**Game Summary**

Game Title

The Tower!

Introduction

This document summarises the design and implementation of the game. Appendix 2 contains the summary of the issues and how they were solved.

Objectives

The objective of the game is to rescue all of the villagers that are being held within the tower by the evil elves which have invaded the tower of the land. The player will need to kill all of the evil elves to eliminate the threat against the village.

Rules

* Rescue all villagers to beat the game
* WASD controls for movement
* Left click to shoot fireballs
* Collecting coins allows for upgrades
* If you lose all your lives the level resets
* Collect keys to unlock doors

Gameplay

The gameplay is based on the game “Hammerwatch”. It has a very similar art style and has inspired gameplay from “Hammerwatch”. The Tower is an top down RPG action game set in a fantasy world in 1400 A.D. It is made to be a enjoyable experience for theplayer.

The outdoor zone is a open level which allows the player to explore the fantasy world giving an RPG feel. Exploring the outdoor zone helps the player in taking the tower back from the evil elves because there are coins scattered around the map which can be used to buy health upgrades. The music is very peaceful to represent lack of danger outside the tower.

The tower level is a lot more dangerous than the outdoor zone, the player has to fight through the evil elves to rescue all of the villagers also the player needs to find keys to open doors. The music used in the tower level is very action packed which gives a more intense vibe while in the tower.

**Research**

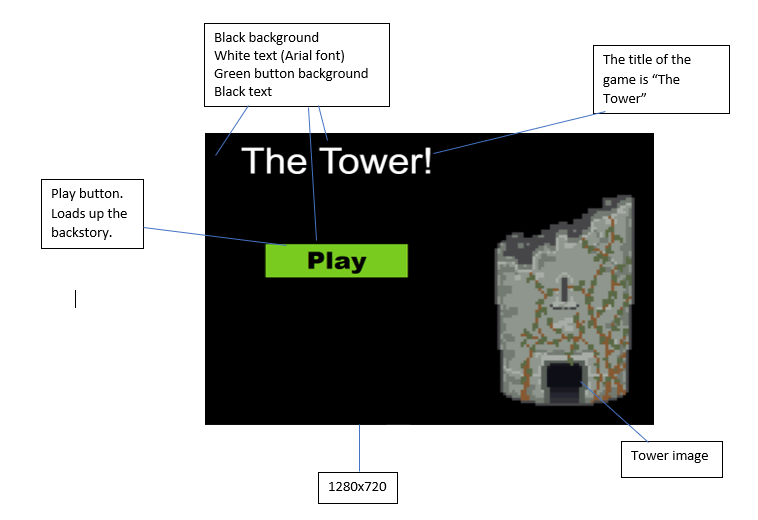
Watched a few videos on how Hammerwatch looked and played, inspiration was taken to create The Tower. The expectations of the game would not be as high as Hammerwatch due to time constraints and needing to learn Phaser coding even further.

For further research, the game was play tested by other colleagues on the course in order to hear about their thoughts if anything needed improving or changing, some input from them was that there should be a clearer path to locate the tower and that the outdoor zone was a bit sparse of things to do. A mossy stone path was included for locating the tower and coins were added to the outdoor zone.

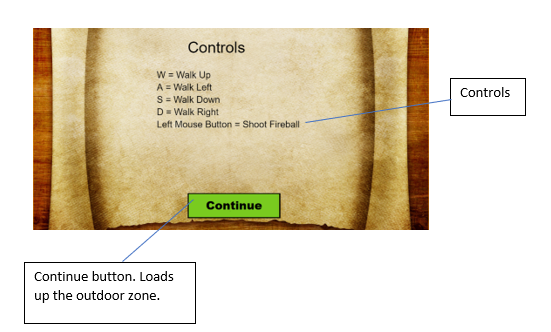
They also commented on the player’s movement speed and difficulty of the tower level. Therefore the players velocity was slightly increased and one of the arrow traps near the start of the level was removed to reduce difficulty.

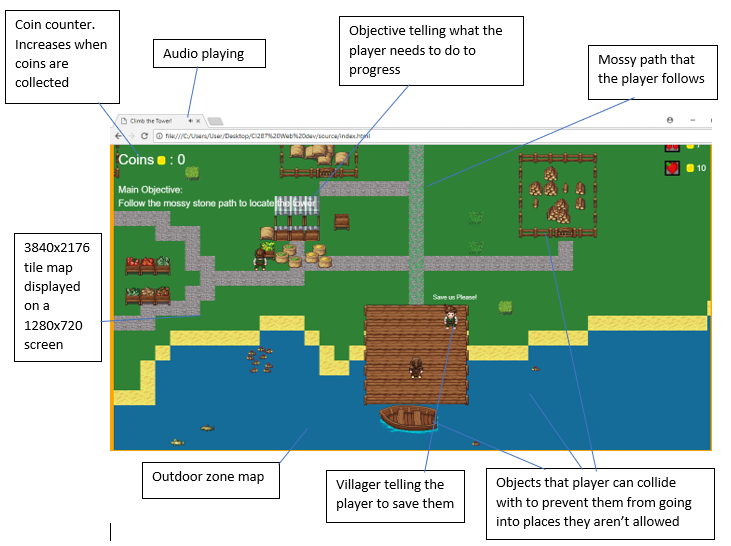
A minor error was noticed in the code which was that the coin sound was only working in the tower level, this error was rectified. Finally they mention that the controls for the game should be displayed somewhere so a controls screen was added when you start the game.

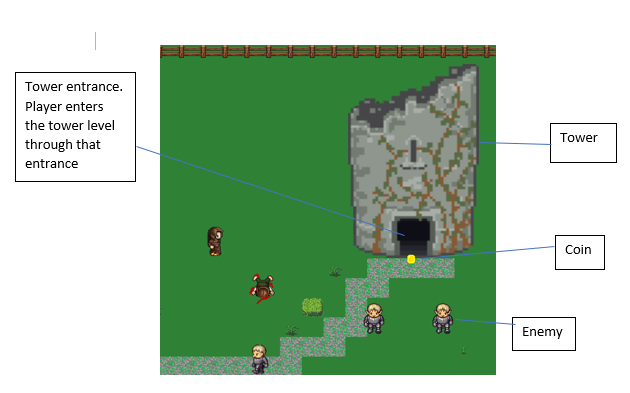
**Screen/Level Map**

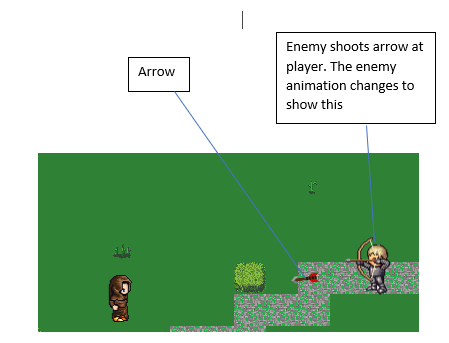
****

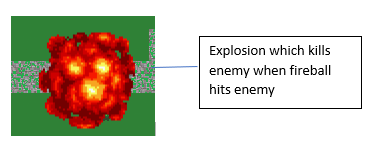
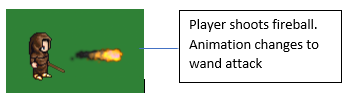


