The current state of cross-platform game development for different device types

Stefan Bruins

October 2014

Preface

This paper is written as part of my master Computer Science at the Vrije Universiteit in Amsterdam. The idea for this paper came to me after seeing the game BrowserQuest, which is a showcase project, initiated by Mozilla, to demonstrate the possibilities of HTML 5. It is a multiplayer game that runs on any platform that has a browser which supports HTML5. I immediately felt the urge to build such a cross-platform game, but I have not been able to do so yet. I decided to dive into the topic of cross-platform game development by writing this paper about the issues and current state of the field.

Abstract

Most people own many different devices, such as smartphones, tablets, laptops and desktops. The gaming experience has always been limited to one particular device but with so many devices available the gaming experience can be extended onto multiple devices. This paper will focus on games that can be played on multiple devices, where it is possible to switch from one device to the other without losing the gamestate. The main focus is on the issues and the current state of the art of cross-platform game development. The future perspective for this genre is that even though it is hard to provide equal gameplay on two inherently different platforms, it is possible to use some platforms to serve limited functionality of the game while other platforms provide the full game experience. For some games it is possible to have the full game experience on each of the platforms but this depends on the specific requirements of the game.

Contents

1 Introduction

Traditionally games were made for one specific device and operating system. With more different devices and platforms being common nowadays, game designers have started looking for ways to create games that are designed to run on different platforms. With the widespread access to networked devices the gaming experience does not have to be limited to one particular device anymore and thus can become much more pervasive. Additionally, it also allows real-life concepts to be integrated into the game, such as a user's physical location.

The concept of a cross-platform game can have two similar but subtly different interpretations. One interpretation simply defines a cross-platform game as a game that can run on more than one platform, while the other more specific interpretation has the additional requirement that the same game instance can be accessed from different platforms. In other words, the user can switch between different devices without losing the game state. This paper will treat all the problems that are present in cross-platform games of both interpretations.

Many of the issues in cross-platform game design are not specific to games but apply to software in general. There are two main difficulties in making cross-platform software. The first one is concerned with how to get software to run on different platforms. Software developed for Android cannot be executed on iOS or on Windows. In cross-platform development there are different solutions to this problem each of which find a balance between several factors including performance, cost of development, flexibility and access restrictions to the platform's available resources. The second problem is that different platforms have different capabilities. A PC has a different input method, network performance and screen size than a mobile phone or tablet. Consequently, the game must be designed with these differences in mind and adapt to the context in which it is used.

In section 2 a short introduction to the topic is given as well as a possible usage scenario for cross-platform games. Section 3 gives an overview of existing work on cross-platform games. Section 4 gives the theory of methods to produce cross-platform code. Section 5 focusses on the issues that are imposed by differences between devices and how to deal with these issues. Section 6 briefly looks at what is required to maintain the gamestate across device transitions. In section 7 an overview of a number of existing frameworks is given. Then in section 8 the focus is on a more recent development called cloud gaming. It gives an overview of the advantages, disadvantages and its relevance to cross-platform development. Section 9 investigates a piece of middleware that has been created to aid the development of cross-platform games. In section 10 a case study on the cross-platform game BrowserQuest is given. Finally the conclusion and perspective for the future is given in section 11.

2 Background

In the literature there are three terms that are often used to refer to games that can run on multiple platforms. These terms are cross-media games, cross-platform games and multi-platform games. Cross-media games refer to games that can run on different types of devices, for instance a smartphone and a desktop computer. The terms cross-platform games and multi-platform games can be used interchangeably and refer to games that can run on two or more platforms. This can be two different platforms for the same device type, for instance Windows and Linux for desktops or two different platforms for different device types, for instance Android for the smartphone and OS X for the desktop. In this paper all of these

terms are used interchangeably and refer to games that are cross-media, meaning they can run on different platforms on different device types, unless stated otherwise. Another term that is often used is pervasive gaming. In a pervasive game the game experience is extended to the real world. Cross-media gaming is a sub-genre of pervasive gaming.

A platform is the environment in which software is run and provides constraints for the software and the facilities that it can make use of. The term platform is a general term which may refer to a hardware architecture, an operating system or runtime libraries. A platform can be seen both as a constraint and as a support. A constraint in the sense that the software is limited by the platform in what it can do and a support in the sense that it provides ready-made low-level functionality. For software to run on multiple platforms it must be able to obey the constraints and make use of the provided facilities for each of those platforms.

Cross-platform game development is a relatively new research area. There are very few games that are truly cross-platform in the sense of the second interpretation presented in the introduction: games that maintain a gamestate between device transitions. The development of these games involves a spectrum of challenges that range from technical issues to user-experience issues. In this paper I categorize these issues in three groups. The first issue deals with getting software to run on multiple platforms. There are several ways of achieving this, each with its own benefits and drawbacks. The second issue is the problem of each platform having different properties. A mobile phone has a different screen size and input method than a desktop computer. It also has a very different network performance. As a consequence, a game designed for the PC can not simply be copied to a mobile phone. It may require that the game gets an entirely different user-interface for both of these platforms. In many cases it will not be feasible to run the same game on multiple heterogeneous media platforms at all. In these cases it is possible to only make a subset of the functionality available for some platforms, while there is one main platform that has all the functionality. The third issue is related to the architecture of cross-platform games. In order to maintain the gamestate the state must be accessible by each device. This implies that the gamestate must be stored server side, which affects the architecture of these games.

Figure 1 illustrates the concept of cross-media games and shows four possible scenarios that may occur when playing such a game. In the initial situation the game has two players playing on a desktop, one using ADSL to connect to the internet and the other using dial up. One player decides to stop playing on the desktop, switches to a mobile phone and proceeds playing the game where he or she left off over a GPRS internet connection. Then the second player is nearby and both players start a local game over bluetooth. Finally, other players join and everyone starts playing on the same wifi network using different devices.

These scenarios show how cross-media games are more pervasive than single media games. Accessibility of the game is not determined by a player's location anymore, it is accessible from everywhere.

