Logo

Description automatically generated**Department of Applied Computer Games (DACG)**

**Mobile Game Development – MHI622947**

**Coursework 1**

**2021-2022**

**Submitted for the Degree of:**

**BSc Computer Games (Software Development)**

|  |  |
| --- | --- |
| **Student Name:** | Stephen Ross Cartner |
| **Matriculation Number:** | S1706321 |
| **Course:** | Mobile Game Development 1 |
| **Word Count:** | 2330 |
| **Github Link:** | <https://github.com/Ubitroll/Mobile-Game-Development-Coursework-1> |

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award*.

*Signature*: Stephen Ross Cartner Date: 10/01/2022

**Contents**

1. Code explanation 3
   1. Intro Screen 3
   2. Sound 3
   3. Background 4
   4. Sprites and Animation 5
   5. Player Action/Interactivity 6
   6. Game Over 9
2. Extension Material 10
   1. Gravity 10
3. Development Blog 11
   1. Entry 1 11
   2. Entry 2 12
   3. Entry 3 12
   4. Entry 4 12
   5. Entry 5 12
   6. Entry 6 12
   7. Entry 7 12
   8. Entry 8 13
   9. Entry 9 13
   10. Entry 10 13
   11. Entry 11 13
   12. Entry 12 13
   13. Entry 13 13
   14. Entry 14 14
   15. Entry 15 14
   16. Entry 16 14
   17. Entry 17 14
   18. Entry 18 14

**1.0 Code Explanation**

**1.1 Intro Screen**

The intro scene is built on a separate HTML file that is loaded first from the MainActivity. It is loaded into the WebView through the url and the relevant CSS file. The body was split using dividers with set ID so that in the CSS they could be applied specific modifiers that would position the background image and the canvas to stretch properly across the screen but then align the text and start button with the centre of the screen for maximum visual clarity.

Text

Description automatically generated

**1.2 Sound**

Mp3 files were found to provide appropriate theme music. Run through a HTML audio source given the link to the appropriate sound file the theme music is given two important attributes. Autoplay makes it so the audio source will start to play the music as soon as the page is loaded in and then loop causes the file to repeat itself when it ends. These two functions work well for theme music that is supposed to be continually going in the background of the menu’s and the game.

Text

Description automatically generated

**1.3 Background**

Tilesets were chosen as the way to draw images to the screen because I wanted to challenge myself and look at a new way to display sprites on a screen. I am very used to taking an individual image and displaying it by positioning it on the screen and setting the scale width and height. This way of displaying the images was new and confusing to say the least.

A tileset is one large image that has all of the images that are going to be used within a game/level, these tiles are then cut out using a source XY position and a height and width in which to cutout.

To make it easier for me to understand, I gave each tile a value, this value represented what part of the grid it was on, for example: The top leftmost tile in in the tilemap is 0, the one next to it is 1 and so on. The values were set like you would read words on a page, left to right and then down a line and back to the left when the right was reached.

Now that the tiles have been given a value for my sake outside of the program, they now have to be cut and displayed onto the screen. This was done by taking the value I assigned, multiplying it by the size of a tile (which is 16px) and cutting out that tile and drawing it via the drawImage. So if I want to draw the 4th tile, it would take in the number 4, multiply it by 16 to get the source and then draw to the screen at a position of my choosing.

This was working well, but then I stumbled up a way to take the cutout tiles and display them seamlessly within a canvas. This was when I found out about tilemaps.

**1.4 Sprite and Animation**

The player object is drawn at a specified position, this position is the player position, using the tile cutting methods mentioned above, the player has an array of sprites that it can use. This array has a cutout XY position and area to be cut, there is also an offset value. This offset value is used because the player sprite is not square, if the offset was not used, there would be strange drawing errors.

A picture containing graphical user interface

Description automatically generatedThis array of cutout positions has been lain out to allow for it to be separated and used for different animation states.

This array is then taken and split into different animation states.

Text

Description automatically generated

When the user presses a directional key, a directional variable is set to 1 or -1. If the direction is 1, the frameset moveRight is called, if it is -1, moveLeft is called.

