**1 per year**	**1 per month**	**1 per week**	**everyday**
1	1	2	5	0

Table 6.2 – Playing frequencies distribution

Protocol

The study follows a repeated-measures design, using the same participants and collecting their assessment of perceived QOE while playing two versions of My Duck hunt game: one including our LoD inspired proposition and the other without our proposition. In the later version, all the entities are synchronized at the same rate. Using a limited number of participants, repeated measure designs are very suitable for pilot experiments, allowing statistical inference to be made with fewer participants.

The experiment proceeds as follows:

— Participants get a quick introduction of game's rules through a demo version of the game. We also tell them what criteria to look for in qualifying the QoE for game sessions: responsiveness, video and motion smoothness and any other factor influencing their enjoyment of the game.

— Participants play one version then the other (the order in which versions are played is permuted from one player to the next). Participants are not acquainted with the difference between the two versions. During the game, participants have to report when they perceive any decrease in QoE or any responsiveness shortage. They do so by holding the space key on the keyboard.

— At the end of each stage, participants evaluate the QoE for the stage by attributing a rate between 1 and 5. Following the MOS rating scheme, 1 indicates a bad QoE whine 5 indicates an excellent QoE.

— At the end of the experiment, participants are individually interviewed and asked to rate the global QoE for each game version using the MOS rating scheme.

Figure 6.13 shows some photos of participants playing the game during the experiment.

Figure 6.13 – The participants playing the game during the experiment

Network conditions

To be able to control the network environment, the pilot experiment is performed on a LAN. In this LAN we have the game server machine and the client machine. Since we did not deploy the server on a real cloud with WAN (Wide Area Network) connections, we need to simulate the instability and poor capacity of network conditions of some WANs. For this reason, we implemented a proxy. For this experiment, we used packet loss as metric of network conditions. The proxy is therefore used for network congestion simulation through packet loss simulation. The proxy forwards all packets from the client to the server and vice-versa. We utilized UDP connections for the client-server link to send state updates. The proxy simulates a congested network by ignoring (dropping) all packets received when a threshold of packets sent per second is reached (when the link reaches its peak capacity). This threshold represents the bandwidth capacity of the communications link. It is the number of packets transmitted per second (in Hz). Thus to artificially change network conditions for the game, we just need to change this threshold value and the dropped packets percentage will change accordingly.

The proxy is started at the same time as the game and it is launched with a configuration file determining the network capacity variations during the game. The first 210 seconds of the settings used for this experiment are given in table 6.3. From 0 to 30 seconds, the proxy forwards 6000 packets per second; from 30s to 60s, it forwards 3000 packet per second, so on and so forth. This network conditions variation is used for both game versions.

Time (s)	0	30	60	90	120	150	180	210
Capacity (Hz)	6000	3000	5000	2900	7000	2500	3500	3100

Table 6.3 – Proxy configuration for network capacity

Objects importance

The organization handling object importance is the one below. A mapping is made between objects' functional roles and their importance values. The attributed importance values are in the range [0,1].

$$\mathsf{Org}\begin{cases} \mathsf{G}_1 & \text{where role} = \{\mathsf{player}\}; \quad \text{imp} = \{1\}; \quad \text{objects} = \{\text{reticle}\} \\ \mathsf{G}_2 & \text{where role} = \{\mathsf{target}\}; \quad \text{imp} = \{\frac{5}{7}\}; \quad \text{objects} = \{\text{ducks}\} \\ \mathsf{G}_3 & \text{where role} = \{\mathsf{protect}\}; \quad \text{imp} = \{\frac{5}{14}\}; \quad \text{objects} = \{\text{flamingos}\} \\ \mathsf{G}_4 & \text{where role} = \{\mathsf{enemy}\}; \quad \text{imp} = \{\frac{5}{28}\}; \quad \text{objects} = \{\text{gombas}\} \\ \mathsf{G}_5 & \text{where role} = \{\mathsf{decoration}\}; \quad \text{imp} = \{0\}; \quad \text{objects} = \{\text{clouds}\} \end{cases} \tag{6.2}$$

Environment

The game server is a Dell Precision M6500 with the following configuration: an Intel Core i7 Q 720 CPU and 4 Go of RAM. This is an experimental setup with a mono player game, meaning that processing cost of the game can be handled by a single server machine with these specifications. The only bottleneck we have is the one simulated by the proxy on the network link. For this experimentation, the game organization remains stable during the game sessions. But we assume that, provided with a changing organization the same results are held.

Hypotheses

In order to evaluate the impact of the proposition, using the Student's t-test, we have stated the following null hypotheses:

— **H.A.0**: There is no difference in player's QoE between the two game versions during each stage.

— **H.B.0**: There is no difference concerning player's QoE between the two game versions for the overall game session.

— **H.C.0**: There is no difference in the ratio of time spent holding the space key per session duration between the two game versions.

Experimental results

The statistical analysis was performed using R[6], version 2.15.0. A repeated measure t-test is used to reject the null hypotheses.

The hypothesis **H.A.0** is rejected for the five game stages with $p - value < 0.5$. Meaning that the difference between player's QoE during each stage when using LoD and without LoD is statistically significant. The results of the t-test are summarized in table 6.4.

The hypothesis **H.B.0** is rejected by the t-test. The difference of player's QoE between the game with LoD based adaptation and the game without LoD for the overall game session is statistically significant. With a mean M = 1.777778, t(8)=8.6298, $p - value$ = 2.521e − 05.

6. http://www.r-project.org

Stage number	M	t(8)	p-value
stage 1	1,1111	3.5921	0.007063
stage 2	1.5555	6.4236	0.0002039
stage 3	2	5.3666	0.0006724
stage 4	1.6667	3.5355	0.00767
stage 5	1.4444	3.5058	0.00801

Table 6.4 – Results

The hypothesis **H.C.0** is also rejected by the t-test. We can therefore conclude that the difference in the ratio of time spent holding the space key per game session duration between the two game versions is statistically significant. With a mean M = -0.1197444, t(8) =-2.4535 and $p-value$ = 0.03972.

Discussion

The results of this experiment, enable us to evaluate the effects of our approach on players' QoE under critical network conditions. In fact QoE plays a significant role in the perceived player experience and the ability of a game to retain the interest of players. All three null hypotheses of the study were rejected. This means that players have perceived a significant enhancement on QoE with the version with the adaptation technique compared to the version without adaptation. The instability introduced by changing the degree of congestion of the network had less impact on the game quality for all participants not only for each game stage but also for overall game sessions.

T-test results have also rejected the hypothesis H.C.0, but with a higher $p-value$ than the others. Because we chose a p-value threshold of 0.5, we were able to reject this hypothesis, but we can see that the results are not as "strong" as in the two first hypothesis. A reason for this is that during the experiment, not all participants were following the guidelines we gave them in the experimental protocol. Some of them did not press the space key every time they noticed a loss of responsiveness or a decrease in the playability of the game. They were complaining verbally instead. Thus for some participants the data we gathered from the space key, did not reflect the QoE they actually experienced.

These results validate our approach on improving the overall player's QoE. They show that adapting the communications load generated by the game to the current network conditions while taking in to account the roles of each game entities, significantly increases the perceived quality of the game. Thus the approach keeps the game playable and enjoyable even in case of drastic variation in network congestion.

One limitation is that, the whole adaptation can be very subjective, since the configuration of the platform is done manually by the game designer. This delicate configuration includes setting entities' roles, mapping roles to importance values and setting QoE utility thresholds. A bad setting of this system can result to an unpleasant player's QoE. In our work, we suppose that the game designer knows what he is doing (the role of each entity) and the LoD parameters are well set.

The number of participants for this experiment was very limited. In fact only 9 participants were recruited for the pilot study. To consolidate our results, a large scale experimentation can be conducted in the future.

In next subsection, we present the experimental protocol used for the evaluation of the adaptation approach with gameplay components.

6.2.2 Gameplay components based adaptation

To complement the validation of our adaptation approach as a whole, it is necessary to also evaluate the impact of our gameplay components based adaptation on player's QoE. For that we designed an experimental methodology similar to the one used for the organization based adaptation.

Participants

As in the previous experiment, we recruited participants with different levels of exposure to video games. The table 6.5 shows the profiles of the 10 participants, aged from 25 to 42 years with an average of 30.4 years. Most of them play video games at least once per week and are aware of lag problems in online games.

Player	Age	Gender	Gaming habits
1	26	F	1 per month
2	32	F	1 per month
3	27	F	1 per month
4	33	M	1 per month
5	32	M	1 per week
6	25	M	1 per week
7	31	M	1 per week
8	29	M	1 per week
9	42	M	1 per week
10	27	M	1 per week

Table 6.5 – Participants' characteristics

In the figure 6.14, you can see a participant playing the crazy runner game during the experiment.

Protocol

They were asked to play two different versions of the game ”Crazy Runner”: One of the version used a GCs based objects' importance adaptation, and the other version ran without adaptation. A game session is divided into five sub-races ”against time” of the runner. Participants, were introduced to the game, and got to train on a demo version of the game without QoE degradation. After the introduction, participants proceeded to play the two game versions, one after another and were asked to give their feedback on QoE. This was done in two ways:

— During the game session: At the end of each sub-race, the participants were asked to rate the QoE of the sub-race.

