Bachelor's Thesis

Information Technology

Software Business

2015

Eetu Pitkänen

DEVELOPMENT OF A FINITE RUNNER MOBILE GAME

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Software Business

2015 | 41

Tiina Ferm

Eetu Pitkänen

DEVELOPMENT OF A FINITE RUNNER MOBILE GAME

The purpose of this thesis was to examine the process of developing a finite runner game. The game was developed for an indie game development company called FakeFish to answer their need of a product that can be easily showcased and used as a reference point of what the company is capable of in a limited amount of time.

The theoretical section of the thesis focused on the game's concept, the endless runner genre's characteristics and history, tools used, potential publishing platforms and the challenges of publishing in the segregated markets of the east and west. The empirical section of the thesis consisted of the game's main programmed features, ad-based monetization, the interconnectivity of the level design and difficulty as well as building to a platform.

Unity was chosen as the development platform due to it having low royalty fees, a big developer community and FakeFish's previous experience with the Unity game engine. The game's publishing in the future will happen in the western world only as publishing in Asia is a complicated and expensive process that FakeFish is not yet ready to undergo. The publishing channel for the game is going to be Google Play and the operating system Android as these match the game's planned monetization model and performance requirements the best.

The product that was developed for this project in the end was a good-looking game with smooth gameplay. However the uniqueness of the game was deemed too lacking for publishing and the game will instead be used as a foundation for a new version. The development of the new version will emphasize on the main problems of the now created game and aim to be unique when it comes to gameplay and overall feel.

KEYWORDS:

Game development, publishing, Unity, Unity3D, game programming, C#, FakeFish, indie

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Ohjelmistoliiketoiminta

2015 | 41

Tiina Ferm

Eetu Pitkänen

ÄÄRELLISEN JUOKSUPELIN KEHITYS MOBIILIALUSTOILLE

Opinnäytetyön tavoitteena oli tutkia mobiililaitteille kehitettävän äärellisen juoksupelin kehitystyötä. Peli kehitettiin vastaamaan FakeFish-nimisen pelialan yrityksen tarpeeseen tuotteesta, jota olisi helppo esitellä muille ja antaisi osviittaa myös siitä mihin yritys pystyy rajallisessa ajassa.

Teoriaosuus keskittyi pelikonseptiin, peligenren ominaisuuksiin ja historiaan, käytettyihin työkaluihin, potentiaalisiin julkaisualustoihin sekä pelien julkaisun eroihin länsimaiden ja Aasian markkinoiden välillä. Projektiosuus koostui pelinkehitystä ohjelmoijan näkökulmasta, jossa pelin ohjelmointiin liittyvien ominaisuuksien lisäksi käytiin läpi suunniteltua mainospohjaista monetisaatiota, pelin tasosuunnittelun ja vaikeustason yhteyttä sekä koontiversion asentamista kohdealustalle.

Pelin kehitysalustaksi valikoitui Unity-pelimoottori sen yhteisön, alhaisten tekijänoikeusmaksujen ja FakeFishin pelimoottorista aikaisemmin hankitun kokemuksen vuoksi. Pelin tuleva julkaisu tulee keskittymään länsimaihin Aasiassa julkaisemisen monimutkaisuuden sekä ylimääräisten kustannuksien johdosta ja julkaisukanavana tulee olemaan Android-käyttöjärjestelmän sovelluskauppa Google Play.

Lopputuloksena syntyi graafisesti miellyttävän näköinen ja pelattavuudeltaan sujuva peli, joka ei kuitenkaan ollut idealtaan ja toteutukseltaan tarpeeksi ainutlaatuinen päätyäkseen julkaistavaksi asti. Työn tuloksena valmistunutta peliä tullaan kuitenkin käyttämään pohjana uudelle versiolle, jonka tarkoituksena on pureutua tarkemmin nyt syntyneen pelin ongelmiin ja saada aikaan lopputulos, joka on pelattavuudeltaan ja tuntumaltaan ainutlaatuisempi kuin edeltäjänsä.

ASIASANAT:

Pelinkehitys, peliala, julkaisu, Unity, Unity3D, peliohjelmointi, C#, FakeFish, indie

CONTENT

PICTURES

FIGURES

GLOSSARY

2D	Two-dimensional computer graphics
2.5D	Two-and-a-half-dimensional, in this case meaning a 3D world with the gameplay restricted to a 2D plane
3D	Three-dimensional computer graphics
AI	Artificial Intelligence
Avatar	The character manipulated by the player
FPS	Frames per Second
GUI	Graphical User Interface
IDE	Integrated Development Environment
NPC	Non-Player Character or Non-Playable Character
SDK	Software Development Kit
UI	User Interface

1 INTRODUCTION

In the summer of 2014 my friends and I established a game development company, FakeFish, to pursue the dream of developing games for a living. The thesis goes through the development process of a game's free version that was developed for FakeFish to be showcased at the Game Development's Conference 2015 in San Francisco, California.

The first part addresses the game's concept, the game design choices we made and the games that inspired us, the platforms we are developing for and the tools we chose to use during the development process. In the last chapter the potential publishing channels for the game are explored.

The empirical section of the thesis goes through the actual game development process from the eyes of a programmer and a project lead. Everything in this project is programmed by me with the exception of the sounds and music that are programmed to the game by FakeFish's audio designer, with me only acting as a consultant if needed. I also partake whenever possible in the level design of the game with FakeFish's full-time level designer having the main responsibility. In addition to the aforementioned tasks I am also responsible for the integration of many of the other elements produced, including the GUI, designed levels, animations and 2D elements in to the game and making sure everything is optimized and running smoothly.

In the closing chapters of the empirical section the possible additions in the future and how the monetization of the game is planned through advertising are discussed.

2 GAME CONCEPT

The game, known as the Twins, developed for this thesis is an endless runner with a finite amount of premade levels, thus the term finite runner. It is set in a world heavily influenced by the Finnish national epic Kalevala, the same one all FakeFish's games are. This interconnection between games is seen to be a powerful marketing tool in the future, with many of the same elements appearing in multiple games and in addition to boosting marketing also reducing the amount of needed assets.

In the game the player controls a frog who cannot stop running and has to satisfy his insatiable hunger while doing so, which in turn creates animosity amongst the places he runs through. The gameplay consists of collecting different varieties of food, avoiding obstacles and hostile creatures all the while running through the fantasy world of the Finnish national epic, Kalevala.

2.1 Finite versus Infinite Runner

There are two categories endless runner games fall into: finite and infinite. In the finite variation the game revolves around premade levels and has a better ability to tell a story or a journey as a whole, whereas infinite runners rely on procedurally generated levels and often character upgrading to keep the player interested.

In the infinite category the player's performance is usually evaluated by a single arbitrary score, usually determined by either the distance travelled or points collected, or a mix of both. On the other hand the finite category relies more on constant scoring like percentage of stage completed and number of total collectibles collected or total secrets found. When it comes to design the finite category is more level design and wholeness oriented, whereas infinite runners often require a considerable amount of more programming to achieve proper generation of levels.

Both of these variations belong to the endless runner genre, which is a sub-category of the platformer genre, and is characterized by the player character's inability to stop running, short play sessions and simple controls. The originator of the genre is not exactly known, but B.C.'s Quest for Tires, a video game developed in 1983 for the computers of its time, can be seen as the first game to introduce the endless running mechanic. In B.C.'s Quest for Tires the player controls a caveman who has to rescue his girlfriend by traveling through several levels on his unicycle while avoiding obstacles. The genre was popularized twenty six years later by the game Canabalt in 2009 which has since inspired many other games like Temple Run, Bit.Trip Runner and Jetpack Joyride to name a few. (Parkin, 2013)

2.2 Difficulty

Challenge is a key element in the endless runner genre, be it a finite or an infinite runner, even though they have completely different reasons for the need of difficulty.

