



**Kassim Sachoo**

**Paintball:**

**A 2.5D Side Scrolling**

**Platformer Game**

**Individual Project**

**CI3330**

**K1014536**

**03/05/12**

Supervisor –Dr Andreas Hoppe

2<sup>nd</sup> Marker – Prof. Paolo Remagnino

**CI3330 Project Handbook**


**Plagiarism Declaration**

The following declaration should be signed and dated and inserted directly after the title page of your report:

**Declaration**

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computing and Information Science. All verbatim citations are indicated by double quotation marks ("..."). Neither in part nor in its entirety I have made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as his or her own work.

Date 03/05/2013

Signature 

**C13330 Project Handbook**

**Viva Declaration**

The following declaration should be signed and dated and inserted directly after the Plagiarism Declaration into your report:

**Declaration**

I have been informed about the Viva, the period in which the Viva takes place, and the objective of the Viva as it has been laid out in the project handbook. I understand this procedure and I also understand that if I do not attend this examination without the university having accepted my claim of mitigating circumstances, I will fail this module.

Date 03/05/2013

Signature 

## Contents

<b>1. Introduction.....</b>	<b>6</b>
<b>2. Aims.....</b>	<b>6</b>
<b>3. Objectives.....</b>	<b>6</b>
<b>4. Literature Review.....</b>	<b>7</b>
<b>4.1. Target Audience.....</b>	<b>7</b>
<b>4.2. Game Design Theory – In Game Rewards.....</b>	<b>9</b>
<b>4.3. Game Design Theory – Competitiveness.....</b>	<b>10</b>
<b>4.4. Game Level Design.....</b>	<b>11</b>
<b>4.5. Game Accessibility.....</b>	<b>12</b>
<b>4.6. Conclusion.....</b>	<b>12</b>
<b>5. Analysis.....</b>	<b>13</b>
<b>5.1. Introduction.....</b>	<b>15</b>
<b>5.2. Methodology.....</b>	<b>13</b>
<b>5.3. Choosing the Game Platform.....</b>	<b>14</b>
<b>5.4. Requirements of the Game.....</b>	<b>14</b>
<b>5.4.1. Requirements.....</b>	<b>14</b>
<b>5.5. Strength.....</b>	<b>15</b>
<b>5.6. Weaknesses.....</b>	<b>15</b>
<b>5.7. Opportunities.....</b>	<b>16</b>
<b>5.8. Strengths.....</b>	<b>16</b>
<b>6. Design.....</b>	<b>16</b>
<b>6.1. Level Design.....</b>	<b>16</b>
<b>6.2. Class Diagram.....</b>	<b>17</b>
<b>7. Implementation.....</b>	<b>18</b>
<b>7.1. Collision Detection.....</b>	<b>18</b>
<b>7.2. Platforms.....</b>	<b>19</b>
<b>7.3. Controllable Character.....</b>	<b>19</b>
<b>7.4. Shooting Mechanics.....</b>	<b>21</b>
<b>7.5. Main Menu.....</b>	<b>23</b>
<b>7.6. Paint Mechanics.....</b>	<b>25</b>
<b>7.7. Camera Controls.....</b>	<b>25</b>
<b>7.8. New Projectile System.....</b>	<b>26</b>

<b>7.9. New Water System.....</b>	<b>28</b>
<b>7.10. New Backdrop and Platforms.....</b>	<b>28</b>
<b>7.11. Waterfall.....</b>	<b>30</b>
<b>7.12. Flamethrower.....</b>	<b>31</b>
<b>8. Testing.....</b>	<b>33</b>
<b>9. Critical Review.....</b>	<b>38</b>
<b>10. Reference.....</b>	<b>40</b>
<b>11. Appendix.....</b>	<b>44</b>

## **1. Introduction**

The game that will be created will be a 3d side scrolling platformer. The game will consist of a single player mode which will be made of multiple levels. The game will feature a character that wields a paintball gun and a water gun. The player can shoot two types of paint, red and blue. The red paint will make the player jump when stepped on while the blue will make the player go faster, the player can clean up the paint by using the water gun if they feel like they have made a mistake. The objective of the game is for the player to go from one side of level to the other while facing obstacles.

## **2. Aims**

- Create an enjoyable game for the player
- Make current players buy future games
- Improve programming skills
- Gain game design skills
- Create a unique and original game

## **3. Objectives**

Having decided briefly in the project proposal what the game should include it's now time to set up a list of S.M.A.R.T objectives of what the game should accomplish.

- Finish analysis and game design by 14 January 2013
  - This should include a class diagram to help start of the game
  - Game development should proceed straight after
- Program using C# because that's the only object oriented language the Unity engine supports
- Research about previous side scrolling games and learn from their mistakes to help improve the game
- Acquire/create at least 5 different types of assets for the game
  - These should include player mesh and animation
  - Terrain
  - Environmental obstacles
- Release the game by 14<sup>th</sup> May

- The game should include at least 2 levels
- The game should be at least 90% bug free
  - Do testing after each iterations
  - Write down the tests performed
- Increase the number of players
  - The game should achieve at least 1000 downloads within 6 months.
  - The game should receive at least 80% of positive reviews for the chosen distributor. This is so that it attracts more customers to download the game
- Release at least 10 new levels every 2 months after the release of the game
  - Either release the levels all at once or over the span of two months will have to be experimented on after the release of the game and after player feedback.
  - Once again the new levels should receive at least 80% of positive feedback
- Provide a quick source of entertainment for the user.
  - The user should play the game for at least 20 minutes when they start playing the game. This is so that the game achieves the role of a casual game

#### **4. Literature Review**

The research consists of finding information about how to make an immersive game and will focus mainly on game design. The reason for this is because the most important thing for any game is the gameplay, if it's not good enough then the players will not come back to play it.

##### **4.1 Target Audience**

Before starting the research on the game design, the first thing to research is the target audience, who are they and what types of games do they like. This will result in designing the right game for the right people. There are two types of games to design; either make it a very challenging platform game or more of a casual game that people can enjoy to pass time on. By making it a very challenging game, the game will alienate a lot of people who don't want that level of stress in a game however by focusing on creating a more casual game will lead to a wider range of audience. Especially people that don't have time to play very long hours

of gameplay and only want to relieve stress by playing a simple game that doesn't cause too much frustration. It would also encourage people who don't usually play games like the older age group or maybe encourage more women to play them. It's shown that while core games have less female players, casual games seem to have a slightly more female players than male with 51.1% female and 48.3% men [5].

However while creating a casual game will open the game to more wider audience making the game too casual will cause the game to get too boring. If a game becomes too easy then it won't be much of a challenge and the players will feel like playing another more challenging game. However by making the game too challenging will cause the players get too stressed which is not always a good thing because they will then identify the game as "the stressful game". This is not what a casual game is supposed to do. *"Games become secondary activities as the player is doing something else in parallel: thinking, eating, watching TV, talking on the phone, waiting for something and so on. The parallel activity is often prioritized; while the player may choose games that do not require much attention and/or use of resources"* [2]. This means that when I design a game I have to make sure that the player can stop it and resume later if they want to. This is where designing the game comes into place.

If the game is too easy then you have to design it by adding other features to it that will hook the player in. One of the most successful games of all times is supposed to be "Angry Birds" which is a casual game with very simple game mechanics. The player has to shoot different types of birds into the buildings and try to destroy them. The game itself isn't very technically complex and doesn't have a very sophisticated gameplay mechanic. That being said the gameplay has this very addicting feel to it and has sold 1 billion copies on a wide range of platforms [6]. Another successful casual game is called Temple Run. The gameplay consists of the player running away from apes while collecting collectibles and jumping obstacles. Once again the gameplay is very simple and yet the game has sold 20 million copies [7]. Another example is the game series Pokemon which has been running for over 15 years and has sold 219 million copies since then.



## 4.2 Game Design Theory – In Game Rewards

One of the research questions is “Why are these simple games so successful?”. This lead to a personal theory that the reason for the success of these games isn’t just the gameplay but something else, another feature, that’s added to the game. There must be other features to the game that make people play it. By thinking about this question lead to the answer that the reason for this is these games have this sense of addiction to it, not too much that the game itself starts affecting the players life but enough to make the player come back and replay the game. So how does a person design a game in this way?