When the player jumps a jumping bool is set to true, if this is true then the jumping frameset is called. Depending on the direction variable, the player with be jumping left or right also.

Velocities are used to move the player (explained later) while this value is greater than 0 (or less than depending on whether the player is moving left or right) the moving framesets will be called and drawn to the screen.

Text

Description automatically generated

**1.5 Player Action/Interactivity**

The player is moved using velocities, these velocities are added to the player whenever they press a movement key. Every frame, these velocities are applied to the players XY position, but if they were applied directly, upon keypress the character would move constantly without stopping.

To prevent this two variables are used to act upon the x and y axis. Friction is used to determine to rate in which the velocity will fall off on the x axis. Gravity is used to bring the player back down after the jump button is pressed.

When the player holds a movement key, a the velocity is increased, this is then applied to the players position. After the key is released the constants friction and gravity bring the player to a gradual stop. To prevent the player from going too fast and forcing themselves through colliders, a maximum velocity was created which the player cannot exceed.

Text

Description automatically generated

Game world collisions

The game world is made up of square tiles, these tiles have 4 sides. Each of these sides can be collided with, but not all tiles can have all 4 sides collided with: A picture containing schematic

Description automatically generated

This means that every single tile can be described using only 4 collision cases, but each tile can have different combinations of each case.

I created these 4 cases: Top, bottom, left, right. The player has 4 sides too, these are checked constantly.

I assigned each tile to have a specific collision, floor tiles are only checked for the top collision, left wall tiles are only checked for right side collisions and so on. There are many different tiles within the game, but they can all be covered with 15 combinations of the 4 top, bottom, left and right collision cases. These cases are stored in a colliderMap array, that is just like the graphics map but with collisions instead.Text

Description automatically generated

Case 1 is the floor tiles, Case 2 is the left walls and so on…

Text

Description automatically generated

If a tile has no, collision case, then it is assigned 0.

So each tile has a collider value and each value has a combination of the collision cases, so how is a collision detected?

So, for example we want to see if the player should be walking on top of a tile: We check the bottom of the player to the top of a tile, if the bottom of the player is less than the top of a tile, then a collision has occurred. If a collision has occurred, we set the players y velocity to 0 and we set the players y position to be at the top of the tile.

Text

Description automatically generated

This worked well until there was a gap underneath certain tiles, when this was the case, the player sometimes phased through the tile and landed on the tile below. This was fixed by comparing the players old bottom position from the previous frame as well as the players current bottom position to the top of a tile.

**1.6 Game Over**

The game over screen is laid out exactly like the intro screen with the change of having its button load the main menu html instead of the game html that the main menu loads. The game over screen also contains a different audio source since the different menu’s actually contain different theme music and the on screen text was also changed to show the player their score.

Text

Description automatically generated

**2.0 Extension Material**

**2.1 Gravity**

Since the player is moving using a velocity-based system it was fairly simplistic to implement gravity by applying a constant downwards velocity. Each frame after the movement key has been pressed and the forces have been applied the Y velocity is also updated by the gravity value. This ensures that any vertical motion like jumping won’t just continually push the character into the air but instead will form a parabolic arc bringing the character back down.

Text

Description automatically generated

**3.0 Development Blog**

**Entry 1 – 11/12/2021**

Started game planning. Taking into account the possible extension material decided on a 2d platformer game.

Game contents:

Left/right motion – Controlled by two on screen buttons at either side of the screen.

Jumping – Button in middle of screen, uses gravity to pull player back down.

Gravity – Extension material and seems fairly simple to implement and necessary if I want a good 2d platformer.

Enemy? – Possible enemy unit that can also walk around? Maybe chases player so they can’t stay still? Or just simplistic goomba esk mob? Basic AI FSM.

Platforms – Stationary, moving, weighted? (One moves down while other moves up?)

Score system – Maybe collectables to increase score? With a time based score to complete level?

Levels? – Unsure how would implement yet but could maybe set up multiple levels? If level pieces were made modularly could randomly generate levels? If randomly generated could be infinite until player dies then check score?