In the finite runner due to the pace of the gameplay the difficulty has to be high to keep the required amount of content at a manageable level. If the difficulty is not high enough, the player will dash through the levels in no time and be done with the game before any real monetization opportunities have arisen.

In the infinite runner variation on the other hand the difficulty has to be very finely tuned to keep the user interested in a way that the game is not too hard but not too easy either. If the game is too hard, the user will not have a great enough feel of progress and ends up quitting and if it is too easy the same feel of progress is lost if the running goes on forever without failing. One popular way to control this difficulty in the infinite runners is character development through upgrades to the user's avatar, which both increases the feel of progression and also makes the game a little less difficult each time an upgrade is received.

3 TOOLS

Game development is a significantly diversified field and thus the tools required by a project are also high in number. In this chapter the tools used by FakeFish in their game development are introduced shortly with the reasons for the chosen tools given, and alternatives being introduced as well.

3.1 Game Engine

Nowadays building an in-house game engine is a very rare occurrence, usually only done by triple-A studios that require very specific functionality out of their engines. For almost all smaller and even some AAA studios the common approach is to license an existing game engine and develop their game with it.

The advantages of using an existing game engine heavily outweigh the disadvantages as the commercial game engines of today are powerful and offer a vast amount of functionality straight out of the box. In addition to the saved development time when using a third-party engine, the developers also have an easier time working together and recruiting new employees as the odds are that others are familiar with the engine already and require little to no training with the engine itself. This is also an important aspect from the eyes of a developer since by having no in-house engine, the knowledge gained while working on a project is not partially lost due to every other studio using their own game engines. (King, 2011)

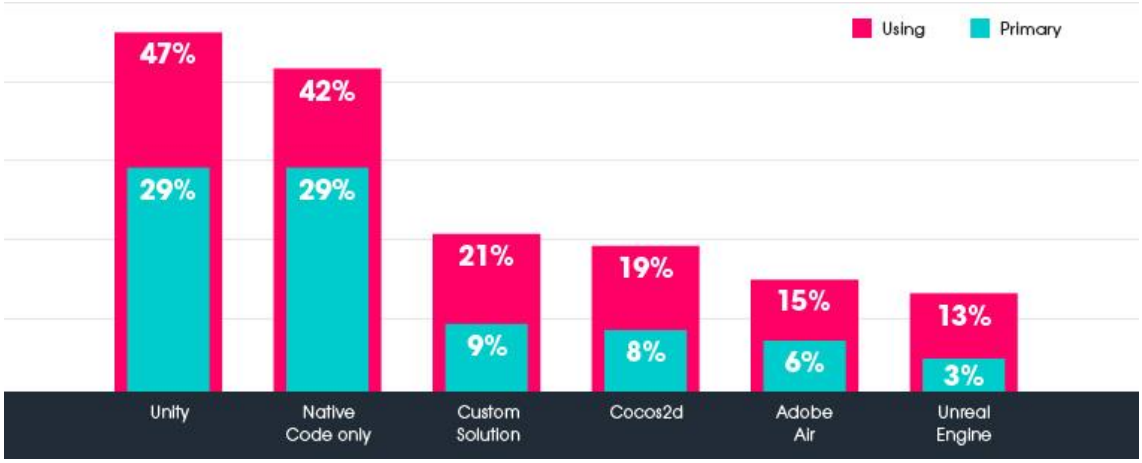


Figure 1. Distribution of Game Engine Use. (Vision Mobile, 2014)

There are multiple third-party game engines available today, but as expected a few have risen to be the more commonly used ones (Figure 1). As FakeFish wanted to use an engine with a big community and established working methods the following game engines were the main options:

3.1.1 Adobe AIR

Adobe AIR is a cross-platform runtime system developed mainly as a platform for building desktop and mobile applications but can also be used to develop games. AIR uses Adobe Flash for the graphical representation and ActionScript or HTML5 and JavaScript for programming to either develop native-like applications that run standalone outside of a web browser, or web-based applications that run without the need of installing any additional software.

A few years ago Adobe AIR would have been a very solid choice to use as a development platform, especially after the addition of cross-platform support in 2011, whereas today the trend has already shifted heavily towards other game engines (Figure 2).

Figure 2. The Demise of Flash in the Jobs Market. (Dunstan, 2014)

In addition to other game engines surpassing Adobe AIR in popularity, also on the application side it is rapidly being replaced by a newer, more versatile HTML5. These declines, both in game and application development, have affected the update intervals and new feature development greatly and has caused the community to disperse. With all these events in mind Flash was not seen as a viable development platform. (Dunstan, 2014)

3.1.2 Cocos2d

Cocos2d is an open source game engine with various versions available developed by different third parties. Cocos2d's Cocos2d-x branch is popular in the Asian game development scene with famous European developers using it as well and is a good choice for targeting multiple platforms. The main drawback is that Cocos2d-x is built with 2D-games in mind, thus not having a good enough support for 3D-elements. Other major disabilities are the fact that most of the documentation of the game engine's own functionality is written in Chinese and that the engine has no graphical editing, making independent level designing for

example too demanding without a vast knowledge of programming. (Cocos2d-x.org, 2015)

3.1.3 Unity

Unity3D is the most popular game engine at the time of writing the thesis, being used to make over sixty percent of current 3D mobile games. Unity offers targeting to multiple platforms, has a good support for both 2D and 3D elements and uses C# as a programming language which are the main reasons for its wide use as a development platform. (Unity3D, 2014)

The documentation available on the internet is good and detailed, thanks to the international use of the game engine. The engine in general is very user friendly, and having a graphical representation of almost every part of the game development process makes it easier as even non-programmers can develop and view it.

Additional assets and extensions offered on the Asset Store make customizing the Unity Editor easy and allow the developer to expand the editor's features in a required direction. With the new Unity 5 also offering all the previously paid content for free for personal use makes it an appealing choice for any type of game developer.

3.1.4 Unreal Engine 4

Unreal Engine is one of the oldest game engines still in use, first released in 1998 together with the game Unreal. Although the engine itself was released 17 years ago, it was made available to the public in November 2009 with the release of the Unreal Development Kit. (IGN, 2009)

As of today Unreal Development Kit is outdated with its over-the-top 25 % of earnings royalty model, clunky asset integration and lack of runtime editing. The new Unreal Engine 4 offers more features than the previous publicly available

versions and is monetized by having a gross revenue royalty fee of 5 % in addition to a monthly subscription of $19. (Epic Games, 2015)

Unfortunately due to the gross revenue based royalty model of Unreal Engine 4, FakeFish does not see licensing the engine being a viable option. For example a sales of $100 000 of the game would be roughly divided as follows: 30 % for the digital distributer, 30 % for Finland through taxes, and then of the $40 000 FakeFish would receive $5000 would go to Epic Games in royalties. Because of the heavy taxation in addition to the digital distribution fees Epic Games' cut of the final sum would be much more than 5 %, which heavily encourages to look at other options.

3.1.5 Conclusion

The top contenders for the game engine of choice were Unreal Engine 4 and Unity, of which Unity was chosen as the development platform. The main reasons for choosing Unity were the lack of royalties, larger community and experience with the platform in previous projects.

3.2 Visual Studio

Visual Studio is an Integrated Development Environment (IDE) developed by Microsoft, mainly intended as a tool to develop programs compatible with the various Windows platforms from phones to PCs. Part of the Visual Studio IDE is a code editor that supports C#, including a code completion component that makes programming easier and more streamlined without the need to google every new function the developer needs. As Unity supports the C# programming language, getting the code completion, also called IntelliSense, of Visual Studio to work together with Unity is easy. To get Visual Studio fully compatible with the Unity engine, only a Visual Studio plugin for Unity called UnityVS needs to be installed. The UnityVS plugin was developed by a company called SyntaxTree before being acquired by Microsoft in July 2014 and since then has

seen major improvements with the UnityVS team having joined the Visual Studio development team. (SyntaxTree, 2014)

At first the $299 price tag of Visual Studio Professional, which is the minimum requirement for the UnityVS plugin to work, might seem a bit much just to use as a code editor. However Microsoft has many deals with educational facilities and startup accelerators so a professional version of Visual Studio is often offered for free for new startups and students.

