|  |
| --- |
| Indie Platform Game |
| Research Phase Report |
| Hugh Desmond – Software Development Year 4 |

Supervisor: Jim Doyle

Second Reader: Byron Treacy

06/12/2012

I hereby certify that this material which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent, that such work has been cited and acknowledged within the text of my work. I understand that my project documentation may be stored in the library at CIT, and may be referenced by others in the future.

Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_

Contents

[**1. INTRODUCTION** 3](#_Toc342229382)

[1.1 The Purpose of the Project 3](#_Toc342229383)

[1.2 Background 3](#_Toc342229384)

[1.3 Project Objectives 3](#_Toc342229385)

[1.4 Learning Outcomes 4](#_Toc342229386)

[**2. RESEARCH** 5](#_Toc342229387)

[2.1 Game Development Approaches 5](#_Toc342229388)

[2.1.1 Personal Development 5](#_Toc342229389)

[2.1.2 The Software Engineering Approach 6](#_Toc342229390)

[2.1.3 The Indie Approach 7](#_Toc342229391)

[2.1.4 Feedback-Driven Development 9](#_Toc342229392)

[2.2 The Platformer Genre 12](#_Toc342229393)

[2.2.1 History 12](#_Toc342229394)

[2.2.2 Sub-Genres 15](#_Toc342229395)

[2.3 Similar Projects 17](#_Toc342229396)

[2.3.1 “*Generic*” by PHL 17](#_Toc342229397)

[2.3.2 Physics Engine Based Platformer 19](#_Toc342229398)

[2.4 Gameplay 21](#_Toc342229399)

[2.4.1 Gameplay & Game Mechanics 21](#_Toc342229400)

[2.4.2 The Science behind “Fun” 21](#_Toc342229401)

[2.4.3 Game Mechanic Categories 26](#_Toc342229402)

[2.5 Game Art/Animation 28](#_Toc342229403)

[2.6 Audio 30](#_Toc342229404)

[2.7 Storyline/Setting 32](#_Toc342229405)

[2.9 Tools & Platforms 35](#_Toc342229406)

[**3. REQUIREMENTS SPECIFICATION** 37](#_Toc342229407)

[3.1 Approach 37](#_Toc342229408)

[3.2 Functional Requirements 38](#_Toc342229409)

[3.2.1 Mandatory (minimum) 38](#_Toc342229410)

[3.2.2 Optional 39](#_Toc342229411)

[3.3 Non-Functional Requirements 40](#_Toc342229412)

[3.3.1 Usability & Humanity Requirements 40](#_Toc342229413)

[3.3.2 Look and Feel Requirements 40](#_Toc342229414)

[3.3.3 Design Requirements 41](#_Toc342229415)

[3.3.4 Performance Requirements 41](#_Toc342229416)

[3.3.5 Implementation and Testing Requirements 42](#_Toc342229417)

[3.3.6 Cultural, Political and Legal Requirements 42](#_Toc342229418)

[3.4 Use Cases 43](#_Toc342229419)

[3.5 Game State Diagram 48](#_Toc342229420)

[**4. OTHER CONSIDERATIONS** 49](#_Toc342229421)

[4.1 Implementation Methodology 49](#_Toc342229422)

[4.2 Resources Required 50](#_Toc342229423)

[4.3 Open Issues 50](#_Toc342229424)

[4.4 Possible Risks 51](#_Toc342229425)

[4.5 Legal Considerations 52](#_Toc342229426)

[4.6 Implementation Phase Plan 52](#_Toc342229427)

[4.6.1 Summary 52](#_Toc342229428)

[4.6.2 Time Breakdown & Estimates 53](#_Toc342229429)

[4.6.3 Implementation Phase Key Milestones 54](#_Toc342229430)

[**5. APPENDIX** 55](#_Toc342229431)

# 1. INTRODUCTION

## 1.1 The Purpose of the Project

“To create an entertaining indie platform game on a low budget, which is to be released on a major platform and enjoyed by many people”

## 1.2 Background

It is my personal goal to design and code a game which is playable on a popular console platform, in this case the Xbox. Microsoft released a special area of the Xbox arcade marketplace (XBLIG) where individual and lesser established game developers can sell their products for minimal publishing/development cost. I plan to use this platform to release my game where it will be available to download by the general public either for free or for a small fee. I aim to make a game which is entertaining for a relatively short period of time but is of a high quality. In terms of how the game plays I aim to challenge the player, rewarding them for overcoming challenges to stimulate them to keep playing. At the same time I don’t want to challenge the player so much so that the game becomes frustrating.

## 1.3 Project Objectives

* Create a story driven 2D platformer game with multiple levels/stages
* Include a shooting mechanic whereby the protagonist must shoot at enemies to survive
* Include Puzzle elements where the protagonist must use math, logic and or problem solving skills to continue progressing in the game.
* Optional Objective: Provide a 2 player option, either co-operative where a second player helps the first through the main game or competitive, where the 2 players compete against each other in a spin-off of the main game, using all of the same mechanics but with different rules and goals.

## 1.4 Learning Outcomes

* Increase in logical and mathematical skills which will be tested heavily during the development of the algorithms which control the game’s behaviour.
* Gain experience in designing class structures for complex projects.
* Gain experience in developing for a specific platform, which is a heavily controlled and regulated environment, requiring the use of an API to control interactions between your project and the platform.
* Increase in creativity both in programming (developing new game features/behaviours) and artistically (creating usable graphical interfaces, interesting in-game environments etc.)
* Gain great experience in a “customer feedback to aid development” type of cycle as this will be my main way of gauging whether my game is entertaining to others and appeals to a wide enough audience.
* Gain experience in testing and debugging a complex project with an aim to removing enough defects to provide and maintain customer satisfaction.
* Gain experience in an aspect of game design which is difficult to achieve: providing the player with just the right amount of challenge so they are stimulated to keep playing but are not frustrated by a lack of progress.

# 2. RESEARCH

## 2.1 Game Development Approaches

### 2.1.1 Personal Development

The success of this project relies on many key factors, one of which I believe is personal ability. Game Development is seen as a subset of software development, but it has so many characteristics which are unique and different to traditional software development that it has its own industry. Although many skills gained in the area of software development are transferable to games development, there are many techniques which one will never have come across in software development but are commonly used in game development. This project is my first real exploration of this subset of software development and as with many firsts in life, it carries inherent risk. Due to the large amount of unknowns it is important that I manage my personal development so that I can learn the required skills to make a game in as quickly and thoroughly a manner as possible.

One of the most common pieces of advice you will find on breaking into game development is to start small. In game development, it is important to control the pace which you set yourself to learn something. By this I mean don’t make the common mistake of trying to create your dream game as your very first game. It’s an easy mistake to make as it’s a safe assumption to make that you have an interest in games and would like to produce something that you would enjoy playing. Game development is all about making mistakes and learning from them, so you are better off spending a year making numerous small complete projects, then spending 2 years trying to build something several levels above your technical ability, only to have all that time wasted on a project you couldn’t finish. Another important aspect which pacing affects is motivation. If you spend an inordinate amount of time trying to perfect something very specific, only to fail, it will lower your morale and increase your likelihood of giving up on the project.

I have learned that progress will generally feel very slow when making games and something which seems simple to implement can easily inflate in complexity once it has been broken down to the pieces which you can work with at a programming level. An example of this would be the typical player jump in a 2d platformer game. At first it seems like it wouldn’t be very difficult or time consuming to program. When a button is pressed there is a response to this event which makes the image/sprite of the character move up and down in quick succession. From looking at this happening in a typical scenario you can tell that the player sprite must move up from the ground, slowing as the peak of the jump is reached and then speeding up as it moves back to the ground. Also you can see that if the player is travelling right, they continue to travel right while in the air, causing an arc and visa-versa for travelling left. When looking at this in a more detailed level, with a view to programming this behaviour, we take into account all the possible scenarios the player may jump in. Say for instance the player jumped at a wall, the sprite would have to detect when its closest edge has hit the wall and from there react by stopping the original movement path and perhaps sliding down the wall to the ground. Say for instance the player jumps from a higher ground to a lower one, the shape of the movement arc would be different, the player would essentially fall below the level from which they originally jumped, thus causing longer airtime. How will this affect the jump animation? As you can see things can quickly expand in complexity so it is best to start as simple as possible to avoid being caught out by deceptively difficult programming tasks.

### 2.1.2 The Software Engineering Approach

It is important to note that although many aspects of game development differ to traditional software development, I must take note of software engineering best practices which are applicable here. Due to a rapid increase in complexity in software, much strain is placed on developers, so much so that it is no longer plausible to work harder, but to work smarter. Typical game software must incorporate gameplay mechanics, physics simulation, sound and music playback, artificial intelligence, compression/decompression, 3 dimensional graphics, networking aspects among many other things. Although compared to traditional software which may have many large databases and information-heavy systems, games usually don’t have a lot of input information from the end user aside from the occasional name input for a leader board (Although this is evolving in recent times with social games and complex multiplayer statistics systems found in major releases). The data manipulated within a game typically comes in the form of textures, models, sounds, music, video, animations etc. Because of all the intertwined media types which within themselves have their own industry (e.g. music production, film industry), mixed with the time-consuming procedures of digitizing this data and making it interactive, you end up with a very complex project which must have a carefully chosen development structure in place to avoid imminent failure.

Iterative Development is commonly practiced in modern software development and is very much applicable to game development. This involves splitting a project into several miniature projects (iterations), starting with the most basic iteration, each one getting more and more complex and feature-full then the last until you end up with the finished product. Each iteration should deliver a working product which can be assessed and analysed to gauge progress towards the overall project goals. Each iteration has a similar flow to a water-fall model, working in the order of requirements gathering, analysis, design, implementation, testing and delivery. When this approach is applied to game development, the early iterations are often used to find and perfect a gameplay formula for the game which is proved to be fun and enjoyable. These iterations often sacrifice immersive elements like detailed sound, art and animations (“programming art” is used, which is usually in the form of placeholder shapes/colours) to have more time to achieve the desirable gameplay and fun factor.

Iterative development reduces risk as early as possible in the project lifecycle and suits game development well. A major risk in game development is when detailed development is continued on top of poorly designed/tested gameplay mechanics which were not proven to be enjoyable or desirable to use by the end user. When the game releases, players find they don’t enjoy playing the game, thus it gets a bad reputation, it doesn’t sell and the project is a failure. It also helps maintain motivation and focus throughout development.

“Always having a product that steadily evolves and moves in the right direction is very important. The opposite -- working months or even years without seeing any progress -- is a certain project killer.” [1]

### 2.1.3 The Indie Approach

The indie developer typically is not limited by the rules laid down by an overruling authority. Indie game development is not tied down by the business requirements of a large publisher. What makes indie developers indie is the core focus on innovation, this freedom from strict business strategies and corporate conformity means they can take greater risks without suffering the consequences of failure to as great an extent as a large company with hundreds or even thousands of jobs and millions of capital at stake. The indie strategy towards developing a game with one or more truly innovative factors differs hugely between different developers and this is part of what makes them indie, there is no common path to follow, it is unique to the individual or sum of the individual’s personalities. [2]

To gain a more in-depth knowledge about how indie developers operate I researched the approaches of some successful leaders in the field. Jonathan Blow is a well-known name in the indie community, mainly for creating the indie puzzle game “Braid”. According to him, one should strive to make as many basic, stripped-down prototypes of gameplay ideas as it takes to find one that is truly engaging to play. He believes you shouldn’t get attached to an idea which doesn’t quite hit the nail on the head or perhaps is at best only good enough. It is easier to throw away a prototype of an idea if you haven’t spent very long creating it, which is exactly why it is encouraged to get to the point as quickly as possible with the prototype.

