An Architecture for Mobile Games with Cloud Computing Module

Mark Joselli
UFF
IC, Medialab

Marcelo Zamith
UFF
IC, Medialab

José Ricardo Silva Junior
UFF
IC, Medialab

Luis Valente
VisionLab/PUC-Rio

Esteban Clua
UFF
IC, Medialab

Bruno Feijó
VisionLab/PUC-Rio

Regina Célia P. Leal-Toledo
UFF, IC

Eduardo Soluri
Nullpointer Tecnologias

Abstract

Applications for mobile devices are getting more and more sophisticated, including features as location based systems, image recognition and speech recognition, for example. Most of these applications uses cloud computing to process these services. In order to use these features, this work presents a novel architecture specially designed for mobile games, which includes a cloud module for interacting with the aforementioned services. This architecture provides layers to control several modules as cloud services, networking, access to social networks, input, rendering and AI processing. Our proposal also allows for robust connections to social networks for publishing player achievements. This work also presents the AlienQuiz Invaders, which is a game built with the proposed architecture that uses augmented reality and speech recognition on the cloud.

Keywords:: Mobile Games, Cloud Computing, Game Loop, Augmented Reality, Real-Time Systems, speech recognition

Author's Contact:

mjoselli@ic.uff.br
mzamith@ic.uff.br
jricardo@ic.uff.br
lvalente@inf.puc-rio.br
esteban@ic.uff.br
bfeijo@inf.puc-rio.br
esoluri@nullpointer.com.br

1 Introduction

Mobile games are applications that run on mobile devices as smartphones and tablets. Those devices offer opportunities to design novel gaming experiences due to their distinct characteristics (as always-on connectivity, multitude of sensors).

As an example, smartphones provide a high degree of convergent features: multimedia capacities (producing and consuming audio, video), networking (local and global), and sensors (camera, accelerometers, GPS, etc). This opens up the possibility to create games as location based games [M1ndLab 2007], voice based games [Zyda et al. 2008], accelerometer-based games [Valente et al. 2009], [Valente et al. 2008] and [Chehimi and Coulton 2008], camera-based games [Park and Jung 2009] and touch based games [Rohs 2007]. In order to develop good mobile games, they must be designed to take advantages of such unique characteristics into gameplay [Zyda et al. 2007]. The game presented in this paper as an example of our architecture uses many of these unique features, as the camera, accelerometer sensor, touch and speech inside a augmented reality game.

Digital games are real-time interactive multimedia applications. In other words, if the application is not able to perform the required tasks on time, it will fail. For game applications, this means not being able to sustain the interactive experience, due to factors like the game taking too much time to process the tasks, or delayed responses for user input. In the case of the cloud services, if the loop has to wait for the services response, the interactivity will fail. A common parameter for measuring game and visual simulation performance is frames per second (FPS). The general lower acceptable bound for a game is 16 FPS. There are not higher bounds for FPS measurements, but when the refresh rate of the video output (a computer monitor or the mobile screen) is inferior to the game application refresh rate, some generated frames will not be presented to the user (they will be lost). One motivation for designing game loops is to better achieve an optimal FPS rate for the application. A new game loop model, called mobile game loop with a cloud module, is one of the contributions of this work.

Mobile phones are connected devices by definition. This means networking is an important feature to consider for mobile applications, especially for games. Mobile phones are able to establish connection with local (co-located) or global peers. Local peers are connected through technologies as Bluetooth, while global peers are reached through WiFi and the mobile operator network (e.g. 3G). This built-in feature makes the use of the cloud as a service on mobile devices a natural move.

The architecture presented in this work uses cloud computing [Armbrust et al. 2009] to offer a set of services. In traditional cloud computing, machines across the internet share resources, software and information, while in our approach the mobile client is able to use resources available in the network to process some services, like speech, location based services (like retrieving map data and other context information based on the device location) and image recognition, enhancing the computational capacity of those services.

1.1 Motivation and Contribution

This work aims at providing a layer to use cloud services in mobile games to improve game quality in general (visual effects, AI processing, game responsiveness, and so on).

This work extends previous works proposed by Zamith et al. [ZAMITH et al. 2011], where the authors present a framework for game loops that use automatic task distribution between CPU cores and the GPU. The present work discusses an approach to mobile game loops that employs a distribution architecture, based on cloud computing paradigm, which also includes the concepts presented in earlier works and new ones as follows:

- Load balancing of cloud computing tasks;
- Use of mobile unique features, like augmented reality, speech recognition and location, in games;
- Mobile games using the cloud to process services;
- A new game loop model, for mobile games with a cloud module;
- A prototype mobile game using augmented reality, speech recognition and location features provided by the proposed architecture, as a proof of concept.

Finally, the organization of this work is as follows: Section 2 presents related works. Section 3 discusses the proposed game loop model that this work proposes. Section 4 presents the architecture developed for this work and Section 5 presents the test case, the AlienQuiz Invaders game, to validate the architecture. Section 6 discusses and analyses the tests. Finally, Section 7 presents the conclusions of this work.

2 Related Work

Mobile devices present constraints (as processing power, memory and battery life) that pose a challenge for developing complex mobile games. On the other hand, cloud computing services allow for processing tasks that have high processing demands (using computer clusters, for example).

Computer games are multimedia applications that employ knowledge of many different fields, such as Computer Graphics, Artificial Intelligence, Physics, Network and others [Valente et al. 2005]. More specifically, computer games are interactive applications that exhibit three general classes of tasks:

- Data acquisition in games is related to gathering data from input devices as keyboards, touch screens, mice and joysticks. This is also known as ”player input”;
- Data processing tasks consist on applying game rules, responding to user commands, simulating Physics and Articial Intelligence behaviors. These tasks are referred to as ”update tasks”;
- Data presentation tasks relate to providing feedback to the player about the current game state, usually through images and audio. Commonly, these tasks are referred to as ”game visualization and auralization”.

This work includes a new class of task: a cloud processing task. This kind of task is made of connectors that interact with services on the cloud.