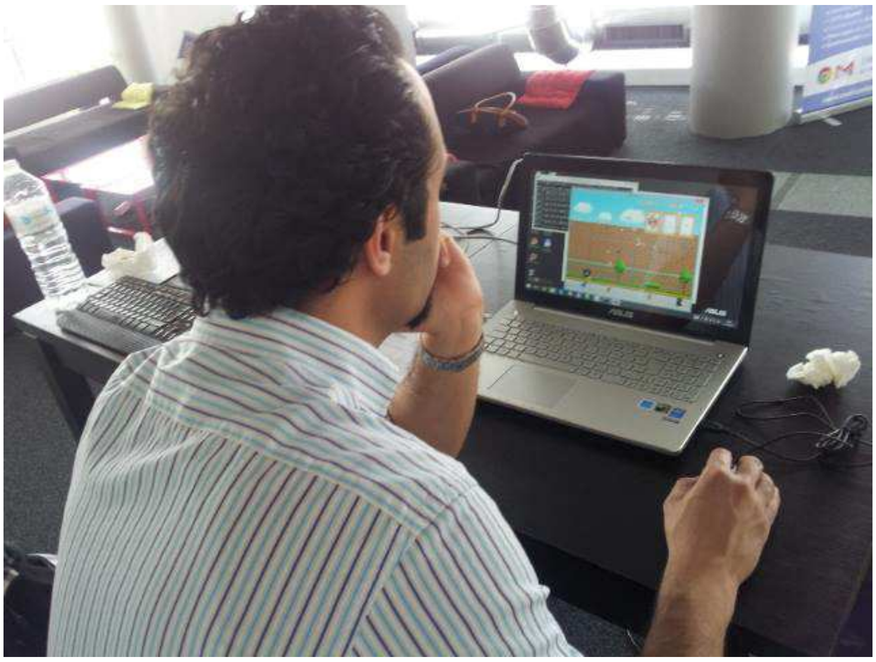


Figure 6.14 – A participant playing crazy runner

— After the game session, through questionnaires: The participants rated the overall QoE of each game version.

Similarly to the previous experimentation, MOS rating scheme is used. The figure 6.15 shows the QoE rating screen at the end of the game session.

Network conditions

Unlike in the previous experiment, we used a free BSD dummynet for network conditions simulation. Dummynet is a live network emulation tool, originally designed for testing networking protocols, and used for a variety of applications including bandwidth management [Carbone and Rizzo, 2010]. Dummynet is very powerful and makes it possible to simulate a wide range of network settings: bandwidth, delay, packet loss, queue size and network protocols. In this experiment, dummynet enabled us to set three network parameters: delay, packet loss and available bandwidth of the network link between the client and the server machines. These settings are dynamically changed during the game session to simulate an unstable network and evaluate how our adaptation technique reacts to various network conditions. For that, we divided the game in 5 steps, each with different network settings:

— For the step 1, there is an abundance of network resources. No network related lag is introduced here. This step introduces the player to the game by letting him (her) enjoy the game without network problems and get the feeling of the game with what we consider a ”good” QoE.

— For step 2, network delay is introduced. Using the dummynet command *ipfw pipe 3 config delay %delay%ms* on the server machine we were able to set the delay for outgoing packets (state updates). Where %delay% represents the requested network delay. The figure 6.16 shows the 3 changes made to network delay for this step.

The length in time of each step depends on the player’s performance. Since the race distance for each step is the same, the player’s ability to eliminate obstacles

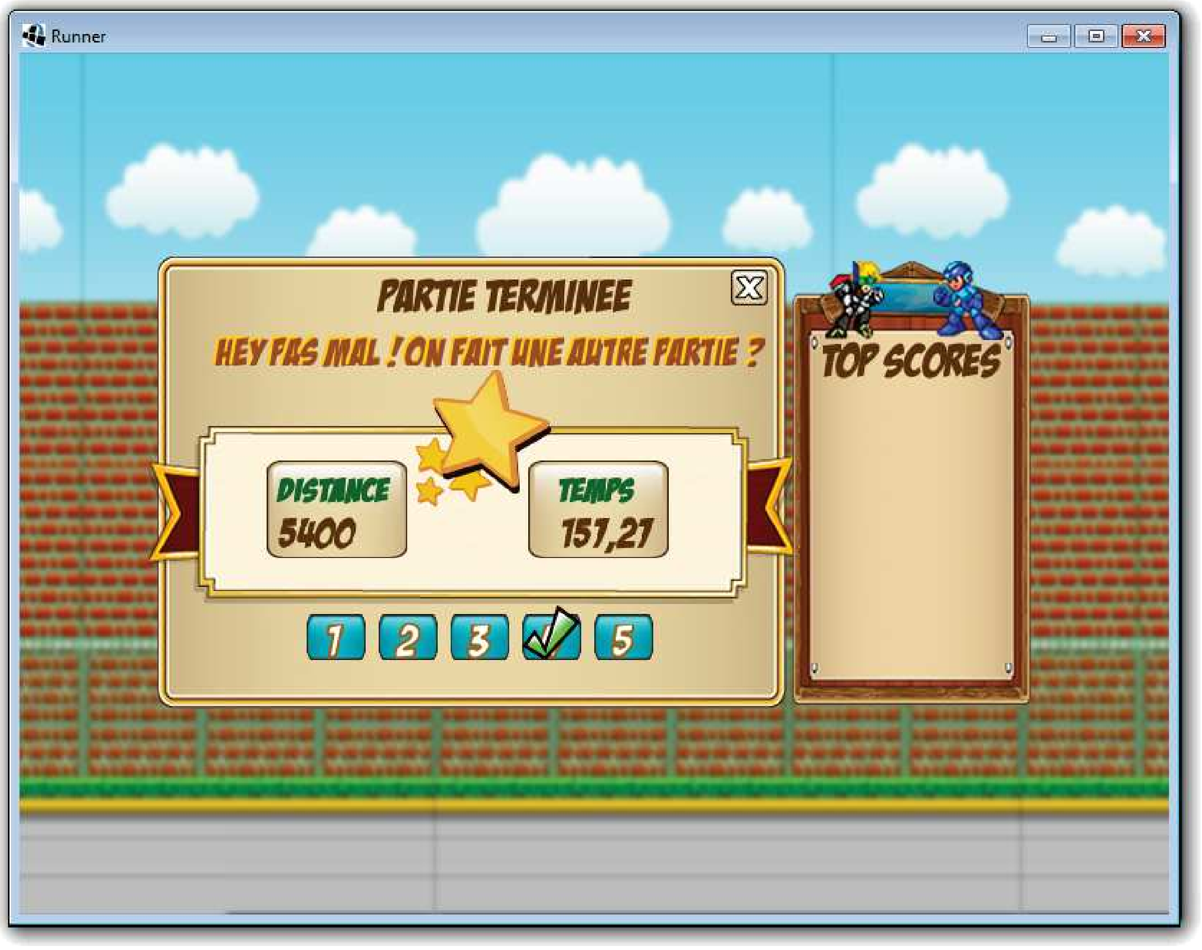


Figure 6.15 – "Crazy runner"'s QoE screen

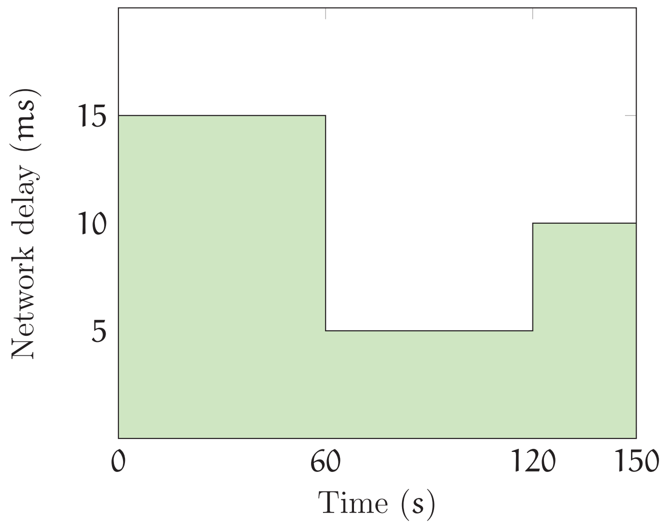


Figure 6.16 – Network configuration for step 2

determine the time it will take him to complete the step. But on average, each step last around 150 s.

— For step 3, packet loss is introduced. Using the dummynet command *ipfw add prob %loss% deny proto tcp out* on the server machine we were able to set the packet loss percentage for state updates. Where %loss% represents the percentage of packet loss requested. The figure 6.17 shows the 3 changes made to packet loss for this step.