Jonathan explains to his audience at an indie games summit how he spent too much time on a detailed prototype which was based on a gameplay mechanic prototype that was proven to be successful. The Basic idea consisted of mouse movements used to simulate hand movements to draw out patterns which translated into magic spells. The original prototype was a static blank screen letting the player draw patterns and then stating what spell the pattern translates to and its properties. He believed this was a successful prototype and wanted to bring it to the next level to give the mechanic some context. He spent an inordinate amount of time creating a 3d world with characters walking around, background scenery, detailed animations, speech systems, a very complex hand/arm animation system which made the in-game character appear as though they were drawing out what you wrote and many other subtle features which helped the prototype exist as a complete package or iteration of what may be a full game. He explains that having looked back on this prototype, he could have could have just as easily gauged the fun-factor of this mechanic in a 2d scenario where all the time-expensive extras like character AI, 3d models, rendering and shading code etc. where left out, to give more focus to the prototype and save a large amount a development time. He originally visualized this game mechanic to be used in a 3d game so naturally the next step was to build the mechanic into a 3d environment but this is where he made a mistake, just because the full game will ultimately be 3 dimensional, doesn’t mean the prototype to evaluate how engaging its core gameplay mechanics has to be also. This is an easy trap to fall into and is very common; your visualization for the finished product skews the focus of your prototypes. [3]

The prototyping style which is generally encouraged among indie game developers is quick and dirty code. By this I mean the idea you are trying to implement in the prototype is disposable, so time is of the essence. Don’t let yourself get caught up in the details. This doesn’t just count for including unnecessarily fancy sound, art and animation, which is an obvious no-no for this type of prototyping, but even the coding style reflects haste in the coding style which disregards coding standards like readability, organization, decoupling, polymorphism and even optimization. The whole point is that you want to get your idea on the screen as quickly as possible so you can jump in and play around with it, tweak from there and evaluate it. This may be a hard concept to grasp for most developers coming from a traditional software engineering background and it is unlike most software projects where the priorities are the very things I listed above. This can even apply to the finished product to a certain extent; if the game isn’t optimized well but still performs to a high standard due to the hardware being so powerful, why waste time optimizing? If there are only a few people on team and they all understand your ad-hoc cryptic code, why waste time refactoring for readability? Often times an indie game is written with no intention of making a sequel or similar future game, so why put time, money and effort into constructing reusable and flexible code? These are all concepts which would not be suitable for use in large companies of 100 or more programmers, as the project would be a big incomprehensible mess, doomed for failure.

### 2.1.4 Feedback-Driven Development

Feedback is of upmost importance in game development. A Game as a piece of software has one key goal, to be entertaining. For a game to sell and make money for those involved in its creation it must attract people to buy it. What is the number one reason a person will buy a game? Fun, it is a commonly used word to describe the degree to which a player is enjoying the game; how engaged and entertained they are. Fun is nebulous and elusive term which is not easily measured and can vary hugely depending on a number of factors, one of the key factors being the individual. So how do you make sure your game is fun to play?

One quick solution is to play the game yourself and see if you enjoy it. Straight off the bat you can see this will not provide an accurate conclusion. First of all you are involved in the development of the game so you are close to the project, you know its ins and outs, you know the solutions to in-game challenges, you have mastered the controls and gameplay mechanics naturally as you have spent so much time on it. It is impossible for you to truly play the game as if it is your first time and having no previous knowledge of it. Secondly, you are just one person, with your own tastes and opinions, the game could be the most entertaining thing you have ever played but the question is, does it appeal to a wider audience?

A Better solution is to provide people an opportunity to playtest your game. You can gather simple statistics based on these playtests to see if the current build is moving in the correct direction in terms of its fun-factor. If the majority of the people you show don’t like it, then it’s a clear indication that something needs to be tweaked, modified or maybe even completely scrapped. Maybe you need to concentrate on a more specific target audience. If you have a sports game and the majority of the people you get to playtest the game are un-sporty and/or possibly non-techy types then your playtest will produce inaccurate results. Perhaps for a puzzle game you could factor in different age categories and IQ levels and take note of who finds certain aspects too difficult or too easy and adjust the level of challenge in the game according to the results so as to appeal to as much of the target audience as possible.

Sometimes you hit the nail on the head with the gameplay mechanics and have already dedicated development towards a particular gameplay formula, but there are so many bugs and interferences that a playtest produces poor results. Often this is the case with large multiplayer games and a proven model to deal with this is having a lengthy beta phase where at first a closed group of the public gets free access to the game and is encouraged to post up any bugs they find, fill out surveys and vote in decision polls. Once the build gets to a certain level of stability it moves to an open beta phase where it is freely attainable by the general public and as frequent as nightly updates are released based on feedback. It is an excellent method of crowd sourcing the finding and reporting of bugs. It works particularly well on large complex games where bugs can be found when carrying out an extremely specific and unlikely stream of events, which would take one man years of testing to find. It also works out as a mutual relationship between the developers and the players, the developers are happy to delegate work to the players and the players are happy to play a free early build of a game they may have been anticipating since its announcement.

During an Android game development presentation at Google I/O 2010, Chris Pruett talked about some interesting approaches to managing user feedback. He mentions how, when he worked in the game industry, they would bring children in to play their games and observe where they were having difficulties and take note of aspects which were not as obvious to them as they were to the game’s creators. He explains that although this worked well at the time, it didn’t apply as well for his individually developed game which was to be released through an internet downloadable game distribution system.

He decided to create an automated system which would get feedback from users around the world, with minimum effort on the part of the user. The system in its simplest form took note of the location of where a user died and had to restart the level. He implemented it in an unobtrusive way by having the option to switch on and off the sending of this anonymous data in the game settings menu. When he gathered the data for all users over a period of time he was able to construct a heat map which overlaid the game map and clearly showed the hotspots for user failure. Areas where a large proportion of users died could then be looked into and tweaked to make the solution more obvious to the user. Maybe a gameplay mechanic that the user should have recently learned was to be used for the solution to the problem; perhaps the introduction/tutorial for this gameplay mechanic wasn’t clear enough? This simple feedback system became a powerful tool in finding and diagnosing design flaws in the game. Something like this could easily be built upon to gather more detailed information about how users are playing your game; perhaps time to solve certain problems could be gauged? Maybe weapon shooting accuracy data could reveal that the controls are too sloppy for aiming? [4]

Chris goes into detail about user comments on his game marketplace page and how useful they are to aid development. He notes that the majority of the feedback he received in these publicly viewable comments was not helpful as people often tend to be more extreme in their views when writing anonymously on the internet. If they like the game they may be clouded by their excitement for it and overlook its flaws, even perhaps defend it at all costs, much like a dedicated fan of anything else. On the opposite end you can get people who hate the game for one reason or another and would not like to be seen giving it any merit at all, even if they genuinely like some elements of the game. He concludes that user feedback and bug reports are highly unreliable based on his experience, especially when they come in the form of a publicly viewable comment. On the other hand, unprompted, privately sent emails from users may provide more useful information. He mentions that sometimes you get the type of user who volunteers to help the developers of the game by sending bug logs, explaining how to reproduce problems, providing honest but constructive criticism etc. Something to take away from this is that it is important to keep communication lines open between end users and the developer, but even more important is that you recognize that certain types of communication systems may not encourage honest, useful feedback. Sometimes feedback should not be taken at face value, but rather analysed and used to recognize feedback patterns across multiple users. [4]

The type end user of a game is typically either a casual player of games or fits the category of “hard-core gamer”. Chris explains that casual players often just want to consume the content of the game, almost as if it were a movie or music, whereas the hard-core audience often seeks a certain level of challenge and longevity in the game. It is important to cater for these contrasting strokes of player. Maybe your game is targeted specifically at one or the other stroke, but quite frequently games attempt to cater for both using a number of methods. Most traditionally the choice of an “easy mode” or “hard mode” is presented to the player, but other methods are used also like dynamic difficulty adjustment, competitive elements (e.g. high scores), reward systems and sub-challenge systems (e.g. can be replayed, but with a different set of goals). Chris mentions that players generally don’t like to know they are being hand-held through the game. His dynamic difficulty adjustment system detected when the player is being blocked from progressing and would make subtle changes in the gameplay scenario to help the player continue progression and avoid becoming frustrated. [4]

Another aspect to take note of is how the game teaches the player something new. Traditionally a tutorial section would teach like a book, giving text or narrative to the player which provides the theory of how to do something and then let them put it into practice. This is slowly being phased out with a new more dynamic style of teaching, which is less intrusive, less cumbersome and is less patronizing to players. Usually it involves giving the players only minimal/optional information and leaving the players to figure it out for themselves, letting the design of the environment and a gradual introduction of new challenges hint at what needs to be learned. An example of this would be a scenario where the player must learn how to climb a ladder; you could have a note appear saying “press and hold the up button when near a ladder to climb it” or you could just have the player encounter a ladder in an uncluttered scene, where climbing it is the only way forward, because there aren’t many other options available, the chances are high that the player will figure out by themselves how to carry out the action. This must also be considered when trying to target both casual and hard-core players, as casual players are less likely to have played a similar game before. Perhaps they aren’t aware that the up button is generally used to climb objects in that type of game?

## 2.2 The Platformer Genre

### 2.2.1 History

A platform game (AKA Platformer) is a video game which is characterized by having the protagonist jump to and from suspended platforms or manoeuvre around obstacles. This is what a Platformer is at its absolute base, however most games build varying gameplay mechanics around this concept. Some common additional mechanics include bouncing off walls, using trampolines or similar to gain extra height in jumps and swinging mechanics. The majority of old games (before the 90s) in this genre were based solely on these core elements but more modern Platformers mix in mechanics like shooting, puzzling, driving and role-playing to provide a deeper experience.

Platformer games first started to emerge in the early 80s, starting out with very basic and limited gameplay. Commonly perceived as the first platform game, *Space Panic* featured a single screen with horizontal platforms, traversable by climbing ladders in between. There is some controversy though, over whether it can be considered a Platformer by modern definition, as the platforming mechanics where limited to simply climbing and falling, with no jumping, bouncing or swinging of any sort. This began a movement of single screen platform games, the most notable of which at the time was *Donkey Kong*. Created by Nintendo, it was released in 1981 as an arcade game. It was the first game to feature the ability to jump over obstacles and across gaps, making it the first true platformer. The game also featured the character Mario as the main protagonist, who is an icon of the genre and is perhaps an icon for games in general. *Donkey Kong* helped Nintendo secure a major foothold in the games industry at the time and allowed them to create 2 successful sequels *Donkey Kong Jr* and *Donkey Kong 3*. Nintendo went on to make *Mario Bros*. It became a smash hit, featuring two-player co-operative play, which was considered ground-breaking at the time. It laid the groundwork for other popular two-player co-operative platformers like Fairyland Story and Bubble Bobble.

By 1982 hardware and software advances influenced the introduction of transitional platformer games (pre-scrolling) which broke up extensive levels into single screens which could be traversed between. *Pitfall,* released for the Atari 2600, was one of the first games to execute this style and became one of the best-selling games on the console. Other advances in the same year included uneven terrain and scrolling pans during screen transitions. Many more transitional games followed, including *Prince of Persia* for the home computer, which chose the transitional style even though scrolling platformers where popular by this time (1989).

The first scrolling platformer game (camera follows the player through the level) was *Jump Bug* released in 1981, just 5 months after *Donkey Kong*. Developed by Alpha Denshi, it contributed to the evolution of the genre, but it was not a popular game. It took many years before scrolling platformers became the trend. By 1984 scrolling platformers became very popular and in 1985, the icon game of the entire genre, *Super Mario Bros* was released for the Nintendo Entertainment System. The game was bundled with the NES gaming console and got a place in the 1999 Guinness book of world records for selling over 40 million copies. It featured horizontal scrolling in a colourful world with more advanced jumping physics and a famous mix of low acceleration with high speed running, which made platforming more difficult and rewarding. By 1987 multi-directional scrolling became popular, it was not seen a major breakthrough but it became a distinguishing feature of the next generation of platformers (16-bit).