The real-time loop represents the heart of real-time simulations and games. However, the literature on this subject is scarce. The works by Valente et. al [Valente et al. 2005], are among the few works that discuss this subject. Dalmau [Dalmau 2003], Dickinson [Dickinson 2001], Watte [Watte 2005], Gabb and Lake [Gabb and Lake 2005], and Mönkkönen [Mönkkönen 2006]. None of them discuss game loop models with focus on mobile devices and cloud services.

The most straightforward approach to modeling real-time loops (for single-player games) is the Simple Coupled Model. Basically, this model consists of sequentially arranging the tasks in a main loop as Figure 1 illustrates. This is a basic approach that mobile games have already adopted in the past, due to its simplicity.

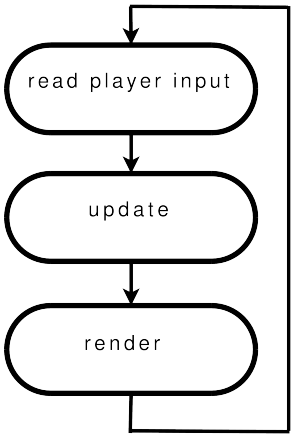


Figure 1: *Simple Coupled Model*

Dickinson [Dickinson 2001] proposed an extension to the Simple Coupled Model, named Single-thread Uncoupled Model. This model has the rendering and updating stages uncoupled, i.e., rendering and updating are running independently of the power processing of CPU. Moreover, the single-thread uncoupled model tries to bring determinism to the game execution by feeding the update stage with a time parameter. For example, existing open-source game engines, as [COCOS2D 2011] adopts this model.

Single-thread Uncoupled Model is an improved model that adapts the game execution according to the capacity of the host machine, so the game runs the same way in different devices. More powerful devices will be able to run the game more smoothly, while less powerful ones should still be able to provide some experience to the user.

Although these are working solutions, time measuring may greatly vary in different hardware devices due to many reasons (such as process load), making it difficult to be reproduced faithfully. For

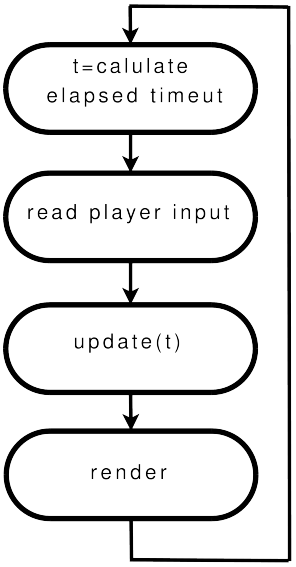


Figure 2: *Single-thread Uncoupled Model*

example, a network module implementation and program debugging [Dickinson 2001] may be easier to implement if the loop uses a deterministic model. Another issue is that running some simulations too frequently, like AI and the game logic, may not yield better results.

Hence, research works as [Valente et al. 2005] propose models that try to address those issues. The Fixed-frequency Uncoupled Model outlined in [Valente et al. 2005] features another update stage that runs at fixed frequency, besides the time-based one. The work by Dalmau [Dalmau 2003] presents a similar model, although not naming it explicitly. Those works describe the model using a single-thread approach. Figure 3 illustrates the Fixed-frequency Uncoupled Model.

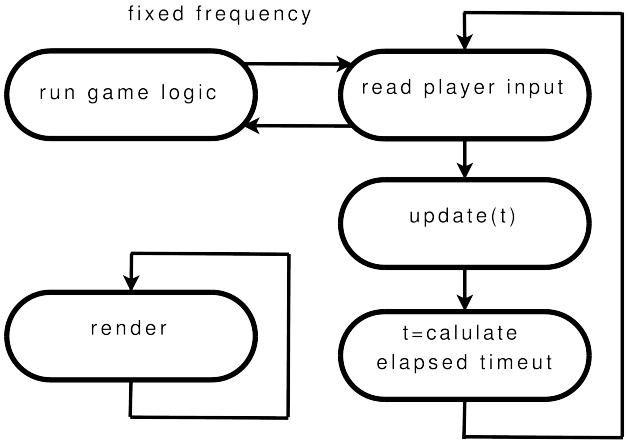


Figure 3: *Fixed-frequency Uncoupled Model*

Dickinson [Dickinson 2001] discusses another approach for fixed-frequency uncoupled models, which presents just one up date stage that runs at a fixed-frequency. The main objective of that model is to attain reproducibility.

Current mobile devices like the Motorola Atrix 2, iPad 3, iPhone 4S, and the Samsung Galaxy S-III, have multi-core processors. For this reason, real-time loops for mobile games that take advantage of those resources are likely to become important in the near future. Therefore, making the tasks parallel in multiple threads is a natural step.

However, dealing with concurrent programming introduces another set of problems, such as data sharing, data synchronization, and deadlocks. Also, as Gabb and Lake [Gabb and Lake 2005] state

not all tasks can be fully parallelized due to dependencies among them. As examples, in a game, characters cannot move until the game logic is computed, and rendering cannot be performed until the game state is updated. Hence, serial tasks represent a bottleneck to parallelized simulation computation.

The Asynchronous Function Parallel Model [Mönkkönen 2006], which separates the tasks (input player, update and render) in three threads. Another example is the Synchronous Function Parallel Model [Mönkkönen 2006], which processes the game physics in a separated thread while the main thread process the characters animations.

The Data Parallel Model [Mönkkönen 2006] uses a different paradigm where data are grouped in parallel sections of the application where they are processed. So, instead of using a main loop with concurrent parts that process all data, the Data Parallel Model proposes using separate threads for sets of data (like game objects). This way, the objects run their own tasks (like AI and animation) in parallel. Figure 4 depicts this approach.