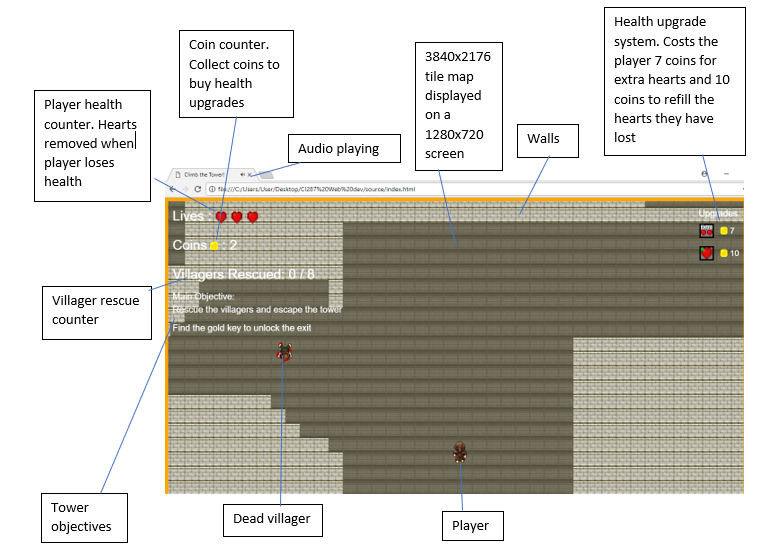
****

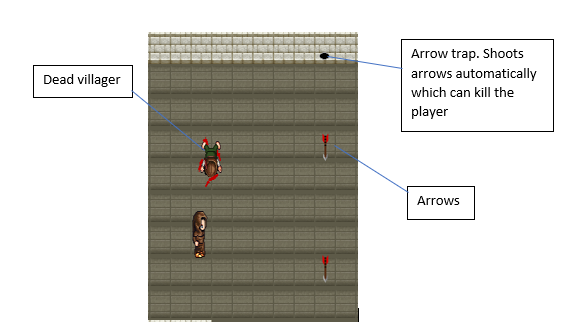
****

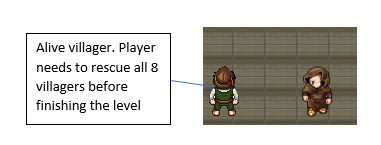
****

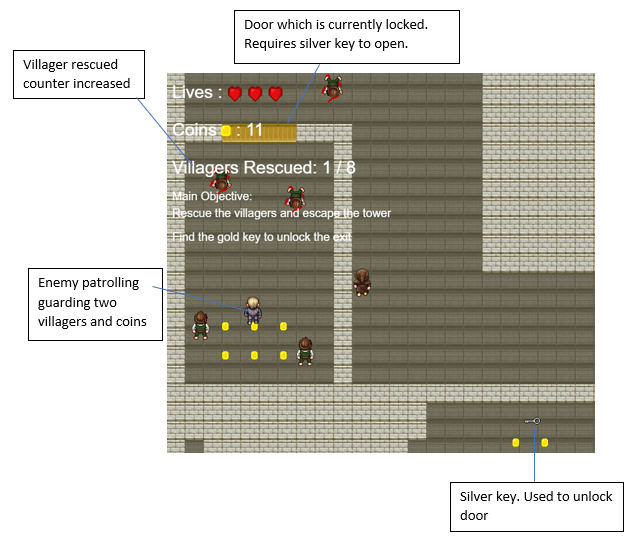
****

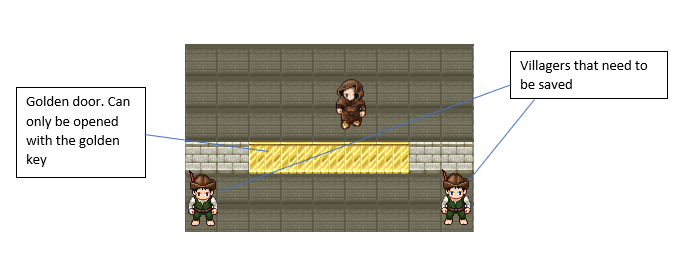
****

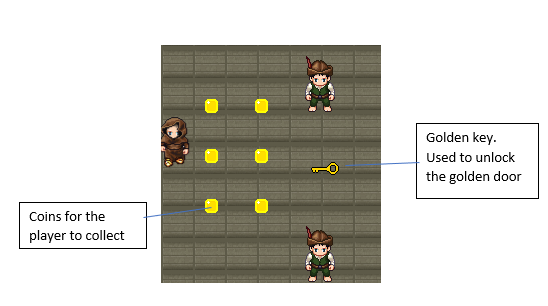
****

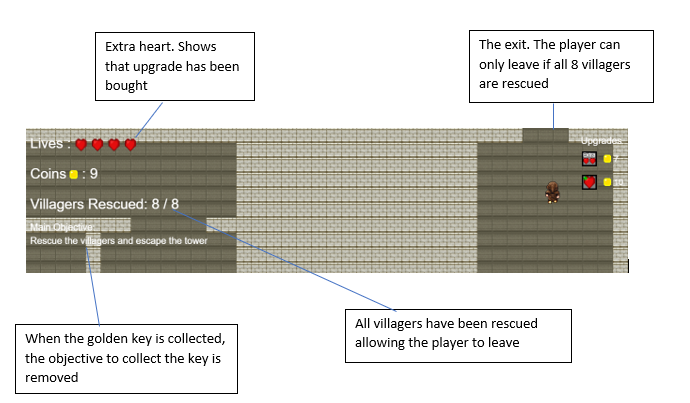
****

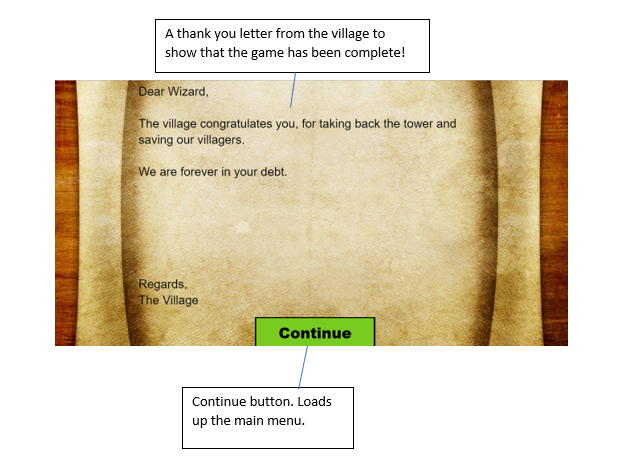
****

****

****

****

****

****

**Implementation Specification**

Functions

**Preload**- loads in all of the sprite assets and audio before the game runs

**Create**- creates and places all of the objects into the game

**Update**- used to change the state of the game while it is running (e.g. moving player when button is being pressed) and constantly checks for collisions

**shootFireball**- function which allows the player to shoot a fireball

**createCoins**- creates the attributes of coins

**createEnemiesUp**- creates the attributes of the enemies which patrol upwards

**createEnemiesLeft**- creates the attributes of the enemies which patrol to the left

**createEnemiesDown**- creates the attributes of the enemies which patrol downwards

**createEnemiesRight**- creates the attributes of the enemies which patrol to the right

**addExplosionToEnemies**- makes the enemies explode when they collide with a fireball

**fireballHitEnemyUp**- kills an enemy\_up object if they collide with a fireball

**fireballHitEnemyLeft**- kills an enemy\_left object if they collide with a fireball

**fireballHitEnemyDown**- kills an enemy\_down object if they collide with a fireball

**fireballHitEnemyRight**- kills an enemy\_right object if they collide with a fireball

**ShootArrowUp**- allows the enemy\_up enemies to shoot arrows upwards

**ShootArrowLeft**- allows the enemy\_left enemies to shoot arrows to the left

**ShootArrowDown**- allows the enemy\_down enemies to shoot arrows downwards

**ShootArrowRight**- allows the enemy\_right enemies to shoot arrows to the right

**arrowHitsPlayer**- causes the player to lose a heart of health when they get hit by an arrow

**collectCoin**- keeps count of all of the coins that the player collects when they collide with a coin

**extraHeart**- gives player an extra heart when they buy the upgrade but removes 7 coins

**heartRevive**- revives all of the hearts that the player has lost when they buy the upgrade but removes 10 coins

**arrowOrFireballHitWall**- kills a arrow or fireball when they collide with any walls

**respawnPlayer**- when the player dies, the game state will restart and respawn the player

**goToTowerLevel**- starts the tower level when player enters the tower

**buildWorld\_outdoorZone**- builds the outdoor zone map into Phaser

**buildWorld\_towerlevel**- builds the tower level map into Phaser

**arrowTrapShoot**- shoots arrows out of the arrow trap

**createVillagers**- creates the villagers that need saving

**villagerSaved**- keeps count of the amount of villagers that have been saved whilst killing the villager objects

**collectSilverKey**- indicates that the silver key has been retrieved and opens the wooden door

**collectGoldKey**- indicates that the golden key has been retrieved, opens the golden door and removes the golden key objective