Other notable code editors compatible with Unity and C# are Notepad++, Sublime Text 2 and MonoDevelop, which is integrated into the Unity installation by default. The main drawbacks of Notepad++ and Sublime Text 2 are the lack of code completion support, although such functionality can be added to Sublime Text 2 by installing community created packages. The biggest drawback of both Sublime Text 2, with the Unity packages, and MonoDevelop on the other hand are the performance issues. Both of these programs are slow to open and have stability issues when multiple scripts are opened, thus being trumped in this field by Visual Studio easily which is very stable and works fast even when several scripts are being edited at the same time.

3.3 Blender

The 3D-models and animations in the game are made by using Blender, an open-source 3D computer graphics software that is completely free to use. As with any open-source software, the main problem of Blender is the lack of a clear roadmap of the future development. Especially as the main team that manages the software consists only of two full-time and two half-time employees, which means that most of the development itself is done by the community with the employees only managing the updates and approving changes. (Blender, 2013)

The most important reason for choosing to use Blender as the 3D graphics software is the high cost of the alternatives. The commonly used commercial alternatives for Blender are Maya and 3ds Max, both of which are developed

and owned by Autodesk. The Autodesk programs cost around $200 each per month, which for a new startup company is way too much for the little advantage over Blender, mainly faster bug fixing and customer support, they provide.

3.4 FMOD

FMOD is a sound effects engine developed by Firelight Technologies and supports many game engines and operating systems, with the most notable engine partners of FMOD being Unity, Unreal Engine and CryEngine. The main feature FMOD provides is the creation and playback of interactive audio throughout gameplay, which makes the repeating sounds in the game seem more alive with small but distinct changes to volume, tone and playback speed during runtime. (FMOD Studio Brochure, 2015)

The pricing model of FMOD Studio is especially lenient for startups as using the sound engine for projects with a budget of under $100 000 is completely free of charge. Even after the budgets grow, the price of FMOD keeps within reasonable limits as it's based on a target platform basis and is completely decidable by the developer on which platforms to release the game on. (FMOD Studio Licenses, 2015)

3.5 Other Music Tools

Whereas FMOD takes care of the playback of sounds during gameplay, Logic Pro X and Audacity are used in the initial creation of the music and effects. Logic Pro X is a digital audio workstation and MIDI sequencer developed by Apple for the Mac OS X platform and in FakeFish's games it is the main tool for music creation with Audacity, a free open source digital audio editor, being used for its static noise reduction capabilities.

Logic Pro X is a relatively expensive application with its retail price currently at $199 but is worth the money and is a widely used and renowned workstation among professionals.

3.6 BitBucket

BitBucket's git revision control system is used during the development to maintain a repository of the game's files to easily add new files and make them available to the development team. Reverting back to older files is easy and efficient with the BitBucket's version control and saves a great deal of work in case a new commit breaks the game. The repository was used by utilizing Windows PowerShell's git command integration and thus no GUI client was used.

BitBucket is initially free for five users in a single private repository, but with a little inviting between the five original users the size of the team can be grown to eight users with no cost. The repository has a 1GB soft and a 2GB hard limit for size. Upon reaching the first limit of 1GB, the users are warned of the repository's growth and given instructions on how to reduce its size. Using the repository can continue as normal, although the repository will not tell a detailed size anymore until the 2GB limit is reached. At 2GBs the repository will be locked and no further file commits can be issued. (Stepka, 2015)

Alternatives to BitBucket were GitHub and Perforce: GitHub, having no free license for private repositories, was quickly dropped and with Perforce requiring a private server to run on was also cast aside at the time. Perforce has not been completely discarded, mainly due to its integration possibilities with Unity, and will likely be used for FakeFish's bigger title's source control in the future.

3.7 Photoshop

Photoshop is a raster graphics editor by Adobe Systems and as of June 2013 has been exclusively part of the Creative Cloud service. With the team's graphic artists having most of their experience in Photoshop, it is considered a definite

choice from the get-go and is used to create concept art, textures and sprites for the game.

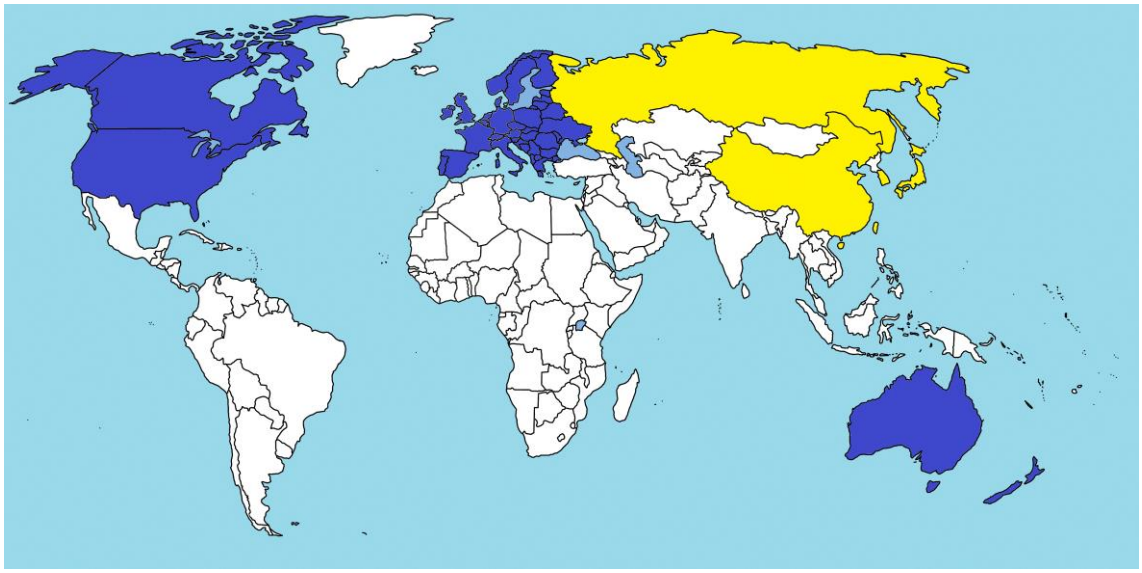
The main competitors for Photoshop used in tandem are Gimp and Pixlr, mainly because they are free and offer non-artists a way to quickly edit and create placeholder graphics. GIMP, GNU Image Manipulation Program, is a traditional desktop application whereas Pixlr is an online graphics editor, both of which offer a large number of the basic functionality like transparency and layers which are required even by many temporary graphics, resulting in a better testing and developing experience.

4 PUBLISHING

Publishing games today is easy thanks to digital distribution which eliminates the need for physical copies and makes game development less risky. In the earlier days of game development, especially in the cartridge era of the 1980s and 1990s, every published game was a huge risk due to the high cost of producing physical copies, no ways of updating content and the logistics required to get them to store shelves. As these risks are completely nullified by digital distribution, many independent game developers have joined the game development scene and have a way to distribute their games without huge investments. This leads to a much greater diversity in games and should also result in a better experience for the customer overall, as game breaking bugs can be fixed later on and new content added to the base game. Unfortunately as the ability to update games post-launch is now a possibility, some developers have seen it more as a moneymaking opportunity: release a cut version of their game and then sell the rest of the game through paid downloadable content. (Walker, 2007)

4.1 East vs West

Distributing games all around the world is easy and relatively cheap, thus one of the notable challenges game developers face when publishing games is localization. Main difficulty when it comes to publishing worldwide is the East, Asia, versus the West, the Western world, confrontation as these markets are completely different when it comes to games. As these two are the most potential ones for games, localizing a game to fit both of these markets is often a possibility that needs to be taken into account in development.