3 Related work

Few studies have been done on games that can run on two entirely different device types. In [2] the focus is on cross-platform development between mobile devices. Part of that work is relevant to this paper, but there is a much higher degree of homogeneity between two different mobile platforms than between a mobile platform and a desktop platform which leads to more complexity in the latter case. The benefits of cross-media applications is

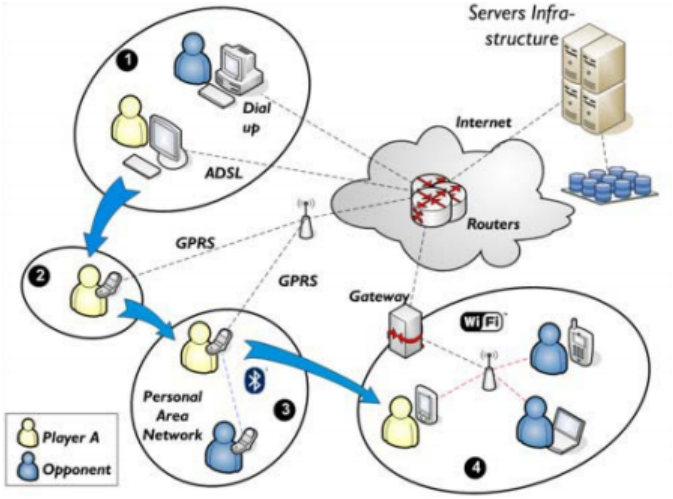

Figure 1: A possible scenario when playing cross-media games. Source: [1]

addressed in [3], specifically for educational purposes. [4] investigates which features make a good learning experience in cross-media games. Some of the elements that make a game fun or interesting imposes challenges for cross-media games, these are addressed in [5].

Only very few works are directly related to the practical side of cross-media game development between different strongly heterogeneous devices. In [1] a piece of middleware is described that supports the creation of cross-media games. This work, called the PM2G initiative, presents a service-oriented architecture that aims to support cross-media game development and execution for role playing games. The middleware of the PM2G initiative will be covered in this paper. Another middleware for cross-media game development is presented in [6], where multiple different device types are supported and the context of a player can be detected through sensors which is then associated with a certain action. This work is very brief and lacks enough detailed information to be usable. .

4 Cross-platform code

The traditional way of getting software to run on multiple platforms is porting. Porting is the process of rewriting a piece of software from scratch to make it runnable on another platform. This is time consuming and therefore imposes high development costs. The porting process is also a very error prone task. This chapter highlights four methods to produce cross-platform software. These methods are just ways of producing software that can run on multiple platforms but do not give solutions to the problem of each platform having different capabilities. The methods mentioned in [2] are cross-compilation, virtual machines, pure web application and hybrid web applications. Below is an overview of the different techniques that exist for creating 'write once, run everywhere' code.

4.1 Cross-compilation

In cross-compilation the code is written once and then compiled for the target platform. This is achieved by having a framework that offers a platform independent API in some language, usually a mainstream language such as Javascript or Java. The developer writes the software in that language using API calls from the framework. The cross-compiler will translate the written code into code that can run on a target platform. This allows developers to write the code once and compile it for each platform it needs to run on. Since the cross-compiler translates the code to native code for the target platform, the software has high performance and has access to all the capabilities of the device. The main drawback is that cross-compilers are hard to build and maintain, especially in the fragmented landscape of mobile devices.

4.2 Virtual machines

A similar approach to using cross-compilation is using virtual machines. In this method, the framework does not only provide the API but it also provides the runtime environment in which the software will be executed. The software is written in some language and uses API calls defined by a framework just like with cross-compilation. The resulting code is interpreted by the framework's runtime environment (ie. virtual machine). This virtual machine takes instructions one by one and calls an equivalent sequence of instructions offered by the target platform. Because the virtual machine translates instructions from the code to platform specific instructions it is itself platform dependent, which means that a version of the virtual machine must exist for each of the target platforms. The advantage of this technique over cross-compilation is that virtual machines are easier to maintain and extend [2]. It is, however, a little slower because the code must be interpreted at runtime by the virtual machine. An example of this method is the Java framework where software is written in the Java syntax and uses API calls provided by the framework. However, Java requires an intermediary step in which Java code is compiled to Java bytecode. The bytecode is then interpreted by the Java Runtime Environment which calls the equivalent sequence of instructions for the target platform.

4.3 Pure web applications

An entirely different solution is web applications. These are applications that run in a browser and use only capabilities offered by the browser. Web applications are written using web languages including HTML, CSS and Javascript. The browser is capable of interpreting these languages. Modern browsers try to conform to the standards for web languages defined by W3C. Because of these standards, code that is executed in one browser should give the same behaviour as code executed in another browser. These standards are the basis for cross-platform development with web applications, because they guarantee similar behavior on different platforms when using the same code.

The advantage of using web applications is that your application is accessible from any platform with a (modern) browser. Other advantages are fast deployment and instant availability. A disadvantage is that despite the standards, browsers are not always implemented equally which may result in different cross-browser behaviour. Web applications are also not very well suited for CPU intensive and visually rich applications such as games as they do not run natively but inside of a browser. Furthermore it is not always possible to emulate the native UI with standard web elements and browsers have limited or no access to

certain resources (for instance microphones or a mobile's notification system). Most web applications also rely on server side code and server side storage and thus require an active network connection.

It depends on the nature of the game or application whether these constraints are acceptable or not. The capabilities of browsers have been expanding rapidly and with the arrival of HTML5 and better support for standards, web applications are a more serious alternative for cross-platform development.

4.4 Hybrid web applications

Pure web applications have a number of drawbacks of which part is taken care of in hybrid web applications. Hybrid web applications try to combine native applications with web applications. It works by creating a native application that embeds a browser view. This browser view runs the web application but since the browser view itself is part of a native application it has access to all the device's capabilities. Upon the installation of the application, the web application can be cached locally so that no network connection is required and there is no loss in responsiveness. Any communication between the web application and the native application happens through a bridge which can either be programmed by the developer or alternatively be picked from one of the ready-made solutions.

Using hybrid web applications rather than pure web applications has the benefit of getting the access to device resources like a native application but the platform independent development of a web application. Furthermore, the application does not run within a browser anymore and can be distributed through the platform's main distribution channels such as the App store on iOS or Play store on android. However, the application may be native, but the web code still runs within a browser view which is another layer between the user and the operating system and thus has a cost in terms of performance.

4.5 Comparison

Figure 2 is a comparative table that gives an overview of the advantages and disadvantages of all four techniques.

While all of these techniques can in principle be used to make software run on multiple platforms, there are limitations that make some techniques more or less usable than others in different scenarios. We are interested in techniques that allow cross-platform development across different device types, for instance mobiles and desktops.

Cross-compilation and virtual machines could in principle be a method for cross-platform development that supports both mobile devices and desktops, but the differences between these two types of platforms are so big that the cross-compiler or run-time environment needs to be extremely complex. Besides that, there are concepts or features of one platform type that do not have an analogous equivalent on another platform type. For instance the notification system on a mobile devices does not have an equivalent on most desktop operating systems. These differences are difficult to deal with. The use of virtual machines also requires that each platform that is to be supported has that particular virtual machine available. The use of a cross-compiler or virtual machine may thus be viable for cross-platform development between similar platform types or for partial cross-compilation of specific components, but it is too complex for full cross-platform write-once, run everywhere code.