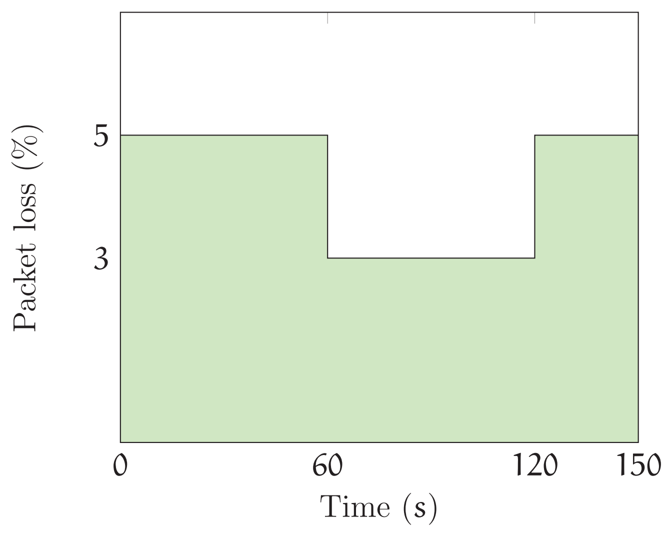


Figure 6.17 – Network configuration for step 3

— For step 4, the available bandwidth is changed. Using the dummynet command *ipfw pipe 3 config bw %bw%Kbit/s* on the server machine we were able to set the available bandwidth for state updates. Where %bw% corresponds to the requested available bandwidth on the client-server connection. The figure 6.18 shows the 3 changes made to the available bandwidth for this step.

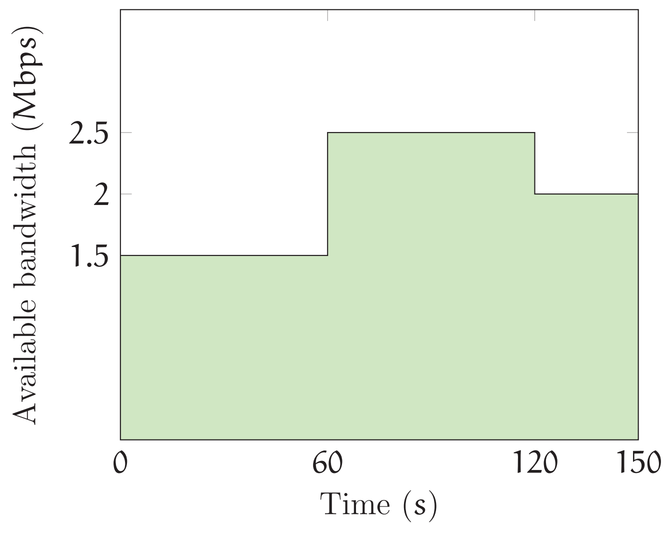


Figure 6.18 – Network configuration for step 4

— For the 5th step, we combined network delay and packet loss. The network delay and packet loss progression for this step is the same as the one of the step 2 and 3 respectively.

Of course at the end of each step, the network conditions are reset to normal, before loading the network configuration of the next step.

Objects importance

As regards to the change in object importance, the distribution of importance values to the gameplay components tree nodes is given by figure 6.19. For this prototype, we have 5 game steps (n=5) with 3 card assignments each (m=3). For each assignment, we have 1 grenade and 11 birds and turtles to eliminate (k=11). An assignment can end for 3 reasons: a card is selected, the runner collides with the grenade, all obstacles are eliminated.

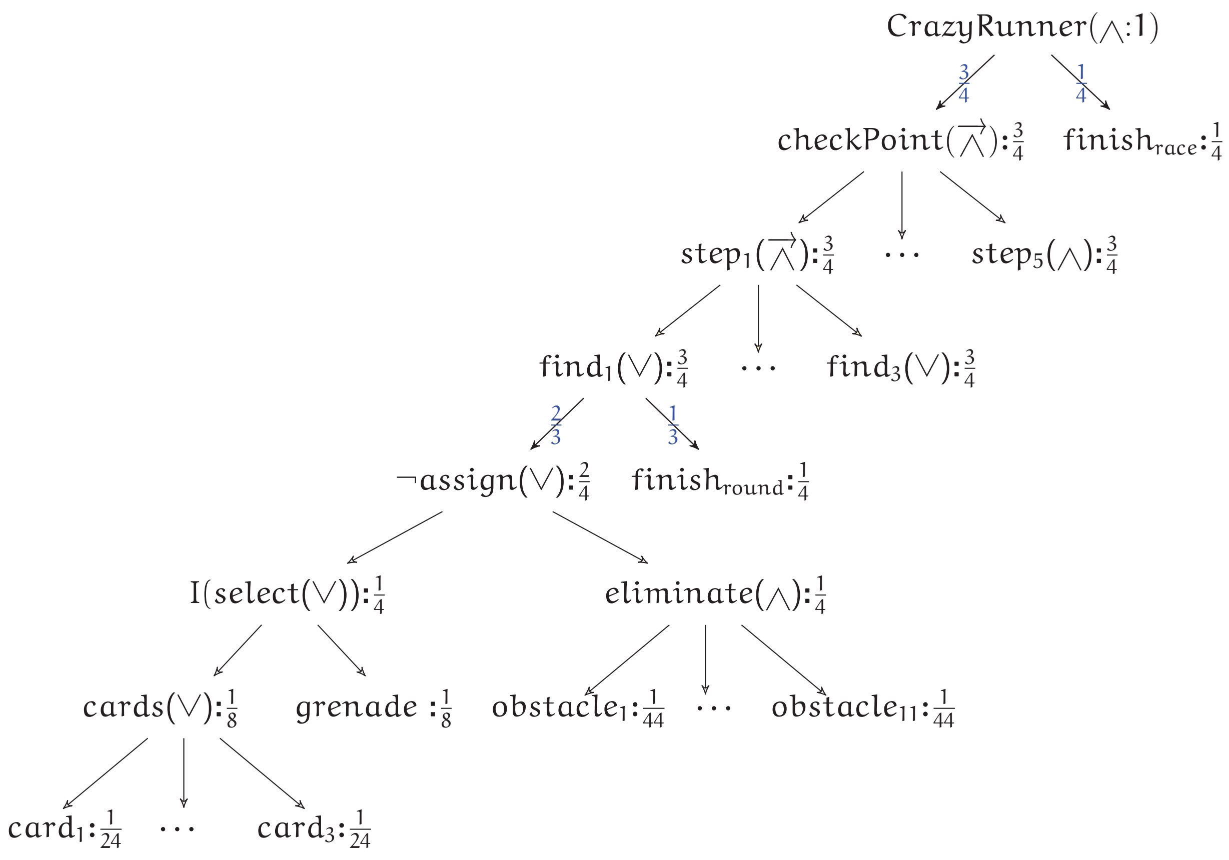


Figure 6.19 – GC tree of the game ”Crazy Runner” at initialization

Background elements such as trees, benches and boxes are in the default set, with an importance value of $\frac{1}{2} \times \frac{1}{44}$.

Given this initial GC tree, we have the initial importance sets below:

$$\mathsf{Sets}\begin{cases} S_1 & \text{where imp } = \frac{1}{4}; \quad \text{objects } =\{\mathsf{grenade}, \mathsf{reticle}\} \\ S_2 & \text{where imp } = \frac{1}{12}; \quad \text{objects } =\{\mathsf{card}_1, \dots, \mathsf{card}_3\} \\ S_3 & \text{where imp } = \frac{1}{44}; \quad \text{objects } =\{\mathsf{obstacle}_1, \dots, \mathsf{obstacle}_{11}\} \\ S_4 & \text{where imp } = \frac{1}{88}; \quad \text{objects } =\{\mathsf{trees}, \mathsf{benches}, \mathsf{boxes}\} \end{cases}$$

Environment

The server machine is a Dell Precision M6500 running Windows 7 with the following configuration: an Intel Core i7 Q 720 CPU and 4 Gb of RAM. The client machine is an ASUS laptop running Windows 8, equipped with an Intel Core i7 - 4710HQ CPU and 8 Gb of RAM. An ad-hoc wired connection was made between the machines, and only transmitted data is the game traffic and the monitoring traffic. For this experimental setup the only bottleneck we have is the one simulated by the dummy net on the server's network adapter. We used the *ping* command to measure the network delay and the packet loss, and the *wget* command to measure the available bandwidth.

Hypotheses

Similarly to the previous experiment, we will use the null hypotheses below to investigate the effects of GCs based adaptation:

— **H.A.0** There is no difference in player's QoE between the two game versions during each sub-race.

— **H.B.0** There is no difference in the number of obstacles eliminated by the player (successful shot attempts) per sub-race between the two game versions.

— **H.C.0** There is no difference concerning player's QoE between the two game versions for the overall game session.

To carry this analysis, the following measures we recorded during game sessions:

— Kinematics measures: Here we monitored the number of obstacles eliminated by the player per sub-race and the time taken to complete each sub race.

— QoE measures: For each sub-race and for the overall game session, players are asked to rate the QoE they perceived.

Experimental results

For the paired student t-test, we have selected a $p - value$ threshold of 5%. Meaning that, all null hypotheses with a $p - value < 0.05$, will be rejected. Following are the results of the experiment conducted on 10 participants using the MOS rating scheme.

The hypothesis **H.A.0** was rejected for four of the five stages. The sub-races 2, 3 ,4 and 5 had $p - values < 0.05$. This indicates that, there is a significant difference in the mean of the rates reported by the players concerning they QoE they experience during the four sub-races between the two game versions. The table 6.6 represents the summary of the t-test results regarding this null hypothesis by giving the mean difference between the two versions, the degree of freedom for the t-statistic, the t-statistic and the p-value for the test. As you can see, the null hypothesis has not been rejected for the first stage. In fact with a $p - value$ of 0.7263 and a mean difference of 0.1, we can conclude that the average QoE of two versions in the first sub-race are significantly similar. Which is normal, because in this sub-race, the network settings were favourable: no latency, no packet loss and sufficient bandwidth. Therefore both game versions perform the same.