Picture 1. Eastern (Yellow) vs. Western (Blue) Markets from the Game Development Perspective. (August, 2008)

Localizing is often not just translating the game into the target market's language, but making it feel local and compatible with their culture, laws and regulations. The more localization, or in this case culturalization, is required, the more time and money is needed. Especially the laws and regulations of the target market have to be adhered to just to have the game in a publishable state, but it is often advisable to at the very least do a partial culturalization. In a partial culturalization the game's atmosphere, music and characters mainly, are tweaked to better match the target market's expectations and to make it easier for the players to immerse themselves in the game. (Skoog, 2012)

4.2 Mobile Publishing Channels

The mobile publishing channels in the western world are tied to their respective platforms and operating systems with Google's Google Play for Android, Apple's AppStore for iOS and Microsoft's Windows Store for Windows. These three are the major channels when distributing mobile content in the West and are the main focus for possible publishing in the future. In the Eastern culture local third party stores, especially in China, are more prevalent and are used

instead of the Google's store due to Google being banned since 2010 for an alleged hacking attempt. These third party app stores are usually maintained by local mobile network operators, contain more targeted content than the Google's equivalent and require extra work for foreign games and applications to be included in the store's selection. (Orsini, 2014)

4.2.1 AppStore

Apple's AppStore is the number one mobile store when it comes to premium apps and games, with iOS users downloading more paid apps than Android users on average. Also the significant difference in 2013's revenue between AppStore, $10 billion and Google Play, $1.3 billion, incentivize developers to primarily target AppStore, especially if the game or application follows the premium model. (Forrest, 2014)

4.2.2 Google Play

Android is the go-to platform for free games with in-app purchases to maximize downloads and the potential customer base, with Android phones having a 77.8 % share of the worldwide smartphone market at the end of 2013. When targeting Android devices, via Google Play or a third party store, the developer has to always keep in mind the wide diversity of Android devices, which makes it very time consuming to publish a version that runs smoothly on all desired devices. (Forrest, 2014)

4.2.3 Windows Store

Windows Store has since its creation had a problem with lack of popular apps, which has likely played a part in the smaller popularity of the Windows Phones. Many popular games and apps are missing from the Windows Store and probably aren't coming there any time soon. (Attkisson, 2013)

Since then the situation has improved with some of the more popular apps and games reaching the Windows Phone platform and keeping the Windows Store still afloat. From the perspective of the developer Windows Store shows some promise: with the average download earning the developer $0.23, it almost rivals the AppStore's $0.24 per download. Compared to the $0.04 per download on Android, Windows Store is looking like a good channel for apps and games with a premium model. (WMPowerUser, 2014)

4.2.4 Conclusion

For a game following the premium model the Apple's AppStore is a clear winner, although Windows Store is showing some promise and is worth keeping in mind. Picking a choice for free-to-play games with in-app purchases on the other hand is a bit more complicated, since both AppStore and Google Play offer potential. With those two stores in mind it comes down to the quality versus quantity in customers: Does the game offer many often optional purchases that influence the game's flow and hope to feed off more on the so called whales, people who purchase excessive amounts of in-app goods, and depend mostly on this marginal group of players for revenue, then the AppStore is a valid primary choice. On the other hand if the goal of the game is to maximize the amount of downloads and players to increase revenue, maybe through in-game ads or purchases that seem mandatory to the player, Google Play is a winner just for the sheer amount of potential downloads. As the game is developed with marketing in mind and is going to be free-to-play, the main platform is going to be Android and the game will be distributed through Google Play.

5 DEVELOPMENT OF A FINITE RUNNER

The development of the game's showcase version started in December 2014 and continued until the end of February 2015, with the time spent on the project hovering at around two months total. During this time two fully playable levels were produced and after that the production was halted until the summer of 2015.

5.1 Selecting a Platform

With the company's current focus being mobile games, the Twins is planned to be released on all possible modern mobile devices. With the game's general design being of the casual variety, a very popular type of game on the mobile devices, a PC version of the game is unlikely to happen due to lack of demand on the platform. Also the simplicity of the controls and quick play sessions are features prominent on the mobile platform.

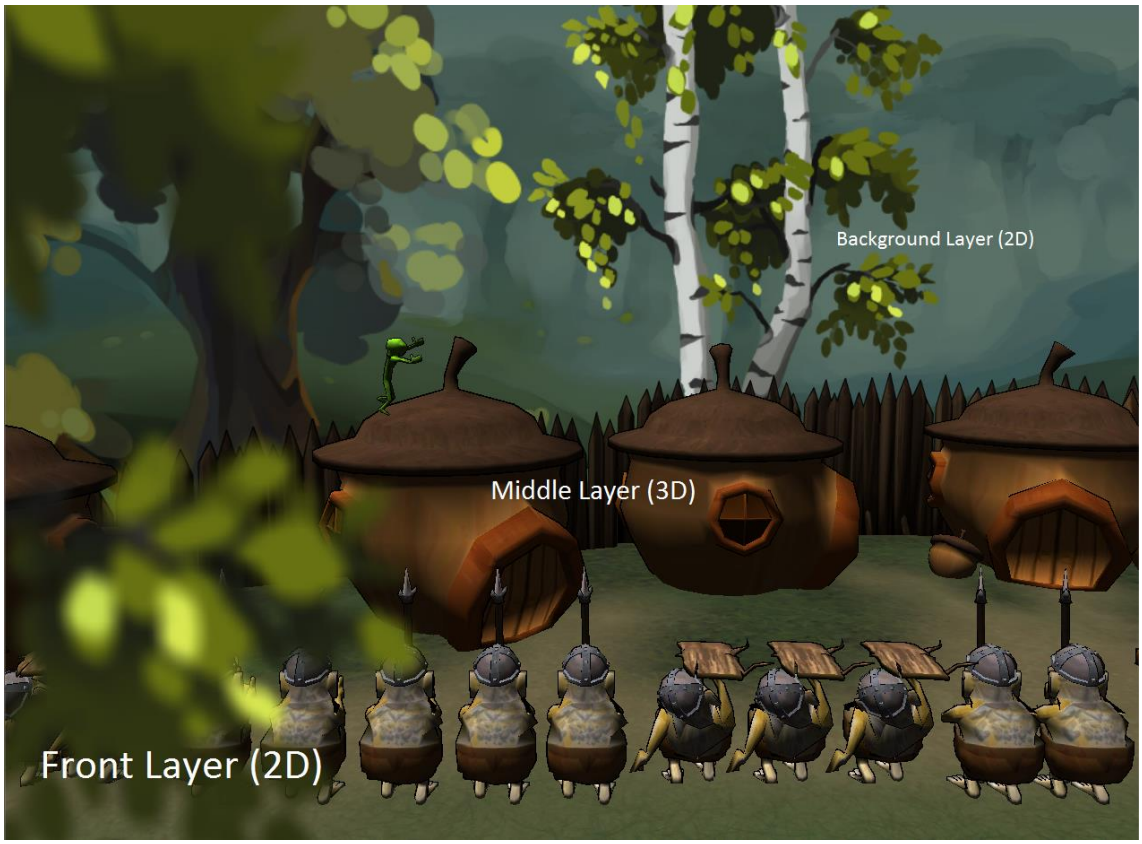
5.2 Story

All the FakeFish products share the same universe of the Finnish epic Kalevala, with each game taking a different point of view into the mythological world. The vision of the world is not an absolute representation of Kalevala by any means as there are creatures, locations and ideas added into it.