By the late 80s the transition from 8-bit to 16-bit games saw the release of 2 consoles, the Super Nintendo Entertainment System (SNES) and its close competitor, the Sega Genesis. At this time platformers where at the peak of their popularity and became the staple game genre, which could make or break a games console. Nintendo’s *Super Mario World* and Sega’s *Sonic the Hedgehog* started years of console wars where one company would try to outdo the other with the next sequel. The 16-bit generation was most notable for major improvements in the look of games and graphical quality and detail became of more importance. Disney’s *Aladdin* famously set the standard for 16-bit animations in platformers in 1993.

The end of the 16-bit generation saw the release of the Sega Saturn, PlayStation and Nintendo 64. These 32-bit consoles introduced the 3rd dimension, which was seen to be the killer of the platformer genre at the time. The difficulties of adapting traditional platformer gameplay to 3D led to the concept of 2.5D, which was somewhere between 3D and 2D, gameplay was limited to a 2D side scrolling plane, but featured the visual flash of 3D.

At the peak of the 32-bit generation developers had gotten the hang of making games in 3D and many forms of 3rd person games like RPGs, action-adventure and Shooters caused ambiguity between genres. Although platforming elements were still prominent in a lot of these games, we saw the decline of the platformer game, i.e. games that were specifically marketed as platformers. Because it was so difficult to create a successful 3D platformer, the genre lost the market dominance it had in the 8 and 16-bit days, with a select few notable titles including *Super Mario Sunshine* for the Nintendo GameCube, *Crash Bandicoot* and *Spyro* for the PlayStation.

The successor to the PlayStation, the PlayStation 2 gained market dominance in 2001, this was around the time where cutting edge games tried to win over gamers with realistic and lifelike graphics. Developers became obsessed with trying to create deep, authentic experiences across many 3D genres. This would often be a deciding factor for the game players, which was bad for the platformer genre. By 2002 platformers had only a 2% share in the market, as compared with a 15% just 4 years ago, in 1998. Despite this, games like *Jak and Daxter* and *Ratchet and Clank* continued to push the boundaries of the platformer, featuring exploration, shooting and RPG elements in vast, detailed 3D environments.

2005/2006 brought us the current generation of consoles (soon to be succeeded), the Nintendo Wii, PlayStation 3 and Xbox 360. At this stage the early release of the Xbox 360, its cheaper price tag and a poor initial game line-up for the PlayStation lead to the decline in Sony’s dominance of the console market. This generation most notably evolved the network integration side of games, with a huge rise in the popularity of online multiplayer games. This ultimately led to the decline in popularity of single player games, especially for the First-Person-Shooter genre. At this stage, popular 3D platformer games were rare.

In recent years a significant niche audience has grown who seek nostalgia platformers, modern takes on the traditional 2D platformers of the hay-days. These games didn’t contain the same depth and complexity found in the typical AAA (big budget) releases, so they became popular as network distributed titles, with small price tags. The side scrolling platformer has also been a popular medium for modern Indie developers with high-grossing innovative releases such as Jonathan Blow’s *Braid*, Playdead’s *Limbo* and Phil Fish’s *Fez*. [5] [6] [7]

### 2.2.2 Sub-Genres

As you might expect, there are many sub genres to the platformer, it is a flexible genre which can easily be mixed with many other genres to provide an original and innovative gameplay formula. Below I will list the most well defined sub-genres.

**Hop and Bop –** This describes a platformer where enemies are primarily defeated by hoping on their head, much like the Mario series. These are predominantly 2D as there are problems in transferring this gameplay mechanic to 3D due to depth of field and 3D coordination having a negative effect on player’s judgment. These games are often characterized by featuring colourful, cartoon-like aesthetics and anthropomorphic animals.

**Isometric Platformers** – These games presented a seemingly 3D environment in 2D isometric projection. They were rare but suited early consoles which didn’t have the hardware capabilities to display true polygonal 3D graphics. An early example is *Snake rattle and roll* for the NES. A popular, more refined example is *Sonic 3D* for the Sega Mega Drive.

**Action-Platform Games** – A somewhat broad category, Action-platform games define the fusion of platforming fundamentals with action adventure and/or RPG elements. These games would typically feature a combination of exploration, progressive elements like gaining new abilities, collectables and a customisable inventory. These games usually have more advanced mechanics for dealing with enemies (as opposed to Hop and Bop) or may not feature enemies at all (for example: *fez*). The original *Ninja Gaiden* series on the NES is a good 2D example. The long awaited 3D sequel to the original *Prince of Persia*, *Prince of Persia: the sands of time* set the bar for 3D action platformers, featuring acrobatic and original platforming mechanics mixed with a deep combat system and a ground-breaking (at the time) time-rewind mechanic.

**Run and gun platformers** – One of the earliest examples of the run and gun platformer was Konami's *Contra*, for coin-operated arcade machines. These games give the protagonist the ability to fend off enemies from a distance with some sort of gun or projectile. They normally put less emphasis on challenging platforming, but more emphasis on shooting, aiming well and picking the right weapon for the right situation to survive and progress. Some early examples only allow shooting straight ahead (horizontally across the screen), but most allow multi-directional shooting, for example the popular *Metal Slug* series. This sub-genre has very strong arcade roots, they are known for being quite difficult, which was great for arcades back in the day, because it meant people would need to put in more money to beat the game. Run and gun platformers typically feature little of no exploration elements and no backtracking, unlike some similar shooting and platforming games which wouldn’t fit into this category, for example the *Mega Man* series.

**Puzzle Platformers** – These are platformers where the primary focus is on solving puzzles. Many games feature puzzle elements. Often they are very basic puzzles and do not feature very frequently. These games would not fall into this category. This genre is very popular with handheld consoles and in recent times, mobile gaming (smartphones). An early 2D example is the *Wario Land* series, a spin-off of the iconic *Mario* series which focused more on puzzles. It is very rare that you would find a 3D game in this genre, even more so from the first person perspective normally associated with first-person-shooters; however Valve’s *Portal* managed to pull this off with great success. It pushed the boundaries of innovative design and technical achievement with a portal mechanic which allowed the player to view the immediate area from a different perspective to the one they currently exist in and created innovative puzzles around this mechanic.

**Cinematic Platformers** – For some platformers the focus may be on the storyline, realism and/or presentation. They would fall under this category. The gameplay elements featured in these games can vary but are often simplified to place more emphasis on the cinematic elements. A common example of this simplification would be situational puzzles where the puzzles are not based on a set of mechanics which the player learns, but may be unique to each situation. An example of this is breaking out of a suspended jail-box by rocking it left and right repeatedly (from *Another World*). A popular example of a cinematic platformer, *Flashback*, featured realistically proportioned humans with smooth, lifelike animations and detailed environments.

**First-person Platformer** – This is perhaps the rarest of all the sub-genres I have mentioned. The first-person platformer is restricted to 3D, platforming focused games, where the world is viewed through the eyes of the protagonist like an FPS game. There is only one notable and successful title which clearly fits this category; DICE’s *Mirror’s Edge*. The game was highly influenced by Parkour which was a recently popularized sport where-by people move quickly and smoothly through varying obstacles using acrobatic techniques. The first person perspective helped increase the immersion factor, making the player feel like they were there. This was executed so well that some players have even experienced nausea and vertigo while playing the game. [5] [6]

## 2.3 Similar Projects

### 2.3.1 “*Generic*” by PHL

A developer from California US, known by the name PHL, has spent many years up until the current day developing his platformer game called *Generic*. It is a similar project to mine in that it is a retro style 2D platformer in a colourful cartoon-like world. He developed the first iteration of the game early in 2007 for a game design class at university. He developed it using C# with XNA game studio, which is a framework for creating games for Microsoft systems. He made the game as a solo class project, but with the intention to expand the game in the future in his spare time. In 2009 he created a blog to share information on the game’s development and post progress updates.

The first iteration of the game featured a little blob shaped creature as the protagonist and a jungle themed environment filled with unique-looking enemy creatures and environmental dangers including pitfalls and spikes. The world was scrollable in all directions (vertically and horizontally) with the player sprite (animated texture) always being the centre of focus for the camera. The game featured a traditional tile-based level system, whereby the playable (collide-able) area of the level was made of same sized tiles fixed to a grid. No form of slope was featured; collision was restricted to walking along even ground (horizontal), which could change in altitude but would still remain even and hitting walls (vertical). This suited the game fine as the levels were traversable in all directions with quick and snappy movement for dealing with gap jumps and enemies, slopes may have interfered with this simple retro style of layout. The game also featured parallaxed backgrounds which involved several layers of background each moving with the player at a slower and slower rate the further back the layer. This method is commonly used in 2D platformers and helps give an effect of depth and distance to the background sprites, be it a mountain miles away or buildings a few blocks away.

The gameplay mechanics were relatively simple but effective, with the typical movement controls of left button corresponding to moving left and vies versa for right and button for jump. Enemies came in 2 forms, the first being the typical unintelligent foe which moves left and right in a fixed pattern. These enemies can sometimes come in packs. The second type being a dinosaur-like creature which moves only by hopping in varied amounts while homing in on the player. These are the more intelligent enemies as their path is not as predictable and they take more hits from the player to defeat. The player can use 1 of 2 attack types to defeat enemies. The first being a bop, which is a sort of short range lunge and can be used horizontally and also upwards in the case that an enemy jumps over the player. The second attack type is a slide which gives the player a sudden and short boost of acceleration either left or right and deals damage to enemies as you pass them. This is particularly effective against packs of enemies. Enemies take 2 – 3 hits with either attack to defeat.

The health system was also simple, with the player having a visible health bar at the top left of the screen containing 6 segments. When all segments have gone the player dies and must restart the current level. 1 segment is lost for each hit taken from an enemy. Health can be replenished by using a heart power-up, of which many are scattered around the levels. The overall structure for the game is simple. A title screen is presented with options to start the game or read info about the game. There are 5 levels which are separated by brief transitions and a boss (high difficulty enemy) at the end of the final level. Defeating this boss completes the game.

In summary this first iteration provided simple, traditional and proven gameplay mechanics which provided a balance of difficulty and reward making for a fun game.

This will be about the scale of game I will be aiming for, for my project, as it is important to keep my goals realistic.

By 2009, from working on the game in his spare time, PHL had overhauled the look and feel of *Generic* and had added several significant new features to increase the fun factor and make the game a more deep experience. The first major addition is a new co-protagonist character called Gene who tags along with the player in single player mode but can also be controlled by another person to create a 2 player co-operative experience. Gene has unique abilities (attacks), which can have their own advantages in certain situations. New artwork was created for the levels (tile-sets, layouts and backgrounds) to make each different level unique. Special environmental features were added including explode-able walls which open up new areas and under-water areas which effected gravity and movement. New enemies where added to the game with more advanced attacks and animations and to counter this new player abilities where added, which were attainable by collectible power-ups. The user interface (or heads up display) now featured a mini-map which helped for navigation and a score display for the newly added score system.

During the transition from the first iteration this more full featured game, PHL decided it was best for the project to be ported from C# over to C++. He felt the limitations of the C# language with XNA framework would eventually make it too difficult for him to expand the game in the direction he visualized for the distant future so he decided to halt progress on development and spend a significant amount of time porting over the game code to C++. For his type of project, with no real time-constraint, this was an intelligent decision as it may have been several times more complex and taken several times longer to port it over a year or 2 later in development.