Sub-race number	**Mdiff**	**t(9)**	**p-value**
sub-race 1	0.1	0.3612	0.7263
sub-race 2	2.1	6.0343	0.0001942
sub-race 3	1.2	2.8823	0.01811
sub-race 4	1,9	5.4596	0.0004006
sub-race 5	1.5	3.5032	0.006689

Table 6.6 – Results for QoE rates per sub-race

The null hypothesis **H.B.0** was rejected as well for four sub-races. This means there is a statistically significant difference between the two versions in the number of successful shot attempts for sub-races 2, 3, 4 and 5. With a $p-values < 0.05$ and a mean difference > 3.0, the adaptation provides a better player-game interaction, enabling players to better control their actions and therefore to eliminate more enemies. The complete results are given in tables 6.7.

Sub-race number	**Mdiff**	**t(9)**	**p-value**
sub-race 1	1.4	1.2206	0.2533
sub-race 2	6.5	4.8079	0.0009631
sub-race 3	3.1	2.72	0.02361
sub-race 4	18.8	5.1973	0.0005661
sub-race 5	24.8	13.4585	2.882e-07

Table 6.7 – Results for number of eliminated obstacles per sub-race

H.C.0 was rejected and the results show that the gameplay component based adaptation has significantly increased the QoE for the overall game session with mean difference = 1.7 , t(9) = 3.7908, $p-value = 0.004277$. Table 6.8 complements these results.

	Mean	**Standard deviation**
adaptation	3	1.3333
no adaptation	1.3	0.6749

Table 6.8 – Results for the overall game session

Discussion

The results of the repeated-measures analyses indicate significant mean differences in participants' QoE for the two game versions under insufficient network resources. Participant's QoE is significantly higher with the gameplay based adaptation compared with the game version without adaptation. This means that our approach effectively adapts to network resources shortage and changes, in order to maintain an adequate player's QoE. With adaptation, players were able to play with no major impairment in the QoE, except for the sub-race 3. As you can see it on the tables 6.7 and 6.6, the sub-race 3 is the one with the highest $p-value$. In this sub-race, packet loss was the added network deterioration, and even though the null hypothesis is rejected, the gain introduced by the adaptation is less effective than with delay and low bandwidth.

This experiment enabled us to notice that, network delay and available bandwidth have almost the same influence on player's QoE in an online game with cloud gaming paradigm. In fact, we found that, there is a cause and effect relationship between both. The diminution of the available bandwidth on a network link causes network congestion on that link. Network equipments such as routers and switches enqueue incoming packets and wait until there is enough space on the link to send new packets. This queueing process results into packets being delayed. Therefore, during our experiment, when we lowered the available bandwidth, the network delay increased as well. But the inverse of this statement is not necessarily true, meaning that an increase in delay, does not necessarily denote the available bandwidth has decreased.

The effects of delay and available bandwidth were very severe for player's QoE. Given that we are in a cloud gaming paradigm, no dead reckoning and client side prediction is utilized. The game client updates its game scene at the same rhythm the state updates are sent. With TCP's control mechanism, the initial delay increases as the game progresses and more packets are enqueued. So at times, we had more than 10 seconds of delay in the displayed game scene. And of course this results in an unplayable game and players were very frustrated. This shows us that, network parameters such as delay and bandwidth can be tolerated in video streaming scenario without any interaction (eg. youtube, dailymotion). But in interactive applications such as cloud games they are less tolerated, because players can have an outdated state of the game and feel as if their actions are not taken into account.

Unlike the normal game version in this study, our adaptation regulates update frequencies by decreasing the update rate of less important game objects, keeping the bottleneck on the network link small with no expanded delay effects. This results into a smooth gameplay and an adequate level of interaction with the game. Some players enjoyed this game version in a point where they were able to develop some strategies to quickly complete the game with the best race time while eliminating more obstacles.

From the questionnaire we carried at the end of the experiment with each participant, 9 of 10 participants reported that the version without adaptation was more difficult and took more time to complete the game objectives. More importantly they also expressed their frustration toward this version because they did not have control of their actions, and did not feel like they are impacting the game, especially in sub-races 2 and 4. This behaviour can be explained by the incapacity of this version to react to bad network conditions and effectively distribute the available network resources. In fact, in sub-race 2 and 4 , delay and low bandwidth were drastically changed and only the version with LoD based adaptation was able to positively cope with these delay which could even reach 10 seconds in the version without adaptation. Meaning that without adaptation, in the sub-races 2 and 4, the player was seeing a replay of the game state as it was 10 seconds ago. Hence there were no interactivity and no direct response to players' actions. This is probably the reason why most cloud gaming systems, have a limit in the tolerable QoE deterioration. Any QoE below that limit, is considered inconsistent and the service is not delivered.

Participants also gave some suggestions for the game itself:

- They suggest to give more importance to player's character and its reticle, because lag affecting other game entities is not as frustrating as the lag affecting player's reticle. We then explained to them the importance hierarchy used in the adaptive game version.
- Some suggest to change the reticle by making it bigger or bolder so that it will be easily visible. Noting that with the version without adaptation, the players had difficulty to determine the position of their reticle.
- Some suggested to have a full screen game, so that they could easily determine the position of their reticle.

— Some went far and proposed to bring more fun to the game by adding different backgrounds and ambiances, different behaviours for game entities as well as different game levels. They also proposed different feedbacks to different actions. And the reported that the game at this stage is a prototype good for demonstration and experimentation, but it needs some improvements to reach a production level. And we completely agreed.

6.2.3 Overall discussion

Cloud gaming has become very popular over the last few years. Most of it due to the ubiquity, the computing power and the high availability of cloud services. The perceived QoE of a cloud game depends not only on psychological concepts such as emotion, immersion and feeling, but also rather objective parameters such as network conditions, and video frame rate. In constrained network circumstances, two main aspects can affect player's QoE: the responsiveness of the game and the smoothness of the video. Our contribution in these studies is therefore to enable people with low network capabilities to enjoy cloud games while maintaining an acceptable QoE. The proposed technique therefore consists of a LoD inspired communications adaptation with the objective of maintaining player's QoE in acceptable range. Our approach adapts objects' synchronization rates, to the network conditions while taking into account their importance in the game. Two versions of adaptation were studied here: an organization based adaptation and a gameplay component based adaptation. Experimental studies focused on assessing the QoE metrics such as responsiveness and smoothness of motion and video.

The first pilot experiment focused on the evaluation of the organization based adaptation. The results demonstrated that, under scarce network conditions the adaptation strategy positively influence the player's QoE compare to a version without adaptation. In fact the MOS ratings the players provided showed that they experienced less lag in their actions and in the game in general with the game running our adaptation technique.

The second experiment, dealt with the gameplay component based adaptation. It revealed that our approach performs better than a version without adaptation under restricted network conditions. Even with low network conditions, players were able to enjoy the game with an adequate QoE. Through a comparison with multiple network metrics, we found that some network parameters cause more damages than others. For the game version without adaptation, low bandwidth and network delay was the most severe compare to packet loss. But paradoxically with the enhancements of our gamplay component based adaptation, packet loss was the most difficult parameter to handle.

It is also important to state that, while this approach reduces the bandwidth needed for an acceptable game quality, it does not eliminate the requirement of a minimum bandwidth for an enjoyable game. It certainly requires less bandwidth than classic cloud gaming services. For instance, we were able to have a good quality game with just 1.5 Mbps as available bandwidth with is not possible with Onlive cloud gaming system for example.

6.3 Conclusion

We focused in this chapter on the design and prototyping of our proposed communications adaptation technique as well as on its experimental evaluation. We used a simple game model implemented in different ways on two game engines. These game engines enabled us to develop two games to evaluate the contributions of our proposition, namely a communications adaptation approach using organizations and a communications adaptation approach using gameplay components.

The experimental validation, involved participants with different gaming histories. The experiments showed that the suggested adaptation minimizes the effects of low and/or unstable network conditions in maintaining game responsiveness and player's QoE. The next chapter will present the contributions' summary, the results and some future works.

CHAPTER 7

Conclusion

Contents

In this thesis, we have presented a level of detail based communications adaptation technique for cloud games. This technique enables game developers to build games that adapt to network resources for the maintenance of an adequate player's QoE. On behalf of this work, we have studied many approaches for online gaming, ranging from cloud gaming with video streaming to cloud gaming with objects replication. We have also identified the potential challenges faced by video streaming cloud game systems namely: the delivery of an adequate QoE in presence of unstable and low network resources. This study convinced us that these problems could be overcome by adapting the game communications to network conditions. Most existing approaches use fixed priority systems to guide the communications adaptation. The objective is to reduce the amount of resources needed for games, without introducing any lagging effect of insufficient network conditions on important (high priority) game entities. However, we believe that a fine grained adaptation with dynamic priorities is possible by exploiting the game structure and scenario. To the best of our knowledge this issue has not yet been addressed by existing approaches.

We answer to this issue by designing a communications adaptation scheme using objects' importance in the game scene and network conditions as adaptation inputs. We believe that, communications adaptation is a key factor in maintaining an adequate player's

QoE in unfavourable network circumstances. For some games, in order to achieve game objectives, objects' importance can change by time. Our adaptation provides models for dynamic importance adjustment as the game progresses. Unlike existing work using static priority for game objects, this dynamic importance opens the possibility to take into account the dynamic nature of game mechanics in adapting game communications.

