**Abstract – 150 Words**

The aim of this project is to use the 2D functionality within Unity to create a simple platforming game. It will show off a range of skills such as animation, game design and software engineering.

This is a retro, indie side scrolling game which involves a human adventurer travelling through a magical forest. It includes several obstacles which test the user’s skills and adds an aspect of fun. It also uses time travel so you can replay the last 20 seconds.

**Introduction – 300 Words**

*Project and its aims.*

This is a 2D game which is inspired by Braid – taking the main game mechanic and adapting it into this project. The main game mechanic is to have the ability to travel your character back 20 seconds. I designed two different levels for the game – the game ends after the second level has been completed.

The main character is an adventurer, who is fully animated – with the ability to jump, climb, and run their way through levels. I did not create the original assets; sources are listed below. I meant this project to be a display of my skills within Unity and game design.

My art inspiration was Ori and the blind forest as I prefer a magical setting. It gives the game more of a unique selling point and keeps the theme open for a range of visuals.

The aim is to create a similar game to Braid in Unity, using 2D physics and tile maps.

**Game Design – 1 page**

*What will they get out of this? Describe the primary mechanic.*

*What makes this project a game?*

This project is game because it has a win and lose state. The win state is when the player reaches the flag – there is one per level. There are several obstacles such as falling rocks and spikes – they cause the player to die – ending the game. It also has a special feature which allows the player to go back in time.

It is a side scrolling, platformer game. It revolves around an adventurer who must traverse ladders and platforms to reach the end goal. The camera follows the player - keeping it centred. This means that the focus is on the player and allows for a complex and bigger gameplay screen.

The game also has a graphic user interface, main menu and the appropriate end screens in which you can navigate the game. It is simple and effective as to not confuse any players.

People will feel motivated to play the game because it is challenging – the player must traverse platforms to reach the goal. A bonus is the nice aesthetic and variety of obstacles – it pays homage to the fantasy art style of Ori and the blind forest.

*How the mechanics fit together and make the user play?*

**Software Design – 4 pages**

All assets are third-party – they are referenced at the end of the document.

Platforms:

Platforms within the levels have varied sprites to give the game more flair and variety. This includes different length and decoration on them.

They are situated on a grid layer which is used for player collision. The player changes state to GROUND when it has collided with the platform. This is important as without it, the player would not be able to trigger the jump method.

Spikes:

Touching the spikes results in a game over state for the player. The sprites are located on the “GAMEOVER” grid layer because the player contains a script which checks for any collision via the layers. This makes adding any spike components easier to manage and add as they are encapsulated within the grid.

I chose to represent it with this sprite because it is like the rest of the art style. It is slightly different though as it is a vector graphic – this was intentional as it draws attention to itself, so the player knows it has a different function than the rest of the decorations within the level.

Falling Rocks:

A rock is game object which has the ‘Rock’ class as a component. It is located on the “GAMEOVER” layer which triggers the game to end when the player collides with it. It has two colliders attached to it, one to detect the player and one to act as a rigid body. Once a player is detected, the rock will fall and crush the player.

Ladders:

The ladder is situated on a separate grid layer and the methods associated with it are activated when the player collides with it. The player automatically changes to the CLIMB state. The player can then traverse up and down, with appropriate physics. The sprite for this is a vine because it fits the theme.

Goal:

The goal is on a separate grid level as the game continually checks if the player has collided with it. When it does, the player tells the environment which triggers the appropriate user interface. The interface is a game object which contains several children which are the appropriate UI elements.

The sprite I chose for this is a rock with a flag embedded in it. It is based off the rock in the overall game but with a small modification. This is done because it fits in with the art style and its function to clear.

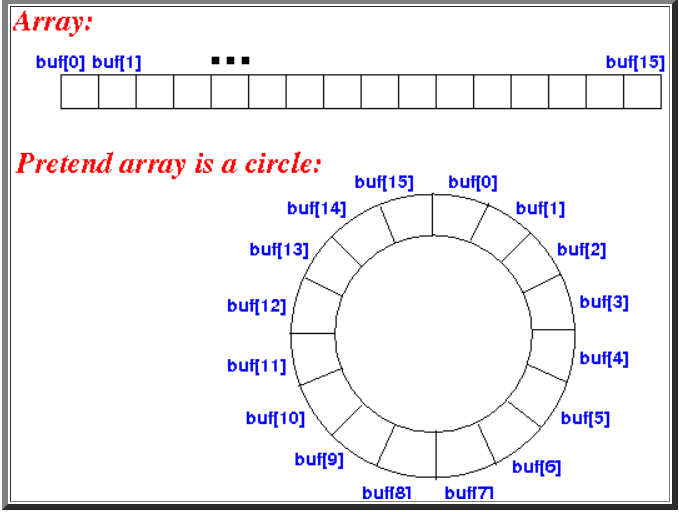
Level Limits:

I used a similar layer technique in order to limit the players movement going off the screen. I chose sprite blocks that blend in with the environment. I did this because it fits the aesthetic, and it does not distract from the immersion of the game.