**Ending**- loads the win screen to indicate the end of the game

Objects

**Player**- the object the user controls to progress through the level

**Enemy\_up**- the object for the enemies that patrol upwards

**Enemy\_left**- the object for the enemies that patrol to the left

**Enemy\_down**- the object for the enemies that patrol downwards

**Enemy\_right**- the object for the enemies that patrol to the right

**Coins**- objects that the player collect to buy upgrades

**Arrows**- the projectile that harm the player when they collide with the player

**Arrow Traps**- produces a continuous amount of arrows

**Hearts**- used to indicate the amount of lives the player has before they die

**Villagers**- innocent people that are in danger. They can be collectable objects which the player must save from the evil elves

**Fireballs**- the projectile used to kill the enemies

**Walls**- the object that prevents the player from going out of bounds

**Doors**- objects that prevents the player from entering certain rooms unless they have collected the key

**Ground**- the object that the player walks on

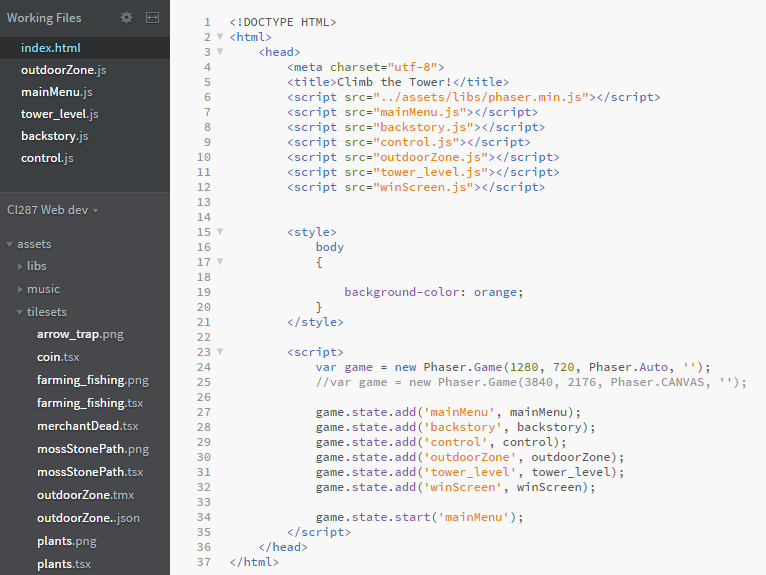
**Villagers Dead**- sprites of any villagers that the evil elves have killed

**Silver key**- used to open the wooden door

**Golden key**- used to open the golden door

Code Structure

Index.html



This screenshot is the index.html file, this is where you set up some basic HTML and include all of your library’s / scripts. Each JavaScript script included is the levels / states within the game for example ‘mainMenu.js’ is the main menu of the game.  
Lower down in index.html on line 24 the Phaser game is created with of width of 1280px and a height of 720px however that's just the viewport on the screen, the actual game is 3840px by 2176px this is done to make the game feel more like an RPG you have to explore the map. On lines 27 - 32 the state machine is created, Phaser requires states if you want to change to a different scene / level. 6 states are created using the game.state.add() function, this function takes 2 parameters, first being the name of that state and then the name of JavaScript file you want to jump to. After the states are set up the function game.state.start() is used which takes a state as a parameter, the main menu state is used so that when the game starts your taken to the main menu.

Initially a JavaScript file called Main.js was created and stored all this information excluding the state machine, however when the game was played many errors would occur such as “Phaser.StateManager - No state found with the key: mainMenu”. After many hours of trying to fix this it turned out to be a scope issue with the states, when they were inside Main.js the states couldn’t find the other JavaScript files, the solution was to put them inside index.html meaning no scope issue could occur.

Main Menu



The ‘GAMEHEIGHT’ and ‘GAMEWIDTH’ variables store the width and height of the game. This is done so that if the width or height needs to be changed, then instead of changing all the code which includes it, only the variable values need to be changed. The variable ‘mainMenu’ stores the entire main menu script because this is the main menu state which is used to load up the main menu, therefore it needs to store all the code inside of it. This is the same for the other Javascript files. The preload function is used to load in all of the images used for the main menu. The create function is used to design the main menu and to add the button which calls the ‘gotoBackstory’ function. This function will load up the backstory screen.

Backstory



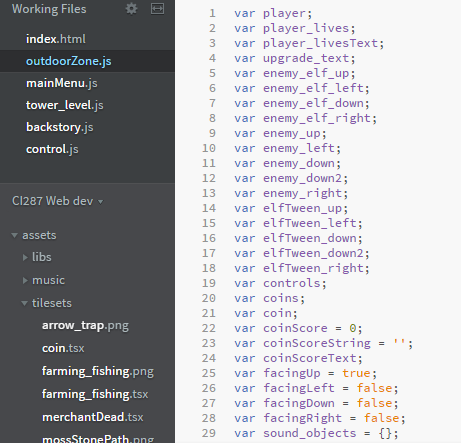
The backstory.js file is very similar to the mainMenu.js file. The preload function loads all of the images before the backstory loads onto the screen. The create function adds the scroll background onto the page and adds the text onto the screen (the villagers message to the wizard). A button is also created which calls the ‘gotoControls’ function. This function loads the controls screen.

Control.js



As the images used for the controls are used for the backstory, there is no need for a preload function as all images are loaded. The create function is also basically the same except for the text being displayed is different. Also, the button calls a function called ‘getOutdoorZone’ which loads up the outdoor zone state.

Outdoor Zone



These are all of the variables used for the outdoor zone.



This function adds the tilemap into the game and makes it accessible to the game. All the tilesets and layers are loaded into phaser and all of the wall layer are made collidable. A program called ‘Tiled’ was used to create the tilemap. The resizeWorld() function sizes the world to the world created in the Tiled program.



The preload function will load all of the assets before the game is loaded onto the players screen. For any sprites that include animation, a spritesheet is used as it requires multiple images are being used for a single object. The spritesheets include numbers after the image files are called. These numbers represent each sprites width, each sprites height and the amount of sprites that are used in the spritesheet. The preload function also loads the tilemap into Phaser via a .json file. For Phaser to read the tilemap, it needs to be converted into a .json file. All of the audio is also preloaded.



The create function includes a lot of different things because a lot of objects are being created. The line of code ‘this.game.physic.startSystem(Phaser.Physics.ARCADE)’ enables physics into the game. All of the sound effects are added to the ‘sound\_object’ variable and then the level’s music is set to play as soon as the game is loaded (on loop). The tower is added to the game using the ‘towerSprite’ image and the physics is enabled, allowing the player to collide with the object. All of the merchants that will be used are also created and added here.