The runner game, working title the Twins, is a story about a frog-man hybrid and his abducted brother. At the start of the game Willie, after a food hoarding gone wrong, sees his brother Hans being taken away and vows to rescue him. To free his brother Willie must travel through the world of Kalevala all the while keeping his insatiable hunger under control by hoarding food wherever he goes. As expected others do not tolerate Willie's actions and try to stop the thief with many traps, obstacles and spells.

5.3 2.5D

A full 3D-game with many high quality elements in the game would too demanding for the current mobile device generation, so the solution is to use 3D-elements in the main gameplay area, with the layers in front and behind of the player having pure 2D-graphics (Picture 1).

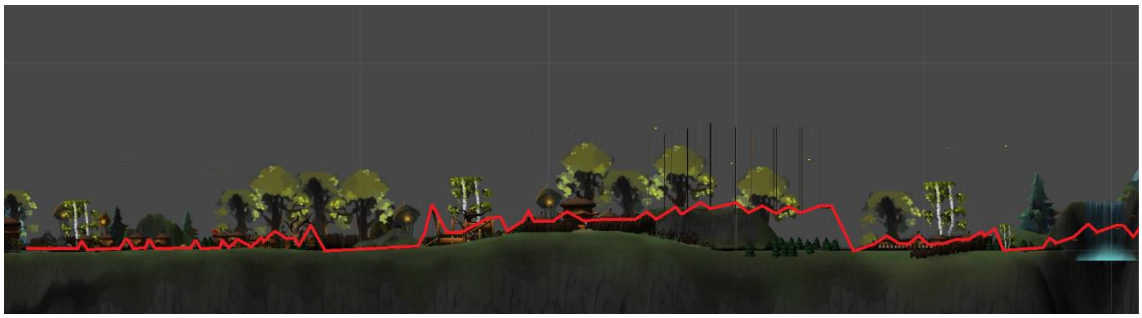


Picture 2. Gameplay Area Layout of the Twins.

By having this division between graphical elements helps highlight the path for the player as the gameplay area is the only one with 3D-elements, and also requires less work since there is no need to 3D-model scenery for the player to run through.

5.4 Level Design

The level design of the game, the Twins, has a clear goal: to make levels that are short, dynamic and difficult. The length of the levels is based on duration of the run, with every level designed to last one minute and thirty seconds. The difficulty and dynamicity of the levels is reached by designing the levels in such a way that the player never has more than a few seconds of time to think, having a soundtrack that adapts to the level's action and creating a running route with enough variety and different types of obstacles.



Picture 3. The Twins, first half of the first level with the route in red.

As seen in Picture 3, the route is very erratic and rarely gives a player any room for thinking, which enhances the difficulty of the level. The levels are designed to ramp up in difficulty fast, with the first level being a sort of a tutorial and easy to complete in comparison to the following levels. In the end most of the levels should take dozens or even hundreds of attempts to get through flawlessly. Even after this level of skill has been acquired, there are also secrets and designed harder paths the player has to take in order to complete everything in the level. This high level of difficulty was one of the game's aims as it helps reduce the amount of required content. As players need to repeat the same area over and over again instead of easily running through. Swiftly finishing all the content the game has to offer is not possible for most people which then increases the lifetime of the game significantly.

5.5 Gameplay

The basic gameplay of the Twins is running endlessly, avoiding obstacles and reaching the end of the level. In addition to just surviving the player can collect two types of food: berries and more exotic food items. Berries are planned to be used as a currency in the future to open new levels in addition to being linked to player score. The more exotic food items, related to the respective level's theme, are planned to be the currency for unlocking other goodies.

5.5.1 Movement

The main gameplay element of the player running is achieved by applying a constant velocity on the x-axis to a Rigidbody2D component (Picture 4).

```
void FixedUpdate()
{
    // If the player is done accelerating, sets the x-velocity to always be at MaxSpeed
    if(!Accelerating && isDead == false)
    {
        if(Jumping == false)
        {
            Rigidbody.velocity = new Vector2(MaxSpeed, Rigidbody.velocity.normalized.y * MaxSpeed);
        }
        else
        {
            Rigidbody.velocity = new Vector2(MaxSpeed, Rigidbody.velocity.y);
        }
```

Picture 4. Endless Running in Script.

The velocity is applied in a FixedUpdate() function instead of the regular FPS dependant Update() to ensure constant updating regardless of what the game's current FPS is, as FixedUpdate() updates always with a fixed time interval adjustable by the developer. For Twins a default time step value of 0.02 seconds is used, which then determines the update frequency of the function, 50 times a second. The y-axis velocity is also set to a constant value by normalizing the y-axis and then multiplying it by the MaxSpeed variable. This is done to ensure constant speed when going up or down slopes and is disabled when the player jumps.

5.5.2 Controls

The screen is divided into two halves, with one half of the screen used as a jump button and the other as an ability button. By default the left side of the screen is used to jump and the right side to trigger an ability, but the sides are dynamically adjusted according to the player's actions. In the first level both of the buttons are used to jump when the level is played for the first time. Data is then collected about the side of preference for jumping: if the player uses the left side of the screen for jumping, come level 2 the ability is going to be placed on the right side of the screen and vice versa.

At this stage the game has only two abilities, jumping (Picture 5) and sliding (Picture 6).

```
public class JumpAbility : Ability {

    float JumpForce;

    void Start()
    {
        JumpForce = GameManager.PlayerCharacter.InitialJumpForce;
        CanJumpWhileActive = false;
    }

    public override IEnumerator AbilityOn()
    {
        yield return null;

        PlayerCharacter.CurrentAbility.DeactivateAbility();
        Anim.SetBool("Landed", false);

        PlayerCharacter.Jumping = true;
        PlayerCharacter.JumpAmount += 1;
        PlayerCharacter.Rigidbody.velocity = new Vector2(PlayerCharacter.Rigidbody.velocity.x, JumpForce);
        PlayerCharacter.Rigidbody.gravityScale = PlayerCharacter.FallingGravityScale;
        FMOD_StudioSystem.instance.PlayOneShot("event:/Twin/twin_jump", transform.position);
    }
}
```

Picture 5. Jumping Script.

JumpAbility is a child of the Ability class and inherits variables from its parent, one of which is the ability to jump while ability is active, and is utiized by the sliding ability during which jump can be used. First the JumpAbility gets values set in the PlayerCharacter script for the initial jump force and also determines that jump cannot be used while jumping. Then nothing is done until an input from the player pressing the jump-button is received and the ability is turned on.

Once input is received, the PlayerCharacter's CurrentAbility is deactivated and the animator is told that the player is no longer in touch with the ground by setting the Boolean "Landed" to false, which starts the jumping animation. Jumping is set to true and the jump is also added to a jump calculator just for the sake of statistics. Jump's effect is added to the Player's Rigidbody2D's velocity by setting the y-velocity to be equal to the JumpForce and gravity is manipulated for a better, more realistic falling speed after the jump. After all this also a jumping sound is played by the FMOD audio middleware engine.