The study made in chapter 2 enabled us to identify the core concepts of video games and cloud gaming, as well as the QoE delivery principles in cloud gaming paradigm. We then used these concepts as foundations for our proposition on player's QoE maintenance. Core optimization techniques including bandwidth conservation and latency compensation provides game developers with tools to specify an optimized architecture during the design phase. Our communications adaptation technique will then stand on top of this architecture.

7.1 Contributions

Our contributions in the domain of video games and network communications can be summarized as follow:

Game models

Our first contribution consist of game models expressing game objects and their communications needs represented by their importance in the game. We provided two different ways to manage objects' importance. These dynamic object importance adaptation models enable us to extend the reach of our proposition by encompassing video games with changing game mechanics during game sessions:

— **Organization based adaptation:** Assuming that each game object has a precise functional role in the game, we designed this version of the adaptation around the notion objects' functional role in the game scene. Each role is mapped with an importance value, therefore a change in object's role steers a change in object's importance. To manage objects' roles in a game scene, we used agents organization. This combination of level of detail and agents organization is known as organizational level of detail. We followed the main principles of AGR organizational model by, assembling game objects with the same functional role in groups. Objects with major roles in the game organization are generally more sensitive to lag impacting player's QoE. Thus our model enables these objects to have more importance than other objects. With our organization, all game objects in a group will receive the same amount of network resources to synchronize their states. Upon changed role, object's importance also changes and game objects can moves from one group to another at any moment.

— **Gameplay components based adaptation:** This model decomposes complex game scenarios into components with simpler objectives called gameplay components. The simplicity and semantics of gameplay components and its operators make it possible to hierarchically design a game in terms of OCR loops. The result is a gameplay components tree also known as OCR tree. Each node of the tree is assigned an importance value that changes as the tree evolves. Each time a gameplay component is added or removed from the tree, and even when it simply ends, the importance values of the rest of the tree are recalculated. Therefore the importance values adapt to the changes in the gameplay components tree. Game objects are then regrouped by importance values to create importance sets, that are used for the selection of the communications level in the level of detail system.

Adapting communications for cloud games

We provided a level of detail approach for managing network resource distribution based on objects importance in the game scene and network conditions. We exploited the dynamic objects importance adjustment models presented above to propose LoD systems adapting to changes during game sessions.

— **Network based level of detail:** The main contribution of this thesis consists of the network oriented level of detail, enabling communications adaptation. The approach proposes an innovative reuse of the level of detail principles in the context of network resources distribution by associating different synchronization rates to game objects through different communications levels. The communications level of an object thus determines the amount of network resources the object gets. Adaptation inputs, used for level selection are network conditions and game objects' importance. These parameters are continuously monitored to compute a composite metric called QoE utility. A list of QoE utility thresholds is provided to determine to which communications level map each game object given the current network situation. As result, when network conditions are favourable, all game objects are updated using high synchronization rates. But when network capacity degrades, only important objects keep a high synchronization rate while less important objects see their network resources gracefully degraded.

— **Dynamic objects importance:** In our adaptation model, object importance can change from one time to another. Unlike existing adaptation schemes with fixed and manually assigned objects importance, we propose models for dynamic adjustment of objects importance during game sessions. Based on concepts of agent organizations and gameplay components, we are able to adapt objects' importance to the game progression. We used the game structure as well as the game objects' roles in the game scene. As the game progresses, its internal structure changes, game objects can appear and disappear or simply change their functional roles in the game scene. These changes are reflected in objects' importance, and steer a reorganization of the level of detail model. From this reorganization, results a redistribution of network resources according to the current state of the game and current network conditions.

7.2 Results

An experimental evaluation of the adaptation approach with organization has been conducted. This evaluation was based on pilot studies including participants with different video game backgrounds. Tests were made with two versions of the game "My Duck Hunt" developed for evaluation purposes. A version with adaptation, and a version without adaptation. The experimental protocol consisted of making participants play both versions of the game in a poor and unstable network. Participants' feedbacks on QoE they experienced during game sessions were collected and statistically studied. Data was collected based on questionnaires and quantitative measures of players' QoE. Game sessions were partitioned in stages at the end of which the player is asked to evaluate the QoE for the completed stage. Three null hypotheses were formulated on the difference between QoE of the two versions of the game for : (i) game stages; (ii) overall game sessions; (iii) quantitative data collected during game sessions (see section 6.2.1).

Using a repeated measure t-test, we found that the three hypotheses we stated were rejected by the test ($p-value < 0.5$). This suggests that, there is a significant difference in player's QoE, between the game versions. This shows that, the version with the LoD communications adaptation technique copes more effectively with low and/or unstable network conditions.

The second experiment evaluated the effects of our gameplay component based adaptation on player's QoE. The tests were carried on two versions of a game we developed called "Crazy runner". One version were using the gameplay component based adaptation and the other one had no adaptation(all game objects were synchronized at the same optimal rate). The experimental protocol was the same as in the previous experiment. The null hypotheses concerned the difference between player's QoE of the versions for each game stage (sub-race) and for the overall game session. We added a null hypothesis on the number of successful shot attempts by players. The network parameters were different per game stage. The first stage was normal, with favourable network conditions; in the second, we added network delay; in the third we added packet loss, in the fourth we diminished the available bandwidth and in the last, we combined network delay with packet loss.

The results we had using a student t-test, demonstrated that, all the null hypotheses were rejected except the one on the first stage. Which is the expected behaviour. We a choosen $p - value$ threshold of 5%, all the results showed significant difference in the player's QoE between the two game versions. Meaning that the adaptation brings significant enhancements on the player's QoE. Delay and available bandwidth caused more lagging effects than packet loss for the version without adaptation. But with our adaptation, we were able to cope with that, and bring the available bandwidth down to 1.5 Mbps while keeping an enjoyable game experience.

7.3 Future works

Our proposition only takes into account object's importance and network conditions as inputs for adaptation. But we believe that, there is no conceptual lock to the integration of new adaptation metrics such as object's distance to camera, object's size, etc. In fact, the distance to the camera is a very popular approach generally used in graphic level of detail. Applied for network resources distribution, the distance to the camera will enable closer objects to have more communications resources than farther objects. This can be very beneficial in certain game scenarios.

Multiplayer support for video games is very important nowadays for game engines. In serious games for example, the acquisition of some serious contents may require to play with others in order to complete a collaborative or competitive quest. Therefore extending our adaptation scheme for multiplayer game sessions could be very rewarding. To do that, it is necessary to extend our adaptation model, to enable multiple LoD managers (one for each client) to cohabit on the server system.

In fact, since different players will have different game objects in the game scene (depending on their positions and their camera angles), objects can have different importance for different players. As far as network conditions is concerned, each client has its own network link with different conditions than the other clients' networks. The adaptation will then deliver to each player a QoE corresponding to his network conditions. Thus two players with different network capabilities can join the same game session, and have different QoE, while the game content remains the same.

Another future objective is to perform a large scale experimentation in a multiplayer cloud gaming environment where each player has his own network condition, his own camera and sees different angles of the game scene. This experiment will enable us to increase the number of participants and have a more quantitative study of the effects of the communications adaptation on a large variety of players with different network settings. Participants should be selected in a random manner while ensuring a variety of gaming backgrounds, genders and ages.

The target of our adaptation technique is the game developer, meaning that, even though our objective is to deliver an adequate QoE to players, game developers are those who will eventually use our technique to build adaptive video games. In the experiments carried out, we only evaluated the benefits of our proposition for the player. But it is also important to assess the usability and acceptability of our framework with game developers. The objective here will be to carry an experiment, letting developers use this framework and get their feedbacks on the ease of utilization, the reusability and the general understanding of the framework.

Appendix A

List of publications

— Richard Ewelle Ewelle, Yannick Francillette, Mahdi Ghulam, Abdelkader Gouaïch et Nadia Hocine. Level of detail based network adapted synchronization for cloud gaming. In proceedings of the 18th International Conference on Computer Games : AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES), IEEE, 2013.

— Richard Ewelle Ewelle, Yannick Francillette, Mahdi Ghulam et Abdelkader Gouaïch. Network Traffic Adaptation For Cloud Games. In International Journal on Cloud Computing : Services and Architecture (IJCCSA), vol. 3, October, 2013.

— Richard Ewelle Ewelle, Yannick Francillette, Abdelkader Gouaïch, Mahdi Ghulam, Nadia Hocine et Julien Pons. Network Aware Traffic Adaptation for Cloud Games. In International Conference on Cloud Computing and Big Data (CloudCom-Asia), IEEE, 2013.

Bibliography

Cited page 72.

Ernest Adams. *Fundamentals of game design*. Pearson Education, 2013. Cited page 62.

Jacob Agar. Ethereal, 2012. URL http://jacobagar.com/Ethereal.htm. [Online; accessed 16-August-2014]. Cited page 36.

M. Albinet. *Concevoir un jeu vidéo: Les méthodes et les outils des professionnels expliqués à tous !* Entreprendre: Développement professionnel. FYP éditions, 2011. ISBN 9782916571638. URL http://books.google.fr/books?id=iwOFZwEACAAJ. Cited page 62.