	Porting	Cross-compilation	Virtual Machines	Pure web applications	Hybrid web applications
Performance	native	native	interpreted	interpreted	interpreted
Access to native resources	yes	yes	yes	no	yes
development cost	high	low	low	low	medium
Development speed	slow	fast	fast	fast	fast/medium
User- interaction	native	native	native	only standard web elements available for UI	native
Distribution channel (for mobile devices)	platform store	platform store	platform store	webbrowser	platform store
complexity of method	simple	complex (the cross-compiler itself)	complex (the runtime environment)	simple	medium (Bridge between web app and OS is needed)
Bounded to similar device types (mobiles or desktops)	no	yes, desktop-mobile conversion is difficult	yes, desktop-mobile conversion is difficult	no	no
Imposed limitations on target platforms	none	none	requires runtime environment	requires webbrowser with good standard support	requires webbrowser engine with good standard support

Figure 2: A comparison of methods to create cross-platform code.

Web apps on the other hand are inherently cross-platform as long as the browsers adheres to the web standards and the developers use no browser specific code. With hybrid applications you can have full access to a device's resources. There is also explicit support to create user interfaces for different screen sizes with CSS. However, the performance cost of the additional layer between the user and the operating system makes web applications much less of a viable option for graphic intensive applications, which many games can be categorized in.

The "write-once, run everywhere" philosophy does not seem to be feasible in all situations at the time of writing. The discussed methods can be used to develop parts of an application or possibly even the entire application if the limitations are acceptable.

5 Platform heterogeneity

Apart from producing code for multiple platforms there is the issue of dealing with the differences between devices and platforms. Different platforms have different capabilities and properties, so simply porting a game to another platform is unlikely to be sufficient. This chapter covers the most important differences that exist between different platforms, especially between the mobile platform and desktop computers. These include technical differences but also differences in the way that users interact with different types of devices.

5.1 Screen size

The screen size on mobile phones is considerably smaller and uses different aspect ratios than a typical desktop computer monitor. Even between mobile platforms different resolutions and aspect ratios are used. The simplest case is when a game needs to be supported by two mobile platforms with different resolutions but the same aspect ratio. In that case the game can just increase or decrease the size of the graphics depending on the screen resolution while maintaining the aspect ratio. The problem with upscaling images is the loss in image quality and the problem with downscaling is the loss in performance that would not be necessary when using images of the right resolution in the first place. To deal with that, images can be stored in different resolutions and the game uses the right images depending on the device resolution.

A more difficult scenario is when not only the resolution differs but also the aspect ratio. Stretching the game scene and graphics to fit the screen is usually avoided as it distorts the images and with that the user experience. Instead a number of scaling techniques exist. One method is to decide what aspect ratio the logical scene of your game uses and then scale up the scene with the same X and Y to make it use the entire width or height of the screen. Consequently devices with a different aspect ratio than that of your logical scene, will have an empty border on the left and right or the top and bottom of the screen. Another method is to scale in the other direction. In other words, instead of leaving part of the screen unused, you scale the content so that the entire screen is used. By doing this some content will fall outside of the screen borders on devices with a different aspect ratio than the logical scene. This is a problem for games that require a player to see the entire logical scene at all times.

These scaling methods work well between different smartphone resolutions, but when used to scale between desktops and smartphones it may lead to images that are too small to view comfortably on a smartphone. The difference between the size of desktop monitors and smartphone displays is huge. One way to deal with this is to make smartphone users see a smaller part of the logical scene than desktop users get to see. Again this is only possible for games that do not require the full scene to be visible for all players at all times. An alternative is to allow smartphone users to zoom in on the scene, which also is not suited for all games and may give one platform an advantage over players from the other platform.

In some cases it will just not be possible to display the exact same game on all screen sizes comfortably. A possible solution is to give an entirely different interface for users of a specific platform. For instance, smartphone users may be served a textual interface instead of a graphical interface. However, this requires extra work and again only works for certain types of games.

5.2 User input methods

Different devices use different input methods. The richness of input methods differ immensely between different devices. The desktop offers a mouse and keyboard with many buttons, while many smartphones only offer a touchscreen and gaming consoles only offer a controller. These input methods are so fundamentally different, that it strongly affects the design of a cross-platform game. The game designer must find a balance between development complexity and unused rich input capabilities. If all functionality is available on all platforms equally, than the development complexity is low. However, the provided functionality is then based on the device with the least rich input capabilities and thus the rich input features on other platforms are left unused. On the other hand if different functionality is offered on different platforms, than the development complexity is higher but no rich input features of a platform are left unused, which increases the user experience.

Dealing with differences in input methods requires platform specific development. This development can be simplified by letting the game deal with input on a higher level. The game should not process mouse clicks or touchscreen swipes directly but instead it should process higher level input descriptions. A separate component can than deal with translating device specific input to higher level device independent input. This is something that is not only done in cross-platform games but also in standard game design.

5.3 Network performance

Another important difference between different devices is the network performance they have. Desktops are stationary devices that usually have a stable non-changing network connection. Smartphones are mobile devices that constantly switch between different network accesspoints and have fluctuating signal strengths. When developing a game that must work on both of these platforms than it must take into account that one of them has an unreliable network connection. Additionally, mobile users have less available bandwidth and often a data transfer limit per month.

Again a decision must be made whether all platforms get equal functionality or not. Equal functionality on all platforms means that the functionality will be tuned to the platform with the most limited network capabilities. Offering different functionality based on the network capabilities of a platform increases development complexity.

5.4 Device usage

Besides differences in hardware, there also differences in how the devices are being used and in what context. There is the element of concentration. As opposed to desktop users, smartphone users are often in an environment with many sources of distraction, for instance in a train or a family filled living room. They often also lack the ability to play with sound due to their environment. These are elements that a game designer can take into account. For instance, by not making sound a key element in the game. By taking these elements into account, the game is better suited for some platforms but at the cost of having a limited experience on other platforms.

5.5 Platform hardware and services

Each platform offers its own set of services. Mobiles have a notification system and a GPS sensor that desktop computers mostly lack. Services of one platform that have no equivalent

service on other platforms, such as a notification system and GPS, cannot be used as a key concept in a cross-platform game. However, they can serve as additional rich features to enhance the user experience on specific platforms that support it.

6 Handling device transitions

This section briefly describes what is required in terms of architecture to make a cross-platform game in which the gamestate is maintained whenever the player switches to another device.