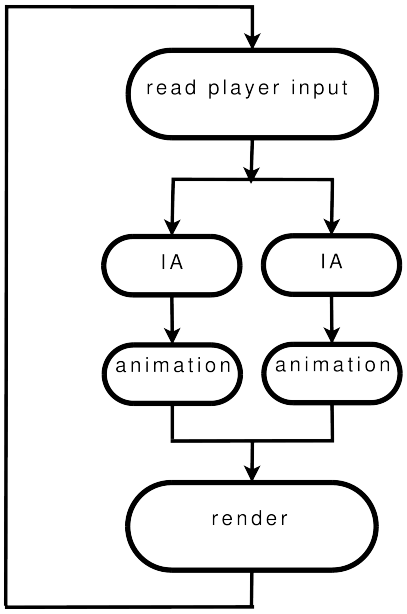


Figure 4: *Data Parallel Model*

According to the author [Mönkkönen 2006], his model scales well because it is able to allocate as many processing cores as they are available. Performance is limited by the amount of data processing that can run in parallel. An important issue is how to synchronize communication of objects running in different threads. Mönkkönen states that the biggest drawback of this model is the requirement to have components designed with data parallelism in mind. Mönkkönens work has been inspirational for the distributed part of our architecture, where it splits a task into threads that run across the cloud.

Barbosa and co-authors [Barboza et al. 2010a] propose a cloud-computing approach for mobile devices, similar to the Onlive architecture [OnLive 2012]. In their proposal, a mobile device sends user input in formation to a server, and later receives back the rendered images for the game as a streaming video. Figure 5 illustrates the game loop by Barbosa and co-authors.

Another approach for game loop architectures adopts the GPU as a new resource in the computer. This resource can be used to process physics or any other massively mathematics problems apart from visualization task. This approach is based on GPGPU, whose importance has been increasing since graphics hardware became programmable. There are some works that discuss using GPGPU with game loops [Barboza et al. 2010b; Joselli et al. 2008b; Joselli et al. 2008c; Joselli et al. 2008a; Joselli et al. 2010]. However, these works concentrate on game loops for desktop computers, and are not applicable yet to mobile devices as current mobile GPUs do not have GPGPU programming capabilities.

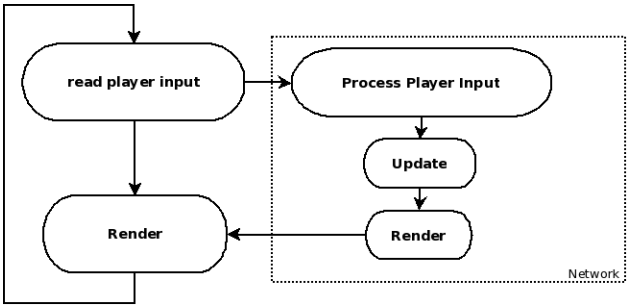


Figure 5: *The cloud game loop*

Our work differs from previous works, as it is concerned mainly with providing a cloud module for mobile applications. By using this approach, a game is able to use tasks and services that could not be processed on a mobile device, due to device and/or network constraints (power processing or memory capacity).

3 The mobile game loop with a cloud computing module model

This work proposes an architecture for game loops that has a cloud computing module. This architecture comprises a server cluster and a set of mobile devices. The mobile devices connect to the server cloud to use cloud computing services. Despite the cloud computing module being uncoupled from the main game loop, the cloud computing module and the cloud itself are coupled, because using cloud services requires a blocking operation (sending and receiving data over a network). Hence, using this kind of service may degrade game performance if the network or the cloud fails. In this case, the game stops and will resume when the cloud communication is back again. On the other hand, using this approach (the cloud computing module) allows for some network delays to occur without affecting the game performance negatively. Figure 6 illustrates this loop model.

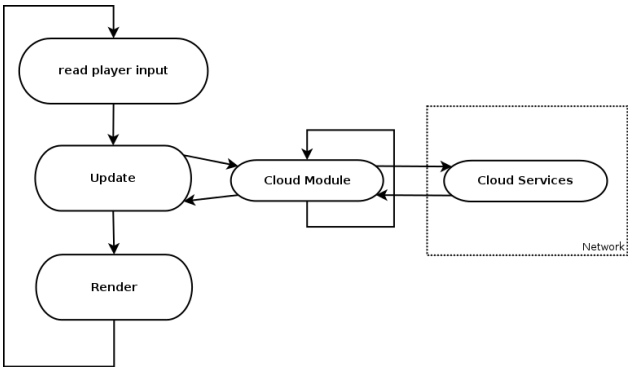


Figure 6: *The game loop with a cloud computing module model.*

The architecture has two main components: the main game loop (responsible for the traditional tasks of the game, like handling user input, presentation tasks and the update tasks), and the cloud module, which is responsible for the communication between the game and the cloud services the game uses.

The current implementation of the proposed architecture uses OpenGL ES 2.0 for rendering and Vuforia [Qualcomm 2012] for augmented reality processing. The implementation uses cloud services for speech recognition and synthesis, with iSpeech [iSpeech 2012].

Vuforia is a Software Development Kit to create augmented reality applications on mobile devices. Vuforia uses computer vision algorithms to recognize image targets and 3D objects in real-time. With this SDK, the game creates virtual 3D objects and superimposes them on the game view, on top of image targets. The recognition part uses a tracker component, which comprises computer vision al-

gorithms used to detect and track real world objects. These objects form a ”image targets database”. The database is a XML file that contains all trackable image targets that the application will use. The Vuforia Target Management System is responsible for compiling these assets, creating the image database.

Using voice for interacting with mobile devices is becoming more common. For example, Apple iOS devices provide the SIRI application and Android devices have built-in packages to use this functionality. These voice-related services require an internet access to work, since they need the cloud to process this task. The present work also needs a cloud server to process voice. We used the iSpeech speech recognition SDK, which is a commercial product (free for mobile platforms, which is the case of the test-case of this work) that process TTS (text to speech) and ASR (automated speech recognition) on the cloud.

4 The Architecture

The core of the proposed architecture corresponds to the Task Manager and the Performance Test class. The Task Manager schedules tasks to be processed either locally, or on the cloud. The Performance Test detects the available hardware configuration capabilities. For example, if a game must use network or location services, this must be available for the game.