This question then lead to the research about addictive gameplay and how to make a person keep on playing a casual game which has no story or overall objective. One of the reasons people kept playing casual games was because of in game rewards. For example in the game of Call of Duty 4 Modern Warfare there’s a multiplayer feature where people from all around the world can play with each other. The objective is to get the highest points possible by killing the players from the opposite team and the team with the highest points win. While playing the game if you kill a certain number of people in a row without dying you get a special ability to do extra damage to the opposite team. The more kills you get without dying the more powerful ability you get. This game turned out to be a major turning point for the game series as this was the first time this game had this feature and it turned out to be one of the most successful first person shooter at that time. Another example is the previously mentioned game Angry Birds. Angry Birds had a reward system where the player had to get the rating of 3 out of 3 stars in an episode. If the player perfects a certain number of levels they would unlock a special hidden level that you can play. By doing this it motivates the player to replay a level until he/she gets a perfect rating just so that they can get the extra level. However having this type of system is very linear in the sense that once the player finishes the whole game and perfects it there’s really no reason for them to go back and play it again.

However the player doesn’t need to have an in game reward. You can reward the player both visually and with sound. An example of this would be to play a cut scene once the player finishes a level or when they receive a reward. In Angry Birds when the player gets a perfect rating a mini cut scene is shown of an egg with a star on it. This lets the player know they have received an extra level. Audio rewards can be something really small but occurs

frequently throughout the game. For example in most first person shooters when a player shoots someone from the other team a sound is played that tells the player they hit someone and then when that person is killed by a player a loud sound is usually played to let the player know they were successful. By doing this the player has something to look forward to and encourages them even more.

#### **4.3 Game Design Theory - Competitiveness**

Another way to keep the player interested in a game would be competition. *“As player skill increases, competition is important to maintain motivation. Whether this is competition with the game, or competition with other players, competition keeps players interested in the game for longer periods of time”* [3]. It is a wide known fact that multiplayer or games with some kind of competition with other players makes a game last longer. A good example is Trials HD which is a 2d platformer game where the player controls a motorcycle and they have to get from one side of the level to the other. It is primarily a single player game with a leader-board system built into it. You can compete with your friends by marking them as rivals and doing that will let you see their ”ghost” version in the game and therefore you can compete with them without the other person having to be online. Having features where players can compete with their friends will make them come back to play it long after the release date. On May 27 2011 the developers announced that they have sold 2 million copies [9] and they did that 2 years after they released the game.

Even really old games have a sense of competition. The original Mario game used collectables as a reward system. If the player collected hundred coins they would receive an extra life. The game also included hidden areas that the player can find and receive extra coins or power ups [2].

However implementing a poor system could have the opposite effect and could cost the developer money in the long term, EA which is one of the biggest game publishers in the world [10] shut down 12 game servers this year so far. This shows that just because you have multiplayer features in your games doesn’t make it a long term investment.

#### **4.4 Game Level Design**

Now that the research into game features was completed it is time to research about how to actually design the levels in the game so that the players find it fun enough to come back. A research paper at a conference proceeding [2] mentions 5 things that a 2d platformer is made up of. These include: platforms, obstacles, movement aids, collectible items and triggers.

Platforms include anything that the player uses to get across the level however any platform that causes the player harm such as a falling platform can also be defined as an obstacle. Obstacles as mentioned before are any objects that cause damage to the player. Movement aids is any object that helps the player progress throughout the level. A collectible item is anything the player collects and rewards the play in some way. Triggers are something that the player activates that changes the level in some way. In Mario this would be the switch that the player activates which changes blocks into coins [2].

That's not the only thing a platformer can include. There's a feature that is seen rarely in 2d platformers and that's the ability for the player to change the character they are using in the game [2]. By implementing such a feature you can open up a wide variety of puzzle elements. A critically acclaimed game called Trine [12] had this very feature where the player can change characters anytime during the game. Doing so enabled abilities that the other characters didn't have.

The paper [2] then moves onto talking about how a platformer should include something called rhythm groups. Rhythm groups are sections in the game where the player has to get across. By setting rhythm group you can set the pace of the player however you want. That can be something that adds more challenge as they go along. However it is important to make sure that the player has a resting point before they carry on because not doing so will cause the player to make more mistakes as a result of frustration. Since this leads to stress for the player, this is not something that is needed frequently throughout the level.

#### **4.5 Game Accessibility**

One of the most critical points of casual games is that the player will be looking for something to play in whatever situation they are in. This means that if the player wants to buy a game in that second they should be able to. The environment we live in allows for that type of lifestyle therefore that need should be catered for. This means the game should be easily available for someone to buy whenever they want to. The problem with releasing a physical game is that the player will have to wait for the game to arrive and therefore they will probably forget about it when the game arrives and will have probably already downloaded a different game. “It is necessary to consider the acquisition of the game product, the price factors, and the concomitant services to smooth the overall experience with a game and make the act of obtaining a game easier to accomplish.” This means the game should be released on a wide variety of platforms. The problem with this is that doing so will cost a lot more money in terms of development. A good example is android, if a game is made for an android device then it has to be compatible with every device out there and since the android operating system is open source a lot of devices will have this operating system. This does mean you have higher potential of gaining more customers but you have to take into account the amount of testing it will take to make sure it works on all devices. However since the iPhone has only one type of device and is more of a closed system; developing and releasing the game on that system will be cheaper and easier. On the other hand less people own a device that has the IOS operating system and will eventually have to expand to other platforms.

#### **4.6 Conclusion**

A lot of things have been learnt while carrying out this research. It is important to first think about what type of people the game should be made for which in this case is everyone since it's a casual game. Then it is important to think about how to make someone keep on playing the game which lead the thought process of either implementing a reward system or creating a competitiveness feel to the game or maybe even both. However the important thing is about how the game should be released after it is created. These to lead to questions like “Do I want to develop for a certain device?” Or “maybe release on all of them?” This lead to the conclusion that the game will be released on PC and if in the future it is successful, porting the game to other platforms will then begin.

## **5. Analyses**

### **5.1 Introduction**

The analysis is going to cover which methodology will be used and why it will be used in the project, it will also include all the requirements, choice of platform and analysis of the project itself.

### **5.2 Methodology**

There were two main methodologies that could've been chosen for the planning of the game, Step Wise Method (PRINCE 2) and Agile. Both of these methodologies can be used for creating a game. Using the Step Wise method will result in a more controlled and really safe if something goes wrong. However since PRINCE2 is meant for larger projects the process of planning can become very intimidating for a smaller project and could cost a lot more money than other modern methodologies.

On the other hand agile methodologies are less intimidating then PRINCE2 since it's meant for a smaller project. It helps remove unnecessary steps in project planning that add no value to the project plan. It also helps produce better software because it divides the project into small segments. Doing this helps reduce risks on a larger scale by problems not affecting other parts of the project.

The methodology that will be used is going to be Extreme Programming for my agile methodology. The reason for this is that it's easier for a small team to develop what they need instead of using time to prepare a very detailed design that can be changed again and again throughout the process of the project (especially with games since some functionality will simply not work with the flow of the game and will need to be reconfigured). This does not mean there will be no designing beforehand only that minimal designing will be done. However to compensate for this there will have to be more testing done after each iteration to make sure there is less problems by the end of the release date. The Extreme Programming method will allow for the development to go at the programmers pace and only program functionalities when it's time to program them

### **5.3 Choosing the Game Platform**

One of the most important parts of developing a game is to decide which platform to release the game. Releasing the game on the wrong platform could cause devastating results financially. For example one platform could have a specific genre that it's very well known. For this example Sony's PlayStation 3 and the genre first person shooter will be chosen. If a game were to be developed for the PlayStation 3 and then choosing to create a first person shooter while knowing that previous first person shooters games have been very successful will increase the likelihood of the game being a success. On the other hand having too many games of the same genre could cause a game genre fatigue [13].

The reason I've chosen to release my game on the pc is because the Unity 3D engine allows games to be developed on that platform for free. This is also a good choice because most people either own a windows/linux/ mac PC at home. By targeting a platform that most people own is very advantageous especially if it is for free. If the game becomes successful then extra money can be invested into to developing games using the unity engine on other platforms by upgrading my license.

### **5.4 Requirements of the Project**

It was stated briefly before what kind of features the game would include in the project proposal. However during the time the proposal was written and now has resulted in a few alterations. By doing the literature review enabled the requirements of the game to be altered to help improve the functionality of the game.

#### **5.4.1. Requirements**

- Controllable character – Must
- Gravity - Must
- Collision Detection - Must
- Shooting mechanics - Must
- Red and blue paintball mechanics - Must
- Health - Should
- Main menu - Should
- Platforms - Must
- Paintball have projectile physics - Should

- Obstacles - Should
- Music/Sound - Could
- Good performance - Should
- Easy to navigate user interface - Could
- Accessibility of game - Should
- Scoring system - Wont
- Tutorials - Wont
- Background settings – Could
- Pause Menu - Could

Now that the requirements have been decided, it is time to analyse the actual project and carry out a SWOT (**S**trengths, **W**eaknesses, **O**pportunities and **T**hreats) analysis.