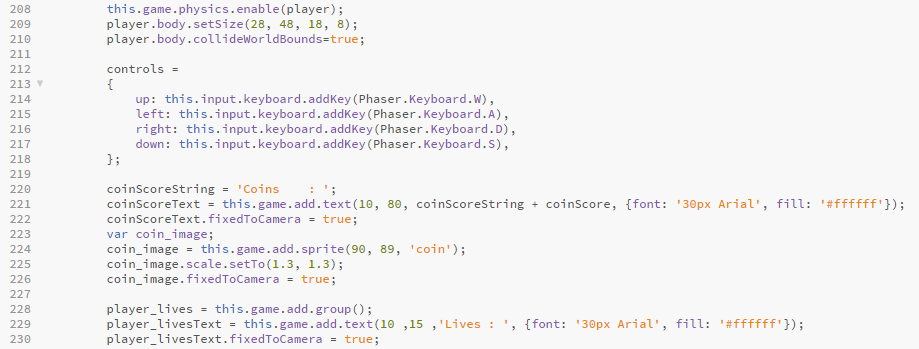


Many groups are created in the create function. The ‘coins’ group is used to group all of the coins used in the game together. It then calls the function ‘createCoins’ in order to spawn the coins into the outdoor zone level. This is the same for the ‘enemy\_elf\_left’ group and the ‘enemy\_elf\_down’ group. However, because the create enemy functions can only create one enemy, multiple enemy functions are needed. This is even in the case that they are a part of the same group. It had to be done like this otherwise only one enemy would be able to shoot arrows at a time and the arrow spawn position would sometimes not be in the same position as the enemy.

The ‘fireballs’ group is different to the other groups displayed above because instead of a function being called to create them, they are created in the create function. The fireballs are given a body and physics is enabled, allowing them to move and collide with other objects. The createMultiple feature tells the game the amount of fireballs that can be put onto the screen (in this case 20). The fireballs are also told that if they hit the bounds of the game then they are killed.



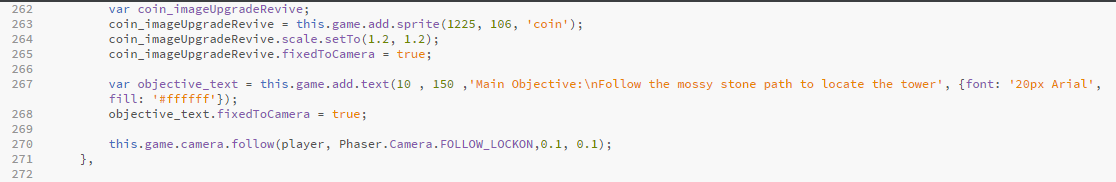
The arrow group is laid out the exact same as the fireballs. The only difference between the projectiles is that more arrows can be displayed onto the screen. After the arrows are created, the player is created and positioned into the world. Animations for the player are created using the sprites in its spritesheet. Any of these animations can be played at any point of the game depending on the players action.



body.setSize sets the size of the collision box for the player. This was used to make the collisions more precise. The controls variable stores all of the movement controls in it so that they are grouped together and can be referenced easier later on. The coin counter is then displayed and fixed onto the screen in order to tell the player how many coins they have.



The screenshot above shows the hearts being created. The for loop positions all the hearts onto the UI and then using ‘.fixedToCamera’ the hearts will follow where the camera is, this is also the same with the outline of the hearts (used to show heart has been lost). The upgrade UI is then created which includes the different types of upgrades and their costs. The coin sprite is used to indicate that the player uses coins to buy the upgrades, the price of the upgrade is displayed next to the upgrades. The different types of upgrades are extra heart and upgrade revive. Both upgrades are buttons but they both call different functions to execute different functions.



The game includes objectives which the player must complete to progress through the game. The objective is created in the create function and is attached to the UI so that when the player moves, the objective will stay on the screen. The code ‘this.game.camera.follow(player,Phaser.Camera.FOLLOW\_LOCKON,0.1,0.1)’ ensures that the camera follows the player at all times, this code includes a ‘lerp’ as well meaning the camera transition is a lot smoother, this smoothness can be changed by adjusting the last two paraments (0.1, 0.1).



The update function will continuously update when the game is running. The first few lines of code check for collision between two different objects and it will check this continuously throughout the game. The collision will become active when the bounding boxes overlap each other. The if statement is for when the player dies. If they have less than one life left then the ‘respawnPlayer’ function will be called. This is done in a timed event which means that it will take a number of seconds (this case 3 seconds) for the function to be called. The players velocity is set to 0 which means that when the game starts, they are not moving.



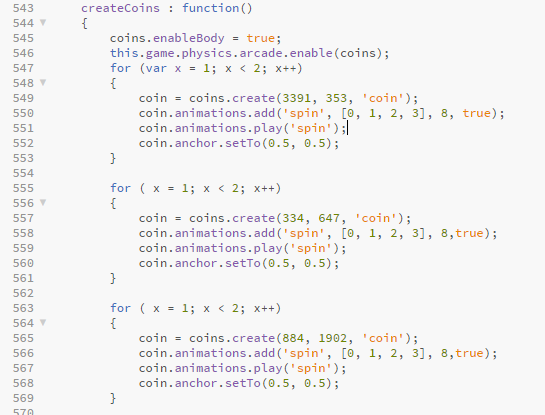
If statements are used to determine where the player is moving. The player can move in 8 different directions. The two directions that are being shown above are North-West and North-East. For the player to move North-West (for example) , the player must be holding the “W” and “A” keys together for that movement to be performed. The animation and velocity of the player will be altered depending on the keys being inputted by the user.



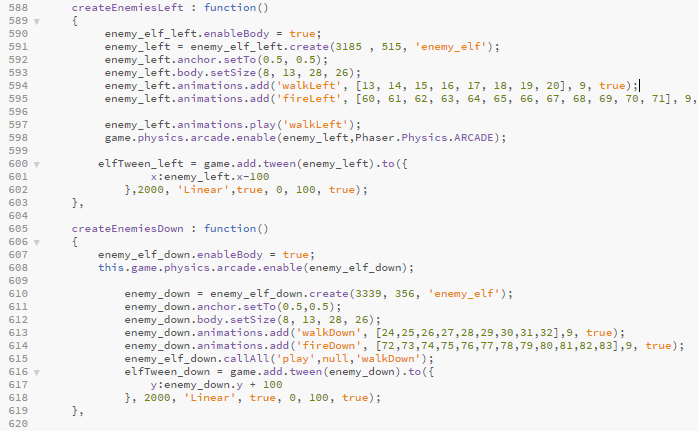
The code above is for the player’s attack. It consists of a if statement which checks if the left mouse button has been clicked. If it has, then a series of if statements will be used to check which direction the player is facing and depending on the direction they are facing, a different animation will play.



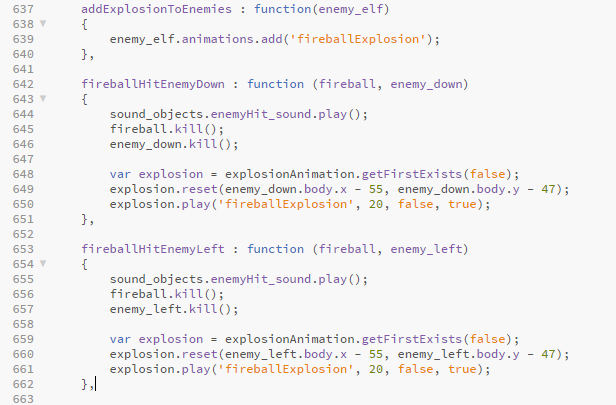
The if statements above determine if the player is in the range for the enemy to attack them. If they are then the enemy will perform the attack animation and the enemy tween will be paused. A function that causes the enemy to shoot an arrow is called which means that the enemy is attacking the player. If they are not in range then they will just patrol as normal. The range setting is dependant on the type of enemy it is (e.g. enemy patrolling upwards).



This function is used to create and place the coins onto the map. Multiple for loops had to be used so that they can be placed in the desired positions without having to create a different variable for each one. Each coin is given a spin animation which causes the coin to spin.