Sergio Alvarez-Napagao, Fernando Koch, Ignasi Gómez-Sebastià, and Javier Vázquez-Salceda. Agents for games and simulations ii. chapter Making Games ALIVE: An Organisational Approach, pages 179–191. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-18180-1. URL http://dl.acm.org/citation.cfm?id=1985721.1985737. Cited pages 59 and 60.

Baik Song An, Manhee Lee, Ki Hwan Yum, and Eun Jung Kim. Efficient data packet compression for cache coherent multiprocessor systems. In *Proceedings of the 2012 Data Compression Conference*, DCC '12, pages 129–138, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4656-8. doi: 10.1109/DCC.2012.21. URL http://dx.doi.org/10.1109/DCC.2012.21. Cited page 27.

K Andresen and N Gronau. Seeking optimal it strategies by the determination of adaptability in domain-specific software applications. in managing modern organizations with information technology. In *Proceedings of the 2005 Information RecourcesManagement Association International Conference*, pages 15–22. Idea Group Publishing, 2005. Cited page 29.

Sander CJ Bakkes, Pieter HM Spronck, and H Jaap van den Herik. Opponent modelling for case-based adaptive game ai. *Entertainment Computing*, 1(1):27–37, 2009. Cited page 59.

Russell Beauregard and Philip Corriveau. User experience quality: a conceptual framework for goal setting and measurement. In *Digital Human Modeling*, pages 325–332. Springer, 2007. Cited page 8.

A.G. Bedeian and R.F. Zammuto. *Organizations: Theory and Design*. Dryden Press, 1991. ISBN 9780030125836. URL http://books.google.fr/books?id=u7iLMQEACAAJ. Cited page 59.

Harlan Beverly. Lag, the barrier to innovation in online gaming. In *Proceedings of the Game developers conference 2009*, 2009. URL http://www.slideshare.net/hbombers/harlan-beverly-lag-the-barrier-to-innovation-gdc-austin-2009. [Online; accessed 20th-Febuary-2014]. Cited page 25.

Jean-Pierre Briot, Alessandro Sordoni, Eurico Vasconcelos, Marta de Azevedo Irving, Gustavo Melo, Vinícius Sebba-Patto, and Isabelle Alvarez. Design of a decision maker agent for a distributed role playing game–experience of the simparc project. In *Agents for Games and Simulations*, pages 119–134. Springer, 2009. Cited page 59.

Roger Caillois. *Man, play, and games*. The Free Press, Glencoe, New York, USA, 1961. Cited page 14.

Marta Carbone and Luigi Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010. Cited page 113.

Cristiano Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In Michael Wooldridge and Nicholas Jennings, editors, *Intelligent Agents*, volume 890 of *Lecture Notes in Computer Science*, pages 56–70. Springer Berlin / Heidelberg, 1995. ISBN 978-3-540-58855-9. URL http://dx.doi.org/10.1007/3-540-58855-8_3. 10.1007/3-540-58855-8_3. Cited page 58.

Yu-Chun Chang, Kuan-Ta Chen, Chen-Chi Wu, Chien-Ju Ho, and Chin-Laung Lei. Online game qoe evaluation using paired comparisons. In *Communications Quality and Reliability (CQR), 2010 IEEE International Workshop Technical Committee on*, pages 1–6. IEEE, 2010. Cited pages 22 and 107.

Stéphane Chauvier. *Qu'est-ce qu'un jeu?* Vrin, 2007. Cited page 15.

Hung-Kuang Chen, Chin-Shyurng Fahn, Jeffrey JP Tsai, Rong-Ming Chen, and Ming-Bo Lin. Generating high-quality discrete lod meshes for 3d computer games in linear time. *Multimedia Systems*, 11(5):480–494, 2006. Cited pages 56 and 58.

Kuan-Ta Chen, Cheng-Chun Tu, and Wei-Cheng Xiao. Oneclick: A framework for measuring network quality of experience. In *INFOCOM 2009, IEEE*, pages 702–710. IEEE, 2009. Cited pages 22 and 107.

Peng Chen and Magda El Zarki. Perceptual view inconsistency: an objective evaluation framework for online game quality of experience (qoe). In *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*, page 2. IEEE Press, 2011. Cited page 22.

S. Choy, B. Wong, G. Simon, and C. Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, pages 1–6, Nov 2012. doi: 10.1109/NetGames.2012.6404024. Cited page 24.

James Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976. Cited pages 52 and 56.

Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, November 2006. ISSN 0001-0782. doi: 10.1145/1167838.1167860. URL http://doi.acm.org/10.1145/1167838.1167860. Cited page 25.

Mark Claypool, David Finkel, Alexander Grant, and Michael Solano. On the performance of onlive thin client games. *Multimedia Systems*, pages 1–14, 2014. Cited page 43.

cloudtweaks. Cloud gaming: Benefits of kalydo cloud and file streaming game technology, 2013. URL http://cloudtweaks.com/2013/08/cloud-gaming-kalydo-cloud-and-file-streaming-game-technology/. [Online; accessed 20th-Febuary-2014]. Cited page 19.

E Comer Douglas. Internetworking with tcp/ip. *Vol. III-Prentice Hall*, 2000. Cited page 55.

Noel Crespi, B Molina, CE Palau, et al. Qoe aware service delivery in distributed environment. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 837–842. IEEE, 2011. Cited page 22.

Eric Cronin, Burton Filstrup, Anthony R. Kurc, and Sugih Jamin. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of the 1st Workshop on Network and System Support for Games*, NetGames '02, pages 67–73, New York, NY, USA, 2002. ACM. ISBN 1-58113-493-2. doi: 10.1145/566500.566510. URL http://doi.acm.org/10.1145/566500.566510. Cited page 19.

Mihaly Csikszentmihalyi. *Flow: The psychology of optimal experience*, volume 41. HarperPerennial New York, 1991. Cited page 22.

D. RavindraKumarI D. Barièeviæ, , and M. Chandrashekar. Gameon: Analysis and implementation of cloud gaming, 2011. URL http://www.cs.ucsb.edu/~manasa/cs276.pdf. [Online; accessed 31-Octobre-2012]. Cited page 20.

Luca De Cicco and Saverio Mascolo. *An experimental investigation of the Akamai adaptive video streaming*. Springer, 2010. Cited pages 44, 45, and 48.

D. De Winter, P. Simoens, L. Deboosere, F. De Turck, J. Moreau, B. Dhoedt, and P. Demeester. A hybrid thin-client protocol for multimedia streaming and interactive gaming applications. In *Proceedings of the 2006 International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '06, pages 15:1–15:6, New York, NY, USA, 2006. ACM. ISBN 1-59593-285-2. doi: 10.1145/1378191.1378210. URL http://doi.acm.org/10.1145/1378191.1378210. Cited page 19.

Macri Dean and Pallister Kim. Building scalable 3d games for the pc, 1999. URL http://www.gamasutra.com/view/feature/3403/building_scalable_3d_games_for_the_.php. [Online; accessed 20th-Febuary-2014]. Cited page 27.

Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2): 223–259, December 2006. ISSN 1556-4665. doi: 10.1145/1186778.1186782. URL http://doi.acm.org/10.1145/1186778.1186782. Cited pages 29 and 30.

Jürgen Döllner and Henrik Buchholz. Continuous level-of-detail modeling of buildings in 3d city models. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 173–181. ACM, 2005. Cited pages 56 and 58.

Charalampos Doukas, Thomas Pliakas, and Ilias Maglogiannis. Mobile healthcare information management utilizing cloud computing and android os. *Conf Proc IEEE Eng Med Biol Soc*, 1:1037–40, 2010. ISSN 1557-170X. URL http://www.biomedsearch.com/nih/Mobile-healthcare-information-management-utilizing/21097207.html. Cited page 5.

Jeff Dyck, Carl Gutwin, T.C. Nicholas Graham, and David Pinelle. Beyond the lan: Techniques from network games for improving groupware performance. In *Proceedings of the ACM Conference on Organizational Computing and Groupware Technologies*, pages 291–300, 2007. Cited page 40.

P. Eisert and P. Fechteler. Low delay streaming of computer graphics. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 2704–2707, Oct 2008. doi: 10.1109/ICIP.2008.4712352. Cited page 18.

Ali El Masri, Ahmad Sardouk, Lyes Khoukhi, and Dominique Gaiti. A preventive traffic adaptation model for wireless mesh networks using fuzzy logic. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, pages 73–81. IEEE, 2011. Cited page 30.

EpicGames. Unreal networking architecture, 2012. URL http://udn.epicgames.com/Three/NetworkingOverview.html. [Online; accessed 20th-Febuary-2014]. Cited pages 31, 41, 42, and 48.

Ben Etzkorn. Data normalization and standardization, 2011. URL http://www.benetzkorn.com/2011/11/data-normalization-and-standardization/. [Online; accessed 27-August-2014]. Cited page 71.

Richard Ewelle Ewelle, Yannick Francillette, Ghulam Mahdi, Abdelkader Gouaich, and Nadia Hocine. Level of detail based network adapted synchronization for cloud gaming. In *Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES), 2013 18th International Conference on*, pages 111–118. IEEE, 2013. Cited page 77.

Jacques Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999. Cited pages 58 and 61.

Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In *AOSE*, pages 214–230, 2003. Cited pages 55, 59, 60, 61, and 141.