### **5.5 Strengths**

Having experience in using object oriented programming is a big benefit to the project. The unity engine uses C# and having previous experience in C++ can be of advantage to the project. Unity has a built in asset store, this is very useful in buying assets such as meshes and textures. The game is very easy to understand therefore could attract a wide variety of players from children to adults. As mentioned in the literature review the game will be available in a wide variety of platforms on the pc by doing this will attract a wider audience.

### **5.6 Weaknesses**

No experience in 3D modelling, animation or art. This could be a huge problem when it comes to the look of the game and in attracting customers to the game. Another disadvantage is that the Unity asset store is very limited in what can be used in the project. Buying assets could be a very costly alternative. Therefore the number of assets used will be very limited. Having such a very simple concept of a game could alienate customers who prefer a more complex gameplay. Only having two different types of weapon could become very repetitive for the player. In the literature review it was mentioned that having competitive features would prolong the life of the game however not a lot of competitive requirements will be added at the release of the game.

## 5.7 Opportunities

The increase in the number of users in the genre could help with the sales of the game and the increase in Indie games being purchased will also help. Future updates to the game engine can improve quality of the game. For example the recent update of Unity 3D to version 4 has added the ability to create games for the Linux operating system, this means the game has the opportunity to expand on a new system. Another example is the ability to implement Direct X 11 features in the game and will help improve the quality of the game.

## 5.8 Threats

The increase in sales of Indie games could cause damage to sales of this game. Recent releases of 2D platformers on PC like Trials HD and Fez could also damage the sales of this game. Customers may also start to get bored of this genre and move onto something else. Lack of experience in releasing games could lead to legal disputes that are resulted because of use of unknown copyright infringement of assets like music or art.

## 6. Design

### 6.1 Level Design

One of the most important things about a 2D platformer is the level design. As discussed in the literature review if the level isn't designed properly the player won't continue playing it. Therefore the platforms have to be specifically designed. For the first level it's a good idea to design the level in a way so that the player understands the mechanics of the game.



Figure A

This represents how the platforms can be designed. The solid boxes represent a platform and the white boxes represent gaps. If the player can jump 5 spaces that means he/she will just barely survive the jump. This would teach the player the limit of how high one can jump when using the red paint.





Figure B

Then when a platform like this is placed after the previous platform, the player should know straightaway that this jump is impossible with the red paint therefore they will have to think about how to pass this using both the blue and red.



Figure C

By designing the platforms like this you can plan out how the player will progress throughout the level.

## 6.2 Class Diagram

To help start of the programming it is a good idea to make a class diagram. Using the functional requirements will help understand what needs to be programmed. Once the most important features of the game are implemented it will then be time to move onto the less important features that will help improve the game. This will be done by updating the class diagram and implementing the newly created classes.

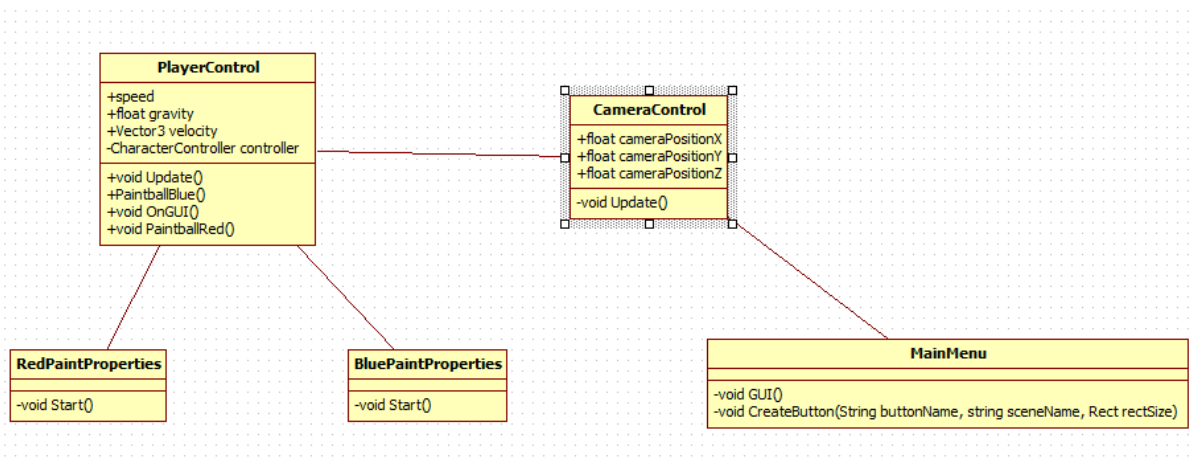


Figure S.1

Since the methodology for this project is Extreme Programming having the classes that need to be programmed first is optimal. Once these classes are finished, an updated version is can then be designed and implemented. This will be done at the end of each deadline which can be seen in the timeboxes.

Since the objective states that the game development should start at 14 January this is when the timebox for the development should start. The timebox below covers the class diagram. In the later stages of the design more timeboxes and improvements on the class diagram will be carried out.

Feature	Player Controls	Camera Control	Red & Blue Properties	Main Menu	Create Platforms	Health System	Testing	Update Class Diagram	Create New Timebox
Deadline	21/01/2013	23/01/2013	30/01/2013	01/02/2013	10/02/2013	11/02/2013	12/02/2013	14/02/2013	15/02/2013

Figure Q.1

The red writing identifies when a problem occurred. This will be talked about more later in the critical review.

## 7. Implementation

### 7.1 Collision Detection

To implement collision detection within Unity 3D is really easy. When you create a game object in the engine, a collision shape is automatically attached to it. This means that any object that has a collision shape to it will collide with the object. As seen below:

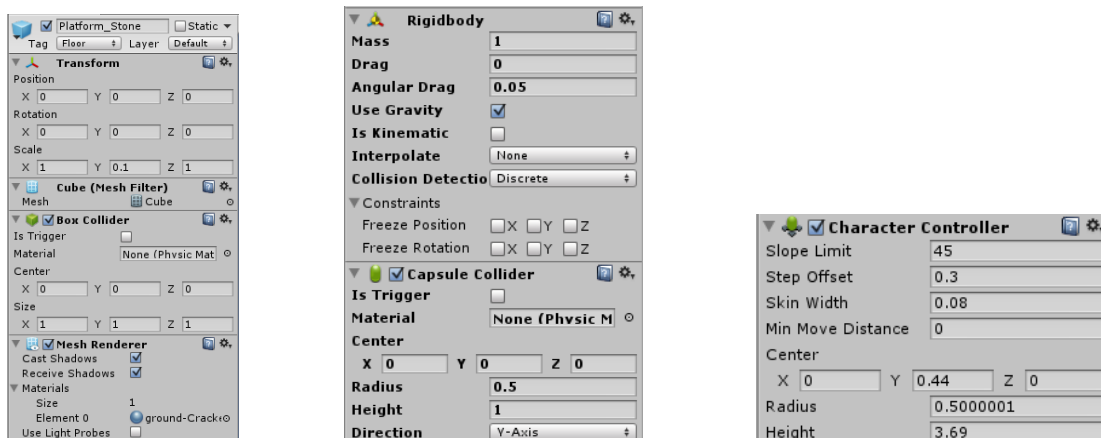


Figure D, E, F

To apply collision detection to the player can be done in either two ways. The first way (which is the chosen way for this game) is to create a character controller as seen above. A character controller has a built in collision detection. The other way to create collision to the character is to apply a rigid body and a capsule collider. This gives the player a more “realistic” physics however this game requires that the player to defy physics and therefore it was decided that the character controller would be the best option.

## 7.2 Platforms

Once the chosen collision detection was chosen it was time to move onto the platforms so that the character has something to walk on. The first thing was to create a platform, this was done by creating a cube and scaling the y axis to 0.1 and duplicating them so that it creates a seemingly long platform that allowed the player to walk on.

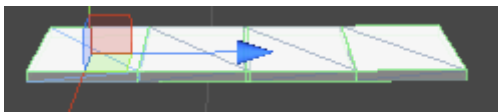


Figure G

## 7.3 Controllable character

Now that the character has something to land on it was time to create the script that will control the character. To determine whether the character was on the ground it is possible to use a built in variable called “isGrounded”. This is so that all the code needed for the character to do must be done when it’s on a platform. Everything else must be done outside of this check. Below is an example of how this variable was used to add velocity to the character only when it’s on the platform.

```
if (controller.isGrounded)
{
    velocity = new Vector3(Input.GetAxis("Horizontal"), 0, 0);
    velocity *= speed;
}
```

```
velocity.y -= gravity * Time.deltaTime;  
controller.Move(velocity * Time.deltaTime);
```

As shown the calculations to move the player is only done when the character is grounded. You first have to get the input from the player. If the player goes right the output value would be 1 and if the player goes left the value would be -1. This way you can determine whether the player is going right or left. You can then apply whatever speed you want the character to go and then add gravity outside the check.