Since the gamestate must be accessible on every device, it must be stored at the server so that all the devices are able to load it. This can be done by placing the entire game simulation on the server. The client sends its user input to the server and the server responds with the resulting gamestate data relevant to that player. This already happens partly in many online multiplayer games to prevent cheating. The gamestate data is stored in a persistent way, for instance a database. Depending on how fluent the transition between devices is required to be, the gamestate can be stored in great detail (including the exact position of the camera) or in a descriptive way (only the most important gamestate information). Whenever a device transition happens, the client can load the last known gamestate from the server so that it can continue from where it left of.

Authentication must be used so that the server knows which gamedata to load. It also works as a prevention mechanism so that players can't accidentally have two sessions open on two different devices that interfere with each other. If a login is sent to the server, all login sessions from the same user are invalidated and a new session is opened with the device that just logged in.

7 Existing frameworks

Most of the frameworks focus on cross-platform development between different mobile devices, but not between desktops, mobiles and consoles. There are quite some frameworks for mobile cross-platform development. In [2], three popular open source frameworks are discussed. These are frameworks for cross-platform mobile software development in general, not specifically aimed at games.

7.1 Mobile-only frameworks

The first is Rhodes, which provides a virtual machine (or run-time environment) that is wrapped around an application. The virtual machine is ported to different platforms and translates the instructions of the application to platform specific commands. It uses Ruby as language for the business logic and HTML, CSS and Javascript to construct the UI. The strength of Rhodes is that it supports many mobile platforms, namely: iOS, Android, Blackberry, Windows Mobile 6.5 and Symbian. The use of Ruby also helps to structure business logic using the MVC and Object Relational Mapper design patterns. Its drawbacks are that when changing the HTML/Javascript code, a complete rebuild of the application must be made. Furthermore, the Rhodes doesn't produce the source code of the final application but only a native package.

PhoneGap is another cross-platform toolbox for mobiles. It only uses HTML, CSS and Javascript. It is popular because of its flexibility and ease of use. This framework is used to

create hybrid web applications. It is especially useful for moving existing web applications to a mobile environment It does not provide much to make the application behave and feel like a native application. Fortunately there are many libraries out there that specialize on this, which can be used in conjunction with PhoneGap. PhoneGap does not force users to structure their application in a certain way, which is good for experienced developers but can lead to poorly structured applications by novices. It has the widest range of supported platforms: iOS, Android, Blackberry, Windows Mobile 6.5, Symbian and Palm.

Appcelerator Titanium uses a cross-compilation technique. The code is written in Javascript and compiled to a native application for a target platform. Because the output is a native application, the application is quick and feels fluent. It has good documentation, online resources and forum community, which makes it easy to get started. Even though Appcelerator Titanium uses Javascript it does not use a browser engine to render the user interface. Its API provides abstractions for most UI elements that are converted to native UI elements in the cross-compilation process. This leads to a richer user experience than the browser based technique. The supported platforms are iOS, Android, Blackberry and Tizen, allthough blackberry is still in beta.

7.2 Mobile-desktop frameworks

There do exist game development frameworks that support the creation of games that an run on multiple device types including desktops and smartphones. However, unlike the mobile-only cross-platform frameworks, those frameworks often require additional work to get a game to work on both smartphones and desktops. This is due to the inherent difference between those platforms as described in section 5. Smart programming can handle some of the issues, but sometimes it may not be possible to get a desktop game to run on a smartphone at all or extensive changes to the input or gameplay are required. There are quite some frameworks available. Since these engines differ mostly on general game development issues and not as much on cross-platform capabilities, I will only discuss Unity and Unreal briefly. Frameworks that I will not discuss include CryEngine, Game Maker, Delta Engine and Mono.

Unity is a collection of tools and services to support game development in general but also provides cross-platform support. It allows you to develop the game using the unity framework and then compile to specific platforms. Unity includes a game engine and an Integrated Development Environment (IDE). It has documentation, a large community and many tutorials available. The platform is considered to be easy to use and offers many features. It provides ways of collision detection without using math. There is a free version and a paid version that provides more features. Using the engine requires you to accept their policy, which among other things means that if a commercial game created with unity produces high profits than you will be required to buy the paid version. It supports game development for many platforms, including windows, Linux, Android and iOS.

The Unreal engine is another framework for creating games for a range of platforms. It also offers many example projects. good documentation and an active supporting community. It is a powerful engine that supports graphic intense games. It requires a license of, at the time of writing, $20 per month. Unlike unity, unreal provides full access to the source code. The User Interface is a bit more complex than that of Unity.

While these frameworks offer cross-platform game support. They mainly provide support by cross-compiling to different platforms. There is little support for dealing with platform differences. Later in this paper, in section 9, a piece of middleware is discussed that is

aimed at cross-media role playing game development. This framework has only been used in one demo project and is limited to role playing games. This middleware offers support for dealing with the differences between platforms, but does not help with the cross-compilation of code to different platforms.

7.3 Web-only frameworks

Besides the mobile-only frameworks and the game engines that use cross-compilation there are also frameworks that are specifically aimed at game design in the form of a pure web application. One of these engines is Quintus, which is a HTML5 game development engine. Quintus is lightweight and uses a Javascript-friendly syntax. It can be used to create cross-platform games that can run in any HTML5 enabled browser. Another interesting option is Facebook Canvas, which is a framework for developing games that run within the Facebook website. It allows game developers to integrate different aspects of Facebook, such as the news feed and notifications. It supports web applications for laptops and desktops and supports the creation of games that take up the full width and height of the screen by using "Fluid Canvas". Besides the framework itself, it also provides the server environment for the game to run on and a popular social media platform to help create a large user-base.

8 Cloud Gaming

Earlier a number of techniques were discussed for developing cross-platform code. These techniques allow a programmer to write code once, rather than for every platform individually. An entirely different approach that also minimizes the amount of platform specific code that needs to be written is cloud gaming. In cloud gaming the game logic is processed on the server. The server sends to each player a stream of video frames, which is specific to that player and based on the current state of the game. The frames form the actual view of the game for that player. The game players run a very thin client which does little more than sending the user input to the server and receiving and display the resulting stream of video frames. The bulk of the game code is written only once and will only run on the server. The game clients on the other hand will need to be platform specific. The sole purpose of cloud gaming is not to aid development of cross-platform games but it is a convenient side effect and therefore this paper will discuss the concept of cloud gaming. Specifically the architecture, advantages, issues and how it supports cross-platform game development.