In the proposed architecture, a task can be anything that the application must process. The Task class is the abstract base class and has four subclasses: Update Task, Performance Test Task, Input Task, Presentation Task and Task Manager. These classes, except for Task Manager, are also abstract classes. The Task Manager is a special class that is responsible for performing the task distribution. Figure 7 illustrates the UML diagram of the proposed architecture, including the task classes that our case study uses.

The remaining of this Section describes the UML diagram elements, with five subsections: Input Task, Presentation Task, Performance Test Task, Update Task and Task Manager. The remaining of this Section details each one of these tasks.

4.1 Input Task

The Input Task classes are responsible for handling user input. Input may come from different sources. Typically for desktop applications this corresponds to keyboards and mice, while mobile applications may use touch screens and microphones as input devices, for example.

The Input class is an abstract class with seven children: Keyboard Task, Mic Task, Touch Task, GPS Task, Camera Task, Compass Task, and Accelerometer Task. The Keyboard task gathers input from keyboards, which can be physical (like PC keyboards and numerical keypads on mobile devices) or virtual (where a mobile device simulates a keyboard on touch screens by drawing the keys). The Mic Task gathers sounds that comes from the microphone. The Touch Task handles input from touch screens, as well as gestures. The GPS class gathers information coming from GPS sensors. The Accelerometer Task gathers input coming from accelerometer sensors. The Camera Task captures images, which can also be used to estimate device motion. The Compass Task reads the device compass to determinate the direction of the player.

4.2 Presentation Task

The Presentation task is responsible for rendering game scenes, usually being a feedback to the player about the current game state, which was calculated by the Update Task. This information can reach the user through several modalities, as visual (Render Task, with images, 3D models and visual effects), aural (Sound Task, with music and sound effects), or haptics (Vibrate Task, controlling the vibration motors in mobile devices). Also on some mobile devices, the Render Task class can control the luminosity of the screen, allowing a different kind of visual feedback.

4.3 Update Task

The Update Task classes and subclasses are responsible for processing the new game state. This game state may include results of physical simulations and/or artificial intelligence processing.

The Update Task is an abstract class and has two children: an abstract Local Task, which is used for tasks that are processed locally on the device, and an abstract Network Task, for tasks that use the network for processing.

4.3.1 Local Task

The local task is responsible for processing that should occur on the device itself. This kind of task corresponds to the ones that require fast feedback, making them unfeasible to being processed over a network. In our architecture, there are three kinds of task that fall into this category: AR Task, Physics Task, and AI Task. The AR Task is responsible for the processing camera images by recognizing image patterns (the targets) and placing virtual objects on them, according to the targets positioning and orientation. The Physics Task is responsible for simulating Newtonian physics in the game, by checking and resolving collisions through Physics laws. The AI Task simulates NPCs (non-player character) behavior using artificial intelligence.

The game could also use cloud services to process those kind of tasks, like previous works discuss in [ZAMITH et al. 2011]. However, this would require high data transfers over the network, which is not ideal for mobile devices that often rely on 3G (or EDGE) networks.

4.3.2 Network Task

The Network Task is responsible for all update tasks that needs to access a network. For example, this includes connecting to social networks, exchanging messages in multiplayer games, and using cloud services. Here are some tasks derived from the base Network Task:

- **Social Task:** The social task is responsible for connecting to the main social networks to publish achievements and scores. This class is abstract and has four children, one for each social client. The Facebook Task connects to the Facebook to gather news from friends, which also played the game and to publish achievements. The Game Center task is responsible for connecting to the iOS game center, which controls player achievements but could also be used for multiplayer games. The OpenFeint Task has the same features as the Game Center Task, but it is also available for platforms other than iOS. The Twitter task is responsible for connecting to Twitter.
- **Multiplayer Task:** The multiplayer task is responsible for connecting different players in the same game. In order to fulfill this task, this class implements a protocol for message exchanging, which is similar to the previous approach discussed in [ZAMITH et al. 2011].
- **Cloud Task:** The Cloud task is responsible for services the mobile game can use, and are processed using cloud computing. For example, Speech Tasks, location based service tasks and Image Recognition Task.
- **Speech Task:** The Speech task is responsible for the speech processing recognition and synthesis. In our architecture this has two main uses: to acquire voice and translate it to game commands, and to synthesize speech to provide audio feedback to the player.
- **Location Task:** The location task is responsible for gathering information from the cloud about the local context where the device is. For example, this could be map data and information about points of interest related to the device location (like nearby shops, bars, other players).
- **Image Recognition Task:** The Image Recognition Task is responsible for image processing and analyzing over a network, using a cloud infrastructure. When an AR Task runs