After the character is able to go right and left you can then add the necessary animation for when the player moves. This is done by checking the “velocity” variable. An example of this is shown below

```
if (velocity.x == 0)  
{  
    aObject.CrossFade("idle");  
}  
if (velocity.x > 1  
{  
    transform.eulerAngles = new Vector3(0, 0, 0);  
    aObject["run"].speed = 1.5f;  
    aObject.CrossFade("run");  
}
```

In this code if the velocity is 0 which means the player isn't moving you play the idle animation and if the velocity is bigger than 1 then you play the running animation and rotate the character to face the right.

One of the great features of Unity 3D is the ability to change the data in a variable without having to access the code. For example if I want the default speed of the character to be higher in a certain level you can just change the variable in the inspector window in the engine. To do this you first declare the variable in the code as public and it will automatically appear in the inspector window. This makes testing out different variables really easily.

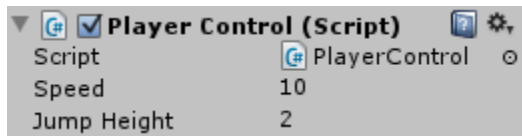


Figure H

## 7.4 Shooting mechanics

The main feature of the game is to be able to shoot a paintball wherever the character chooses to. This means that if the character wants paint on a certain tile then the paint should be created on that tile. This is one of the reasons why the platform isn't made up of one very long cube but instead smaller tiles.

The first thing that was created were three functions:

PaintballRed(), PaintballBlue() and CleanPaint().

The code in these function were all very similar and therefore only the RedPaintball function will be shown.

```
if (Input.GetMouseButtonDown(1))
{
    RaycastHit hit;
    if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit))
    {
        if (hit.collider.gameObject.tag != "Player")
        {
            if (hit.collider.gameObject.tag == "Red Paint")
            {
                GameObject gameObject;
                gameObject = hit.collider.gameObject;
                Destroy(gameObject);
                Instantiate(bluePaint, new
                Vector3(hit.transform.position.x, hit.transform.position.y, 0),
                transform.rotation);
            }
        }
    }
}
```

```

        else
        {
            Instantiate(bluePaint, new Vector3(hit.transform.position.x, hit.transform.position.y,
            0), transform.rotation);
        }
    }
}
}

```

First you get the input of the mouse. Then you get what object the user clicked on to determine whether you can create paint on that surface or not. You do this by casting a ray which gets all the colliders from the view of the camera. If the user clicks on an object with a tag called floor you create (or in the Unity engine term “Instantiate”) paint at the same position the player clicked on with the same rotation.

The object that is instantiated is a sphere that is flattened in the y axis and is the colour red. This is what the result looks like.



Figure I

The result is red oval shape on the surface of the floor. To delete an object you do exactly the same thing except when the player clicks on the paint you delete the object with

```

if (hit.collider.gameObject.tag == "Red Paint")
{
    GameObject gameObject;
    gameObject = hit.collider.gameObject;
    Destroy(gameObject);
}

```

## 7.5 Main Menu

The Unity engine has a built in user interface system that can print whatever the developer wants. These features include buttons, labels and even textboxes. Creating the menu was a very simple process thanks to this built in feature

To print data on the screen you have to use the function “OnGUI()”. This function lets the developer input anything they want on the screen.

```
void OnGUI()
{

    GUI.BeginGroup(new Rect(Screen.width / 3+100 , Screen.height / 3 , 400, 300));
    GUI.Box(new Rect(0, 0, 400, 300), "Main Menu");

    CreateButton("Start Game", "LevelSelector", new Rect(160, 30, 80, 30));
    CreateButton("Instruction", "Instructions", new Rect(160, 65, 80, 30));

    if (GUI.Button(new Rect(160, 100, 80, 30), "Exit"))
    {
        Application.Quit();
    }

    GUI.EndGroup();
}

void CreateButton(string buttonName, string sceneName, Rect rectSize)
{
    if (GUI.Button(rectSize, buttonName))
    {
        Application.LoadLevel(sceneName);
    }
}
```

This is the code that displays the main menu. After creating the file you just attach it to the camera and the menu is shown.

This is the result:

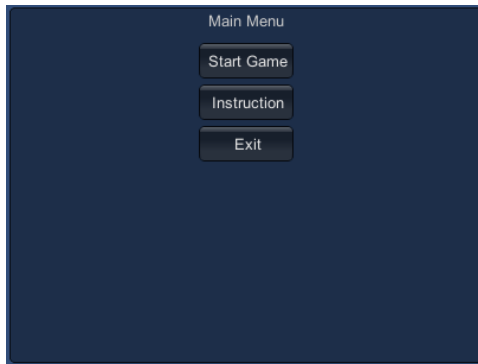


Figure J

## 7.6 Paint Mechanics

Now it was time to create the feature to make the player jump or move faster when it collides with the paint. First thing was to apply the tag “Red Paint” and “Blue Paint” to the different spheres. Then in the collider properties you tick the trigger box. This notifies the engine that the spheres are triggers and no collisions should be applied to them. This was done using built in functions called “OnTriggerEnter()” and “OnTriggerStay()”. These functions allow specific code to be run when it hits a trigger. When that trigger is hit you find out what tag it has and run that piece of code for it as shown below:

```
switch (objectCollidedWith.tag)
{
    case "Red Paint":
        jump = true;
        break;
}
```

The reason for the use of switches even though there is only one case is because in future development of the game might be added to the game and the code would be more organised then using if statements.



Once the variable “Jump” is returned blue the code in the player controls script knows that the player has to jump and runs that specific code for it

```
if (playerTriggers.jump)
{
    velocity.y = speed / 1.5f + jumpHeight;
    aObject["jump"].speed = 4f;
    aObject.Play("jump");
}
```

The height of the jump is determined by the speed of the player, if the player is going really fast then the player will jump higher. This will help the player think about where to place the 2 different types of paint.

## 7.7 Camera Controls

Unity allows the ability to attach either game objects or data of the objects as child's to other objects this can be very useful if you just want a position of the object but don't want the components that are attached to it. I used this feature to attach the transform of the player to the camera. This way I can get the real time update of the players' position and move the camera accordingly.

```
transform.position = playerObject.position + new Vector3(cameraPositionX,
cameraPositionY, cameraPositionZ);
```

This simple code makes the camera follow the player around. The different camera position variables determine how far away or how close the camera should be to the player. Since they were declared as public they can be modified through the inspector.

## Update Class Diagram and New TimeBox

Now that the main features of the game were implemented it was time to update the class diagram to include new features that can improve the game. The new diagram is shown below

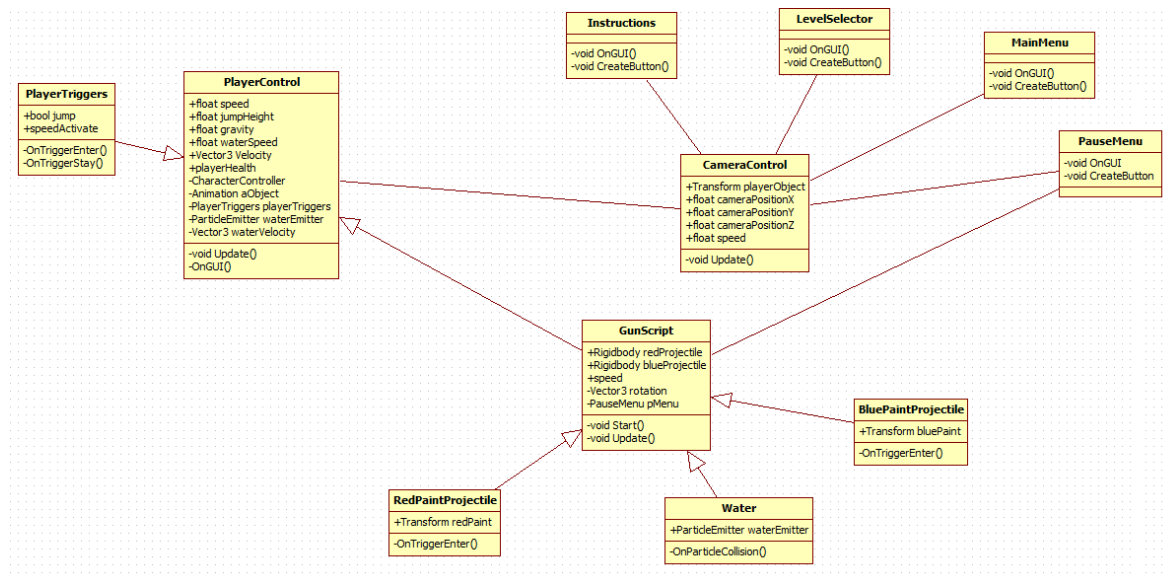


Figure S.2

Then create a new timebox to set deadlines for new features and make improvements to current ones.