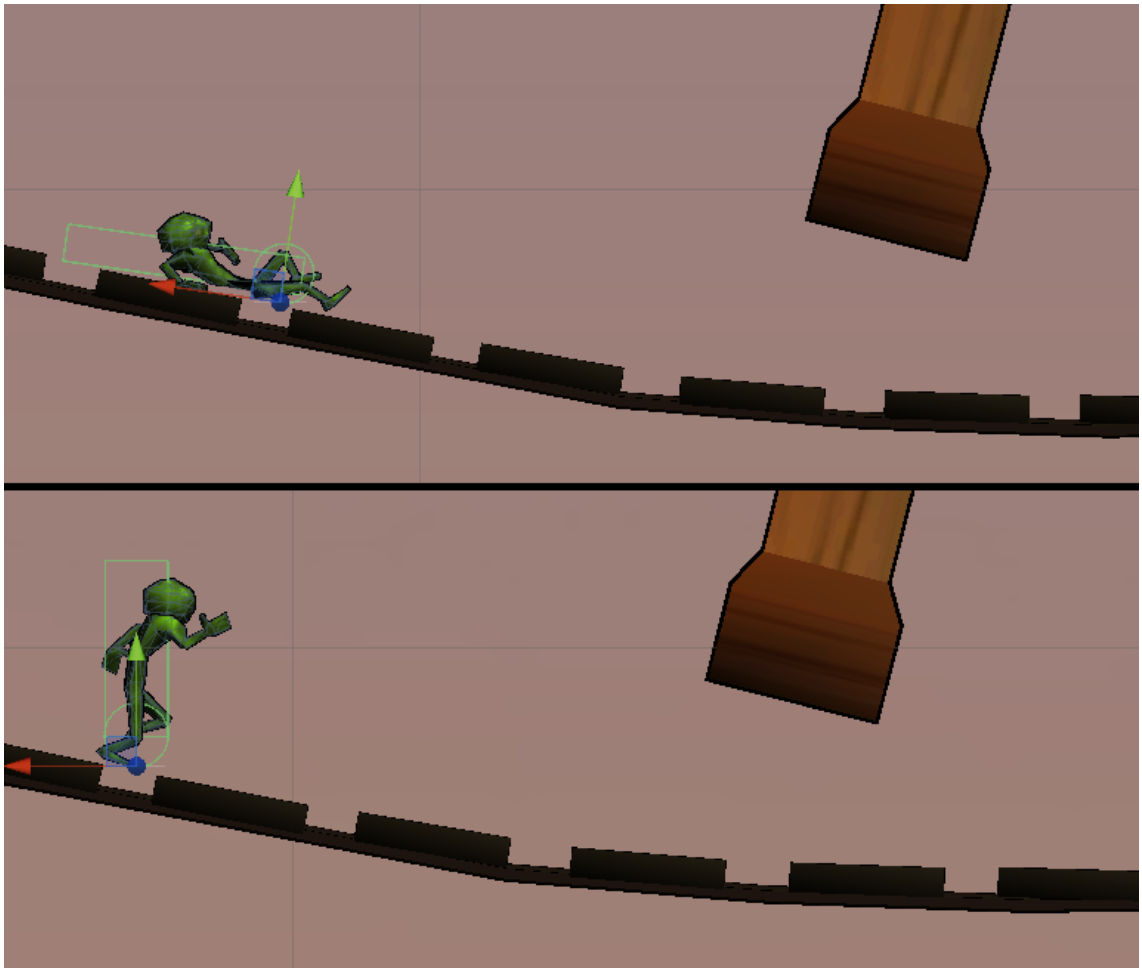
```
public override IEnumerator AbilityOn()
{
    yield return null;
    Anim.SetBool("Sliding", true);
    PlayerCharacter.Rigidbody.fixedAngle = false;
    Active = true;
    // Changes the collider to be of horizontal orientation
    PlayerCharacter.ChangeColliderSize(TargetSize, TargetCenter);
}

public override void DeactivateAbility()
{
    PlayerCharacter.transform.localEulerAngles = new Vector3(0,180,0);
    PlayerCharacter.Rigidbody.fixedAngle = true;

    // After a wait (Slide duration) changes the collider back to normal
    PlayerCharacter.ChangeColliderSize(TargetSize, TargetCenter, true);
    Anim.SetBool("Sliding", false);

    if(CanJumpWhileActive == false)
    {
        PlayerCharacter.CanJump = true;
    }

    Active = false;
    PlayerCharacter.CanAbility = true;
    StopCoroutine(AbilityOn());
}
```

Picture 6. Sliding Script.

SlideAbility is also a child of the Ability class and has the same basic functionality as described in the JumpAbility paragraph. The main differences in functionality are the Player's avatar rotating according to the platform's angle, being able to jump while sliding and changing colliders. Slide is used to go through narrow passages that are impossible to run through (Picture 7).

Picture 7. Sliding Under an Obstacle.

The rotation is achieved by simply disabling the Rigidbody's FixedAngle Boolean, which allows the Player's avatar to rotate and match the platform's angle. The rotation is based on the colliders' position when compared to the platform's collider, which is why during sliding the colliders change significantly, thus allowing the Player to go through much narrower passages (Picture 8).



Picture 8. Sliding Colliders versus Running Colliders.

The two collider-types, standing and sliding colliders, are premade and saved to the TargetSize and TargetCenter variables, changing the colliders' dimensions position respectively. The colliders are controlled by the PlayerCharacter's ChangeColliderSize function which first changes the colliders to match the sliding requirements when the slide ability is called, then returning them back to the upright position whenever the ability is deactivated.

5.5.3 Collisions

The collisions in the game are relatively simple and only have two different outcomes: player landing on an object or running into an object (Picture 9).

```
void OnCollisionEnter2D(Collision2D coll)
{
    // If the collision is a horizontal collision, the player has landed
    if(coll.contacts[0].normal.x <= 0.71 && coll.contacts[0].normal.x >= -0.71f)
    {
        Rigidbody.gravityScale = 30;
        if(Jumping == true || Falling == true)
        {
            FollowJump = false;
            CanJump = true;
            Jumping = false;
            FallingCheck = false;
            Anim.SetBool("Landed", true);
            CameraController.PlayerYValue = transform.position.y;
            if (a_Landing == true) {
                print("A_landing!");
                FMOD_StudioSystem.instance.PlayOneShot("event:/Twin/twin_landing", transform.position);
                a_Landing = false;
            }
        }
    }
    else
    {
        // If the collision is vertical, the player is considered dead and the run reset
        StartCoroutine(Death());
    }
}
```

Picture 9. Collision Script.

First upon collision the contact points of the Collision2D object are evaluated and checked if they are between the acceptable range of greater than -0.71 and smaller than 0.71. These values have been acquired by testing and represent the angle of the collision, with 0 being a completely flat collision and 1 a completely vertical one. What this does is any collision with a collision angle of equal or less than 70 degrees is acceptable and does not end in the Player's demise. If acceptable, the end result is then interpreted as the player landing and treated accordingly by setting the required animation and jumping related Booleans to correct values, applying the default gravity scale when not jumping to make falling off objects possible and playing a landing sound.

5.6 Falling

Falling is a distinctive state that needs to be differentiated from jumping to have the camera follow the player's position in a way that the player is always visible on the screen, disable jumping as player runs off a cliff for example and when landing play a sound associated with a long fall (Picture 10).

```
// Compares the raycast distance and determines if the distance is greater than the maxFallingTreshold.
// If true determine that the player is falling and act accordingly.
// This is done until the player either lands or it is determined that the player is actually falling
IEnumerator CheckFallHeight()
{
    FallingCheck = true;
    while(FallingCheck == true)
    {
        float dist = 0f;
        if (GetHitDistance(out dist))
        {
            if (initialDistance < dist)
            {
                //Get relative distance
                float relDistance = dist - initialDistance;
                //Are we actually falling?
                if (relDistance > maxFallingThreshold)
                {
                    print("Falling");
                    CanJump = false;
                    Anim.SetBool("Landed", false);
                    FallingCheck = false;
                    if(Jumping == false)
                    {
                        CameraController.FollowPlayer = true;
                        Falling = true;
                        if (relDistance > 3f) {
                        a_Landing = true;
                        }
                    }

                }
            }
        }
        yield return new WaitForSeconds(0.1f);
    }
}
```

Picture 10. FallingCheck Script.

The CheckFallHeight function is called every time a player jumps or falls off an edge of an object or terrain and is iterated through until it is determined if the Player is falling or not. This is evaluated by shooting a RayCast down from the Player's current position every 0.1 seconds and comparing the length of the RayCast to the maximum falling threshold, if bigger the player is falling, otherwise just jumping off an object or falling a small distance.