From 2009 through till the current day, in his spare time, PHL has been constantly implementing new features to the game, making adjustments to gameplay mechanics, fixing bugs and adding new content in the form of levels, enemies, power-ups and goals.

Major improvements include an RPG-like inventory system for power-ups/abilities/collectables, new enemy types, several boss changes, effects like rain and screen rumble, two-player mode over a network and more detailed sound and music.

The project is still going strong, all thanks to keeping development at an appropriate pace from the start. This case study has shown me a successful path for developing a game from scratch, specifically a platformer. By analysing the order in which features have been implemented and the time between each, I can get a good sense of direction for my own project. This has thought me how important it is to get the core mechanics and structure of the game right, as early as possible in development. His project is a great example of this. The latest version of his game, although much more feature-full and complete then the iteration of the game from 5 years ago, still operates on the same core mechanics as the original class project. [8]

### 2.3.2 Physics Engine Based Platformer

While researching design and implementation of platformer games I found the most common method used was a tile-based system where levels were made up of a grid of tiles which determined the collision bounds for the game. Another method which can be considered is a more modern one, making use of open source physics engines. These normally come in the form of a set of APIs which allow the creation of bodies or shapes that can have their physics properties set. Normally these engines will try to alleviate difficulties involved in collision detection and reaction. While I have found that these engines will save a lot of hard work and make the level design more flexible, they also have their problems when being used to make a traditional 2D platformer. The problem is that many physics properties in a platformer, which are normally hard-coded in, are not realistic. The most common example is in-air control of the protagonist. Physics engines normally try to simulate real life physics and therefore cause problems when trying to create an unrealistic behaviour such as the commonly implemented ability to change direction in mid-air. I have looked at some similar projects which have chosen the physics route, to see how they handle this problem.

Eduardo Miranda has implemented a solution which tackles the limitations of using physics engines to implement traditional 2D platformers. Large images are used for the ground tarrain and attached (fixated) onto a physical body (shape) which matchs the shape of the tarrain, for example a sloped piece of ground would have a sloped rectangle body sitting over it, perfectly in line with the image. The physics properties for these bodies are set so that they have relativly high friction, little or no restitution (bouncyness) and are static, ie. do not move. For the player a sprite must be attached to a body. Naturally you may think of creating a simple rectangle body for the player, but the problem here is that a rectangle with a flat bottom will never be able to move on flat ground, as this behaviour breaks the laws of physics. Because simulating a realistic walking motion with separate bodies for legs attached to a motor of some sort is extremely difficult to do effectively, Eduardo has put a rectangle on wheels. The wheels have a motor attached so that the body is controlable (essentially like a car), but the player wont see this, they will just see the animated sprite.

For the player to jump a spring mechanism is used, where the bottom part of the player body can suddenly exert force downwards towards the ground, resulting in the whole body to be launched into the air. Natural variables make the jump height, speed and distance much less predictable. This is another shortcoming of using “real” physics for a platformer as normally jumps are sized exactly to work with the height and distance seperation of the platforms used to traverse a level. Predictability of the jump allows more control in the hands of the player which in turn lets the player learn and master the skill as aposed to relying more on luck or an unrealistic degree of precision, which would result in frustration.

Eduardo’s prototype showed me the difficulties in using physics engines to implement a traditional style 2D platformer. These difficulties are often overlooked when compared to the great amount of advantages, in terms of level design and collision detection, over a pure tile-based, hardcoded physics solution.

## 2.4 Gameplay

### 2.4.1 Gameplay & Game Mechanics

First we must define what gameplay actually is. Wikipedia defines gameplay as:

“*T*he specific way in which players interact with a game, and in particular with video games. Gameplay is the pattern defined through the game rules, connection between player and the game, challenges and overcoming them, plot and player's connection with it.” [9]

But during my research I have found it to be quite an ambiguous term, for which many different definitions can be found. A few notable definitions include:

*"A series of interesting choices."* - Sid Meier

*"One or more causally linked series of challenges in a simulated environment."* - Ernest Adams and Andrew Rollings

*"A good game is one that you can win by doing the unexpected and making it work."* - Dave Morris and Andrew Rollings

Game mechanics are constructs of rules intended to produce enjoyable gameplay. Essentially, good gameplay is made up of one or more game mechanics, which work together to produce an enjoyable and engaging experience for the game player(s).

### 2.4.2 The Science behind “Fun”

As discussed before, the main goal of a game is to produce an engaging and enjoyable experience. A good game will stimulate the player to continue playing as it is “fun” to continue playing. Games achieve this in many different ways, usually through effective game design, which is the process of producing “fun” gameplay. “Fun” is not easily defined and one person’s concept of “fun” can be much different to the next person. The term describes a feeling of stimulation and engagement at a physiological level. In this section I will explore the science behind the production of this feeling.

It all starts with the basic human needs which have been developed over time since the beginning of human existence. These basic needs are closely related to our survival, existence and evolution. Although it may not seem like it, in some way everything we do in our daily lives is in some way directly/indirectly related to these needs. Our needs come in two contrasting forms, deficient needs and growth needs.

Deficient needs are those which are closely related to our survival and can be put into the following categories; physiological needs (eating, breathing), safety/security needs, love/belongingness needs and esteem needs (self-esteem and to be respected by others). Growth needs are more closely related to self-improvement and evolution. They can be categorized as follows; cognitive needs (knowledge, exploration), self-actualization needs (realize potential) and transcendence (realize potential in others).

As humans we are programmed to concentrate on fulfilling the needs which are not fulfilled. The number one priority is survival, so the lower needs (deficient) must be fulfilled before we can move our attention to our higher needs (growth). An example of is a competitive cyclist taking a break from his sport for a while to let an injury heal, the physiological need takes priority over the growth needs.

Evolution has developed our brain to encourage good/healthy behaviour. Dopamine is a neurotransmitter associated with pleasant feelings. Our brain releases dopamine when it senses good behaviour, so that repetition of this good behaviour is encouraged. A simple example is when you are thirsty, drinking water makes us feel better and therefore we can see that having a drink is good behaviour in this scenario. Lower, physiological needs are quite simple and easy to understand, but higher needs related to psychology are much more complex. Because everyone has their own opinion on what their psychological needs are, there can’t be a defined set of behaviours which are rewarded like with physiological needs. From an objective perspective it doesn’t matter what, how or why we do something, as long as we are doing what we feel is the right thing and getting a good result. This is all that is required to get a release of dopamine and fulfil our psychological needs. Take for example winning the lottery, there is no actual difference in the initial feeling someone gets when they have won and when they think they have won but they actually haven’t.

On a personal level, the ultimate goal of all our activities is essentially to get pleasure and avoid pain. The amount of pleasure/pain which is derived from an activity can be judged by answering the following questions: What have I learned? Was it interesting to me? Was it okay for me to do? Did I do well? In modern society, due to the abundance of options and vague boundaries between right and wrong, it is often very difficult to decide what we should be doing and ever more difficult to answer the above questions afterwards. Games, not just video games (e.g. sport, board games), are designed to ensure pleasure in the person(s) participating. They all have set goals and rules which govern decisions. These rules set boundaries which constrain the options available, making decisions easier to make and outcomes more easily measurable.

Roger Caillois is a theorist who came up with a list of the types of experiences present in games. The first and most common is Competition, where pleasure is derived from developing the skills needed to beat the opposition. Secondly is Chance, often associated with gambling, pleasure is derived from attempting to predict an outcome. This can give the illusion of controlling the future. The third is Vertigo, commonly experienced on theme park rides, it disrupts the normal perception of the world resulting in a pleasurable state of dizziness. The last is Make-believe; these experiences derive pleasure from existing in an alternate reality, one which is not bound by the constraints of the real-world, like a dream.

*“Pleasure comes in the area between boredom and anxiety at the distinct moment where our options are in line with our abilities”*

There are a number of factors which can determine how pleasurable an activity is. With games, one of the key factors is the level of challenge. Challenges require a certain skill to complete and if we don’t have the required skill, the activity can feel frustrating or meaningless. Pleasure comes in the area between boredom and anxiety at the distinct moment where our options are in line with our abilities. The level of our skill must be close to the required level of skill, be it under or over this level. If reaching the goal is too easy then we get bored, if the goal is much too difficult then we become frustrated and find it hard to derive pleasure. Level of concentration in carrying out the activity is also very important. When a person’s skill is required to perform a task, they become absorbed in the activity and regress into a state where their actions are spontaneous and they are immersed in the activity. To remain in this state it is important that distractions do not occur. An example of a distraction in a video game could be poor/unintuitive controls; they make it difficult to do what we want to do in the game, therefore making us focus more on the controls then the game. Feedback is another important factor, as we need to be informed of our progress towards our goal so that we can analyse what we are doing right and wrong and learn from our mistakes.

A pleasurable activity can distract us from everyday worries; the level of concentration required can often switch focus from negative aspects of life to the outcomes of performing the activity. Participating in a game can be a form of escapism. It is also important that the activity offers much control to the participant. Games which offer very little control require less of the person’s skill and therefore are less engaging. Another important aspect to note is our perception of time; this is normally based on changes occurring around us. When concentrating fully on an activity there is no room to register any of these changes, therefore hours can feel like minutes. This leads to the phrase “just passing time”.

In terms of video games, the core activities include learning, interaction, recognizing patterns, understanding how they fit together, exercising control over them and improving our skills in doing so. This core is typically mixed in with many forms of media like art, sound, film and text which act as a concrete supporting role. These give the game context and meaning which humans can relate to, understand and enjoy. A well designed game caters for its target audience’s needs, desires and abilities. It should be designed in such a way that the player feels comfortable and in control. The player should be given a chance to learn a skill before being challenged to use it to complete a task and they should be rewarded for being successful. Another major factor to consider is that humans respond to change, they are excited by new things, and this is why innovation is so important when designing game mechanics. There are a number of methods commonly used in the games industry which try to maximize the pleasure the target audience gets out of a game. I will outline these below.

Game design is similar to attempting to buy a present for someone when you have no idea what they like. You are trying to create a gameplay experience which your target audience will like but you and quite often the players themselves don’t know what they like. A method to tackle this problem is creating user profiles, which involves taking a portion of individuals from your target audience and collecting each of the individual’s personal information such as age, gender, what they are good/bad at, what games they play, how often they play and what kind of music they listen to and so on. By having a varied set of profiles represent your target audience you can get feedback and make changes based on how the majority of them feel. A lot of the time people don’t know what they want until they see it, this way you can guess what they want, show them and then adjust based on feedback.

The term “Game width” describes how broad an audience a game is targeting, the range of needs and desires which are catered for. Wide games cover a broad range of interests and when executed correctly have more potential customers then a game which is designed to target a more niche audience. An example of a successful broad game is the *Grand Theft Auto* series, in which you can do many of the activities of real life inside a fictional world, for example walk around, race a car, pilot a plane, shoot weapons etc. On the other hand a game series like *Fifa* would not be described as broad, as it is a football game and the majority of people who would enjoy this game would need to have some interest in the sport of football. Broad games do have their disadvantages though. Trying to create one is like trying to be a jack of all trades. In a lot of cases, instead of doing a few things very well, they do a lot of things poorly.

Much pleasure can be derived from the process of learning. The desire to understand and seek knowledge is the reason we enjoy hearing stories and watching the news. A common way of deriving pleasure from this desire is to present something which we don’t fully understand but are interested in. This then encourages us to try to find out more about it and ask ourselves questions like why? how? when? An example of this would be a movie where someone has been murdered and clues are shown throughout the film as to what happened and eventually revealed at the end. In games this desire can be used to maximize pleasure by ensuring the player is always learning new things throughout the game, not just at the start. A common mistake made in games is too much hand-holding of the player, telling them exactly what they need to do, not letting the player discover and figure things out for themselves. When a player overcomes a challenge using a solution which did not seem obvious or required a level of skill which was significantly higher than required before, the feeling of reward is much greater than if they were guided through the challenge.