Feature	New Projectile System	New Water System	Create Backdrop	New Menu Screen	Music Sound	Update Class Diagram	Create New Timebox
Deadline	22/02/2013	01/03/2013	05/03/2013	07/03/2013	09/03/2013	16/02/2013	17/02/2013

Figure Q.2

## 7.8 New Projectile System

The projectile system was a feature that would change how the player played the game. Before if the player wanted to create paint on the floor he/she would just click on the platform and paint would appear out of nowhere. This new system makes the game more challenging as it encourages the player to use more skill. The new system consists of the player aiming where they want the paint to go and a paintball would fly out of the weapon and land on the platform which would then create the flattened sphere.

The first thing to do was to create a new game object. This was a sphere with an attached rigid body. The old function in the player controller script was no longer needed and therefore was removed. This is how the new system worked. An empty game object was placed around the chest of the character; a new script called “gunScript” was attached to this object. The idea was that the player would click the mouse button and an instance of the object would be created. However the complexity came from the fact that the script had to get the position of the mouse on the screen and shoot the projectile in that direction. This was the

hardest part of the project. Because of not being able to figure out how to implement this feature resulted in the deadline being missed by a few days however fortunately I was able to find help on a website [14]. This is the code I derived from it

```
float distanceFromCam =
Camera.main.transform.InverseTransformPoint(transform.position).z;
Vector3 mousePos = Input.mousePosition;
mousePos.z = distanceFromCam;
Vector3 targetPosition = Camera.main.ScreenToWorldPoint(mousePos);
Vector3 targetDelta = (transform.position - targetPosition);
Vector3 launchVelocity = targetDelta.normalized * speed;
transform.LookAt(targetDelta);

if (Input.GetButtonDown("Fire1"))
{
    Rigidbody paint = (Rigidbody)Instantiate(redProjectile, transform.position,
transform.rotation);
    paint.velocity = -launchVelocity;
}
```

The first thing that had to be done was to find the distance from the camera to the player. This is so that the projectile fires in the right direction in the z –axis. Then to get the position of the mouse you use the built in function. However this gets the position of the mouse on the screen and not in the actual game world. You have to convert the value using the function Camera.main.ScreenToWorldPoint(). Taking away the location of the player and the position of the mouse will not be enough to get the direction you want to shoot. You have to normalize it and multiply it by the chosen speed. After that you get the direction and all that's left it is to add that as the velocity to the rigidbody on the projectile so that it flies in that specified direction.

## 7.9 New Water System

The new water system includes a particle system that acts like water and when middle mouse button is clicked instead of being able to click on any paint to get rid of it. This makes it harder for the player to clean the paint which in turn will make them think before they shoot the projectile. The code to determine the direction for the flow of the water used the code from the projectile shooting. Particles in the Unity engine can have collision detection you just have to tick the box that enables it in the inspector and use the function `OnParticleCollision()`. It's used exactly like the previous collision detection.

```
void CleanPaint()
{
    if (Input.GetMouseButton(2))
    {
        waterEmitter.worldVelocity = waterVelocity;
        waterEmitter.enabled = true;
    }
    else
    {
        waterEmitter.enabled = false;
        waterEmitter.ClearParticles();
    }
}
```

This code just enables the particles and applies the velocity to the code when the middle mouse button is pressed.

## 7.10 New Backdrop and Platforms

Since most of my game was completed at this stage, it was time to improve the look of the game. Using the Unity store is a good way to find free or cheap assets. These are some of the assets I created/bought.

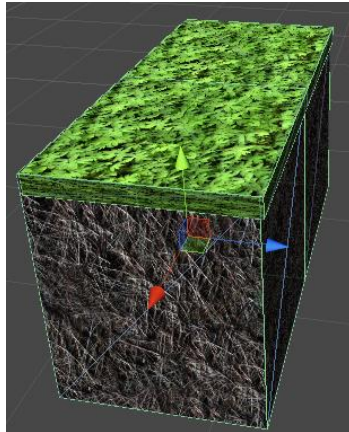


Figure K



Figure L

For the backdrop a terrain was created, using the terrain tools trees and hills were created.

## Update Class Diagram and New Timebox

Now that all the features from the timebox have been created it was time to update the class diagram to include more features that were planned like obstacles.

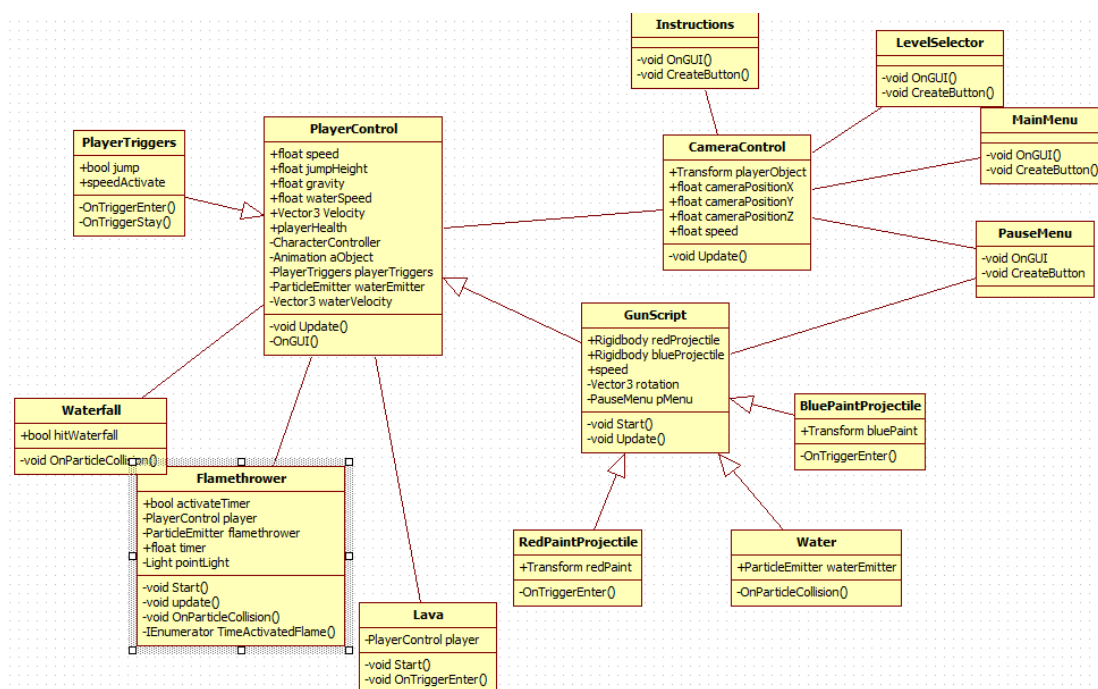


Figure S.3

Feature	Waterfall	Flamethrower	Lava	Testing	Update Class Diagram	Create New Timebox
Dealine	25/02/2013	03/03/2013	07/03/2013	10/03/2013	12/03/2013	13/03/2013

Figure Q.3

As shown above the new features include environmental hazards. This is the new timebox to help meet deadlines.

### 7.11 Waterfall

The waterfall is an obstacle that pushes the player of the platform in the z –axis. This will give the game a feeling that it may seem 2D but that doesn't mean that's the only direction you can go in.

```

if (waterfallObj)
{
    if (waterfallObj.GetComponent<Waterfall>().hitWaterfall)
    {
        velocity.z = -3f;
    }
    if (waterfallObj.GetComponentInChildren<WaterfallTrigger>().exitedWaterfall)
    {
        if (transform.position.z < 0)
        {
            Vector3 pos = transform.position;
            pos.z = 0;
            transform.position = pos;
        }
    }
}

```

This code is run when the player hits the triggers placed near the waterfall and the longer they stay in the water the more it pushes them on the z axis. If they exit the waterfall without falling of the platform the z-axis on the player is returned back to normal.