5.7 Graphical User Interface

At this point in development the GUI consists of a Main Menu and In-game UI, both of which are at a very early stage of development and are not to be considered final.

5.7.1 Menus

The main menu is simple set of different screens: level selection, settings, achievements and stats (Picture 11).

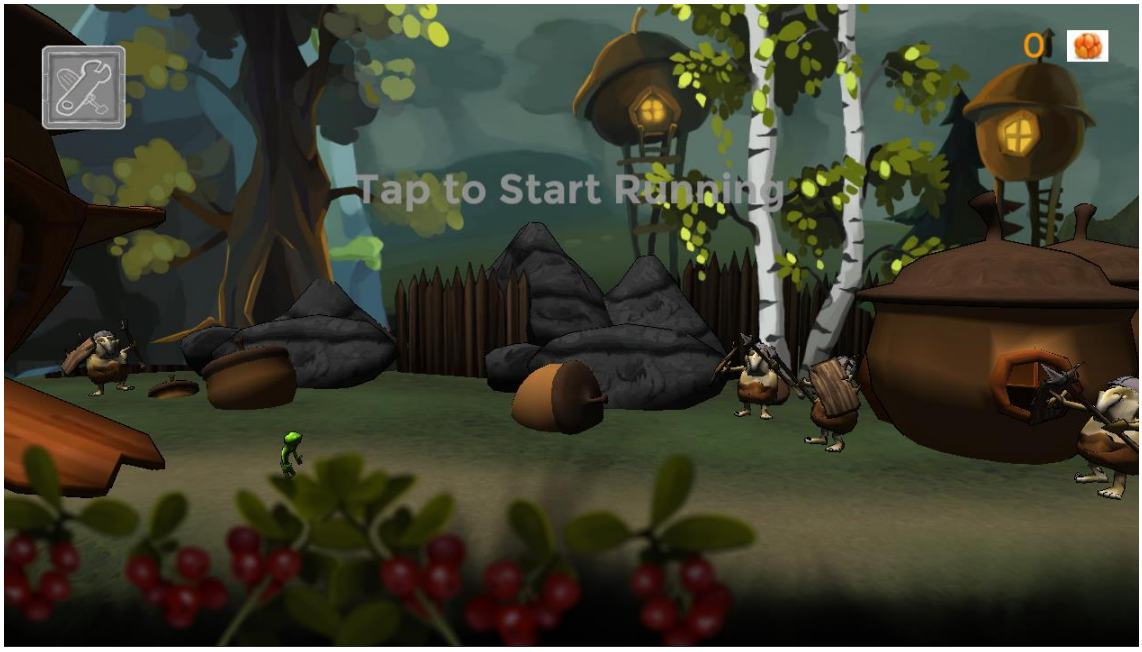
Picture 11. The Main Menu.

The levels screen contains a basic level selector with the currently available levels. Settings screen has a variety of adjustable values like music, effects, in-game button placement and a How to Play –section. Achievements screen at the moment is just a list of possible achievements to be added in the future with no functionality. Stats screen shows statistics of gameplay that are saved to an XML file on the device and read on application start.

5.7.2 In-Game

The visible part of the in-game UI is a very minimalistic representation, with only a button for going back to the main menu, collected berries display and a

prompt in the middle of the screen to tell the Player how to begin the run (Picture 12).



Picture 12. The In-Game UI.

Hidden part of the UI consists of the two hidden buttons, one taking up the left half of the screen and another the right part of the screen, which control the character's jumping and ability use. Both of these buttons forego the Unity's GUI button logic by being custom made event triggers that are created upon starting the game (Picture 13).

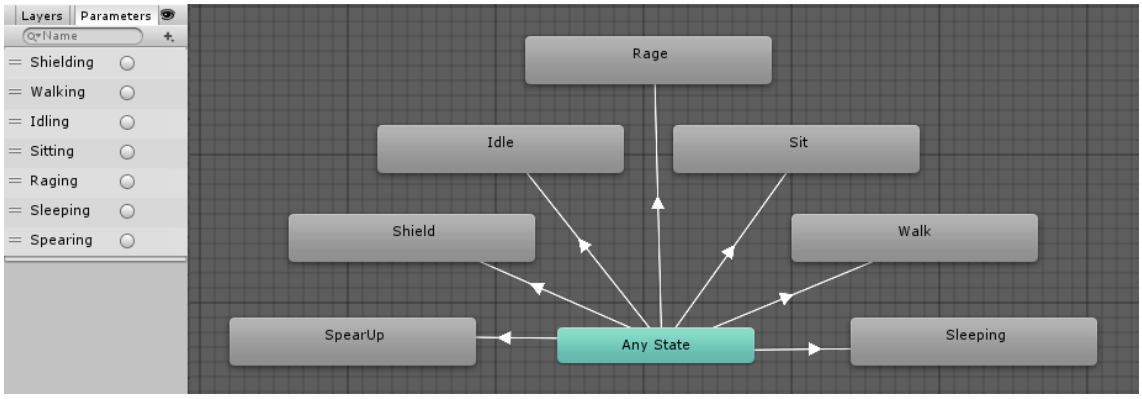
```
// LeftSide function linking start
EventTrigger trigger = LeftSide.GetComponent<EventTrigger>();
EventTrigger.Entry entry = new EventTrigger.Entry();
entry.eventID = EventTriggerType.PointerDown;
entry.callback = new EventTrigger.TriggerEvent();
UnityEngine.Events.UnityAction<BaseEventData> call;
if(GameManager.CurrentLevel == "Level11" || GameManager.SettingsDetermined == false)
{
    call = new UnityEngine.Events.UnityAction<BaseEventData>(LeftSideClickLevel1);
}
else
{
    call = new UnityEngine.Events.UnityAction<BaseEventData>(LeftSideClick);
}

entry.callback.AddListener(call);
trigger.delegates.Add(entry);
// LeftSide function linking end
```

Picture 13. Dynamic Buttons.

The triggers are attached to an object with an EventTrigger component already in place and these triggers' type is of the PointerDown type, which is a feature only in EventTriggers as Unity's own button implementation are of the Pointer-Up type. PointerUp has a delaying effect when calling functions as it is only called the user touches the screen and then ends the touch, instead of calling a function immediately when a user starts touching the screen. This delay is the reason for using EventTriggers with a PointerDown type to make jumping in the game have no extra delays.

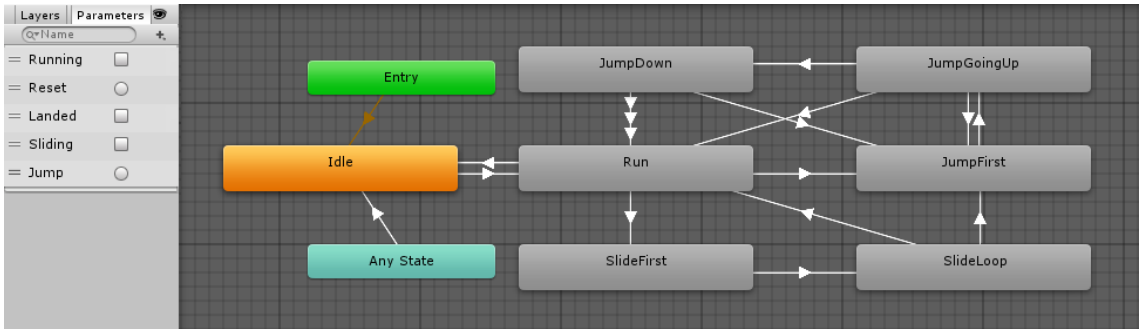
5.8 Animations

The animations in the game are imported together with the model from Blender and controlled in Unity with the Animator component. In the game the NPC's AI is simple as it usually only needs to do a single task endlessly. One example of this very simple AI and its controller is the Troll Animator Controller (Picture 14).