Player – Needs animations to look good. Collision detection so doesn’t pass through floors or walls,

Parallax Background? – Would make game more visually appealing.

UI – use a good font to show score. Time alive? Slightly transparent control buttons so doesn’t block any useful details?

Sound – Background music for level, appropriate sound effects

Intro screen – Play, adjust sound settings,

Pause screen? – pause mid game to end run early or adjust sound settings?

Game over screen – When player dies or runs out of time show game over? Show score, time lived? And maybe give option to tweet about it??? (Unsure what this would entail just yet and may change mind later if too complex)

**Entry 2 – 14/12/2021**

Set up project in android studio and began to work on some base systems. Began to work on basic player that can be controlled to move around the screen. Started with a basic square character. Decided to use a velocity-based system as opposed to a coordinate movement to make much smoother movement.

**Entry 3 – 16/12/2021**

Started work on gravity system just making it so it couldn’t fall off the screen. Had to fiddle with the values of the up arrow to create a ‘jump’ rather than the character slowly moving up the screen.

**Entry 4 – 20/12/2021**

Foreseeing some future issues began setting up an engine system that could create a pseudo delta time and allow for smooth movement based on the frame rate of the device.

**Entry 5 – 27/12/2021**

Started abstracting appropriate code into their own scripts to make the code much easier to look through. Movement based stuff was put into the controller script whilst the delta time and other useful engine-based code was put through into the engine script.

**Entry 6 – 29/12/2021**

Decided on a tile-based system for background since it would allow for a simpler level designing. Decided on creating an array to help with my own visualisation for the scene I wanted to make. Used different coloured squares to start with

Swapped from scroller platformer to level-based platformer since would be quicker to produce a map for single screen levels than it would to produce a longer one and then figure out how to make it scroll along the level.

**Entry 7 – 3/1/2022**

Found a suitable sprite sheet that had tile sets, character animations and some bonus details like grass and collectables and some menu themes and game theme.

**Entry 8 – 5/1/2022**

Implemented sprite sheet into tile-based background to make an appropriate looking pixel art game. Though currently character is just restricted to walking on the ‘floor’.

**Entry 9 – 7/1/2022**

Started work on collision system. Plan is to assign a value to each tile and use it in a case-based array to tell the player what to do when they interact with specific cases. Such as being unable to pass through a floor you are stood on top of or walk through a wall.

**Entry 10 – 11/1/2022**

Managed to get collision working on a wall piece. Discovered I needed several different cases for the different types of platform and angles. Also managed to make a platform that could be passed through from the bottom to create a platform that could be useful in level designs. Also implemented player sprite animations for movement to make the player look more interactive with the world and appropriate for a platformer based game.

**Entry 11 – 12/1/2022**

Scrapped enemy idea. Was taking too much time to implement and was being a general annoyance when I tried running it… UI was being uncooperative so was only able to get the exit button on and working. Because I had developed the game system so intuitively around using keyboard inputs can’t seem to get on screen buttons to work correctly.

**Entry 12 – 15/1/2022**

Began development of menu scene and game over scene. Centred buttons and put a background image that was thematic to the game.

**Entry 13 – 16/1/2022**

Started work on collectibles. Set up array of locations for collectibles to spawn and assigned their frames from the tile sheet. Added animation for them to spin a little.

**Entry 14 – 17/1/2022**

Tried to add in grass at the same time as the collectibles but can’t seem to get them to work. Not sure what’s wrong since project is getting quite complicated at this point but without the collectible then there wont be a way to increase the score.

**Entry 15 – 18/1/2022**

Set up background theme music for each scene.

**Entry 16 – 19/1/2022**

Implemented Both menus in their current state. Although unable to get scoring system working put in temp place holder to show where it would be and set restart button to take back to the first screen. Game is playable as a platformer level but couldn’t get it to be more gamified.

**Entry 17 – 20/1/2022**

Finalised some of the code to be submitted for the github

**Entry 18 – 21/1/2022**

Polished up documentation.