## 7.12 Flamethrower

Like the waterfall if the player hits the flame some of their health drops. However there is a timer on this flamethrower so that the player has to be strategic on when to go

```
IEnumerator TimeActivatedFlame()
{
    if (flamethrower.enabled)
    {
        yield return new WaitForSeconds(timer);

        flamethrower.enabled = false;
        flamethrower.ClearParticles();
        pointLight.intensity = 0;
    }
    else
    {
        yield return new WaitForSeconds(timer);
        flamethrower.enabled = true;
        pointLight.intensity = 1.5f;
    }
}

if (activateTimer)
{
    StartCoroutine(TimeActivatedFlame());
}
```

## Create 2 More Levels

Now that all the features of the game were added, it was time to create new levels to show all the features of the game. 3 Levels were created in the end(refer to Figure Q.4 for the timebox):



One of the levels was created in the setting of a mountain; this level featured a waterfall and really high platforms

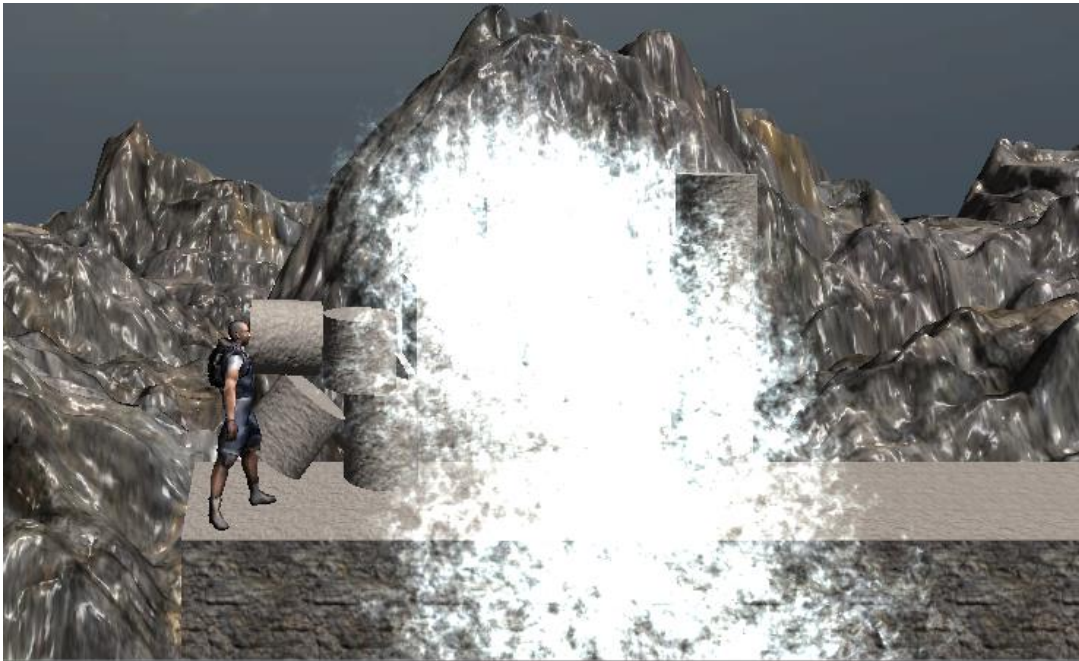


Figure M

The last level is set in the underground; this level features flamethrowers and lava.



Figure N

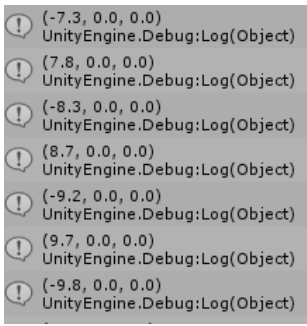


## 8. Testing

During the implementation of this project tests were carried out on the functionality of the features after they were created. You can see in the different timeboxes that at least one test was done before moving on; this helped contain errors during the development. The rating system is made up of 1-4 and is defined as follows:

1. Needs fixing now
2. Its problematic but not game breaking
3. It's a minor problem and can be looked at later
4. No problems

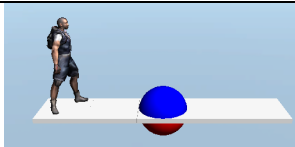
### Timebox 1

Test Carried Out	Rating	Detail	Evidence	Solution
Moving character	1	For some reason when I wanted to run left the character would keep rotating. However running right would work perfectly	<p>By using the debug function I could output the rotation of the player and work from there.</p> 	<p>I found that the function <i>transform.TransformDirection(velocity)</i> was the problem. This function was supposed to transform the position from local space to world space but it was also inverting the velocity and so commenting the function out fixed the problem</p>

Camera following player	4	There were no problems, the code worked perfectly fine. The camera followed in a smooth transition		
Does paint make character jump and go faster	2	As expected the red paint made the player jump and blue made the player go faster. However when placing the blue paint before red wouldn't make the player jump higher. Also the player didn't return to normal speed		<p>Because of this problem I made this equation <math>velocity.y = speed / 1.5f + jumpHeight</math>. Depending how fast the player moves will cause the player to jump higher or lower. It also led me to think of a way to gradually reduce the speed of the player. It can't be too quick to not make the player jump high when a blue paint is before a red paint but reducing the speed too slowly to decrease would also cause problems. This is the solution I came up with.</p> <p>if (speed &lt; 10f)</p>

				<pre> {     speed = 10f; }      speed -     = 14f *     Time.deltaTime; </pre>
Main Menu	4	The main menu loaded the game. No problems		
Health System	4	Falling down would cause the player to start from the starting point		

## Timebox 2

Test Carried Out	Rating	Detail	Evidence	Solution
Projectile system	3	The paint was displayed at the right position. However every time a projectile was thrown on a paint that was already displayed. It would merge together with it		<p>The solution was very simple. Put a new case in the switch statement that checked to see if there was already paint on the ground. If there was destroy it</p> <pre> Destroy(objectCollidedWith.gameObject); </pre>
Water	4	No problems		


system				
--------	--	--	--	--

**Timebox 3**

Test Carried Out	Rating	Detail	Evidence	Solution
Waterfall	2	The waterfall didn't push of the player but instead kept on repositioning the character back to normal on the z –axis.		<p>This was a simple fix. The code that was supposed to reposition needed a Boolean check to solve the problem. Creating a trigger that detected when the player left and then reposition him solved the problem</p> <pre> if (waterfallObj.GetComponentInChildren&lt;WaterfallTrigger&gt;().exitedWaterfall) {     if     (transform.position.z &lt; 0)     {         Vector3         pos = transform.position;         pos.z = 0;     } } </pre>
Flamethrower	4	No major problems. There was a minor problem of the timing of the flame turning		

		on and off but that was a simple variable fix		
Lava	4	No problems		

#### Timebox 4

Test Carried Out	Rating	Detail	Evidence	Solution
Full play through of game	3	Some minor problems of positioning of platforms. However during the play through of level there was a point in the level where the character was really high and because of the position of the camera the player couldn't see the platforms below		First trying to have an adaptive rotation of the camera every time a character entered a trigger seemed like a good idea however a simpler solution was to pull the camera back.

As you can see there weren't a lot of errors that caused problems to the development of the game. After each iteration a test was carried out to make sure the new features worked and if it didn't then the problem had to be fixed before moving on to cause less problems later on. If a different methodology had been used that required little testing then the problems would've

built up and would've caused a lot of problems cause a chain reaction of issues. By testing on a regular basis made sure that errors were reduced to a minimum.

## **9. Critical Review**

During the development of the game numerous problems were faced, either from technical issues or poor judgement. These problems however were solved but did cause problems with the deadline.

One of the first problems encountered was during the first timebox. While in the middle of the development of creating the properties of the red and blue paint, there was a discovery of new animation system in the Unity 3D engine called Macanrim. Development of the game was halted while more information of this new system was researched. After being satisfied with the collected information implementation of this new system into the game began. While this new feature resulted in very fluid animation in the game, this new feature also started causing problems to other features of the game like jumping and increasing the speed of the player. Another major problem that occurred was that the deadline was missed by one week and therefore the timebox had to be updated refer to Figure R. As a result of this sound and music had to be excluded from the final version of the game however a simple patch will be released in the future that will add music and sound to the game.

Another issue that occurred that was briefly mention during the implementation section of the report was the projectile system. Being unable to successfully create a projectile system resulted in using someone else's code to make it work. This also led to further delays in the timebox but only by a few days, refer to Figure R.

While looking back at this project all features mentioned in the initial proposal were created however some features that could've have been implemented would have probably improved the quality of the game. One of these features was the inclusion of wall jumping. Right now the player can only place paint on the floor however when the player hits a wall no paint is created. By implementing wall jumping could've have opened to a more complexity to the level design which would've increased the enjoyment of the player.