Modern games can be compared with movies in that the quality of the visuals and sound have improved to such a level that they can achieve a realistic, immersive experience for the player. Now, more than ever, we can take advantage of this immersive effect by creating emotional impact on the player, just like they experience when watching a movie. In movies a common effect to maximize viewer emotion is to match an emotional peak of the backing music with an emotional moment in the story. The pleasures experienced when viewing the moment and listening to the peak of the music are combined to create stronger effect on the viewer. This concept can also be applied in many ways with games, but it is not just restricted to a combination of story and music. An example would be in the *Need for Speed: Underground* series, which introduced a blurring effect when driving at high speeds, which was intensified more, the faster you went. Other arcade racers expanded on this by adding gradual fading out of music and sounds other than the engine and gears and the loudening of the sound of the player’s heart beating, to simulate the feeling of adrenaline at when driving at dangerously high speeds.

In conclusion, games can be fun to play for hundreds of reasons, most of which we simply don’t know, so the best approach to creating a fun game is through trial and error. Having an insight into how the human mind works when playing games will be of great benefit when attempting to design game mechanics which can produce good gameplay. [10] [11]

### 2.4.3 Game Mechanic Categories

Game mechanics commonly used in video games can be categorized based on the type of psychological effect they have on the player. I will outline some notable categories below.

**Behavioural Momentum** – This is the tendency of players to keep doing what they are doing. They will only continue doing something if they feel it was useful and/or meaningful.

**Blissful Productivity** – This is the idea that when playing a game makes the individual happier working hard then if you were relaxing and doing very little. This occurs because humans are optimized to work hard and do meaningful, rewarding work. An example of this is that many dedicated *World of Warcraft* (a large scale multiplayer RPG) players would easily spend more hours playing per week then they would be working if they had a part-time job. This is because they enjoy playing the game but what they often don’t realize is that while they play they are actually working their brain just as hard as if they were doing real work.

**Cascading Information Theory**– This is the theory that information should be given to the player in the minimum possible snippets, to let the player gain the appropriate level of understanding at each point during the game narrative. This essentially means that overloading of information should be avoided and a gradual, step-by-step learning process should be encouraged.

**Countdown**– A time constraint is one of the most basic methods to introduce challenge to an activity. This will generally cause the player to become more concentrated and reactive as time runs out and the pressure increases, often leaving a positive climax of relief if the task is completed just in time.

**Discovery**– Naturally, humans get a kick out of exploring and discovering something new, by themselves, with little or no help. Often in game design discovery is rewarded with something. Perhaps the player finds a secret area in a level which when searched, a bonus item can be found.

**Reward Schedules** – These are timelines and delivery mechanisms where players are rewarded regularly for doing different types of activities. For example, these rewards can come in the form of extra health, new items or even a simple virtual badge which states that you have achieved something and you can show it off to your friends. These schedules can come in 2 forms; fixed ratio and variable ratio. Fixed ratio means rewards are given after a fixed number of actions e.g. kill 10 enemies to unlock the next level. This generally causes a burst of activity closer to the point of being rewarded. Variable reward schedules give rewards after a roughly consistent but unknown amount of actions. This plays on the idea that the player becomes more active and engaged when they are about to achieve a reward, as there could always be an award after the next action.

**Loss Aversion** – This influences the player not by reward but by not instituting punishment. A simple example is avoiding falling into a pit in a platformer, as doing so will result in death and having to restart the level (loss of progress). This can be very effective when played against a positive reward, for example the player must risk their life in order to get to a useful power-up.

**Ownership** – This is a powerful dynamic which creates loyalty. The player may be presented with many options with contrasting characteristics, but the player must only pick one (or a select few) and the choice the player makes may have long term implications. Because the player doesn’t have much freedom to change their mind about the decision, they must put more thought into it, thus making the player grow to be attached to the option they chose. An example of this is choosing a character in *World of Warcraft*; the player may use the same character for years and become attached to it over time.

**Virality** – This describes a game element which requires or encourages play with multiple human players to achieve something. Often co-operative multiplayer games will encourage teamwork to complete tasks, which can be a great source of pleasure and social interaction. Competitive multiplayer lets players or teams of players test their skills against one another, often in under a set of rules which balance the difficulty for all parties involved. [12]

## 2.5 Game Art/Animation

Games can be made up of many forms of media. The decision of which types of media will and will not be included in the game is entirely up the developer, but one element which is critical for modern games is game art. This describes all graphical and visual elements of a game, without which it can be very difficult for humans to perceive what is happening in the game. Some rare games don’t involve visual art. They may be purely text-based or even sound-based, but for most games this is a critical element. It allows humans to relate to what is happening on screen and can help them guess or quickly learn the behavioural characteristics of something which they recognize from real life. As developing heavily on this element of my game is outside the scope of my project, being from a software development standpoint, I will not go into as much detail on it.

Designing game art is not the same as designing any other kind of art. When creating art for games it is important to consider the impact it has on the player and how they perceive it. Attributes like the position of something on the screen, it’s size, it’s colour palette and it’s animation all have a major effect on how a player perceives it. Take for example the difference in the look of the foreground of a platformer and the background. The background may be more faded, with less clear lines and dampened colours, whereas in contrast the foreground, which is the playable (collide-able) area, may feature more sharp and high-contrast textures. This makes it clear to the player what is in the background and what is in the playable area, so they don’t get confused between the two. An example of possible confusion is if they tried to walk on a block that was seemingly in front of them but was actually in the background, they may be surprised to find they fall right through it.

How the player perceives the world they play in has a huge effect on how hard or easy the game is to play. Generally, it is more appropriate to introduce challenge in a game by requiring skills and/or knowledge from the player via gameplay mechanics, rather than by creating ambiguity through poorly designed visual game art. An example of this is if the character on-screen which the player is in control of looks the same or similar to many other things in the world. Perhaps the character doesn’t visually pop-out enough, the player will easily get confused as to who they are controlling, leading to frustration.

Animation must also be optimized to suit gameplay. It is important to aim for the healthiest balance of visual appeal and compatibility with the mechanics of the game. The timing of an animation can have a critical effect on how the game plays. An example is if the player wants to open a door but then suddenly decides they must defend themselves from a nearby enemy, the animation for opening the door should immediately stop, letting the player deal with the threat. If the opening door animation had been a lengthy, cinematic, highly appealing one and the player had to wait till the end of the animation before regaining control, then this would become the reason for the player’s death. This leads to high frustration and the appeal of the animation would quickly be lost.

As well as appealing to players particular interests, visual game elements can also increase the immersion factor, making the player feel like they are in the world they see on screen. This is effectively the make-believe experience which was mentioned earlier. As humans, we love to re-enact activities which we can’t do in normal circumstances, we take great pleasure in imagining an alternate reality. Modern 3D graphics technology has enabled us to simulate deep, realistic experiences through games. Rally Simulator game *Richard Burns Rally* from 2004 famously set the bar for realism and the immersion factor in driving games. The game featured realistic graphics, physics, sounds and a cockpit view which went hand-in-hand with a wraparound screen (allowing peripheral viewing) and real steering wheel, gear and pedal controls. Some Simulator setups even went as far as having hydraulic movable cockpits which shacked and vibrated in sync with the car you were driving in the game. This combination of synchronized feelings greatly heightened the sense of realism and even produced adrenaline in some players.

As digital 2D and 3D graphics technology improved and games became more visually appealing and realistic, from approximately 2000 onwards, their place in society began to rise. No longer a niche activity, the games industry developed to a similar scale to that of movies and music. With this new found recognition came the controversy of weather games are art or not. Developers pushed the boundaries of graphical realism so much in recent years that a screenshot of game could often be mistaken for a real photograph. Animations in AAA (high-budget) games are now often performed by real actors with special suits that translate physical movement and expression into digital data which can be used as guidelines for animators. Commonly accepted art forms such as novels, paintings, sculptures, film and poetry can all be present in games, however one critical element in games is not present in any of these art forms and that is interactivity. This is often the source of the controversy in this argument. The fact that video games are a somewhat recent form of media (around 30 years old, as opposed to Filmography which is more than 100 years old) also contributes to this controversy, due to society’s natural resistance to change.

Due to the focus of my project being on the software and programming side of game development, I will likely be using programmer’s art, which is visual art that is basic (often comprising of simple coloured primitive shapes) and is the minimum needed to test gameplay elements. Although my game may consist of manipulating basic, possibly unappealing art assets, it is important that I have this background knowledge so that I know how the visual elements in my game affect how it is played. How on-screen items are perceived by the player will have a great influence on their gameplay experience. [13] [14]

## 2.6 Audio

Audio has been a critical element in games since the days of hardware limited chip-tunes and its importance is often overlooked. If you were presented with a game with standard sound effects which fit the on-screen action you would probably not pay much attention to these sounds, but had there been no audio at all you would have sensed that the game didn’t quite feel right. Audio will never be as noticed and praised by game players as visual appeal, but it is an essential element without which gameplay feedback would be reduced and the effect on the immersion factor would be detrimental.

Sounds provide emphasis on visual feedback or in some cases are the only forms of feedback, for a specific action. An example which commonly makes use of this is game menu navigation. Each time the player selects a different option, that option is visually highlighted to make it standout as being selected. This feedback will normally be backed up with a clear, abrupt sound to emphasize the change in selection. At a basic level, each type of sound should relate to particular action in the game. The easiest way to relate a sound to an action is for it to sound like the visual entity being represented, for example a gun being used on-screen should sound like a gun. This can seem quite obvious, but what if the game world takes place in an abstract, perhaps alien world, with entities you don’t recognize from real life. Humans naturally categorize different sounds, recognize sound patterns and relate them to an action. A weapon with inconsistent sounds, perhaps it sounds like a gun most of the time, but the very odd time it makes a ringing sound, is an example of poor sound design. When the sound changes we are alerted and immediately wonder what action does this new sound relate to? We try to match the sound with an on-screen action and it is not obvious to us that it is related to the gun. This can lead to confusion and possibly annoyance.

Sometimes sounds can help a player focus on the important aspects of a game without having to look out for a visual indication. For example in many driving games manual gears can be used and the player must know the best time to go up or down a gear. Usually the RPM meter is represented on screen and can be used by the player to know the optimum to shift a gear. Constantly glancing at this meter can be distracting for the player. The use of accurate and realistic engine sounds enables the player to gauge when a shift in gear is needed without having to glance at the meter.

A commonly used effect (especially in 3D games) is special awareness using sound. Things that are far away sound quieter then things that are near. Furthermore a realistic 3D effect can be achieved (best suited to surround sound systems) where sounds can seem as if they are coming from any specific angle in 3D space, for example a helicopter hovering over you coming from far behind, to far in front. These techniques help make the player feel like they are in the game world, especially in 3D games viewed from the first person perspective. Many player skills require special attention the location of where a sound is coming from, for example in modern multiplayer first-person-shooters, players can hear enemy player’s movement coming from behind them as they sneak up on them.

Many more audio effects can be used in games such as atmospheric sounds, vocals and environmental sounds. These types of audio contribute significantly to the level of realism and/or immersion in the game. The *Battlefield* series is famous for having outstanding sound design. In 2007 they introduced a new sound technology they called High Dynamic Range Audio which emphasized loud and important sounds. In a simple tech demo they showed a car parked in the foreground with its radio on. The radio was quite low in sound, but because we were close to it and there were little other sounds, we could hear it load and clear. A tank in the background fires a shell round at the car in the foreground. To emphasize the loudness of the explosion when the shell hits, the sound of the radio is completely drowned out, and your attention is directed towards the explosion. As the sound of the explosion dies out, the sound of the radio fades back in to full volume again. This effect greatly enhanced the audio realism and gave the impression of a higher range of volume in 3D space.