S. Ferretti, M. Roccetti, and C. E. Palazzi. An optimistic obsolescence-based approach to event synchronization for massively multiplayer online games. *Int. J. Comput. Appl.*, 29(1):33–43, January 2007. ISSN 1206-212X. URL http://dl.acm.org/citation.cfm?id=1735675.1735680. Cited page 19.

Stefano Ferretti and Marco Roccetti. Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, pages 405–412, New York, NY, USA, 2005. ACM. ISBN 1-59593-110-4. doi: 10.1145/1178477.1178570. URL http://doi.acm.org/10.1145/1178477.1178570. Cited page 19.

Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-475-8. Cited page 4.

Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008. Cited page 4.

Yannick Francillette, Abdelkader Gouaich, Nadia Hocine, and Julien Pons. A gameplay loops formal language. In *Proceedings of the 2012 17th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES)*, CGAMES '12, pages 94–101, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-1-4673-1120-5. doi: 10.1109/CGames.2012.6314558. URL http://dx.doi.org/10.1109/CGames.2012.6314558. Cited pages 11, 55, and 62.

G-Cluster. G-cluster official web page, 2012. URL http://www.gcluster.com. [Online; accessed 31-Octobre-2012]. Cited page 20.

Gamasutra. Raknet chosen by bigpoint as networking solution for upcoming multiplayer rpg drakensang online, 2011. URL http://www.gamasutra.com/view/pressreleases/148567/RakNet_Chosen_by_Bigpoint_as_Networking_Solution_for_UpcomingMultiplayer_RPG_Drakensang_Online.php. [Online; accessed 20th-Febuary-2014]. Cited page 38.

Gamespot. Tribes 2, 2001. URL http://www.gamespot.com/tribes-2/. [Online; accessed 27-August-2014]. Cited pages 40, 41, and 141.

GarageGames. Torque3d- multi-player made easy, 2007. URL http://www.garagegames.com/products/torque-3d/overview/networking. [Online; accessed 20th-Febuary-2014]. Cited page 40.

GarageGames. Tnl - official website, 2009. URL http://www.opentnl.org/. [Online; accessed 20th-Febuary-2014]. Cited pages 29, 30, 31, 40, 48, and 56.

Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Šimlovič, Radim Vansa, Michal Štolba, et al. Pogamut 3 can assist developers in building ai (not only) for their videogame agents. In *Agents for Games and Simulations*, pages 1–15. Springer, 2009. Cited page 59.

Peter Groen. Max healthcare is 1st hospital in india to receive 'stage 6' recognition from himss, 2012. URL http://www.openhealthnews.com/hotnews/max-healthcare-1st-hospital-india-receive-stage-6-recognition-himss. [Online; accessed 22th-January-2014]. Cited page 5.

Meirav Hadad and Avi Rosenfeld. Adapt: abstraction hierarchies to succinctly model teamwork. In *AAMAS*, pages 1177–1178. IFAAMAS, 2011. Cited pages 59 and 60.

N.A. Hawes. *Anytime deliberation for computer game agents*. PhD thesis, University of Birmingham, 2004. Cited page 14.

Richard Held and Nathaniel Durlach. Telepresence, time delay and adaptation. *Pictorial communication in virtual and real environments*, pages 232–246, 1991. Cited page 23.

Nadia Hocine. *Adaptation dans les jeux sérieux pour la rééducation fonctionnelle*. PhD thesis, Université de Montpellier 2, 2013. Cited page 30.

Ole-Ivar Holthe, Ola Mogstad, and Leif Arne Rønningen. Geelix livegames: Remote playing of video games. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, pages 758–759, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2308-8. URL http://dl.acm.org/citation.cfm?id=1700527.1700724. Cited page 19.

Simon Hurst. The characteristic function of the student t distribution. *Research report: statistics research report/Centre for mathematics and its applications (Canberra)*, 1995. Cited page 108.

Yoshiaki Ida, Yutaka Ishibashi, Norishige Fukushima, and Shinji Sugawara. Qoe assessment of interactivity and fairness in first person shooting with group synchronization control. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, page 10. IEEE Press, 2010. Cited page 22.

IGN. Slant six games chooses raknet for multi-platform project, 2011. URL http://uk.ign.com/articles/2011/02/08/slant-six-games-chooses-raknet-for-multi-platform-project. [Online; accessed 20th-Febuary-2014]. Cited page 38.

ISO. *Ergonomics of Human-system Interaction: Part 210: Human-centred Design for Interactive Systems*. ISO, 2010. Cited page 22.

JenkinsSoftware. Raknet - official website, 2011. URL http://www.raknet.com/. [Online; accessed 20th-Febuary-2014]. Cited pages 31, 38, and 48.

Thomas Judd and Robert W Levi. Dead reckoning navigational system using accelerometer to measure foot impacts, December 10 1996. US Patent 5,583,776. Cited page 28.

A. Jurgelionis, P. Fechteler, P. Eisert, F. Bellotti, H. David, J. P. Laulajainen, R. Carmichael, V. Poulopoulos, A. Laikari, P. Perälä, A. De Gloria, and C. Bouras. Platform for distributed 3d gaming. *Int. J. Comput. Games Technol.*, 2009:1:1–1:15, January 2009. ISSN 1687-7047. doi: 10.1155/2009/231863. URL http://dx.doi.org/10.1155/2009/231863. Cited page 18.

Jesper Juul. The game, the player, the world - looking for the heart of gameness. In Marinka Copier and Joost Raessens, editors, *Level up: Digital games research conference.*, pages 30–47+. Utrecht University, 2003. Cited page 14.

Antal Kozak. *Introductory probability and statistics: applications for forestry and natural sciences*. CABI, 2008. Cited page 70.

Nir Kshetri. Cloud computing in developing economies. *Computer*, 43(10):47–55, 2010. ISSN 0018-9162. doi: http://doi.ieeecomputersociety.org/10.1109/MC.2010.212. Cited page 5.

LibGDX. Libgdx introduction, 2013. URL https://github.com/libgdx/libgdx/wiki/Introduction. [Online; accessed 27-August-2014]. Cited page 100.

Ricardo Lopes and Rafael Bidarra. Adaptivity challenges in games and simulations: a survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2): 85–99, 2011. Cited page 29.

David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. ISBN 1558608389. Cited pages 56, 57, and 141.

Ghulam Mahdi. *Level of Detail in Agent Societies for Video Games*. PhD thesis, Université de Montpellier 2, 2013. Cited pages 58 and 76.

Ghulam Mahdi, Yannick Francillette, Abdelkader Gouaich, Fabien Michel, Nadia Hocine, et al. Level of detail based ai adaptation for agents in video games. In *ICAART'2013: 5th International Conference on Agents and Artificial Intelligence*, 2013. Cited page 30.

Chris McEntee. Rational design: The core of rayman origins, 2012. URL http://www.gamasutra.com/view/feature/167214/rational_design_the_core_of_.php. [Online; accessed 22-July-2014]. Cited page 62.

Peter Mell and Tim Grance. The nist definition of cloud computing. 2011. Cited page 4.

Alan Henry David Miller, John Philip McCaffery, Colin Allison, Lisa Dow, and Iain Angus Oliver. Measuring server qos in open virtual worlds: relating qos to qoe for opensim servers on the hyper-grid. *Proceedings of pgNET 2014*, 2014. Cited page 8.

MobyGames. Unreal, 1998. URL http://www.mobygames.com/game/game/unreal. [Online; accessed 16-August-2014]. Cited pages 41, 42, and 141.

MobyGames. Tribes 2, 2001. URL http://www.mobygames.com/game/tribes-2. [Online; accessed 16-August-2014]. Cited page 40.

Mogwai. Ethereal - 2d multiplayer action game, 2012. URL https://www.youtube.com/watch?v=455UnqQEqBU&feature=youtu.be. [Online; accessed 20th-Febuary-2014]. Cited pages 37 and 141.

MojOS. Stroke patients addicted to video games, 2009. URL http://euromov.eu/site/2012/02/609/. [Online; accessed 31-Octobre-2013]. Cited page 7.

Katherine L. Morse, Lubomir Bic, and Michael Dillencourt. Interest management in large-scale virtual environments. *Presence: Teleoper. Virtual Environ.*, 9(1):52–68, February 2000. ISSN 1054-7460. doi: 10.1162/105474600566619. URL http://dx.doi.org/10.1162/105474600566619. Cited page 36.

Nvidia. The power of cloud gaming, 2014. URL http://www.nvidia.com/object/cloud-gaming.html. [Online; accessed 27-August-2014]. Cited page 6.

Onlive. Onlive official web page, 2010. URL http://www.onlive.com. [Online; accessed 31-Octobre-2012]. Cited pages 7, 8, 20, 43, 48, and 141.

Onlive. Onlive's partners, 2014. URL http://www.onlive.be/corporate/partners. [Online; accessed 27-August-2014]. Cited page 6.

Reinhard Oppermann. *Adaptive user support: ergonomic design of manually and automatically adaptable software.* CRC Press, 1994. Cited page 29.

Lothar Pantel and Lars C Wolf. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 79–84. ACM, 2002. Cited page 28.

Martin Prangl, Ingo Kofler, and Hermann Hellwagner. Towards qos improvements of tcp-based media delivery. In *Networking and Services, 2008. ICNS 2008. Fourth International Conference on*, pages 188–193. IEEE, 2008. Cited pages 45, 46, 47, 48, and 50.