A few other features that could've been improved included music, art and animation. Doing this project has taught me that you shouldn't rely too much on finding assets by searching

online or using the Unity Asset Store. Doing so will either cost too much or result in very bland assets. By finding other students to create the background art, 3D models, music and animation would've have improved the overall quality of the game.

Another lesson taught while doing this project is that if a deadline is set then I shouldn't divert away from it and do something that will hinder the quality of the game. Doing this resulted in delays and those days that were spent doing something else could've been spent implementing more features in the game.

## 10. References

[1]

Kultima, A. and Stenros, J.

Futureplay

Kultima, A. and Stenros, J. (2010) "Proceedings of the International Academic Conference on the Future of Game Design and Technology", paper presented at Futureplay, Vancouver, May 06 - 07. New York: ACM, p.66-73.

[2]

Smith, G. et al.

MindTrekKultima, A. (2009) " Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era", paper presented at MindTrek , Tampere, September 30 - October 02. New York: ACM, p.58-65.

[3]

Harrigan, K. et al.

Futureplay

Harrigan, K. et al. (2010) "Proceedings of the International Academic Conference on the Future of Game Design and Technology", paper presented at Futureplay, Vancouver, May 06 - 07. New York: ACM, p.127-133.

[4]

Smith, G. et al.

Sandbox

Smith, G. et al. (2008) "Proceedings of the 2008 ACM SIGGRAPH symposium on Video games", paper presented at Sandbox, Los Angeles, August 09 - 10. New York: ACM, p.75-80.

[5]

Casual Connect

Casual Games Market Report 2007

Casual Connect (2007) Casual Games Market Report 2007. [online] Available at: <http://casualconnect.org/mag/CasualGamesMarketReport-2007.pdf> [Accessed: 01 Apr 2013]



[6]

Nextgenplayer.com

NextGen Player - Canada's Premier Entertainment Blog: Angry Birds Clears One Billion Units Sold

Nextgenplayer.com (2012) NextGen Player - Canada's Premier Entertainment Blog: Angry Birds Clears One Billion Units Sold. [online] Available at: <http://www.nextgenplayer.com/2012/08/angry-birds-clears-one-billion-units.html> [Accessed: 02 Apr 2013].

[7]

Temple Run: More copies sold than CoD: Modern Warfare 3 on consoles

Bunker, A. (2012) Temple Run: More copies sold than CoD: Modern Warfare 3 on consoles. [online] Available at: <http://www.electricpig.co.uk/2012/01/17/temple-run-more-copies-sold-than-cod-modern-warfare-3-on-consoles/index.html> [Accessed: 02 Apr 2013].

[8]

Metro

GTA sales hit 125 million, as 2K promise no yearly sequels

Metro (2012) GTA sales hit 125 million, as 2K promise no yearly sequels. [online] Available at: <http://metro.co.uk/2012/11/28/gta-sales-hit-125-million-as-2k-promise-no-yearly-sequels-551368/> [Accessed: 02 Apr 2013].

[9]

RedLynx - Trials HD Reaches Two Million Unit Sales

Jenkki (2011) RedLynx - Trials HD Reaches Two Million Unit Sales. [online] Available at: <http://redlynx.com/press/96-trials-hd-reaches-two-million-unit-sales/> [Accessed: 03 Apr 2013].

[10]

Alexander, L.

Electronic Arts Breaks Into Fortune 500

Alexander, L. (2012) Electronic Arts Breaks Into Fortune 500. [online] Available at: [http://www.gamasutra.com/view/news/28250/Electronic\\_Arts\\_Breaks\\_Into\\_Fortune\\_500.php](http://www.gamasutra.com/view/news/28250/Electronic_Arts_Breaks_Into_Fortune_500.php) [Accessed: 04 Apr 2013].

[11]

EA shutting down online services for 12 games

Gamespot (2013) EA shutting down online services for 12 games. [online] Available at: <http://uk.gamespot.com/news/ea-shutting-down-online-services-for-12-games-6401956> [Accessed: 04 Apr 2013].

[12]

IndustryGamers

Trine Sells 1.1 Million Copies Ahead of Sequel Release

IndustryGamers (2011) Trine Sells 1.1 Million Copies Ahead of Sequel Release. [online] Available at: <http://www.industrygamers.com/news/trine-sells-11-million-copies-ahead-of-sequel-release/> [Accessed: 19 Apr 2013].

[13] <http://www.gamesindustry.biz>

Call of Duty: "Significant concerns" that it's peaked, "genre tired" says analyst

Unknown. (2013) Call of Duty: "Significant concerns" that it's peaked, "genre tired" says analyst. [online] Available at: <http://www.gamesindustry.biz/articles/2012-08-03-call-of-duty-significant-concerns-that-its-peaked-genre-tired-says-analyst> [Accessed: 20 Apr 2013].

[14] Answers.unity3d.com

Side Scrolling Shooter. How to aim my bullet trajectory at the mouse cursor? - Unity Answers

Answers.unity3d.com (n.d.) Side Scrolling Shooter. How to aim my bullet trajectory at the mouse cursor? - Unity Answers. [online] Available at:

<http://answers.unity3d.com/questions/13093/side-scrolling-shooter-how-to-aim-my-bullet-trajec.html> [Accessed: 22 Apr 2013].