Other notable audio effects present in modern games include; the muffling of sounds coming from inside a busy room when the player is outside, followed by the gradual clearing and loudening of the sounds as the player enters the room; the distortion of sound when a player in a first-person-shooter is heavily injured from explosions; echoing of sound when in open and sound reflective spaces.

Last but not least in terms of audio is music. It is hard these days to find a game without any form of music. Much in the same way as in movies, music complements the on-screen action and can often contribute greatly to player emotion, just like a score. As mentioned earlier, music can be synchronized with the on-goings in the game, to emphasize emotional moments in the plot. Music can add character to a game, even it is a short simple loop, for example the famous *Tetris* theme. Ironically, despite audio often taking a back seat to visuals as mentioned above, it is usually the music and sounds of a past game which conjures up nostalgic thoughts and memories. Music is sometimes used in gameplay elements also, a famous example being *Bust-A-Groove* from 1998 where players had to match combinations of button presses with the rhythm of the music.

Similarly to art design, audio design will not be the main focus of my project, however there won’t necessarily be a complete absence of audio either. I learned that audio is important to emphasize feedback to the player; be it a loud notification indicating the unlocking of a reward or the subtle click heard at the end of a gun’s cartridge, letting the player know they need to reload. [15] [16]

## 2.7 Storyline/Setting

The truth is, some games have plots and others don’t. A plot isn’t always a crucial part of a game and the importance of the plot is usually dependent on the genre of the game. There are some genres of games which simply aren’t compatible with a storyline. These are typically abstract games like puzzle games where players manipulate shapes which don’t have meaning in the real world. In the early days, the platform game was the most common genre in which a story was expected. These days, stories are told in many types of games including first-person-shooters, action adventure games, horror games, RPGs, strategy games and even driving/racing games.

Games in some genres typically have weak plots which mostly exist to give the appearance of a complete package. They are typically weak due to the scope within which the story is bounded by, for example a story in a racing game is typically going to be about an individual’s racing career, which can be hugely limiting in terms of storyline creativity. Because game genres are typically categorized by activity or functionality many games in a genre can feature very similar storylines, due to innovation in this area being very difficult to achieve without negatively impacting gameplay.

Developing a good plot for a game is often very difficult to achieve. This is because a storyline is typically linear, like in novels and film, the story has a beginning, middle and end. Everything is in a particular order and if this order is changed it may render the plot meaningless and nonsensical. Try writing a story which is split into many, many segments, the order of which can be changed around however the reader likes (within some bounding rules) and the plot will still make sense. This is similar to attempting to write storylines for modern games. A player may make hundreds or thousands of potential plot changing decisions while playing a game. On top of this they may absorb only a small fraction of the plot related information which is around them as they progress through the game. These constant forks in storyline and how the player perceives it can make writing plots for games a much different craft to writing plots for movies.

Some games place more emphasis on plot and can be taken as an example of how it is important to get the balance of good story and good gameplay right. Most games are an example of a very high gameplay focus to story focus ratio. Let’s take an example of the extreme opposite, *Heavy Rain*, which is a cinematic psychological thriller in video game form. The game focuses heavily on its storyline, the characters and how they interact. The developers aimed to create a plot based on film noir, with a mature and intricate storyline based around 4 key characters and an unknown serial killer known as the Origami Killer. The story played out very much like one in a movie with the key difference being player decisions which have a direct effect on the storyline path.

Because of this focus on storyline, a lot of dialogue had to be delivered and many action sequences, just like in films, would contain sporadic, once-off and varied actions. These types of actions do not suit games as generally game mechanics are based off repetitive actions which can be gradually learned, combined and manipulated to overcome challenges. To put it another way, is it really worth spending time training the player to use certain mechanic for say pushing another person when it only occurs once or twice in the game? Maybe you could have the player guess how to use the mechanic at the time it is needed, but you must give some indication as to how to perform the action, so neither of these methods will work. Heavy rain found a balance between these 2 extremes by implementing a system of button, axis and movement prompts (the six-axis system detects angle and movement of the controller) which are requested of the player in a pressurized situation. The player is punished for missing or mismatching a prompt which could even result in changing the path of the plot. As for dialogue, simple options like how you should answer another character’s question can have an effect on a future element of the storyline.

This series of time constrained button prompts, often referred to as QuickTime events, became a standard way to handle cinematic sequences in games, where such a variety of actions occur in such a short space in time that other game mechanics would not suit such a sequence of events. Generally the patterns of prompts are loosely connected to the actions which they effect, due to the variety and range of the actions. QuickTime events have been the target of much criticism, especially from the hard-core gaming audience, who don’t see it as a legitimate or engaging game mechanic.

Getting the player invested in the game’s storyline can be a great contributor to the immersion factor and will often help drive a players progress through the game. Frequently games may lose their appeal in terms of the gameplay experience towards the end of the game, due to the player mastering most elements of the game and becoming aware of the repetitiveness of their actions. A good plot can help drive the player through this stage of the game as they will want to find out how the story ends. Plot elements can be used throughout the game as a reward mechanism, incentivizing the completion of certain tasks. A very simple example can be found in games as early as *Super Mario Brothers* where Mario must save the princess from the antagonist Bowser. At the end of a level he expects to find the princess but is famously told *“Thank you Mario! But our princess is in another castle.”* This drives the player to progress through the next level to find the princess.

Another important aspect to consider is the setting of your game. Your game may not have a plot but it must have a setting of some sort, an overall theme to define the scope of the in-game world and the characters and objects which inhabit it. For example a realistic game set in a specific year in a specific place will be constrained by the characteristics of that place in that particular time, such as the clothing people wore, limitations in technology, buildings that were present in that place at the time and even the way in which language was spoken. In contrast to this a fantasy game set in the future is less constrained and has more freedom in the creation of character, objects and the scenarios they are found in. This however does have its disadvantages including too many design choices, causing a loss of focus and creating too many entities which the human player cannot relate to, causing confusion and/or disinterest.

Once again, like many other forms of media found in a game, the plot is taking a backseat for the purpose of my project. I will likely implement a basic plot to drive the player through the game, but will not spend a significant amount of time developing on this aspect. [17] [18]

## 2.9 Tools & Platforms

To release a game to the public, one or more delivery platforms must be chosen and which one(s) you choose could have a major effect on the sales of the game. There has always been strong competition between game platforms, with much of the main manufacturers gaining market dominance at some point such as Nintendo, Sega and Sony. Famous platform battles from the heydays went from coin-operated arcade machines vs. Nintendo’s NES home console to Nintendo’s N64 struggling to fight off competition from Sony (PlayStation) and Sega (Saturn). These days there is an abundance of platforms available for games, ranging from portable handheld consoles to multi-functional touchscreen tablets/phones, from long standing consoles operating with 7 year old hardware to the latest home pc with cutting edge graphics technology. This range of choice means deciding which platform to release a game on can be critical to a project’s success.

Besides from platform popularity, many other factors can determine the most suitable device for a game to be created for. Because the games industry has expanded so much since the early days of gaming, each different platform has become optimized for a particular type of target audience. It is critical that the typical type of player that uses the platform your game is available for, matches the game’s target audience. Currently, the types of games which are available for multi-function handheld devices such as tablets and phones are typically aimed at the casual player, usually used as a quick time-killer. Handheld consoles cater for the more dedicated gamer who wants portable games which more closely resemble full console games. The Nintendo Wii console stands out from other consoles as being targeted towards the casual market, catering for family and party (same room multiplayer) gaming. Microsoft’s Xbox 360, Sony’s PlayStation 3 and the home PC platform cater for casual and hard-core gamers, with the latter having the largest proportion of hard-core players.

Another key factor to consider is the hardware itself. Generally handheld devices are much more limited in terms of hardware capabilities then consoles and PC, the most limited being the phone. When creating a game for a phone many things must be considered in the design of the game, for example if the user receives a call during the game or if the user is low on battery, the game must give priority to the more important functions of the phone. Many games may be created to take advantage of a particular platform’s input and output devices. A good example of this is the Nintendo Wii’s motion detection technology where controls for a game can be in the form of particular movements like a swing for bowling a ball in a bowling game. Graphics capabilities may be a major factor when considering a release platform. If so, you may look at releasing for the PlayStation 3, Xbox 360 or home PC. If you plan to be on the cutting edge of graphics technology the home PC may be the only viable option, as computer hardware including graphics cards are upgraded many times each year. In contrast to this, consoles aren’t modular, so graphics technology is only upgraded with each new console release, which can be several years apart.

Another factor which is generally of somewhat lesser importance is the development environment and development tools. With consoles and handheld devices you may be limited to a select few well supported programming languages and development environments, but with PCs there may be a much larger range of choice. For modern phones/tablets and handhelds the commonly supported languages include Java, C and C++. Supported languages are C and C++ for the PlayStation 3, C++ for the Nintendo Wii, C# and C++ for the Xbox 360.

Traditionally, games have always come in a physical form of some sort, from the cartridges of the old days to the compact disk and its modern advances such as Blu-ray and HD-DVD. In recent times the distribution of downloadable games through an online marketplace has been popularized. Generally these types of games are suited better to small time developers and budget projects, as the marketing and manufacturing side of the project doesn’t cost as much with digital distribution. With the integration of the internet with games came the ability to release updates to a game post-release. This also spawned a new component-based distribution system, where certain components of a game could be purchased separately as add-ons and users had the ability to pay extra for exclusive content if they wanted.

How the player can try out a game before buying must also be considered. Some games have demos which may consist of a small portion of the game which gives the player an idea of what the full game would be like, perhaps one level of a platformer or one track with one car in a driving game. Trials can also be used, this is where the full game or a limited version of the full game is available to the player for a limited period of time, after which they must purchase it to continue. Beta demos may also be released to players, pre-release. Open betas are just like a demo, they are free to download, but receive regular updates. Closed betas usually have a code protection system which only allows those who have exclusive access to the beta (have acquired a passcode) to download and play it.

After researching available platforms, I decided the Xbox 360 would be a suitable platform for my game. To keep the budget for this project down I wanted to choose a device which I already had, so that I didn’t have to purchase it to debug my code during development. I have a PC, an Xbox 360 and an android smartphone. One key factor was the controls; I felt that a smartphone would be awkward for controlling a platform game and I prefer a traditional gamepad controller. The Xbox indie games marketplace was attractive to me as it was designed to cater for developers on a minimum budget and the C# with XNA framework development environment is particularly suitable for game development novices, who have just begun to learn the trade.

# 3. REQUIREMENTS SPECIFICATION

## 3.1 Approach

Due to the fact that game development differs somewhat from typical software development, my approach to gathering requirements will involve the typical meetings with stakeholders and customers to define what it is they need. It is more a case of attempting to define what is required of my game, to successfully cater for the needs of my target audience.

For this project my target audience are players below they age of approximately 25, who have an Xbox 360, with access to the Indie Games marketplace. I am targeting casual gamers who want a quick thrill for free or cheap. I expect them to have a basic knowledge of and an interest in platform games. My audience may include those who seek a traditional style platformer, with similar aspects to the retro platform games of the late 80’s and early 90’s.

It is very difficult to make out the requirements for a game without having made a prototype or at least a similar existing game to playtest first. At the initial requirements stage it only makes sense to outline the core needs of the game, as game mechanics which I have planned to implement may prove unsuccessful after playtests. Below I will list what is required of my game at a minimum and some optional requirements which may be implemented if I have the time.