8.1 Advantages

While cloud gaming does have its own set of implications it also offers a number of advantages. Most of the game in a cloud gaming setup runs on the server. This means that the server does more work and the client less work than in a classical game. Modern games can have a size of dozens of gigabytes but with cloud gaming the client does not need to store the entire game on hard disk. Besides saving on hard disk space, cloud computing also relaxes the pressure that the game exercises on the CPU, memory and the graphics card. Since the input processing. game logic calculations and the rendering of the frames is done server side, the client's hardware is not bothered with these tasks anymore. This all comes at the prices of having a much higher usage of network bandwidth, since rather than sending small pieces of information it sends the entire frames instead. Note that the reduction of hardware requirements for clients has huge implications for game developers as

clients do not need to own expensive, powerful gaming systems anymore. This means that game developers are now mostly limited by the processing power of the servers on which they host the game, rather than by that of their target audience's clients.

Another advantage that most game players will appreciate is that many of the cheats that work for classical games are not possible in cloud games anymore. A taxonomy of eleven different cheating methods is given in [7]. Even though only one of these cheating methods is mitigated by cloud gaming, namely: cheating by modifying game software, it is the cheating technique that has the most direct impact on the game and is very prevalent in online gaming. This method of cheating abuses the fact that anything that runs on the client can be tampered with. For instance, the game can be disassembled and edited to give the player some kind of edge over the other players. In cloud gaming this technique is not as effective anymore since all of the game logic is processed on the server and the client does not need to know anything about the state of the game. All cheats that rely on game state information or on affecting game settings that should be immutable, do not work, because the client does not have access to this information and does not render its own video stream.

Additionally, in [8], the current CEO Brian Farrell of the game development company THQ, mentions that in cloud gaming it is significantly easier to add server capacity or shutdown under-utilized servers, meaning that gamers will get reliable service and publishers only pay for what they use.

Furthermore, with cloud based games the spreading of illegal copies of the game is impossible as there is no client code to spread.

8.2 Disadvantages

The toughest issue to overcome for cloud gaming is arguably related to internet connections. The issue is two-fold: bandwidth issues and latency issues. The streaming of video frames to clients requires much more data to be sent across the wire than with traditional games. Bandwidth of modern internet connections is often enough to cope with these high bandwidth requirements, but since many providers dictate a bandwidth cap, cloud gaming is often not feasible. With bandwidth still increasing rapidly, this is arguably not an unsolvable problem in the near future.

Latency is the time it takes for a packet to get from one point to another. As opposed to bandwidth, the latency improves rather slowly. There are four types of delays that cause latency [9], but the most limiting one is the propagation delay. This is the delay induced by the distance that data needs to travel which has an absolute bound imposed by the speed of light. As a result, the latency between the game player and the cloud server will never fully disappear. Placing additional servers to reduce latency is a solution for single player games, but this is not a solution for multiplayer games between players located far away from each other. In multiplayer games data will always need to travel to the server and then to the other players, either in the form of a stream of video frames or packets containing game information. London and New York are approximately 5600 km apart, which, given the speed of light, means there is a minimal round-trip-time of 56ms. A maximum network latency of 80ms is often considered to be required to have an unimpaired game, but it depends on the precision and deadline requirements of the specific game [10] and this value may be lower for the competitive gaming scene. Furthermore, this value has been determined for a classical game setup where the user input delay is not affected by the latency. In cloud gaming, the user input delay is affected by latency, which may have a severe impact on user experience. Latency mitigation techniques may decrease the effects

of latency on the game play, such as automatically correcting the user input based on the latency or placing additional servers with the only purpose of decreasing input delays so that a player's own input will be processed sooner based on the information that is available at that time. While latency requirements are tough to accommodate for, often games do not have strict latency requirements.

Another disadvantage of cloud gaming is that no game modifications can be developed by the community. Also, in older games, communities may keep the game active even after official support has been dropped. In cloud gaming this is not possible as the game only exists on the servers of the game developer and once these are taken down the game becomes inaccessible.

8.3 Architecture

Most of the cloud gaming providers use proprietary software to offer their cloud gaming service. More recently, an open source cloud gaming system has entered the market. To give an impression of the tasks that a cloud gaming system performs, this section will focus on the first open source cloud gaming system, named "GamingAnywhere" [11]. In the setup of GamingAnywhere there are clients, servers and portal servers (see figure 3). The clients are run on the machines of the game players. The client initially logs into a portal server and receives a list of available games. After selecting a game the portal server will select an available gameserver, launch the game on the server if that hasn't been done yet and then return a URL for that game to the client. The portal server has a RESTful interface that can be interacted with through the regular HTTP and HTTPS protocols. Upon receival of the URL the client can use the URL to connect to the game server.

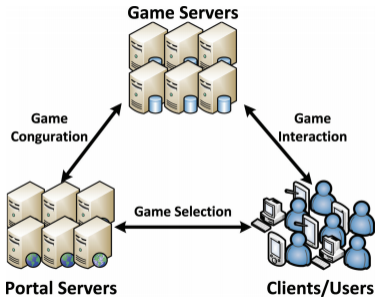

Figure 3: The main elements of the open source cloud gaming system: "GamingAnywhere" [11].

Figure 4 shows the client-server interaction. Once the client and game server are connected, the server streams the video frames to the client while the client captures the user's input and sends it to the server. The games that run on the server are just regular games, but they have an agent running alongside of them. This agent can have the form of a standalone process or a shared object or DLL injected in the game. The agent has two tasks. The first one is to capture the audio/video frames from the game that is running on the server, then encode those frames and send them to the client similar to streaming a video. The second task of the agent is to have the agent perform the user interaction with the

game. The agent does so based on the user input that the client sends to the server. The server replays the user input as if it is the player himself.

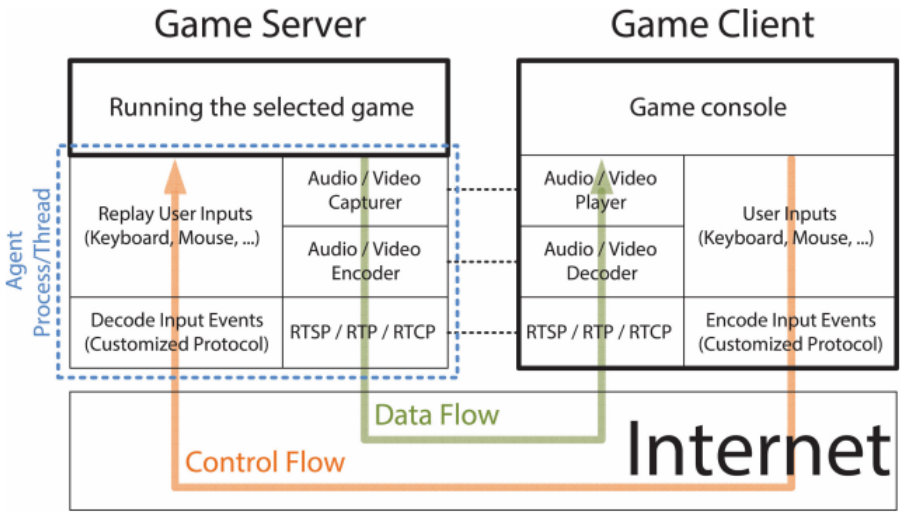

Figure 4: The client and server architecture of the open source cloud gaming system: "GamingAnywhere"[11].