11. Appendix

Figure A



Figure B



Figure C

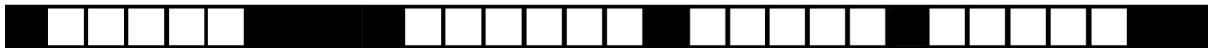


Figure D

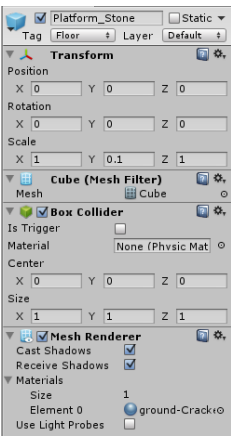


Figure E

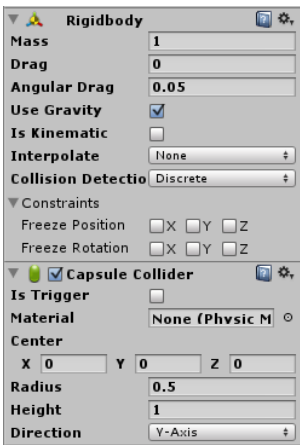


Figure F

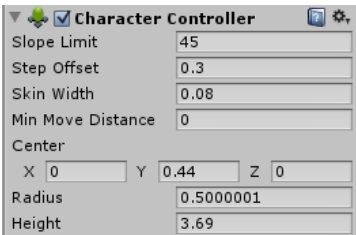


Figure G

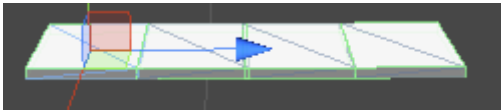


Figure H

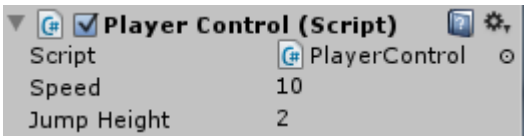


Figure I



Figure J

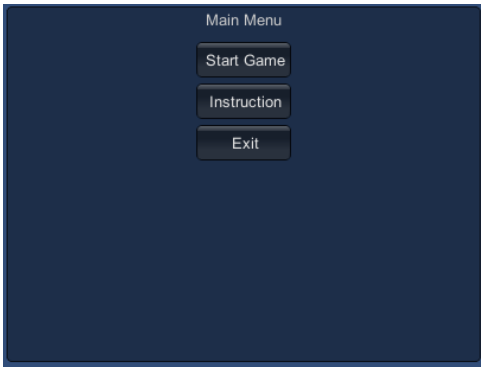


Figure K

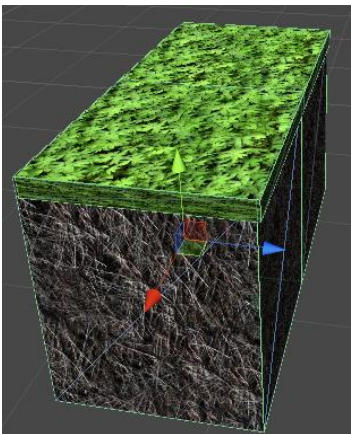
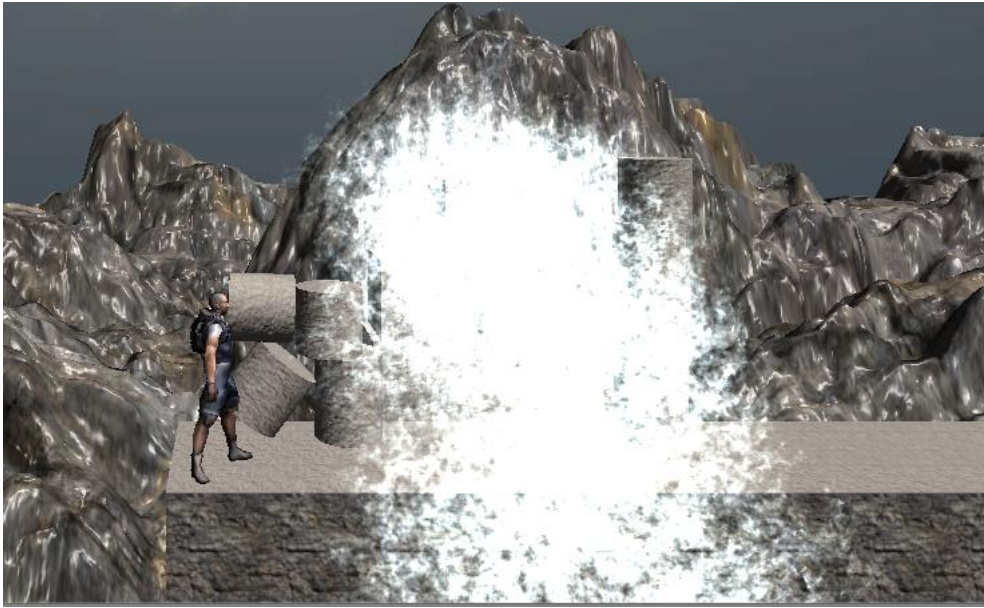


Figure L



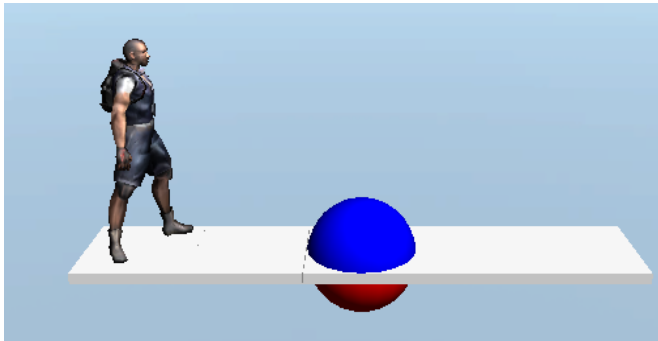
**Figure M**



**Figure N**



**Figure O**



**Figure P**





Figure Q.1

Feature	Player Controls	Camera Control	Red & Blue Properties	Main Menu	Create Platforms	Health System	Testing	Update Class Diagram	Create New Timebox
Dealine	21/01/2013	23/01/2013	30/01/2013	01/02/2013	10/02/2013	11/02/2013	12/02/2013	14/02/2013	15/02/2013

Figure Q.2

Feature	New Projectile System	New Water System	Create Backdrop	New Menu Screen	Music Sound	Update Class Diagram	Create New Timebox
Dealine	22/02/2013	01/03/2013	05/03/2013	07/03/2013	09/03/2013	16/02/2013	17/02/2013

Figure Q.3

Feature	Waterfall	Flamethrower	Lava	Testing	Update Class Diagram	Create New Timebox
Dealine	25/02/2013	03/03/2013	07/03/2013	10/03/2013	12/03/2013	13/03/2013

Figure Q.4

Feature	Add 2 new levels	Finish of report
Dealine	18/03/2013	04/05/2013

Figure R

Feature	Player Controls	Camera Control	Red & Blue Properties	Main Menu	Create Platforms	Health System	Testing	Update Class Diagram	Create New Timebox
Dealine	21/01/2013	23/01/2013	06/02/2013	08/02/2013	15/02/2013	18/02/2013	19/02/2013	22/02/2013	23/02/2013

Feature	New Projectile System	New Water System	Create Backdrop	New Menu Screen	Update Class Diagram	Create New Timebox
Dealine	29/02/2013	08/03/2013	12/03/2013	14/03/2013	22/02/2013	01/03/2013

Feature	Waterfall	Flamethrower	Lava	Testing	Update Class Diagram	Create New Timebox
Dealine	09/03/2013	10/03/2013	12/03/2013	14/03/2013	15/03/2013	16/03/2013

Feature	Add 2 new levels	Finish of report
Dealine	21/03/2013	04/05/2013

Figure S.1

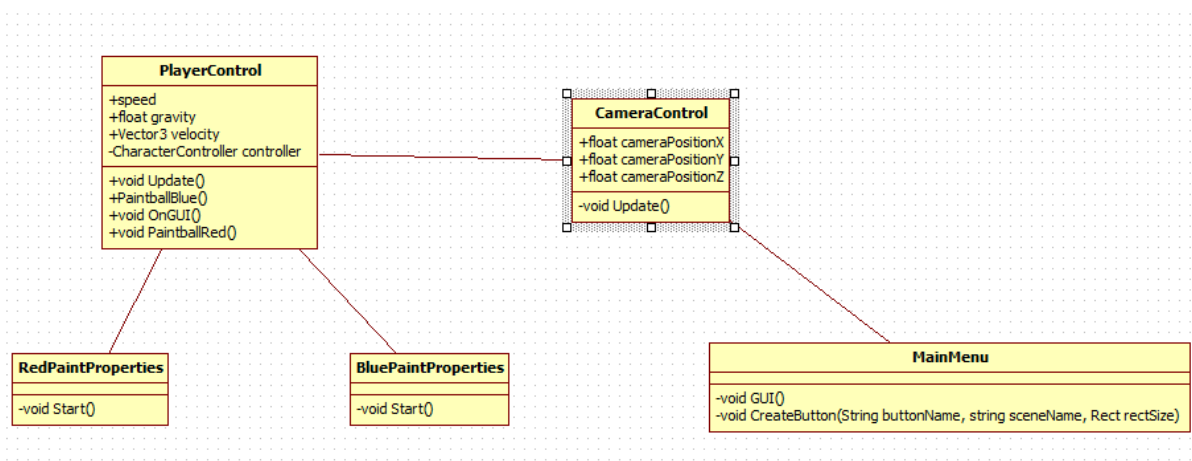




Figure S.2

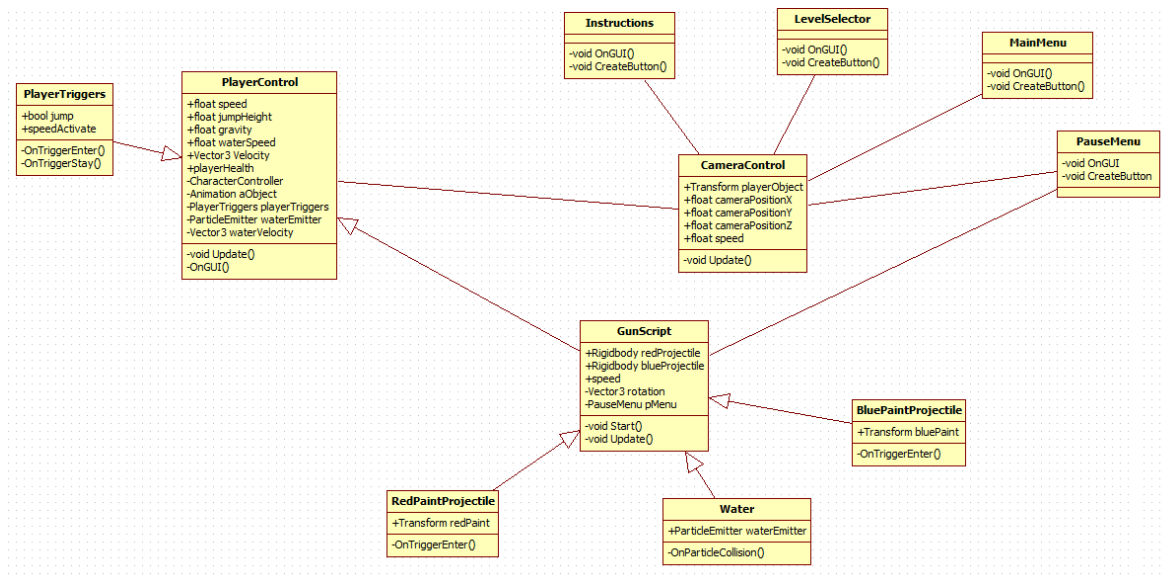


Figure S.3