## 3.2 Functional Requirements

### 3.2.1 Mandatory (minimum)

* The game will take place on a 2 dimensional plane.
* The game will have a menu system which appears before the game play begins.
* The game can be paused at any stage during game play.
* The menu will have an option to view the control scheme for the game.
* The player will be able to control at least one character on-screen (the protagonist).
* The environment will come in the form of foreground terrain which the player traverses and one or more background scenes.
* The player can move the protagonist right and left along the terrain and up and down terrain via jumping and falling.
* The protagonist will feature animations for idle, running, jumping, falling and shooting.
* The game will feature 3 unique levels.
* A transition will occur at the end of each level, showing a new aspect to the storyline.
* The game will feature both horizontal and vertical scrolling, with the camera focusing on the protagonist as he/she moves through the environment.
* The plot of the game will be delivered through animation and sound at a minimum.
* The protagonist will have 2 unique weapons at a minimum.
* Weapons will feature projectiles, which can be fired in a straight line horizontally and vertically at a minimum.
* The game will feature 2 unique enemies at a minimum.
* Each enemy will behave differently.
* The protagonist can die from enemy attacks or environmental dangers.
* Enemies can die from player attacks or environmental dangers triggered by the player.
* When the player dies, they are reset back to some point in the past, enemies defeated between that point and the point where the player died will reappear.
* The game will feature at least 2 environmental dangers which can be used by the player to defeat enemies.
* The game will feature 3 puzzles at a minimum.
* The game will feature sounds for navigation of the main menu, enemies dying and the protagonist jumping and shooting at a minimum.
* The game will feature at least one music track.
* The game will feature an in-game user interface, which is always displayed and shows the player’s remaining lives at a minimum.
* The game will feature a credits sequence, giving credit to those who helped in the development of the game.

### 3.2.2 Optional

* The game will have a 2 player option in the menu.
* The 2 player option would enable a second player to play the main campaign with the first player, playing co-operatively to progress through the game.
* The co-op 2 player mode will have increased difficultly due to the challenge being spread between 2 players.
* The 2 player option would enable competitive multiplayer where a small sized level is used an arena where the 2 players must fight and attempt to defeat each other.
* The competitive 2 player option would feature rules and goals which are different from the single player campaign.

## 3.3 Non-Functional Requirements

### 3.3.1 Usability & Humanity Requirements

* The main game menu should be legible and easy to navigate.
* The control scheme for the menu and in-game should reflect the standard seen across similar games for the Xbox.
* Controls for controlling the protagonist should be intuitive.
* Learning game mechanics should not be difficult for the player.
* The player should be able to relate to the items seen in the game’s environment.
* The player should always know the current state of the game (for example feedback should be provided if the game is waiting for something to load).
* The player should be able to relate to the sounds in the game and understand what the sound indicates.
* The music should not interfere with the player’s perception of in-game sounds.

### 3.3.2 Look and Feel Requirements

* The look and feel of the menus throughout the game should be consistent to prevent confusion and make them more intuitive.
* The look and feel of the in-game user interface should be consistent throughout the game.
* All of the text in the game should be in English, should be clearly legible and should be spelled correctly, with correct grammar.
* In game objects which represent items that the player should be able to recognise, should be illustrated well enough so the majority of players will recognise them.
* The player should be able to tell what objects are in the foreground (playable area) and what is in the background.
* The player should be able to understand what actions are occurring in the in-game animations.
* The player should be able to clearly differentiate between the controllable protagonist and other characters/objects.

### 3.3.3 Design Requirements

* Microsoft guidelines should be considered when designing the game.
* Important information should be displayed within the inner 10% of the screen space, to cater for televisions which may cut off a portion of what should be displayed.
* The class structure should be designed in such a way that each cohesive function has its own class.
* The structure for implementing levels should be designed in such a way that it is easy to redesign a level or add new levels.

### 3.3.4 Performance Requirements

* The game should take no more than 1 minute to load up from the point where it is launched on the Xbox dashboard, to the point where the main menu is navigable.
* Splash screen animations appearing on start-up, before the main menu, should be skip-able.
* The player should be able to skip parts of the game where player control is paused (for example cut-scenes) for over 2 minutes.
* The credits sequence should be skip-able.
* The game should run at an absolute minimum of 30 frames per second, with allowance for the occasional temporary drop below this.
* The game should not hang/sleep for longer than 10 seconds without giving feedback to the player.
* The size of all the game files including sound and graphics related assets should not exceed 2 GB to adhere to Microsoft’s minimum requirements.
* The amount of memory used by the game at any one time should not exceed 512 MB to adhere to Microsoft’s minimum requirements.

### 3.3.5 Implementation and Testing Requirements

* The game must run on the Xbox, but can be tested on a windows computer.
* The game should be controlled with the Xbox gamepad on the Xbox and also when debugging in windows.
* All code should be formatted so it is easy to read, using whitespace to separate groups of related code.
* All code should be self-describing and variable names should be short in length but should accurately describe the information it holds.
* Areas of the code which are not self-describing should be well commented.
* Unused/invalid/depreciated code should not be present in the final build.
* Testing should be done throughout the entire implementation process.
* All new game mechanics should be play tested by several people representing the target audience and should receive at least 60% positive feedback before being implemented for the final build.
* When a particular game mechanic has been decided on, test cases should be written to ensure the game mechanic has been implemented correctly.
* The final build should not contain any more than 2 rare game breaking bugs (bugs which do not occur frequently but when they do, they force the player to restart the game in order to resume normal behaviour).
* The above requirements related to code do not apply to prototype builds, as speed is the priority with prototypes.

### 3.3.6 Cultural, Political and Legal Requirements

* The game should not contain any profanities or potentially offensive material.
* Text Content should be clear and not contain inappropriate lingo/abbreviations.
* Third party software/resources used in the development of the game or in the game itself should be used with permission of the owner(s).
* Third party software/resources used in the development of the game or in the game itself should be appropriately credited.
* Digital Certificates used for using third party software/resources should be legitimate and obtained legally.

## 3.4 Use Cases

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Use Case:** View Controls Layout | | **ID:** UC1 |
| **Goal:** The player views the layout for the controls of the game | | |
| **Prerequisites:** The player has downloaded the game and is at the dashboard of the Xbox. | | |
| **Flow of Events:** | | |
| **User:** | **System:** | |
| 1 | Launches the game from the Xbox dashboard |  | |
| 2 |  | Displays initial loading screen while the main menu loads | |
| 3 |  | Displays the main menu | |
| 4 | Chooses the option to view the controls |  | |
| 5 |  | Displays a diagram of the Xbox controller showing which buttons, triggers and axis functions correspond to which in game actions | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Use Case:** Start the Campaign | | **ID:** UC2 |
| **Goal:** The player starts the single player campaign | | |
| **Prerequisites:** The player has downloaded the game and is at the dashboard of the Xbox. | | |
| **Flow of Events:** | | |
| **User:** | **System:** | |
| 1 | Launches the game from the Xbox dashboard |  | |
| 2 |  | Displays initial loading screen while the main menu loads | |
| 3 |  | Displays the main menu | |
| 4 | Chooses the option to start the single player campaign |  | |
| 5 |  | Displays loading screen while loading the first level | |
| 6 |  | Displays opening cut scene or narrative to begin telling the plot | |
| 7 |  | Displays the first playable part of the game | |
| 8 | The player can now control the protagonist |  | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Use Case:** Navigate the Environment | | **ID:** UC3 |
| **Goal:** The protagonist moves through the in-game environment as controlled by the player. | | |
| **Prerequisites:** Use case ID UC2 | | |
| **Flow of Events:** | | |
| **User:** | **System:** | |
| 1 | Requests to move the protagonist right |  | |
| 2 |  | The protagonist moves right if they are not blocked by a wall | |
| 3 |  | Displays a protagonist animation corresponding to moving right | |
| 4 |  | The camera follows the protagonist | |
| 5 | Requests to move the protagonist left |  | |
| 6 |  | The protagonist moves left if they are not blocked by a wall | |
| 7 |  | Displays a protagonist animation corresponding to moving left | |
| 8 |  | The camera follows the protagonist | |
| 9 | Requests to make the protagonist jump |  | |
| 10 |  | The protagonist jumps to a set height, unless he/she is blocked by terrain during the jump | |
| 11 |  | Displays a protagonist animation corresponding to jumping and then falling | |
| 12 |  | The camera follows the protagonist | |
| 13 | Requests to make the protagonist jump while moving horizontally |  | |
| 14 |  | The protagonist jumps to a set height, unless he/she is blocked by terrain during the jump. During the jump the character will continue to move horizontally | |
| 15 |  | Displays a protagonist animation corresponding to jumping and then falling | |
| 16 |  | The camera follows the protagonist | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Use Case:** Firea Weapon | | **ID:** UC4 |
| **Goal:** The protagonist fires a weapon in the aiming in the direction desired by the player | | |
| **Prerequisites:** Use case ID UC2, the protagonist has a weapon to fire | | |
| **Flow of Events:** | | |
| **User:** | **System:** | |
| 1 | Requests to fire the equipped weapon while aiming right |  | |
| 2 |  | Displays an animation for firing the weapon | |
| 3 |  | Displays the weapon’s projectiles moving from the weapon horizontally across the screen to the right | |
| 5 | Requests to fire the equipped weapon while aiming left and moving right |  | |
| 6 |  | Displays an animation for firing the weapon | |
| 7 |  | Displays the weapon’s projectiles moving from the weapon horizontally across the screen to the left | |
| 8 |  | Displays a protagonist animation corresponding to moving right but aiming left | |
| 9 |  | The camera follows the protagonist | |
| 10 | Requests to fire the equipped weapon while aiming up |  | |
| 11 |  | Displays an animation for firing the weapon | |
| 12 |  | Displays the weapon’s projectiles moving from the weapon vertically towards the top of the screen | |
| 13 | Requests to fire the equipped weapon while aiming down, jumping and moving left |  | |
| 14 |  | Displays an animation for firing the weapon | |
| 15 |  | Displays the weapon’s projectiles moving from the weapon vertically towards the bottom of the screen | |
| 16 |  | The protagonist jumps to a set height, unless he/she is blocked by terrain during the jump. During the jump the character will continue to move horizontally | |
| 17 |  | Displays an animation corresponding to jumping and then falling | |
| 18 |  | The camera follows the protagonist | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Use Case:** Interact With an Intractable Object | | **ID:** UC5 |
| **Goal:** The protagonist Interacts with the object the player desires to interact with. | | |
| **Prerequisites:** Use case ID UC2 | | |
| **Flow of Events:** | | |
| **User:** | **System:** | |
| 1 | Moves the protagonist to within the proximity of the intractable object where it can be interacted with |  | |
| 2 | Requests interaction |  | |
| 3 |  | The object carries out a behaviour specific to the type of object, its location and the context in which its interaction was triggered | |
| 4 |  | Displays animation(s) related to the resulting behaviours of the object | |

## 3.5 Game State Diagram



# 4. OTHER CONSIDERATIONS

## 4.1 Implementation Methodology

I have chosen to go with a flexible, prototype-heavy approach to developing my game. After researching implementation methodologies for games I found it makes the most sense to start development early, with a heavy emphasis on prototyping. The idea is that you are inevitably going to get things wrong, so it’s best to make all the mistakes as early as possible and keep play testing your prototypes until you get something right and when you do, evolve the idea from there. Both for me (inexperienced) and in general, game development has a lot of unknowns and often the best solution, for better recognizing these unknowns, is through trial and error.