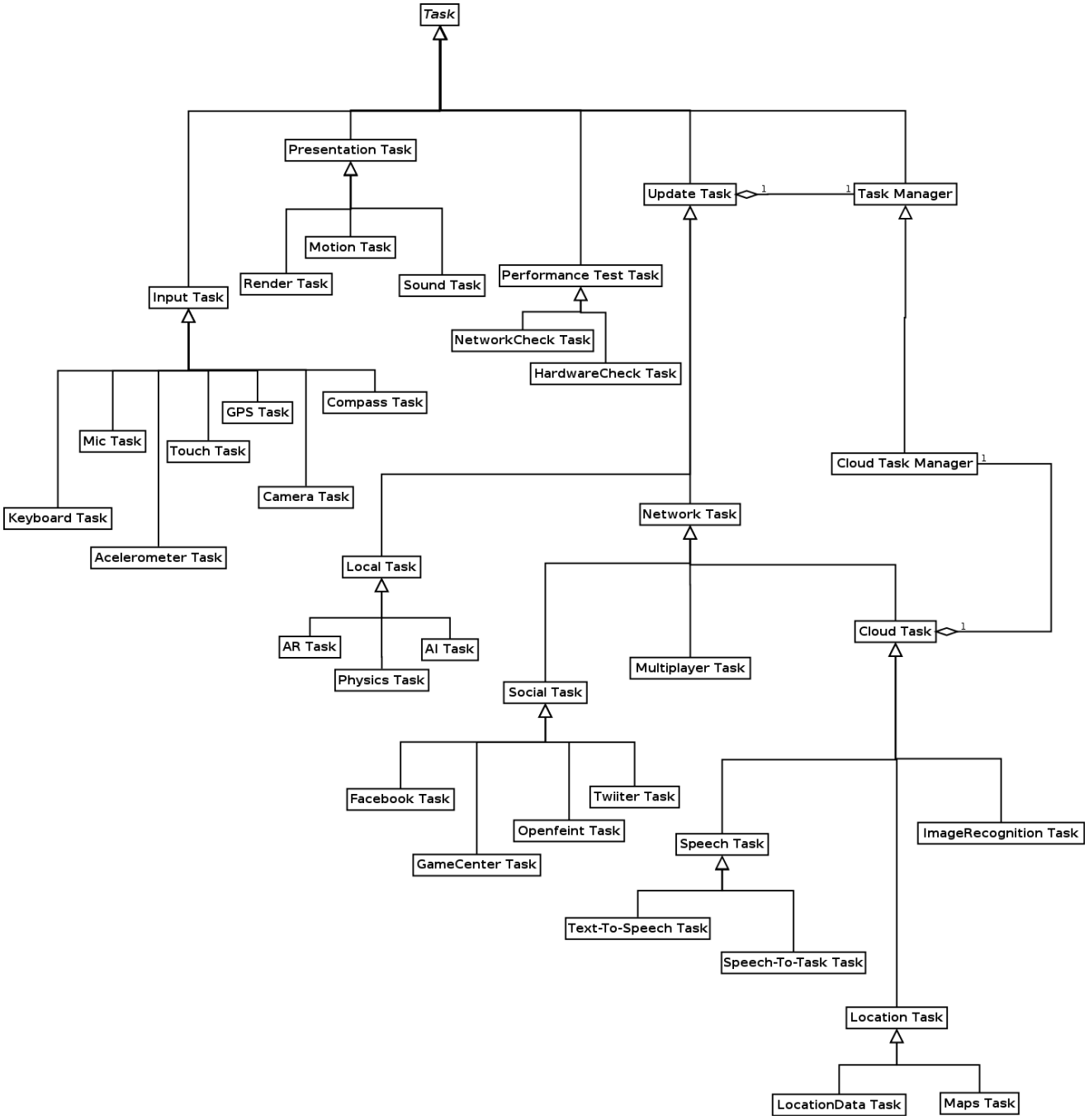


Figure 7: *Architecture UML Diagram.*

locally, the number of image targets it is able to handle is limited due to device constraints. However, when using the Image Recognition Task the application is able to use a cloud service, making it possible to use a virtually unlimited image target database.

4.4 Performance Test Task

The Performance Test class is responsible for gathering and testing all the resources that the game would use. This class has two children, one for the hardware tests and another for network tests.

4.5 Hardware Test Task

The Hardware Check is responsible for gathering information about hardware features available to the game. There is only one instance of this class in the application. This class checks the available hardware and keeps track of the configuration, like available inputs mechanisms, the number of processing cores and the model of the device.

By using this class, the Task Manager is able to get to know the available hardware without previous knowledge. The Hardware Test Check class instance is always created at the beginning of the game execution.

4.6 Network Test Task

The Network Test Task is responsible for checking the network resources available for the device. There is only one instance of this class in the application. This class performs connectivity tests to estimate the network bandwidth available to services the game needs to use.

For example, when using this class the architecture is able to query the network about the available bandwidth, so the game is able to determine which network services it can use. This class also pro-

vides an error handling mechanism for receiving messages about network errors and failures. The unique Network Test Task class instance is created at the beginning of the game execution.

4.7 Task Manager

The Task Manager (TM) is the core component of the proposed architecture. It is responsible for instancing, managing, synchronizing, and finalizing tasks. The Task Manager uses a XML configuration file to determine the ordering and dependencies among the threads. In order to configure the execution of the tasks, each task has control variables described as follows: UNIQUEID: the unique id of the task. It is used to identify the tasks; TASKTYPE: the task type. The following types are available: input, update, presentation, and manage; DEPENDENCY:a list of the tasks(ids)that this task depends on to execute.

With that information, the TM creates the task and configures how the task is going to execute. A task manager can also hold another task manager, so it can use it to manage some distinct group of tasks, which is the case of the Cloud Tasks Manager. The XML bellow illustrates a speech-to-text task configured by the XML.

```
<Config>
  <Tasks>
        <uniqueId>
        speechToTextTask
        </uniqueId>
        <taskType>
        Cloud
        </taskType>
        <Dependency>
        MicTask
        </Dependency>
  </Tasks>
</Config>
```

4.8 The Cloud Task Manager

The purpose of this class is to manage all the tasks that uses cloud computing. The class may change execution ordering of some task services (like the speech recognition service) when the application is running, which characterizes a dynamic task distribution. Figure 8 illustrates the process.

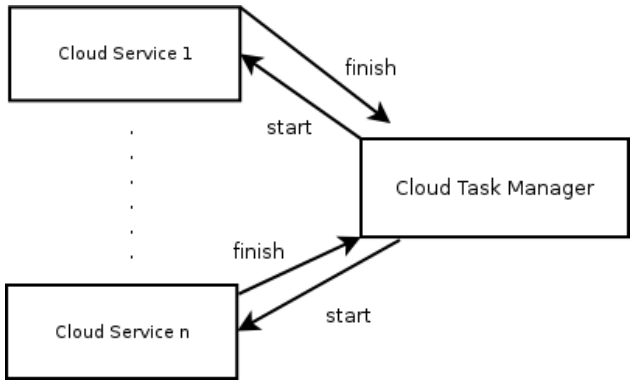


Figure 8: *The Cloud Task Manager.*

This class always keeps the current bandwidth of the services, in order to evaluate if the service should be changed during execution. The Cloud Task Manager acts as a server and the cloud tasks act as its clients, as every time a task ends, it sends a message to the Task Manager. The Task manager then checks which task it should execute in the thread.