The client itself contains a keyboard and a mouse logger, as well as a RTSP/RTP supporting multimedia player. The media player receives the encoded video frames sent using the Real Time Streaming Protocol, it then decodes and displays them to the user. The keyboard and mouse logger capture the input events. These events are then encoded and transmitted to the server. Upon receipt of the encoded input events, the server decodes them and lets the agent apply them to the game, which in turn will automatically cause the video and audio stream to be based on the current state of the game.

8.4 Relevance for cross-platform development

As mentioned earlier cloud gaming inherently supports cross platform development. The game code is only present on the server and thus any client that supports some form of media streaming (RTSP in the GamingAnywhere system) can make use of the game. The process to make games available depends on the cloud gaming system that is used. In the GamingAnywhere system, the only requirement is to hook a shared object or dll into the game which can intercept video frames and interact with the game by feeding it user input. Game developers do not have to deal with intricate ways of making their game runnable by a cloud gaming system. This paper makes the distinction between a game that can run cross-platform and a game that adapts to multiple platforms. The former is automatically the case in cloud gaming, but the latter is not and requires changes to the actual game which would adapt to the features of the target device. For instance by changing the user interface for mobile users. This is something that needs to be done by the game developers. The

game would have to decide how to deal with different screen sizes and hardware features and render the game differently based on the features of that specific device. It may be difficult to adapt a cloud game to different platforms in a way that gives a native experience, because the game runs only the server and therefore does not have access to the resources and input methods of the other platforms. To summarize, cloud gaming makes game development much easier because it can stream the game to any device or platform as long as the cloud gaming system can deal with the input of that platform and the client supports the type of media streaming used by the system. Specific adaptations to a target platform still need to be programmed by game developers but at least the code needs to be written only once.

9 Middleware

With so little publicly available research on cross-platform games, there is also a lack of available middleware. To the best of my knowledge, there is no framework for cross-media game development that has been applied in practice, with the exception of the PM2G initiative [1] which created a middleware layer that has only been used for their own demo project. The PM2G initiative is a research effort which stands for "pervasive multiplayer multiplatform game initiative". It introduces a middleware layer which facilitates crossmedia game development by offering a set of services. While this initiative was aimed at role playing games, the general idea may be extensible to other genres. This chapter is a mini case study on the PM2G initiative.

9.1 Concept of the PM2G middleware

The middleware specifies a number of services that aid the development of cross-platform role playing games. The middleware services provide functionality that can be reused in any other role playing game project. Four concepts are introduced: perception, context, interaction and mini-world.

The *perception* of a player is its area of interest. It depends not only on the location of the player and game rules but also on the device type of the user. User's with small screens may have a different area of interest then users with big screens. The *context* consists of relevant information about a player. This includes elements such as connection type. device type and preferences. This context is important because a player's perception is based on it. The *interaction* concept offers a high level way to describe a user's interaction with the game world. Rather than describing an interaction by a mouse click, a touch screen swipe or a keyboard press, it is translated to a higher level representation of that user interaction. Finally the middleware has the concept of a *mini-world*, where players can play over a local connection outside of the global game-world. Whatever happens in a mini-world is at some point integrated with the game state of the global world.

9.2 PM2G middleware architecture

The architecture, shown in figure 5, is a client-server architecture, where the game simulation and PM2G services are handled on the server. The architecture is divided in three layers. Below is a brief explanation of the architecture to give an idea of how the middleware works, for a more detailed description see [1].

In the bottom layer there are two components. The first component is called the "PM2G simulation framework", which manages the game simulation. The other component, called

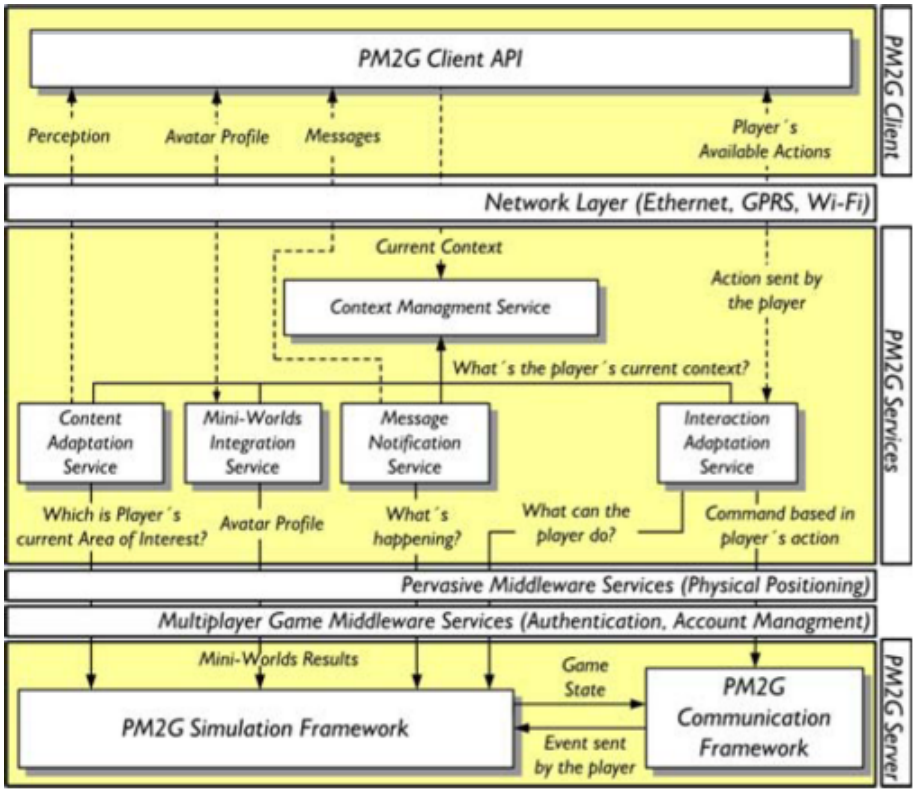

Figure 5: PM2G middleware architecture. Source: [1]

"PM2G communication framework", handles the communication between the game clients and the game simulation. The communication framework receives events from the game clients and sends them to the simulation framework, which updates the game state based on that event. The simulation framework contains the entire game state including player statistics and object positions. The relevant part of the game state can be sent to a player's game client via the communication framework. Some services of the PM2G architecture may also request the gamestate from the simulation framework.