These are the functions used to create the enemies. For this to work, each enemy had to have its own function. Each function consists of all the attributes the enemy will have when they are created. They are all given bodies and physics so that they can collide with other objects and they are given different animations depending on their patrol path. Each enemy will have a tween which they will follow. A tween is what is used to allow the enemy to patrol the area. The enemy will continuously move up and down or left and right while they are patrolling the area.



The ‘addExplosionToEnemies’ function takes in the enemy\_elf as a parameter and adds an explosion to them, this will apply to all enemies. There are 4 different types of ‘fireballHitEnemy’ functions which are ‘fireballHitEnemyUp’, ‘fireballHitEnemyLeft’, ‘fireballHitEnemyDown’ and ‘fireballHitEnemyRight’ (screenshot only shows two of them). These functions take the parameters fireball and the type of enemy (e.g. enemy\_left). In the functions, the sound effect of the enemy being hit is played and the enemy is killed alongside the fireball. The explosion animation will then play around the enemy’s position.



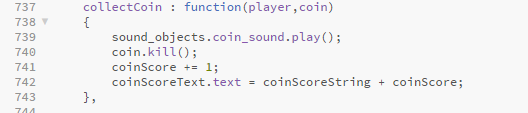
The functions above are to enable the enemy to shoot arrows at the player. The function is called in the update function so that it will be called every time it is needed (just like most of the other functions).

‘arrow.rotation = this.game.physics.arcade.angleBetween(arrow, player);’

will rotate the arrow so that the point of the arrow is aimed at the player. This was a massive problem at first because when the arrows were shot, they would be facing the wrong direction and it looked very unrealistic. This code fixed the problem. Also, the code ‘game.physics.arcade.moveToObject(arrow,player,200);’ will cause the arrow to move to the players position when it is shot (it does not follow the player). It includes three parameters, the moving object, where to move the object to and the speed it moves at.



The ‘arrowHitsPlayer’ function determines what happens when the player is hit by an arrow. It takes in the player and arrow as parameters. The arrow object is killed as well as a heart. The sound effect of the player being hit will be played as soon as the collision happens. If the player has less than one heart and is hit, then all the controls are disabled and the player dies. The code ‘game.add.tween(player).to( { alpha: 0 }, 2500, Phaser.Easing.Linear.None, true); will fade the player off the screen before they respawn.



This code happens when a coin is collected by the player. The sound effect will be played and the coin object will be killed so the player cannot get it again. Lastly the coin total will be incremented by one.



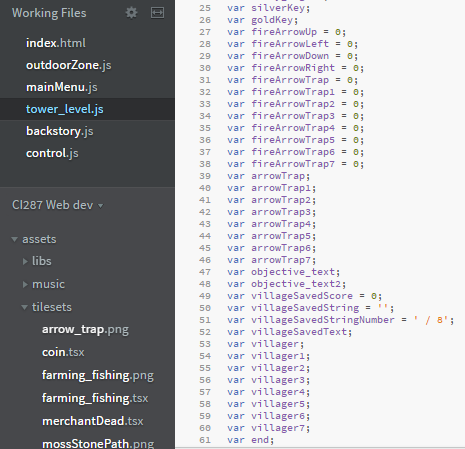
The extra heart function is activated when the player clicks on the extra health upgrade button, it will give an extra heart to the player when it is bought. However, it is only limited to two extra hearts. If the player has not bought this upgrade yet, then a heart will be position next to the third heart but if it has been bought then it will be placed next to the position of the fourth heart. Before the upgrade is bought, the game will check the amount of hearts the player has and the amount of coins that they have. If they have not got seven coins then the upgrade will not be able to happen as they cannot afford it. If they do have seven coins, then seven coins will be removed and the heart will be added to their total lives.



The revive upgrade allows the player to regenerate any hearts that the player has lost. This requires the player to have ten coins and if they do then ten coins will be removed and the upgrade will commence. ‘arrowOrFireballHitWall’ will kill the arrow or fireball object if they hit a wall. It only occurs when a collision between one of these objects and a wall is triggered. The ‘respawnPlayer’ function will stop the music and will reset the state of the game. Resetting the state means that the game will refresh and the player will have to start the level again. ‘respawnPlayer’ only occurs when the player dies. The ‘gotoTowerLevel’ function is called when the player enters the tower that’s placed in the outdoor zone. It will load up the tower level.

Tower Level

The code of the tower level is very similar to the code in the outdoor zone so this will only cover the code which has not been covered previously.



These are all of the variables that the tower level contain that aren’t in the outdoor zone.



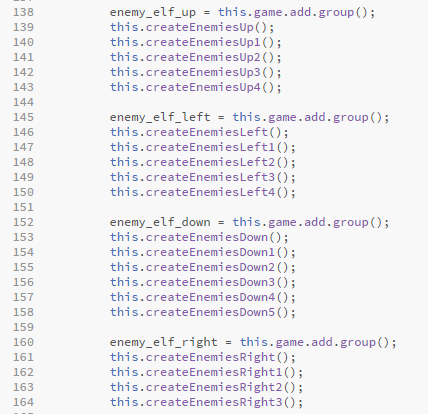
Var world\_towerLevel includes all of the variables that are apart of the tower level made in the program Tiled. The function below it is like the function ‘buildWorld\_outdoorZone’ but for the tower level instead of the outdoor zone.



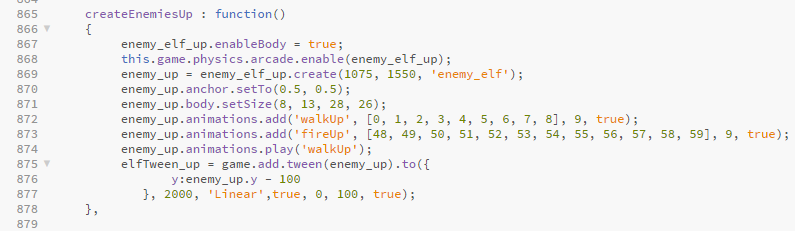
Most of the objects are preloaded for the outdoor zone so there is no need to preload them again. Objects that are not included in the outdoor zone will need to be loaded in (e.g. the tower level map).

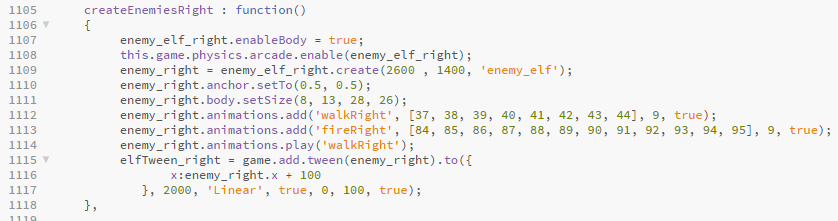


The create function is very similar to the create function in outdoor zone. All the objects are created into the game in this function. However more objects are included in the tower level so this create function is bigger. Each arrow trap is created individual so they are all declared as their own variable and positioned onto the tile map using the arrow trap sprite.

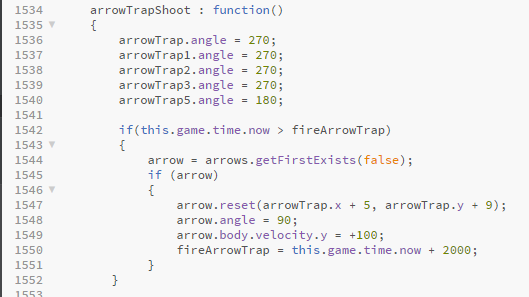


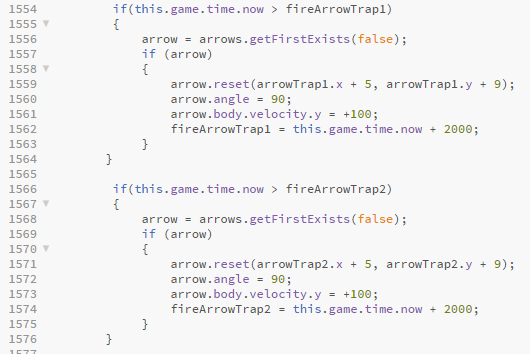
Every enemy is apart of one of the four groups. However, each enemy has their own create function in order to work properly and all of these functions are called here.