As the Cloud Task Manager uses a multi-thread loop model, it is necessary to apply a parallel programming in order to identify the shared and non-shared sections of the tasks, because they should be treated differently. The independent sections compose tasks that are processed in parallel, like the communication with the cloud. The shared sections, like the update tasks, need to be synchronized in order to guarantee mutual-exclusive access to shared data and to preserve task execution ordering.

The processing of shared objects needs to use a synchronization object (as a mutex), as it is common in concurrent programming. Concurrent programming is a complex subject, because the tasks in the application run alternately or simultaneously, but not linearly. Hence, synchronization objects are tools for handling task dependence and execution ordering. This measure should also be carefully applied in order to avoid thread starvation and deadlocks. The Cloud Task Manager uses semaphores as the synchronization object. Lastly, as in this case it is required to specify the execution ordering, this constraint must be described in the XML configuration file. .

5 Test Case: AlienQuiz Invader

The game prototype that uses the proposed architecture was implemented with Objective-C in the iOS platform. The cloud services this game uses are: a speech-to-text service, a text-to-speech service (provided by iSpeech), and a custom-made location service that indicates available locations for augmented reality objects. This location service is used to guide the player to the location of the nearest image targets.

The game uses the AudioToolbox framework from the iOS SDK to implement audio features, and the Vuforia SDK to implement augment reality. The 3D model the game uses is a simple disc with different textures applied. Figure 9 illustrates some game screenshots.

5.1 Game Design

The AlienQuiz Invaders is an augmented reality game inspired on the classic Space Invaders. Besides being an action game (like the original), this version also features a speech-based quiz game.

The game is simple, fun, and different from traditional games. The player walks around using the mobile device camera, which displays on the device screen the location of ”portals”, where aliens invade the Earth. Whenever the player points the camera to a place where there is an image target, he/she sees a portal and aliens coming out of it. In order to survive, the player must avoid hitting the bullets and colliding with the aliens, as these events consume the player’s energy. The player is able to eliminate the aliens by touching them on the device screen.

The portals have another feature: they are active entities that keep asking math questions to the player, like ”How much is five times six?”. This happens once every minute, using speech. If the player answers the question correctly (through voice) , the portal diminishes its size and reduces the number of aliens spawn from it. If the player misses the question, the portal increases its size and increases the number of aliens spawn from it. The players mission is to sustain his energy (not letting it become zero), while closing game portals.

The AlienQuiz Invaders game uses the device camera (as the players aim), touch input to fire lasers (to eliminate the aliens) and speech to both ask and answer questions.

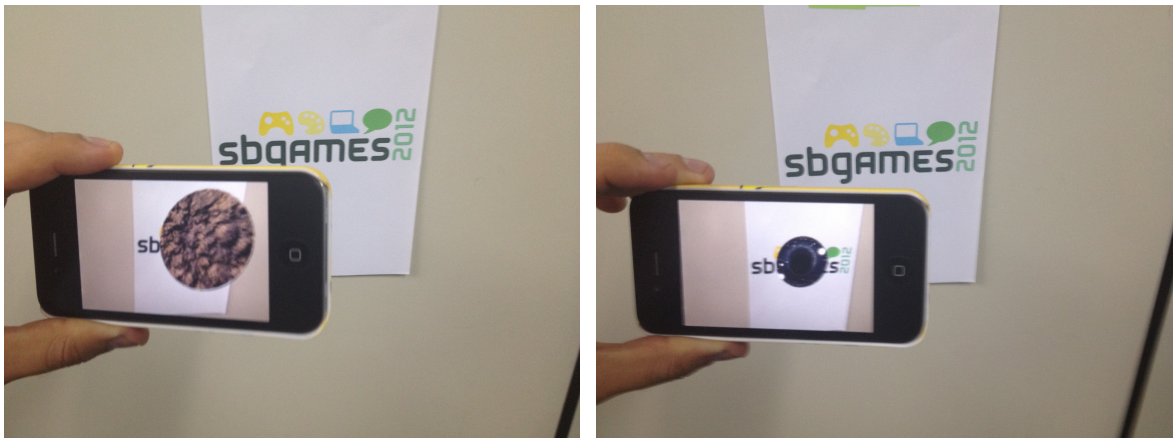
6 Results

The test device is an Apple iPhone 4S, which is a mobile phone with a dual-core 1 GHz Cortex-A9 processor, a PowerVR SGX543MP2 GPU, and 512 MB of RAM, running the iOS operation system version 5.1.1.

6.1 Game Execution

This Section describes the execution flow of the proposed architecture, so the reader is able to better understand it. Figure 10 illustrates this process.

Firstly, the architecture queries the mobile device about available hardware. Next, the architecture queries the game about services



(a) (b)