The middle layer contains the five PM2G services. The Context Management Service holds the contextual information on players including the device being used, the personal preferences and possibly the geographical location. Other services can ask the context management service about the context of any player. The content adaptation service is used to provide the player specific game state information based on the context of that player. In other words, it uses part of the game state that is of interest to the player and transforms it into a format that is most suitable for the player's device and preferences. The transformation includes two types of processing: logical procedures and presentation procedures. The first are procedures that modify which game state information is relevant for a player's context. The second prepares the data that is sent to the client. For instance, one of the logical procedures transforms three dimensional coordinates to two dimensional coordinates. Mobile phones that do not have enough processing power for three dimensional scenes can offer a two dimensional representation of the world. The interaction adaptation

services facilitates the execution of tasks for mobile users. Based on the game state it determines the available user actions which are sent to the player. After choosing which action to perform the service sends the action to the communication framework as an event, which in turn signals the game simulation to update the virtual world. The interaction adaptation service is only used for mobile users and adapts the gameview for mobile devices. The message notification service listens for events in the game simulation that are relevant for each player. When such an event occurs that player is notified in one way or another based on their current context. Finally there is the mini-world integration service which integrates the results from a mini-world into the virtual world. All of these services are build on top of existing middleware layers related to pervasive computing and multiplayer games.

The upper layer represents the game clients of the players. It has a specific API which can be used to communicate with. The client interacts with the middleware through an authentication service. Whenever the player starts playing it needs to create a session that requires authentication. This session setup happens every time the user switches from one device to another. This way the PM2G middleware knows that the player's device has changed and that any open session with another device can be closed.

9.3 Evaluation of the PM2G middleware

The middleware service has been tested using a simple game. They have evaluated the game that uses the middleware by doing experiments on the performance of the services and the amount of data exchanged between the client and the server. Additionally, they have aimed at discovering how much the PM2G API really helps developers in developing cross-platform games.

The performance of the middleware services was concluded to be sufficient. The performance is better for clients that use a PC than for clients that use a mobile. This has to do with the additional adaptation that needs to take place for mobile users. The amount of data that is transferred between client and server is also considered to be sufficient. The effectiveness of the PM2G API for developing cross-media games is evaluated by their creators by looking at how many of the classes in the source code are specific to that game. Another metric they used to measure the effectiveness of the PM2G initiative is by counting the portion of the source code that was part of the middleware. While the results of their evaluation was positive it only says something about how much of the code is reusable. It does not indicate whether using the middleware actually decreases development time nor does it show how much limitations are imposed by using the middleware. While the middleware has been used to build a game with, which has proven to be a true cross-platform game, the evaluation method of the middleware seems unreliable. More research is needed to show the limitations of the PM2G framework and whether it can be extended upon to work for other types of games.

10 Case study: BrowserQuest

BrowserQuest is a web based game developed by Little Workshop at the request of the Mozilla Foundation. The Mozilla Foundation is involved in promoting openness and innovation on the web and has initiated the BrowserQuest game development as a showcase project for what is possible on the web platform using HTML5. BrowserQuest is a cross

platform game that runs in the browser. Even though it uses the narrow definition of a cross-platform game, where the game state is not maintained across device transitions, it uses some interesting technologies and shows how cross-platform games are feasible using standard web elements.

10.1 The technology

The game consists of one big game-world where people from all over the world can play in. The player controls its own character and can see the movements and actions of other players. The goal is to explore the map, defeat artificial opponents and collect better weapons. It is not a very rich game in terms of game play and is aimed to be a showcase project rather than an actual full fledged game.

The player's browser runs the web client, which is written in Javascript and uses a canvas for the rendering engine. The canvas element was introduced in HTML5 and allows for scriptable rendering of 2D shapes and images. The canvas API is accessible through javascript and offers methods for controlling and drawing on the canvas. The canvas is a low level model that updates a bitmap. More complex engines can be built on top of this low level canvas API, for instance engines that support scene graphs to create hierarchies of objects.

Another HTML5 capability that is used is local storage. Traditionally local storage was achieved using cookies. Cookies are name-value pairs that are stored on the client and transmitted to the server upon a server request. Cookies are limited in size and waste bandwidth by including all the cookie data on every request, regardless of whether it is used. Besides cookies there have been a number attempts at providing local storage in web-applications, but these were often browser specific or relied on plugins. With HTML5 the functionality of local storage is natively build in the browsers and can be used by any browser that is up to date on the web standards. This local storage is not transmitted upon a server request and has the same API and limitations on every browser. In BrowserQuest, local storage is used to store the player information such as name, achievements and weapons. To play audio the game uses the HTML5 audio API. This API offers methods to start audio and interact with it. It also makes it possible to ask the browser whether it will be able to play a certain audio format. BrowserQuest handles all in-game sounds and music using HTML5 audio.

Figure 6: In-game screenshot of the game BrowserQuest

The web has been built around the request/response paradigm of the HTTP protocol. Initially webpages could only do full page reloads and not partial page updates. Partial reloads are possible using AJAX, but all the communication is still initiated by the client. Consequently, clients are required to periodically poll whether the server has new data available. Techniques exist to send data to the client at the moment it becomes available on the server, such as long polling. However, all of these techniques carry significant overhead of HTTP but latency sensitive applications such as games cannot accept this amount of overhead. Websockets address this problem with the ability to create persistent TCP connections between the client and the server. Since websockets were designed to work in the existing infrastructure of the web, a connection setup is performed through a regular HTTP connection request which is then upgraded to a websockets connection. Websockets must be supported both by the client as well as the server. An established websocket connection provides a real-time full duplex channel over which both client and server can send data at any moment in time. Websockets make real-time applications feasible by offering a real-time connection as well as handling proxy servers and firewalls. The web engine of BrowserQuest is build using websockets.

Since BrowserQuest is available for any websockets enabled browser, it must adapt to the properties of different types of devices with high degrees of heterogeneity in terms of screen sizes. The game developers achieved this in two ways: rendering optimizations and CSS media queries. The exact rendering optimizations they have used is unspecified but it may include rendering a smaller portion of the map depending on the size of the screen. HTML4 introduced media types, which made it possible to apply a stylesheet only for a specific media type, such as a screen or a projector. Media queries are an extension of this concept and allow more specific queries on the properties of the currently used device. For instance, a media query can query the exact screen size of the device and apply a style only if it falls within the given range of screen sizes specified. By using media queries the game-world can automatically be rendered based on the current screen size.