These create enemy functions displayed above are very similar to the ones shown in the outdoor zone code but they is specified for the enemies facing up and facing right, unlike the other ones which were specified for the enemies facing left and down. The main difference between these ones and the other ones are the animations being added and played. As well as this, the direction of the tweens are different. For example, the enemy left tween will decrease the x axis by 100 to move the enemy to the left. For the enemy to move up, the y axis is decreased by 100 while the x axis is not altered.

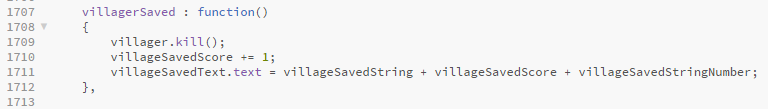




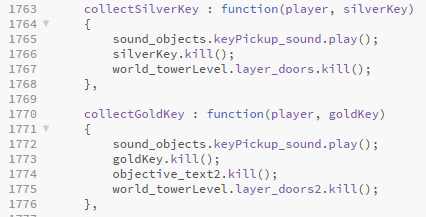
The ‘arrowTrapShoot’ function is very similar to the ‘shootArrow’ functions for the enemies. Each of the arrow traps are rotated to face the direction that they are shooting out of. The if statement ‘this.game.time.now > fireArrowTrap’ is checking to see if current time is greater than the delay between each arrow shot, if this is true then the next arrow can be shot. Then the function checks if there's an arrow available in the ‘arrows’ group and if there is shoot an arrow out of the position of the arrow trap and angle it to the direction of the arrow trap. Lastly give the arrow a velocity and the direction it needs to move in using either ‘arrow.body.velocity.x’ or ‘arrow.body.velocity.y’ and then reset the delay between each arrow by 2 seconds using ‘this.game.time.now + 2000’.



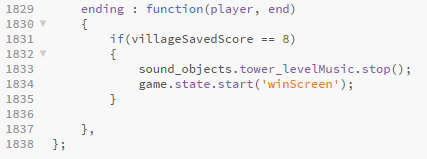
The ‘createVillagers’ function is called in the create function, this function spawns in all of the villagers into the tower level. Firstly the villager are physics enabled so that the player can collide with them then each, then each villager is created and added to the ‘villagers’ group with different positions on the tower level. Lastly some villagers have different sprites either ‘merchant’ which is facing forwards or ‘merchantBack’ which is facing backwards.



The ‘villagerSaved’ function is called when the player collides with a villager, when this occurs the villager is killed and then the villager saved score is increase by 1.



Both of these functions are called when the player collides with either the gold or silver key, when the collision happens the key pickup sound effect is played and then the key itself is killed. After that depending on which key was collected one of the door layers will be killed which basically unlocks that door. Silver key unlocks ‘layer\_doors’ and the gold key unlocks ‘layer\_doors2’. Lastly the gold key function also kills ‘objective\_text2’ which removes the gold key objective from the screen.



The ending function is used as a check to see whether all of the villagers have been rescued when the player collides with the invisible walls at the end of the game called ‘end’ which is inside the function parameter, using the code ‘villageSavedScore == 8’. If this is true then the music is stopped and you swap to the winScreen state/level.

Win Screen



winScreen.js file is very similar to the mainMenu.js file. The preload function loads a sound effect. The create function adds this sound effect to the ‘sound\_objects’ variable and then that sound effect it set to play when this level loads up. The scroll background is added to the page and then text is drawn onto it (the villagers message to the wizard). A button is also created which calls the ‘gotoMainMenu’ function. This function loads the main menu screen.

Assets

Fireball sprite sheet, this is used for the player’s projectile, this asset was from the internet (linked in references).

Fireball explosion sprite sheet, used when a fireball hits an enemy, this was from the Phaser website.

Player hearts sprite, this is used to indicate the player’s health, this asset was from the internet (linked in references).

Tower sprite, this is used for the tower on the outdoor zone map and also used as a image in the main menu, this asset was from the internet (linked in references).

Scroll background, this is used for the background in most menus, this asset was from the internet (linked in references).

Villager / merchant sprites, this is used for merchants and villagers in both levels, this asset was from the internet (linked in references).

Enemy elves sprite sheet, this is used for enemies in both levels, this asset was from the internet (linked in references).

Player sprite sheet, this is used for the player, this asset was from the internet (linked in references).

Outdoor zone tileset, this tileset was used in the Tiled program to create the outdoor zone, this asset was from the internet (linked in references).

Tower tileset, this tileset was used in the Tiled program to create the tower level, this asset was from the internet (linked in references).

Music outdoor zone, this music is played when your in the outdoor zone, this asset was from the internet (linked in references).

Music tower level, this music is played when your in the tower level, this asset was from the internet (linked in references).

Music win screen, this music is played when you beat the game and enter the win screen, this asset was from the internet (linked in references).

Sound effect fireball, this is used when the player shoots a fireball, this asset was from the internet (linked in references).

Sound effect key pickup, this is used when the player walks over a key, this asset was from the internet (linked in references).

Sound effect player dead, this is used when the player loses all their lives and dies, this asset was from the internet (linked in references).

Arrow sprite was made by Sam using GraphicsGale, this is used as the enemy’s projectile.

Coin sprite was made by Sam using GraphicsGale, this is the currency in the game.

Both gold and silver key sprites were made by Sam using GraphicsGale, their used to open doors.

Mossy stone path was made by Sam using MS Paint, edited the original stone path and added the green to it to make a mossy effect. Used as a path to locate the tower.

Arrow trap sprite was created by Bradley using MS Paint, this is used throughout the tower level as a trap.

The play and continue button sprites were created by Bradley using MS Paint, their used in the menus of the game.

The heart outline sprite was made by Bradley using MS Paint, this is used when the player loses a life.

Extra heart upgrade sprite and the heart revive upgrade sprite were made by Bradley using MS Paint, these are the upgrade icons.

Upgrade sound effect was made by Bradley using the BFXR game sounds program, used when the player buys an upgrade.

Player hit sound effect was made by Bradley using Audacity, edited the player death sound and took part of that. Used when the player is hit by an arrow.

Enemy hit sound effect was made by Bradley using the BFXR game sounds program, used when an enemy gets hit by a fireball.

Coin collect sound effect was made by Bradley using the BFXR game sounds program, used when the player walks over a coin.

**Implementation Evaluation**

Phaser is a great library for creating browser games because it’s designed for Canvas and WebGL also it’s free. Phaser can be for Windows, Mac OS and Linux. The program Tiled which is compatible with Phaser allows you to create level maps with ease because Phaser supports JSON and CSV file types.

Groups are a feature in Phaser which allows for an object to be duplicated multiple times with the same properties for example bullets would use a group because each bullet wants to act the same way. However groups can be used for objects that need slight changes for example a gun, one gun in the group may have a fire rate of 10 however another gun in that group may have a fire rate of 20, groups allow for a lot of control over objects.

Issues with Phaser have been errors codes that you receive in the browser console, most of the time they don’t help at all and aren't relevant to the actual issue. Furthermore as Phaser uses Javascript you can still run the game with errors and not even know about it, which will give you errors in the console as well making it hard to debug the code. Lastly Phaser seems to have audio issues with Google Chrome.