Figure 9: *ScreenShots of the game.*

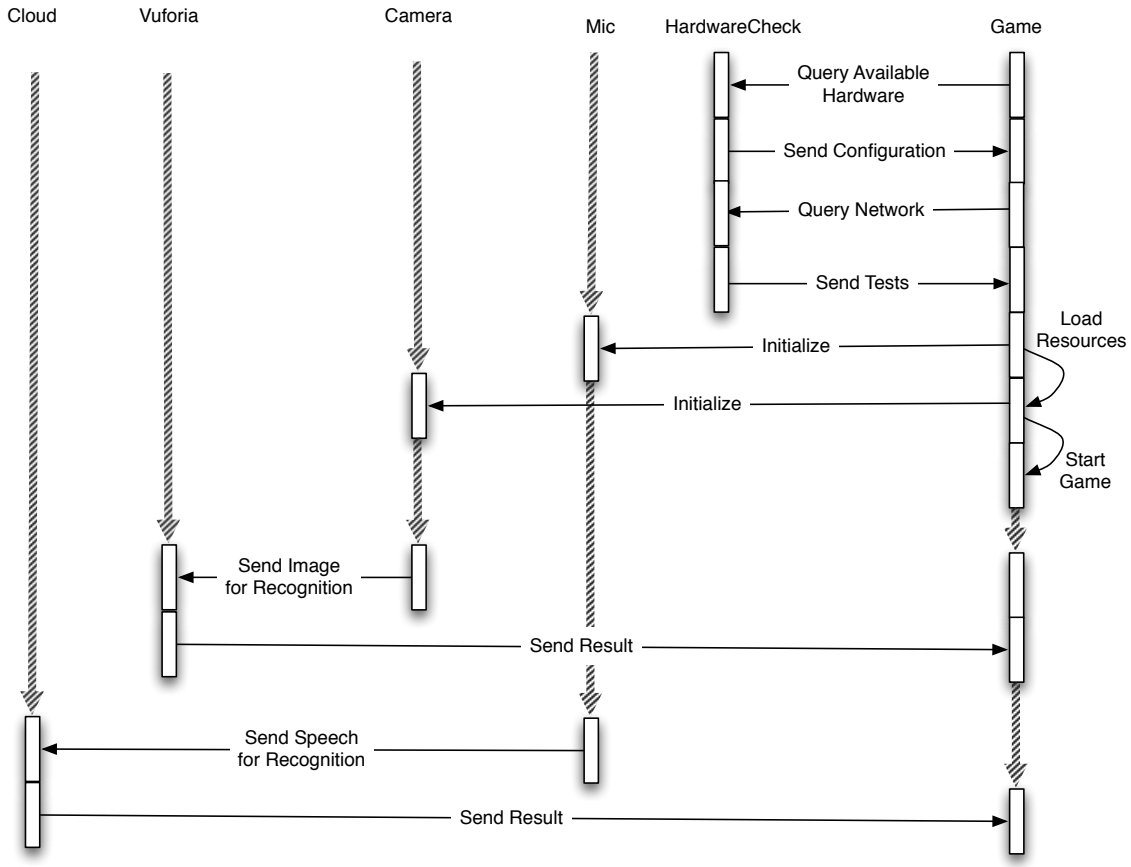


Figure 10: *Sequence diagram of the Game.*

that use the network, in order to estimate the required bandwidth. After this procedure, the game starts loading resources and initializing. The next step is starting the game loop, using Vuforia to process augmented reality and the cloud module to process speech.

6.2 Image Targets

The game uses image targets to place portals and enemies in the real world, creating an augmented reality layer. The image targets correspond to patterns that the game recognizes in the video stream coming from the device camera. The game uses the Vuforia SDK to process image targets.

The AlienQuiz Invaders game uses two image targets: the SBGames conference logo and a stone floor picture. Figure 11 illustrates these two image targets. The Vuforia SDK works better with visual elements with sharp edges and high contrast, which is not the case of the SBGames conference logo image.

When Vuforia generates patterns on images, it calculates a numerical property known as effective. Images with a low effective value will not track well and may not be robust to occlusions. The effective property for the SBGames conference logo is 40%, while this property for the stone floor is 100%. Figure 11 illustrates the images and the outlined patterns (represented as yellow crosses).

In order to test game architecture performance, this work has per-

(a) SBGames logo (b) SBGames logo with outlined patterns

(c) Stone floor (d) Stone floor with outlined patterns

Figure 11: *The different image recognitions.*

formed tests in three different stages. The first one occurs at the beginning of a game session, before the portals are open, when the player has not found yet image targets (this stage is referred to as ”game recognition”. The second stage occurs when the player has found an image target, but there is no cloud connection in use (this stage is referred to as ”in game, no cloud”. Finally, the third stage occurs when the player has found an image target and there is a cloud connection in use (this stage is referred to as ”in game, using cloud”. Table 1 illustrates these tests. To assure that results are consistent, all tests of this work were repeated 10 times and the standard deviation of the average times was confirmed to be within 5%.

Table 1: *Numerical results from the game on different steps.*

Test	Game recognition	In game, no cloud	In game, using cloud
Time	3.62 ms	1.61 ms	2.66 ms
CPU usage (%)	66%	38%	55%

The results describe that the ”game recognition” stage takes more time to run than the other two stages. This happens mainly because Vuforia spends a fair amount of time trying to recognize image targets when there are no image targets tracked. When the game uses a cloud service, this request happens in a non-blocking thread, which helps to minimize the negative impact in game performance.

Table 2 lists the time the game spent using cloud services for speech synthesis (text-to-speech) and speech recognition, along with network usage and CPU usage. These parameters were measured in the mobile device. This measured time includes gathering input (audio from the microphone for speech recognition and text for speech synthesis), preparing this data to send over the network, sending the data over the network, and receiving a response from the service (a sound file from speech synthesis services, or text from the speech recognition service). These results present an one time operation, for each cloud service.

These results suggest that these services are time consuming and could not be implemented in a single-threaded game loop, as they use blocking operations (sending data over the network and waiting for the response). These results also describe a low CPU usage that is mainly regarded to audio processing. The network usage in Table

Table 2: *Numerical results from the game on different steps.*

Tests	Speech synthesis	Speech recognition
Time	2.2s	1.5s
CPU usage %	17%	13%
Network usage	2MB	1.2MB

2 regards the audio file transfers due to using the speech service.

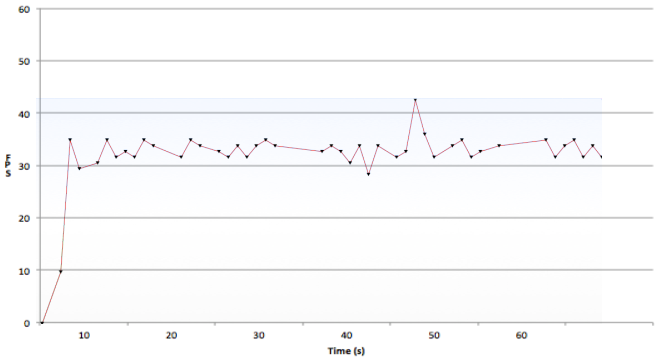


Figure 12: *Performance of the game.*

Figure 12 illustrates the game performance (in FPS) for one minute of game playing, which includes the three stages described in Table 1.