Time Travel

Time travel is a key aspect in this game as it is loosely based on Braid – see introduction for more details. It tracks the last 20 positions of the player and gives the user the ability to take the player back to those. It is mapped to the ‘E’ key and it also effects the states of the rocks.

I chose to store the positions in a ring buffer – both for efficiency and performance reasons. It is a fixed length array which works as if it is connected from end to end. It points at the current position; moving in either direction depending on whether you are reading or writing to the list. This fixed size buffer works well for the feature as it is not too performance intensive. A visual representation is below:



<http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/8-List/array-queue2.html>

Initially, I started tracking the player positions in a list. It was with a lot of research during my implementation that I found that it was more computationally expensive to add and remove an object each frame so I switched to the ring buffer.

*Describe how the components fit together and contribute to the whole.*

There are two challenging levels for the player to beat – which combine all the components specified in the brief.

I took a modular approach to designing the levels, starting by designing small components that could fit together to create the level. Examples of these are below. I got a small test group to rate these components in fun and difficulty – I took those ratings and drafted levels from there. They then got play tested and tweaked – to make sure the levels were beatable and fun.

*Use UML diagrams to explain key points.*

The rocks and time travel are encapsulated in their own classes because they store their relevant methods and data separately. This is for ease of use when developing the features further; both for debugging and readability for the other classes.

Within the hierarchy, the player object has a script called “playerController” attached to it which controls the states and actions of the player. It observes the different interactions between the features and the player.

There is a word controller object which has the “environmentController” script as a component. The use of this is to relay the players game state and show the appropriate graphical user interface within the scene. This component is present in each of the levels.

The last controller script is the “UIController” and that contains all the methods for the buttons to access. That is used on each level and on the main menu screen.

*What alternative designs do you consider (or try) and what are the pros and cons of these different choices?*

Originally, I was going to create all the assets by hand – based on a storyline containing dreaming cats. This however changed after consideration of the brief and timelining the development of the project. This change improved the game as the features are more coherent and make sense in the submitted project – it provided a better context and visual aspect.

There were several levels designed for the game – they had to be re-designed as many were either impossible or too easy to complete. Crowd testing also showed that certain types of levels with more strategy were more popular. An example of this is a level where the player must go back on themselves to reach the goal.

As I used third party graphics, I had to bear in mind the sizing of these sprites as this affected the hit boxes and the positioning for example. This meant re-designs happened after filling in the correct sprites.

**Testing**

I did a mixture of testing including:

* Manual Testing
* Crowd Testing

Testing happened during implementation of each feature and before merging the future branch. I did this to make sure the software was stable and working before pushing it to the main branch.

Player:

* Player moves right on the horizontal axis.
* Right running animation plays when moved on Horizonal Axis
* Left running animation plays when moved on Horizonal Axis
* Player moves left on the horizontal axis.
* Player jumps correctly on the Y axis.
* Jump animation plays when Player jumps.
* Player dies when it touches spikes
* Game over animation plays when Player dies.
* Player dies when it touches rocks.
* Player wins the level when touching the flag.
* Climb animation plays when Player climbs.
* Player climbs when it touches ladder.

UI:

**Discussion and Reflection – 1 page**

***What are the primary strengths of your project?***

The main strengths of the project are the element abstraction as I spent a lot of time organising all the components in the project. This made it easier to develop further and debug if any issues arise. All the components are linked where they need to be and only contain the minimum amount of methods as to not cause any future developers confusion.

Another strength was the planning and development process. This is because I used a private Github repository along with a Kanban board to track issues and create pull requests for features. This also allowed me to work on multiple aspects at the same time – without affecting the stability of the overall game project. This helped greatly with time management.

***What are its weakness?***

A weakness of the project had to be prioritisation. I spent a lot of time designing and implementing the visuals of the game – placing tiles, configuring animations. This meant I spent less time making the features future-proof and stable. A way to improve in the future would be to focus on core features first. This ties in to how much polish is still yet to be done. This is reflected by the lack of prioritisation and time constraints unfortunately.

***What have you learned during this project?***

***What would you do differently next time? If during self reflection you have identified an issue, e.g. time management, what actual steps could you take to address this issue?***

On reflection, I need to stick more closely to the project plan and reduce the amount of deviation when it comes to feature development. I could use milestones to achieve this – creating smaller deadlines to ensure I stick to the project plan.

References:

* *https://rvros.itch.io/animated-pixel-hero?download*
* *https://trixelized.itch.io/starstring-fields*