The game was development for the PC, meaning the keyboard and mouse is available, this allows for better controls and more options for key bindings also it’s very likely that the PC’s monitor is going to be big so the game worlds can be bigger. PC’s will also have audio and a internet connection which is good because Phaser games are meant for the internet. If the game was on a mobile the design is limited to a touch screen so there's less options on what you can do, also the phone may not have access to the internet meaning the Phaser games can’t be played at all. Latest the game can have better graphics because a PC will have a lot more CPU / GPU power to process the game.

**References**

Enemy sprite-

<http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/#?weapon=bow&sex=male&eyes=blue&legs=none&clothes=longsleeve_white&hat=none&armor=chest_plate&shoes=boots_metal&=eyes_gray&body=tanned&nose=big&ears=elven&ammo=arrow&hair=plain_white&gloves=metal&shoulders=none&mail=chain&greaves=metal>

Player sprite- <http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/#?weapon=wand_wood&sex=male&eyes=blue&legs=robe_skirt&clothes=longsleeve_brown&hat=hood_cloth&armor=none&shoes=boots_golden&=shoes_brown>

Villager sprite <http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/#?clothes=formal&formal-shirt=1&formal-vest=1&formal-pants=1&hair=bedhead_raven&belt=leather&hat=cap_leather>

Tower tileset- <https://forums.rpgmakerweb.com/index.php?threads/lunas-tiles.84/>

Player Movement- <https://www.youtube.com/watch?v=KT4GqDlFMhk>

Animations- <https://www.youtube.com/watch?v=kMZ2QkHwOTE>

Coin collection <https://phaser.io/examples/v2/tilemaps/create-from-objects>

Phaser documentation seeing what function parameter are <https://phaser.io/docs/2.4.4/Phaser.Tilemap.html>

This forum talked about using the GID ID

<https://stackoverflow.com/questions/42039879/createfromobjects-not-working-in-phaser>

Outdoorzone Music <http://soundimage.org/wp-content/uploads/2015/06/The-Concrete-Bakes.mp3>

How to add animation to sprites <https://stackoverflow.com/questions/30090258/how-to-add-animation-to-a-group-sprite-in-phaser-js?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>

Issue with music in Google Chrome, used this code to fix the issue <https://github.com/photonstorm/phaser-ce/issues/437>

OutdoorZone tileset <https://opengameart.org/content/lpc-farming-tilesets-magic-animations-and-ui-elements>

Heart image <https://opengameart.org/content/heart2-1616>

Fireball image <https://opengameart.org/content/fireball-spell>

Fireball sound effect <https://www.freesoundeffects.com/free-track/fireball-1-466500/>

Scroll image <http://www.pptbackgrounds.org/advanced-blank-scroll-paper-backgrounds.html>

Tower sprite <https://www.scirra.com/forum/cancelled_t92628?&start=50>

Player death / hit sound effect <https://opengameart.org/content/hurt-death-sound-effect-for-character>

Fireball explosion animation from phaser website

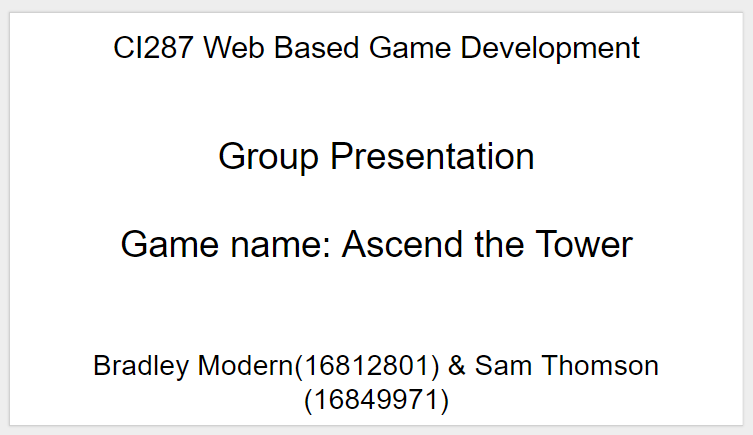
Key sound effect <https://opengameart.org/content/key-pickup>

Tower level music <http://soundimage.org/wp-content/uploads/2017/06/Guitar-Mayhem-2.mp3>

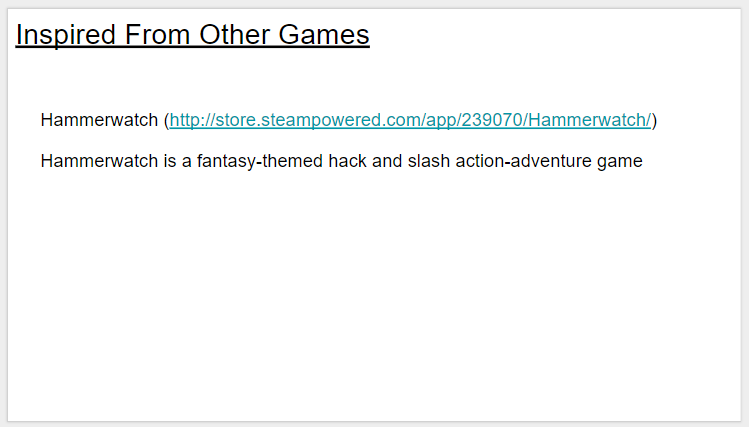
Winscreen sound effect <http://www.freesfx.co.uk/rx2/mp3s/4/16055_1460555775.mp3>