From this figure it can be seen that the performance of the game ranges on an average of 30 frames per second (FPS), which is considered optimal in a game.This figure shows that the speech features and the recognition tasks of the game do not affect the frame rate (the speech feature was called one time during this test).

7 Conclusions

The development, evolution and use of cloud computing in mobile applications is a trend. However, using this kind of service in mobile games is still in the experimental stage.

This work proposed a new architecture for mobile games that use cloud services as a part of the game. Also this work presented a game example that implements the proposed architecture - Alien-Quiz Invader. This game uses AR and cloud services to provide a better interactivity in the game.

This work also discussed the concept of game loops, a subject that is not very often discussed in the literature. Our contribution lies on extending a previous work, by providing an architecture for game loops with a cloud module. This cloud module has a cloud task manager to provide a layer to interact to the cloud services, without affecting the main game loop (or keep the affect to a minimum).

References

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. 2009. Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb.

BARBOZA, D. C., JUNIOR, H. L., CLUA, E., AND REBELLO, V. E. F. 2010. A simple architecture for digital games on demand using low performance resources under a cloud computing paradigm. *Proceedings of the IX Brazilian Symposium on Computer Games and Digital Entertainment*, 38–45.

BARBOZA, D. C., JUNIOR, H. L., CLUA, E. W. G., AND REBELLO, V. E. 2010. A simple architecture for digital games on demand using low performance resources under a cloud computing paradigm. *Games and Digital Entertainment, Brazilian Symposium on 0*, 33–39.

CHEHIMI, F., AND COULTON, P. 2008. Motion controlled mobile 3d multiplayer gaming. In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, ACM, New York, NY, USA, ACE, 267–270.

COCOS2D, 2011. Cocos2d. Avalible at: http://www.cocos2d-iphone.org/games/. 30/09/2011.

DALMAU, D. S. C. 2003. *Core Techniques and Algorithms in Game Programming*. New Riders Publishing.

DICKINSON, P., 2001. Instant replay: Building a game engine with reproducible behavior. Available at http://www.gamasutra.com/features/20010713/dickinson_01.htm/ .

GABB, H., AND LAKE, A., 2005. Threading 3d game engine basics. Available at http://www.gamasutra.com/features/20051117/gabb_01.shtml/ .

ISPEECH, 2012. http://www.ispeech.org/.

JOSELLI, M., ZAMITH, M., CLUA, E., PAGLIOSA, P., CONCI, A., MONTENEGRO, A., AND VALENTE, L. 2008. An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu. *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, 115–120.

JOSELLI, M., ZAMITH, M., VALENTE, L., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., FEIJÓ, B., DORNELLAS, M., LEAL, R., AND POZZER, C. 2008. Automatic dynamic task distribution between cpu and gpu for real-time systems. *Proceedings of the VII Brazilian Symposium on Computer Games and Digital*, 48–55.

JOSELLI, M., CLUA, E., MONTENEGRO, A., CONCI, A., AND PAGLIOSA, P. 2008. A new physics engine with automatic process distribution between cpu-gpu. *Sandbox 08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, 149–156.

JOSELLI, M., ZAMITH, M., CLUA, E., LEAL-TOLEDO, R., MONTENEGRO, A., VALENTE, L., FEIJO, B., AND PAGLIOSA, P. 2010. An architeture with automatic load balancing for real-time simulation and visualization systems. *JCIS - Journal of Computational Interdisciplinary Sciences*, 207–224.

M1NDLAB, 2007. Alien revolt: Location-based massive-multiplayer online rpg. Avalible at: http://www.alienrevolt.com.

MÖNKKÖNEN, V., 2006. Multithreaded game engine architectures. Available at http://www.gamasutra.com/features/20060906/monkkonen_01.shtml .

ONLIVE, 2012. http://www.onlive.com/.

PARK, A., AND JUNG, K. 2009. Flying cake: Augmented game on mobile devices. *Comput. Entertain. 7*, 1, 1–19.

QUALCOMM, 2012. Vuforia http://www.qualcomm.com/solutions/augmented-reality.

ROHS, M. 2007. Marker-Based Embodied Interaction for Handheld Augmented Reality Games. *Journal of Virtual Reality and Broadcasting 4*, 5 (Mar.). urn:nbn:de:0009-6-7939, ISSN 1860-2037.

VALENTE, L., CONCI, A., AND FEIJÓ, B. 2005. Real time game loop models for single-player computer games. In *Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment*, 89–99.

VALENTE, L., DE SOUZA, C. S., AND FEIJÓ, B. 2008. An exploratory study on non-visual mobile phone interfaces for games. In *Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems*, Sociedade Brasileira de Computação, Porto Alegre, Brazil, Brazil, IHC '08, 31–39.

VALENTE, L., DE SOUZA, C. S., AND FEIJÓ, B. 2009. Turn off the graphics: designing non-visual interfaces for mobile phone games. *J. Braz. Comp. Soc. 15*, 1, 45–58.

WATTE, J., 2005. Canonical game loop. Available at www.mindcontrol.org/~hplus/graphics/game_loop.html/ .

ZAMITH, M., MONTENEGRO, A., CLUA, E. W. G., JOSELLI, M., FEIJO, B., LEAL, R., AND VALENTE, L. 2011. An distributed architecture for mobile digital games based on cloud computing. In *X Brazilian Symposium on Computer Games and Digital Entertainment, SBGames 2011 - Computing Track*, 1–8.

ZYDA, M., THUKRAL, D., JAKATDAR, S., ENGELSMA, J., FERRANS, J., HANS, M., SHI, L., KITSON, F., AND VASUDEVAN, V. 2007. Educating the next generation of mobile game developers. *IEEE Computer Graphics and Applications 27*, 2, 96, 92–95.

ZYDA, M. J., THUKRAL, D., FERRANS, J. C., ENGELSMA, J., AND HANS, M. 2008. Enabling a voice modality in mobile games through voicexml. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, ACM, New York, NY, USA, Sandbox, 143–147.