ITUT Rec. P. 800: Methods for subjective determination of transmission quality. *International Telecommunication Union, Geneva*, 1996. Cited page 22.

Vineet Richharya, Shweta Shrivastava, and Naman Agrawal. Image compression technique using different wavelet function. *IJRCCT*, 3(5):617–621, 2014. Cited page 44.

Carlos Oberdan Rolim, Fernando Luiz Koch, Carlos Becker Westphall, Jorge Werner, Armando Fracalossi, and Giovanni Schmitt Salvador. A cloud computing solution for patient's data collection in health care institutions. In *Proceedings of the 2010 Second International Conference on eHealth, Telemedicine, and Social Medicine*, ETELEMED '10, pages 95–99, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-3950-8. doi: 10.1109/eTELEMED.2010.19. URL http://dx.doi.org/10.1109/eTELEMED.2010.19. Cited page 5.

Jörg Rüppel. zoidcom - zoidcom automated networking system, 2011. URL http://www.zoidcom.com/. [Online; accessed 20th-Febuary-2014]. Cited pages 31, 36, 48, and 56.

Katie Salen and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. The MIT Press, 2003. ISBN 0262240459, 9780262240451. Cited page 14.

Katie Salen and Eric Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2004. Cited page 62.

Lee Salzman. Enet - official website, 2014. URL http://enet.bespin.org/. [Online; accessed 20th-Febuary-2014]. Cited pages 38, 48, and 56.

SilentWings. Silent wings simulator, 2006. URL http://www.silentwings.no/article/articleview/96/1/2//. [Online; accessed 16-August-2014]. Cited pages 39 and 141.

Nic Simmonds. Battlefield 3 pc bandwidth usage, 2011. URL http://mygaming.co.za/news/features/15417-battlefield-3-pc-bandwidth-usage.html. [Online; accessed 15-August-2014]. Cited page 47.

SMILE. Agent game development engine, 2012. URL http://gforge-lirmm.lirmm.fr/gf/project/agde/frs. [Online; accessed 27-August-2014]. Cited page 99.

SMILE. Game agent mechanics engine, 2013. URL http://info-depot.lirmm.fr/repos/SMILE/GAMEDEV. [Online; accessed 27-August-2014]. Cited page 100.

StreamMyGame. Streammygame official web page, 2007. URL http://www.streammygame.com. [Online; accessed 31-Octobre-2012]. Cited page 20.

Eileen La Susa. Cloud computing brings cost of protein research down to earth, 2009. URL http://www.eurekalert.org/pub_releases/2009-04/mcow-ccb040909.php. [Online; accessed 22th-January-2014]. Cited page 5.

T5-Labs. T5-labs official web page, 2007. URL http://www.t5labs.com. [Online; accessed 31-Octobre-2012]. Cited page 20.

Nicolas Tizon, Christina Moreno, Mihai Cernea, and Marius Preda. Mpeg-4-based adaptive remote rendering for video games. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 45–50. ACM, 2011. Cited pages 44 and 48.

Unlagged. A solution, 2002. URL http://www.ra.is/unlagged/solution.html. [Online; accessed 20th-Febuary-2014]. Cited pages 25 and 141.

Valve. Source multiplayer networking, 2004. URL https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking. [Online; accessed 20th-Febuary-2014]. Cited page 28.

Valve. Latency compensating methods in client/server in-game protocol design and optimization, 2009. URL https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization. [Online; accessed 20th-Febuary-2014]. Cited page 28.

Adrián Juan Verdejo, Katrien De Moor, Istvan Ketyko, Karen Torben Nielsen, Jeroen Vanattenhoven, Toon De Pessemier, Wout Joseph, Luc Martens, and Lieven De Marez. Qoe estimation of a location-based mobile game using on-body sensors and qos-related data. In *Wireless Days (WD), 2010 IFIP*, pages 1–5. IEEE, 2010. Cited page 22.

Village. Projet village, official site, 2013. URL http://www.projet-village.fr/. [Online; accessed 27-August-2014]. Cited page 105.

Vormetric. Data security in the cloud, 2012. URL http://www.vormetric.com/sites/default/files/wp-data-security-in-the-cloud.pdf. [Online; accessed 22th-January-2014]. Cited page 9.

Joost Westra, Frank Dignum, and Virginia Dignum. Scalable adaptive serious games using agent organizations. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 1291–1292, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9826571-7-X, 978-0-9826571-7-1. URL http://dl.acm.org/citation.cfm?id=2034396.2034531. Cited pages 59 and 60.

Whitaker. The game loop, 2014. URL http://rbwhitaker.wikidot.com/the-game-loop. [Online; accessed 22-July-2014]. Cited page 15.

Michael Wissner, Felix Kistler, and Elisabeth Andre. Level of detail ai for virtual characters in games and simulation. In *Proceedings of the Third international conference on Motion in games*, MIG'10, pages 206–217, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16957-0, 978-3-642-16957-1. URL http://dl.acm.org/citation.cfm?id=1948395.1948423. Cited page 11.

Matthias Wloka. Lag in multiprocessor virtual reality. *Presence*, 4:50–63, 1995. Cited page 23.

Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3:285–312, 2000. Cited page 59.

Wanmin Wu, Ahsan Arefin, Raoul Rivas, Klara Nahrstedt, Renata Sheppard, and Zhenyu Yang. Quality of experience in distributed interactive multimedia environments: toward a theoretical framework. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 481–490. ACM, 2009. Cited page 8.

M. Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9):25–32, 2005. Cited page 14.

List of Figures

List of Tables

Abstract

With the arrival of cloud computing technology, game accessibility and ubiquity have a bright future. Games can be hosted in a centralize server and accessed through the Internet by a thin client on a wide variety of devices with modest capabilities: cloud gaming. Some of the advantages of using cloud computing in game context includes: device ubiquity, computing flexibility, affordable cost and lowered set up overheads and compatibility issues. However, current cloud gaming systems have very strong requirements in terms of network resources, thus reducing their widespread adoption. In fact devices with little bandwidth and people located in area with limited network capacity, cannot take advantage of these cloud services. In this thesis we present an adaptation technique inspired by the level of detail (LoD) approach in 3D graphics. It is based on a cloud gaming paradigm in other to maintain user's quality of experience (QoE) by reducing the impact of poor network parameters (delay, loss, bandwidth) on game interactivity. Our first contribution consist of game models expressing game objects and their communications needs represented by their importance in the game. We provided two different ways to manage objects' importance using agents organizations and gameplay components. We then provided a level of detail approach for managing network resource distribution based on objects importance in the game scene and network conditions. We exploited the dynamic objects importance adjustment models presented above to propose LoD systems adapting to changes during game sessions. The experimental validation of both adaptation models showed that the suggested adaptation minimizes the effects of low and/or unstable network conditions in maintaining game responsiveness and player's QoE.

Keywords: *Level of detail, quality of experience, cloud gaming, agents organizations, gameplay components, network, communication, synchronization*

Résumé

Le Cloud computing émerge comme le nouveau paradigme informatique dans lequel la virtualisation des resources fournit des services fiables correspondant aux demandes des utilisateurs. De nos jours, la plupart des applications interactives et utilisant beaucoup de données sont développés sur le cloud: Le jeu vidéo en est un exemple. Avec l'arrivée du cloud computing, l'accessibilité et l'ubiquité du jeu ont un brillant avenir; Les jeux peuvent être hébergés dans un serveur centralisé et accessibles via l'Internet par un client léger sur une grande variété de dispositifs avec des capacités modestes: c'est le cloud gaming. Le Cloud computing dans le contexte de jeu vidéo a beaucoup attiré l'attention en raison de ses facilités d'évolution, de disponibilité et capacité de calcul. Cependant, les systèmes de cloud gaming actuels ont des exigences très fortes en termes de ressources réseau, réduisant ainsi l'accessibilité et l'ubiquité des jeux dans le cloud, car les dispositifs clients avec peu de bande passante et les personnes situées dans la zone avec des conditions de réseau limitées et/ou instables, ne peuvent pas bénéficier de ces services de cloud computing. Dans cette thèse, nous présentons une technique d'adaptation inspirée par l'approche du niveau de détail (Level of detail) dans les graphiques 3D. Elle est basée sur un paradigme du cloud gaming dans l'objectif de fournir une accessibilité multi-plateforme, tout en améliorant la qualité d'expérience (QoE) du joueur en réduisant l'impact des mauvaises conditions réseau (delai, perte, gigue) sur l'interactivité et réactivité du jeu. Notre première contribution se compose de modèles de jeu reliant les objets du jeu à leurs besoins en termes de communication représentés par leurs importances dans le jeu. Nous avons ensuite fourni une approche de niveau de détail pour gérer la distribution des ressources réseau basée sur l'importance des objets dans la scène et les conditions réseau. Nous validons notre approche en utilisant des jeu prototypes et evaluons la QoE du joueur, par des expériences pilotes. Les résultats montrent que le framework proposé fournit une importante amélioration de la QoE.

Mots clefs : *Niveau de detail, qualité d'expérience, cloud gaming, organisations d'agents, composants de gameplay, réseau, communication, synchronisation*

LIRMM — 161, rue Ada — 34095 Montpellier cedex 5 — France