My approach will be closer to an agile methodology then for example a waterfall model. The traditional waterfall model is better suited to projects which have a more predictable outcome and are carried out by people who have the experience to boot. This scenario is quite the opposite. I will be constantly reviewing and revising each stage of the development cycle, but in general the revisions will become less frequent and more minor as time passes and deadlines approach.

**Research Phase Prototype**

The source code and visual studio project folder for the research phase prototype is on the project CD which contains this document. The prototype can be run on windows by opening the project in visual studio and pressing the Start Debugging button or pressing the F5 key. You should see a scene with a few green coloured platforms and a character. You can control the character using the left and right arrows to move left and right and the space bar to jump. The character should walk along and the ground and hit off walls appropriately. Basic animations should also be seen for walking and jumping. The platform at the top centre of the screen is an example of a platform which is in the first background, 1 layer behind the player sprite. The character should not collide with this platform. Pressing the back button on the gamepad or the escape key on the keyboard will exit the prototype.

## 4.2 Resources Required

* Microsoft XNA Game Development Studio
* Microsoft Visual Studio Express 2010 for C Sharp
* A Laptop with windows 7 and the above software installed. an Xbox controller receiver device and an Xbox controller for developing, debugging and demoing the project.
* A Microsoft Xbox 360 console with controllers, internet access and a developer account to debug and demo the project. Note: Developer account to be attained for free through the CIT Dream-Spark (software alliance) account.
* Adobe Photoshop and/or similar image editing programs.
* FL Studio and/or similar audio editing programs.

## 4.3 Open Issues

Time constraints are proving to be a major issue with for this project. This is due to the project taking place in 4th year along with other modules which have an advanced difficultly level and large workloads. To make this problem worse I have chosen to take on an extra elective module this semester so that I have 1 less module to take up valuable design and implementation time for the project next semester.

Another issue which involves my own health and well-being is eye-strain. I have been suffering from severe eye-strain and soreness due to prolonged periods of time working with a computer screen. This is common amongst people who work in IT, but in this case it seems to be a serious problem, as it could elevate to a stage where I could no longer continue working and be forced to take a significant break, during which valuable progress will be lost. To alleviate this I should be taking regular and lengthy breaks in between sessions of work. Trying to keep the right balance of getting as much work done as possible, while trying to prevent eye-strain (by taking breaks) is proving difficult due to tight deadlines.

After doing some prototyping, a significant amount of detail which I thought the prototype would help me decide on, has not yet been decided on. One of the major decisions is the use of a physics engine to create the terrain collisions and character behaviours. Having tried both this method and the traditional tile-based method with hard-coded, basic physics, I felt I could get a lot more work done and more effectively with the physics engine approach. I later found out the physics engine approach has many of its own issues including inefficiency for a platformer game. I would have to code in a non-standard way to by-pass these inefficiencies, which would mean I will have less guides and documentation to reference during development and less support overall.

Other detail indecisions which may be seen as issues at this time, but in my opinion are not critical to the project’s success, include; no plot or setting has been decided on yet (just rough ideas); the protagonist and his/her enemies have not been decided on (just rough ideas); the name of the game has yet to be decided on (relies on plot, theme and characters).

## 4.4 Possible Risks

The most critical risk is in relation to time constraint. This is my first attempt at developing a game. In game development, the first game is infamous for being very difficult to finish in a relatively small time-frame or even at all. I have set myself a great challenge, but one which I feel is realistic. There is always the risk of hitting a major progress blocker, like for example hitting an intermittent memory error which is very rare and difficult to debug. It could take 2 weeks to fix, setting the project schedule back hugely. This is something I learned to anticipate based on experience from the group project I did last year. In the last 2 weeks of the project, we encountered a memory error which crashed our application after a variable amount of time between 0 and 30 seconds. This caused a lot of panic and had we not resolved the problem, it would have been detrimental to the project and would mean we couldn’t even demo it successfully.

Another risk which is heavily related to time-constraint is technical difficulty. Game development, due to the broad range of skills required and maths, physics and graphics heavy programming, is known to be very difficult. Not knowing every fine technical detail about something I have specified at this stage means it is quite possible that after spending much time trying to fulfil that specification, I find that I just don’t have the experience or technical ability at this stage to fulfil it.

Play testing is a critical aspect of game development, but it requires many people, who are willing to give up some of their time to help the developer. The risk related to this is that I may not be able to get an appropriate number of people and/or an appropriate variety of people to playtest my game and give me feedback on it. Another risk related to this is that the people have become so used to high quality, high budget games that they expect a certain level of quality in a game which I couldn’t possibly achieve on my own, at my current level of skill and experience. This may lead to none of my game mechanics passing the playtest stage. If this occurs I will have to choose a stage in development where I will continue to finish my game regardless, so that I can at least gain a full project cycle experience, which would be of much more benefit to me than an unfinished project.

## 4.5 Legal Considerations

Copyright can be a major issue when releasing games for public availability. This is due the vast amount of data and information contained in games. All documentation in relation to the development of the game should have all sources of information appropriately referenced. In the game itself, it is customary to include a credits sequence, just like in movies, which gives credit to all individuals and organisations which contributed to the project.

All third party software and content used in the game should be used with the appropriate licences and permissions. It is common for games to include royalty free music, sounds and graphics, often for elements which have been done many times before and spending time reproducing the same artefacts would be wasteful. It is important that only content which is free for commercial use is used, if the game is to be published publicly. There is sometimes much ambiguity in whether something is royalty free for just personal use or for both personal and commercial use; this must be confirmed before using it.

It is also important when integrating your game into another organisation’s platform (in this case the Xbox system), that the platform’s security is not compromised because of security holes in your game. The Xbox platform is very secure and will cater for most issues in this area, however it is the responsibility of the game developer to ensure that their game cannot be used to disable or change a user’s access and/or cause malicious behaviours in the system.

Another aspect to consider is the use of user information. If the game makes use of the player’s personal information it must do so under an agreement with the user and should not abuse the right to manipulate this information.

## 4.6 Implementation Phase Plan

### 4.6.1 Summary

In this section I will approximate how much actual time I may have to spend working on this project (i.e. Carrying out any project related task that takes time). I attempt to take into account class hours and time outside of class hours which will be spent completing labs and assignments, reviewing lecture notes, studying for exams, breaks (meals), commuting, work (part-time job) and rest periods. I will not plan to fill every spare hour I have, to work on the project but rather state a safe (or near-guaranteed) amount of hours which I have free and I will normally work within these time slots. I also state overflow hours which when added to the safe hours results in the maximum amount of hours I can expect to be free that day. I will reserve these overflow hours as fall-back time for when the project is behind schedule and I need to crunch to catch-up.

### 4.6.2 Time Breakdown & Estimates

I declare the 15th of October 2012 as being the official starting point of the project as it is the point where the proposal was confirmed, a plan was put into place and work had started on the project. From this point till the end of semester 1 is 8 weeks long. During the interval between semesters there will be 1 week off over Christmas and 3 weeks in January (taking into account 1 week in January for exams), leaving a total of 4 free weeks. In the context of this project, I will call a week which I have off from college a free week, as I have significantly more free time to work on the project during these weeks. In semester 2 I have 13 college weeks and 2 free weeks (Easter break), leaving a total of 15 weeks. So in total, the development cycle will have a duration of 27 weeks (of which 6 are free weeks), ending on the final week of April 2013.

Due to the large string of free weeks with minimal distractions from other modules present during the Christmas break; I plan to get the majority of the implementation done during this time. Specifically these are the days between the 4th of January (the date of my final semester 1 exam) and the 28th of January.

**Average Semester 2 Weekly Hours to Spare**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Mon | Tue | Wed | Thu | Fri | Sat | Sun | **Total** |
| Max | 5 | 5 | 7 | 5 | 6 | 4 | 10 | 42 |
| Safe | 3 | 3 | 4 | 3 | 4 | 0 | 6 | 23 |
| Overflow | 2 | 2 | 3 | 2 | 2 | 4 | 4 | 19 |

**Average Free Week Hours to Spare**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Mon | Tue | Wed | Thu | Fri | Sat | Sun | **Total** |
| Max | 10 | 10 | 10 | 10 | 10 | 4 | 10 | 10 |
| Safe | 6 | 6 | 6 | 6 | 6 | 0 | 6 | 36 |
| Overflow | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 28 |

### 4.6.3 Implementation Phase Key Milestones

I will be tracking my project meetings, milestones and delivery dates on Google calendars. I have made the calendar publicly available for the sake of simplicity and transparency. Here is an agenda summary of the key dates:



[Click here to see the full calendar](https://www.google.com/calendar/embed?src=mycit.ie_kvbhr98k556rusc47vh07mmq6k@group.calendar.google.com&ctz=Europe/Dublin&gsessionid=EDFY0H4o6fLwb4QNvJCOrg)

# 5. APPENDIX

1. **Norneby, Johannes and Olsson, Tobias.** A New Attitude To Game Engineering: Embrace Change, Re-Use, Fun. *gamasutra.com.* [Online] 2009.

2. **Wikipedia.** Indie Game. *Wikipedia.com.* [Online] 2012. http://en.wikipedia.org/wiki/Indie\_game.

3. **Blow, Jonathan.** Indie Prototyping Talk @ Independent Games Summit. *youtube.com.* [Online] 2007. http://www.youtube.com/watch?v=ISutk1mauPM.

4. **Pruett, Chris.** Writing Real-Time Games for Android Talk @ Google I/O. *youtube.com.* [Online] 2010. http://www.youtube.com/watch?v=7-62tRHLcHk&feature=relmfu.

5. **Racketboy.** Platforming Games 101: Running, Jumping & More. *racketboy.com.* [Online] 2011.

6. **Wikipedia.** Platform Game. *Wikipedia.com.* [Online] 2012. http://en.wikipedia.org/wiki/Platform\_game.

7. —. PlayStation 2. *Wikipedia.com.* [Online] 2012. http://en.wikipedia.org/wiki/PlayStation\_2.

8. **PHL.** "Generic" Blog. *Blogspot.com.* [Online] 2012. http://generic2d.blogspot.ie/.

9. **Wikipedia.** Gameplay. *Wikipedia.com.* [Online] 2012. http://en.wikipedia.org/wiki/Gameplay.

10. **Hejdenberg, Anders.** The Psychology Behind Games. *gamasutra.com.* [Online] 2005.

11. **businessballs.com.** Maslow's Hierarchy of Needs. *businessballs.com.* [Online] 2012. http://www.businessballs.com/maslow.htm.

12. **gamification.org.** Game Mechanics. *gamification.org.* [Online] 2012. http://gamification.org/wiki/Game\_Mechanics.

13. **Valentine, Jake.** Mutant Mudds is a Charming Platformer. *geekparty.com.* [Online] 2012. http://geekparty.com/mutant-mudds-is-a-charming-platformer/.

14. **Antel, Barzan.** The Art and Psychology of Gaming. *devhardware.* [Online] 2008. http://www.devhardware.com/c/a/PC-Gaming/The-Art-and-Psychology-of-Gaming/1/.

15. **Kobic, Simon.** The Importance of Sound Design in Video Games. *gamingenthusiast.net.* [Online] 2012. http://www.gamingenthusiast.net/the-importance-of-sound-design-in-video-games/.

16. **DICE.** Battlefield Bad Company Tech Demo. *youtube.com.* [Online] 2007. http://www.youtube.com/watch?v=-V3Qaj\_cixY.

17. **Rosa, Sergio.** Storytelling in videogames AKA games are NOT movies. *gamasutra.com.* [Online] 2012.

18. **cdoubleu.** Great Moments in Gaming. *biglargehuge.com.* [Online] 2012.

19. **Miranda, Eduardo.** XNA Platform Game using Farseer Physics Engine. *youtube.com.* [Online] 2011. http://www.youtube.com/watch?v=HesRvU19biE.