10.2 Discussion

BrowserQuest is a good example of how pure web applications can be used to create cross-platform games. All those new web technologies (canvas, local storage, audio, websockets and media queries) make it feasible to create cross-platform games that can run within a browser. Even though this game does not maintain gamestate across device transitions, this is the result of a design choice rather than a fundamental limitation of web applications.

The game is programmed once and with little effort the game is playable on all platforms that have a websockets enabled browser. However, the game is relatively simple and does not make use of the available input methods of the different platforms. There is a browser window bordering the game which, especially in mobile browsers, breaks the user immersion as it covers a significant part of the screen.

11 Conclusion and future perspective

This paper has given an overview of the issues of cross-platform development and the current state of the art. n conceptual overview is given of cross-platform code production techniques and the handling of platform heterogeneity. For a more practical view a number of existing frameworks have been discussed that can support the development process. Cloud gaming

and the PM2G middleware were discussed to show some recent developments that can aid cross-platform game development.

Ideally creating a cross-platform game would only involve writing code once without considering any platform specific issues. Current existing frameworks come close to this and allow a large part of the development to be done in a platform independent way. The biggest challenges that remain are dealing with differences between platform like input methods, screen sizes, network performance and hardware limitations. Additionally, providing an equal and fair experience on many different platforms is something that is part of the game design.

So what is the future perspective for cross-platform games? Will all games be cross-platform in the future? Probably not, but why? The problem is not that it isn't possible to make any game cross-platform, the problem is that it comes with a cost. When supporting smartphones the game needs to be playable with the hardware limitations of a smartphone, leaving all the rich and powerful capabilities of the desktop unused. Furthermore, some games require big screens to offer a good experience and getting the same kind of experience on two entirely different devices is hard to achieve. It is also difficult to make the game equally difficult on different platforms, which may give user of some platforms advantages over users of other platforms. This is especially the case in competitive online multiplayer games.

Simple games or games that are particularly well suited to work on different devices can already be developed with currently existing frameworks, although the development process is still more complicated than that of regular games. For more complex games or games that are not well suited to be cross-platform, the development is difficult and often not possible without extensively simplifying or compromising in gameplay. For these types of games the future is more in having one platform be an extension of another. This can be a mobile application which offers a subset of the functionality or extra functionality not available on the desktop version. An example is a role playing game where you can play the full game on the desktop or console but can only give high level strategic commands and access the shop and online player statistics on the smartphone version.

A promising development is cloud computing as discussed in section 8. Although the heavy network requirements are still a problem today, this is is likely to change in the future. Latency, however, is limited to the speed of light and thus can not improve indefinitely. Latency mitigation techniques and additional server placement to improve input response times may be able to eliminate latency problems. The many advantages of cloud gaming make it a good option for the future.

References

[1] Fernando Trinta, Davi Pedrosa, Carlos Ferraz, and Geber Ramalho. Evaluating a middleware for crossmedia games. *Comput. Entertain.*, 6(3):40:1–40:19, November 2008. This article presents the PM2G initiative, which is a service oriented architecture that aims to support cross-media game development. It is one of the few documented cross-media middleware services available that provide content and interaction adaptation. It is mainly aimed at role playing games.

[2] Gustavo Hartmann, Geoff Stead, and Asi DeGani. Cross-platform mobile development. 2011. This work looks at cross-platform development for mobile platforms. It gives an

overview of conceptual techniques for cross-platform development and it discusses a number of existing frameworks.

[3] Shalom M. Fisch. Cross-platform learning: On the nature of children's learning from multiple media platforms. *New Directions for Child and Adolescent Development*, 2013(139):59–70, 2013. This work discusses the benefits of cross-platform learning. This work aims at educational media projects in general and not specifically at games.

[4] Robert Fohlin. A cross-media game environment for learning, 2010. This work identifies what features play an important role in cross-media educational games.

[5] Kalle Jegers. Elaborating eight elements of fun: Supporting design of pervasive player enjoyment. *Comput. Entertain.*, 7(2):25:1–25:22, June 2009. This work presents the validation of the Pervasive GameFlow (PGF) model, which describe what elements are important for the fun and enjoyment of a player when playing a game. It also presents the comparative importance of each of the eight elements from the PGF model. The paper also presents how some of the important elements impose challenges and possibilities for cross-media games.

[6] JungHyun Han, Hoh Peter In, and Jong-Sik Woo. Towards situation-aware cross-platform ubi-game development. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, APSEC '04, pages 734–735, Washington, DC, USA, 2004. IEEE Computer Society. This work presents a middleware layer for the development of cross-media games that work on a number of device types. The article is too concise to be of much use, but is included as reference for the sake of completeness.

[7] Knut Håkon T Mørch. Cheating in online games–threats and solutions. *Publication No: DART/01/03. Norwegian Computing Center/Applied Research and Development*, 2003. This article gives a categorization of cheating techniques in online games. It also discusses the prevention techniques for those cheating techniques. It is used as a reference to show which type of cheats can be avoided by cloud gaming.

[8] John Gaudiosi. Future of cloud gaming: Industry leaders' thoughts. *FC Business intelligence*, 2011. This article gives an overview of the views of a number of industry leaders' thoughts on cloud gaming and its future. All of those leaders give their idea on what they think of cloud gaming, its advantages and challenges.

[9] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games*, page 2. IEEE Press, 2012. This article looks at the latency requirements of online games and whether they can be fulfilled by the existing network architecture. It further looks at how placing servers near the end-user can help to improve the situation.

[10] Mark Claypool and Kajal Claypool. Latency can kill: Precision and deadline in online games. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, pages 215–222, New York, NY, USA, 2010. ACM. This work looks at how latency affects online gameplay. It presents a way to determine how much a game is affected by latency by looking at its precision and deadline requirements (how accurate an action must be and how big the deadline is within which the action must be completed).

[11] Chun-Ying Huang, Kuan-Ta Chen, De-Yu Chen, Hwai-Jung Hsu, and Cheng-Hsin Hsu. Gaminganywhere: The first open source cloud gaming system. *ACM Trans. Multimedia Comput. Commun. Appl.*, 10(1s):10:1–10:25, January 2014. This work presents the first open-source cloud gaming system. It describes the architecture and it presents experiments on the performance.

12 Appendix A: online resources

- Scaling methods: `http://v-play.net/doc/vplay-different-screen-sizes/`
- Dealing with platform differences: `http://docs.unity3d.com/Manual/HOWTO-PortingBetweenPlatforms.html`
- Quintus (a HTML5 game engine): `http://www.html5quintus.com/`
- Facebook Canvas (for cross platform game development): `https://developers.facebook.com/docs/games/canvas`
- Browser support of HTML5 features: `http://mobilehtml5.org/`