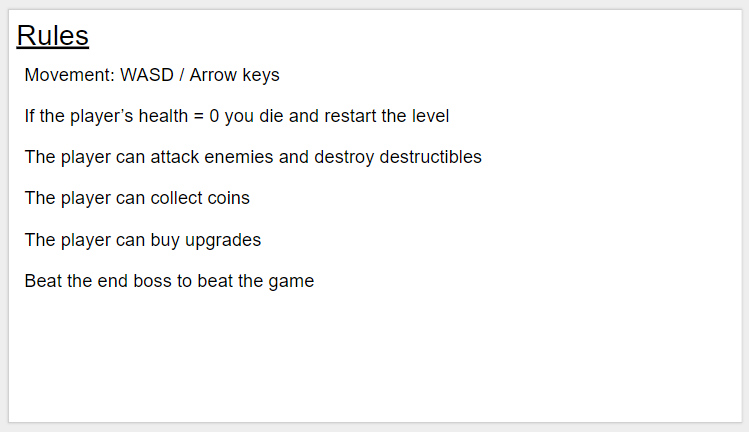
**Appendix 1**

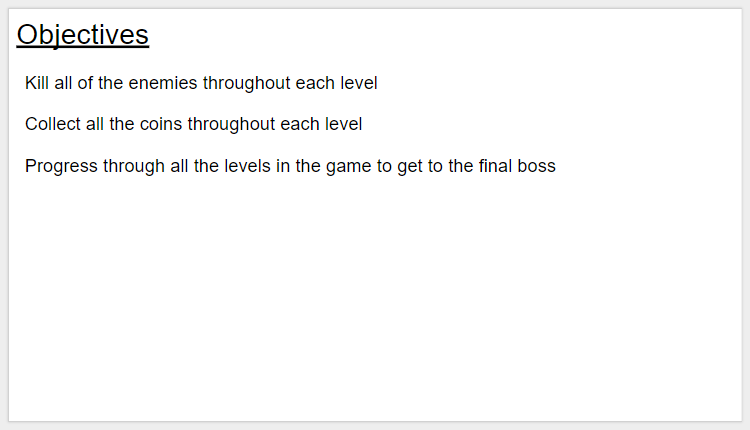
Game Concept Presentation

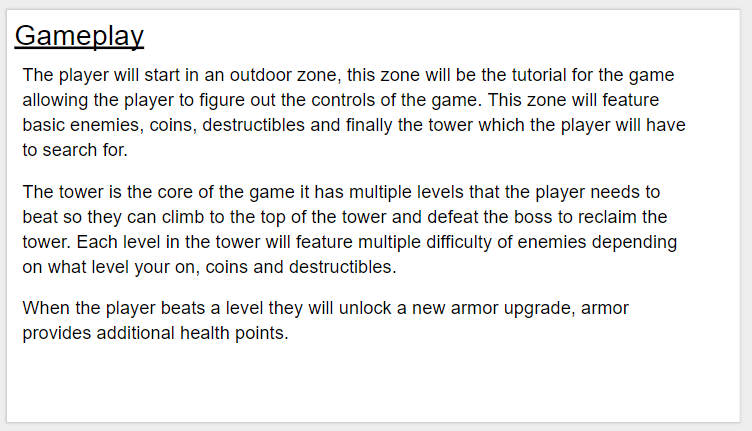
****

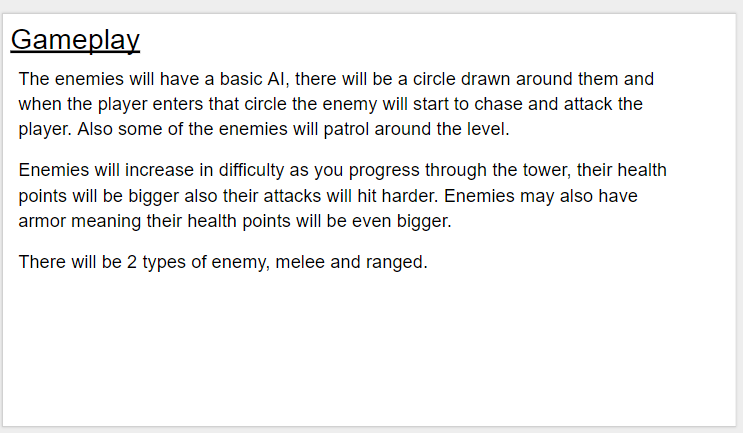
****

****

****

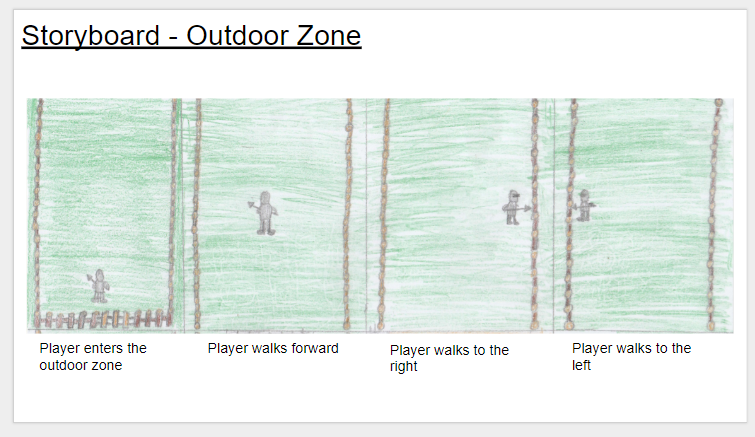
****

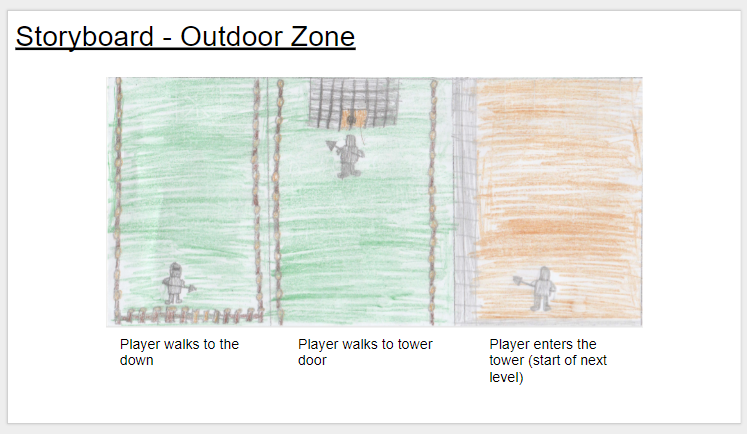
****

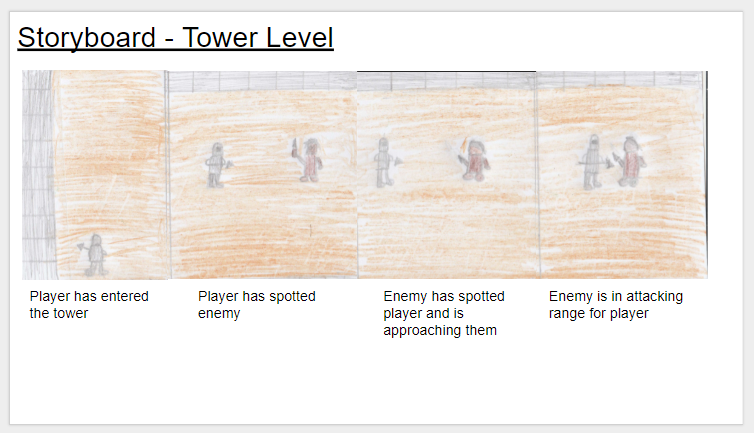
****

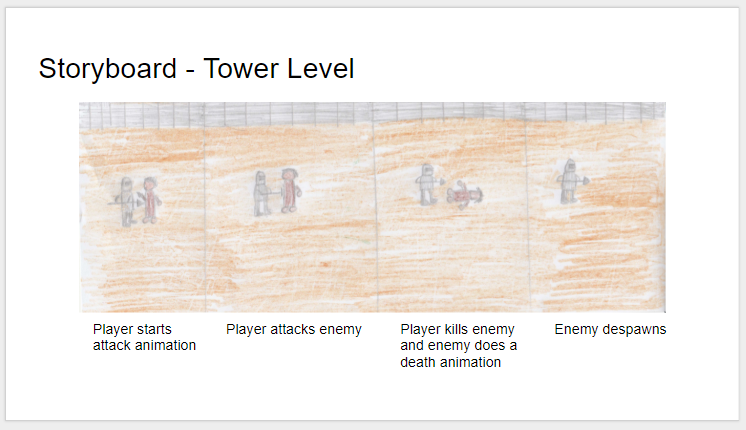
****

****

****

****

****

****

**Appendix 2**

ALL ISSUES

* Trying to get the tile map working using JSON files then swapped to CSV files then back to JSON files because CSV files doesn’t work with layers, spend ages working out which names and IDs go where for the tile map layer names.
* Wall collision never worked, tried CSV files, then JSON files. Tried all the setCollison functions, using debug mode showed the collision box. Also tried making the player 32x32 as well and given it a collision box still didn’t work. Finally got wall collision working, basically the player was using player.x and not player.body.velocity.x for movement meaning the player wasn’t using physics and physics was needed for collision dectection.
* Coins using groups, having to work out which id to use. Tileset set id which should be 0, then in the collision layer there all different ID’s 2,3,4,5,6 then finally i found a forum that said you need to use the GUI instead.
* Enemies are unable to move while doing the animation (using a tween has caused this to work)
* Enemy update: enemies change animation but continue to patrol when doing the firing animation, used tween.stop() to fix this
* Enemy update: enemies stop patrolling and start doing firing animation however this only works for one enemy, not in groups. To fix this the enemy needed separate groups.
* Arrow shooting: When enemy shoots arrow, it goes diagonally and shoots multiple arrows at once. Fixed by using a timer on each arrow shot and the changed the angle of the arrow to face the player x,y position. Had to change the arrow sprite itself originally it was drawn facing upwards this had to be editing for it to be fasting right, this fixed all the issues with the angle when it was being fired.