Picture 14. Troll Animator Controller.

The Troll Animator has a series of toggle values, represented by circles on the left side of the screen, that are set before the game starts which dictate the behaviour of the troll for the whole game. Each state has a related toggle value that triggers the animation and plays it until a new toggle value is set. As the behaviour of the trolls is very simple, no complex cross-state interactions are required unlike in the Player Animator Controller (Picture 15).



Picture 15. Player Animator Controller.

The Player's Animator Controller is more complex and in addition to toggles uses Booleans, represented by squares on the left side of the screen, to have a more consistent behaviour since purely using toggles can be problematic when there are many cross-state interactions. What makes the Player Animator Controller more complex is the user whose actions are impossible to predetermine, and thus requires a great deal of animation blending based on the Player's actions to make everything look seamless.

5.9 Monetization

With the game being completely free to play, the main monetization planned for the game in the future is going to be from ad revenue, which is now, with the release of Unity 5 and UnityAds in the Game Developer's Conference of 2015, very simple to implement and efficient. Watching ads is completely obligatory, but it is incentivized by rewarding the user for each advertisement watched with in-game currency. The advertisements are fifteen second long game ads which can be targeted to the most potential users by the genre they are currently playing, their device and country. (UnityAds, 2015)

5.10 Profiling

The main tool for optimization in Unity 5 is the Profiler, which reports where the time is spent on running the game, for example rendering, animating or scripts.

If one of these areas is significant enough to slow down the game, it will be visible on the profiler and give information on where the problem may lie. Running the profiler every now and then when developing is a good practice to catch resource hogging elements before they are hard to fix.

5.11 Building to Android

During the development it is crucial to constantly keep building to the target platform one is developing for to avoid unexpected problems. Especially when the target platform is not PC, testing anything where the target platform's controls play a significant role is important to not develop something that is not compatible with the target platform at all.

The build process with Unity itself is simple, just choose a few basic settings and the building is started. What matters is the target platform and the software needed to get the build compatible for the device in question. On the device itself a developer mode needs to be applied so that the device sees the incoming data as a legitimate application instead of an install from an unverified source, thus blocking the build from succeeding.

For Android, only the SDK Tools package is required, which can be downloaded from https://developer.android.com/sdk/ and installed with ease following the instructions provided. In addition to the packages featured in the instructions it is likely that a Google USB driver package needs to be installed as well, which is found from the SDK under the Extras folder. After the packages are installed building from Unity can start. Upon the first initiation of the build process Unity will inquire the user of the location of the Android SDK and once the correct folder is selected, a build will be initiated. The first time a game is built it takes longer than the consequent builds as everything is not needed to be included in the build installation.

Briefly for comparison the other mobile operating systems, iOS and Windows Phone, both require their respective PC counterparts, either a Mac or a Windows PC, for building. On Macs the Unity build process will be finalized in

xCode, a free IDE provided by Apple, whereas on Windows finalizing is done by Visual Studio, an IDE by Microsoft with a price tag of $299 for the cheapest version.

6 CONCLUSION

Even though the endless runner genre has been one of the most popular game genres in the few years of its official existence, it still has potential for new innovations and creative solutions. Unfortunately the Twins in its current form will likely never be developed with the game's concept as is, since it does not bring enough innovation to the already saturated genre. The game's concept has already been honed further and the development of a new version with improved concept, mainly focusing on gameplay mechanics, has started and so far proven to be more unique and innovating.

The Twins might have not been even close to the best of success stories, but for FakeFish it was almost as important as if it was. The people who worked on the project learned to estimate the work hour requirements of their tasks much better and the importance of continuous testing. FakeFish also now has a demo of what it can do in two months' time, which will prove useful when seeking customer projects in the future.

Once FakeFish's main project is well underway and the level designers and graphical artists have extra time, working on the Twins' new version will be resumed. Most of the programming for the game was already done for the first version and can be reused, thus the main focus of the work is going to be on game and level design.

REFERENCES

Attkisson, A. Windows 8 Store Lacks 75 Percent of Most Popular Apps (Infographic). Referenced 10.05.2015. http://blog.laptopmag.com/windows-store-lacks-popular-apps

August, R. 2008. Black and white political map of the world. Referenced 12.05.2015. https://en.wikipedia.org/wiki/File:Black_and_white_political_map_of_the_world.png

Blender 2013. History. Referenced 02.05.2015. http://www.blender.org/foundation/history

Cocos2d-x.org 2015. Cocos2d-x. Referenced 25.01.2015 http://www.cocos2d-x.org/wiki/Cocos2d-x

Dunstan, J. 2014. From Flash to Unity. Referenced 23.04.2015 http://jacksondunstan.com/articles/2632

Epic Games 2015. Frequently Asked Questions (FAQ). Referenced 29.04.2015 https://www.unrealengine.com/faq

FMOD Studio 2015. FMOD Licenses. Referenced 02.05.2015 http://www.fmod.org/sales/

FMOD Studio 2015. Introducing the Game Changer for Audio Professionals. Referenced 02.05.2015 http://www.fmod.org/files/public/FMODStudioBooklet.pdf

Forrest, C. 2014. Google Play v Apple App Store: The battle for the mobile app market. Referenced 08.05.2015. http://www.techrepublic.com/article/google-play-v-apple-app-store-the-battle-for-the-mobile-app-market/

IGN 2009. Epic Games announces Unreal Development Kit, powered by Unreal Engine 3. Referenced 29.04.2015 http://www.ign.com/articles/2009/11/05/epic-games-announces-unreal-development-kit-powered-by-unreal-engine-3

King, Y. 2011. Opinion: Why On Earth Would We Write Our Own Game Engine? Referenced 23.04.2015 http://gamasutra.com/view/news/128765/Opinion_Why_On_Earth_Would_We_Write_Our_Own_Game_Engine.php

Orsini, L. 2014. Google Brings The Play Store To China, Kinda Sorta. Referenced 08.05.2015 http://readwrite.com/2014/11/20/google-play-store-china-support-kinda

Parkin, S. 2013. Don't Stop: The Game That Conquered Smartphones. Referenced 14.05.2015 http://www.newyorker.com/tech/elements/dont-stop-the-game-that-conquered-smartphones

Skoog, K. 2012. When to Forgo the Culturalization of Video Games: Contextualizing Globalization within the Mobile Marketplace. Referenced 07.05.2015 http://www.gamasutra.com/blogs/KarinESkoog/20120702/173371/When_to_Forgo_the_Culturalization_of_Video_Games_Contextualizing_Globalization_within_the_Mobile_Marketplace.php

Stepka, J. 2014. Repository size limits. Referenced 24.03.2015 https://blog.bitbucket.org/2014/05/30/repository-size-limits/

SyntaxTree 2014. Microsoft Acquired SyntaxTree. Referenced 30.04.2015 http://syntaxtree.com/

Unity3D. The Leading Global Game Industry Software. Referenced 07.02.2015 http://unity3d.com/public-relations

UnityAds. Ads That Users Love. Growth for Your Game. Referenced 17.05.2015 https://unityads.unity3d.com/

Vision Mobile. Developer Economics Q3 2014: State of the Developer Nation. Referenced 28.03.2015 http://www.visionmobile.com/product/developer-economics-q3-2014/

Walker, J. 2007. RPS Exclusive: Gabe Newell Interview. Referenced 05.05.2015 http://www.rockpapershotgun.com/2007/11/21/rps-exclusive-gabe-newell-interview/

WMPowerUser 2014. Windows Phone Store revenue said to rival that of the the Google Play store. Referenced 10.05.2015. http://wmpoweruser.com/windows-phone-store-revenue-said-to-rival-that-of-the-the-google-play-store/