LIFE OF DRAGON

Prepared by

Matthew Bogdanov (270041214)

Nae (270028568)

Manny Kwong (270036328)

CS106 Assessment 2 - Final Project Report

2020 - 2021

Submitted to

YOOBEE COLLEGES

ABSTRACT

In this report you will read about our objectives such as: creating a working game, passing 106, and to challenge ourselves in programming for Android. We decided to split the project into three separate parts which later would be working and communicating with each other. This report outlines what we have learned in past months, how we dealt with problems, how we used maths and a lot of other new code that we never used. This report also summarises what we accomplished each week and what goals or tasks we finished. Finally, this report reflects on planning and struggles during this project, providing insight on our progress and individual experiences, our inspirations and our conclusions.

Table of Contents

1 Weekly Summary

Week 1

Time was spent finishing off Assignment 1 for 105 (Individual Game Project) and the Game Design Document (GDD) for 104 so the implementation of the features was delayed for 1 week. However we did end up reusing much of the code (for the main game loop and game over screen) from Assignment 1 and the GDD was very useful for the planning for this project so the week was not a total waste.

Week 2

We implemented:

- parallax for the ground to give the illusion of moving the dragon horizontally
- basic building spawning and drawing.
- basic NPC drawing and movement towards a target.

Getting something on the screen as soon as possible was important. We also refined the GDD into the Game Proposal for 106.

Mid-Semester Break

The animation for the dragon's movement (flying and walking) and its fire breath was implemented.

Week 3

We implemented:

- the extension of the basic NPC to include idle movement, walking animation (bobbing up and down) and fleeing (bobbing up and down faster) to and from a target position.
- the extension of the basic building to fortress, farm and house.
- the integrated projectile physics from a previous 105 lab into the project for arrows.
- gold collection for the dragon and gold physics (exploding and bouncing).

We also did the game asset planning and references for visuals for the game.

Week 4

We merged all the separate features from the previous weeks (NPC, buildings and dragon) together so they can interact with each other.

Such as:

- combining the projectile physics with NPC to create the archer NPC (though they still look like sheep).
- combining projectile physics and buildings to create the turrets
- collision between the fire breath with NPC and buildings with the latter being damaged (losing health points) and destroyed.
- farmers moving from a house to a farm to work (one way).

We also implemented an object pool for spawning NPC and returning to the pool upon the NPCs death.
Completed the Prototype Report for 106.

Week 5

We implemented:

- the heads up display (HUD) (health and mana bar, displaying the number of gold coins collected)
- the upgrade menu
- the pause menu
- A basic start menu
- day night cycle
- farmers moving to farms at morning and then moving back home at sunset
- archers were changed into dragon slayers (very aggressive, hard hitting and hard to kill) and wizards (slow but never miss)
- the fortress generating gold over time
- buildings spawning npc based on certain conditions such as money and fear.,

We also had to do some collision debugging and optimization.

Week 6

We implemented:

- the lair where the dragon can sleep, deposit gold and open the upgrade menu to grow stronger.
- the extension of NPC to thieves (going to the lair, stealing gold, returning home and depositing the gold at the fortress)
- the fortress taxing individual buildings and using the gold taken to spawn NPC and more buildings.
- the fortress surrendering when the fear reaches maximum and offering daily gold to the dragon in exchange for mercy
- the fortress repairing destroyed buildings,

We also did the first draft of sprites for the game assets in the form of sprite sheets.

Week 7

We refined the sprites and combined art styles so the game assets fit together visually on the same screen.

We implemented:

- the theme music that plays on the start screen and under certain conditions during the game
- the game over music that plays upon the dragon dying
- extra animations such as the NPC turning into charred corpses
- the dragon being able to eat the corpses of NPC to regain health and mana
- UI improvements from play testing (buttons, hud, menus)
- More drawing and physics optimizations (pooling, reducing sprite sizes, reducing colliders etc)

We also did extensive playtesting to balance the difficulty and for debugging.

Week 8

We implemented:

- start menu animations
- more UI improvements (buttons, HUD, reducing input lag)
- sound effects

We also did extensive debugging and the final presentation.

2 Individual Report

2.1 Matthew

Planning

At the beginning we started planning how our project development would go and after some thinking and brainstorming we made a game development plan that we were supposed to complete in specific dates.
As for my part in a nutshell my general plan was like this: Create general class NPC => Make sub classes of other NPCs from the main class => make quality updates for these classes, such as sounds and visuals.

Start

Unfortunately we started late by one week, so me and the others had to keep up with the plan.
At the beginning it was hard for me to understand what exactly I was supposed to create and how everything should work and interact with other classes but the more we made the more understanding I had how it was supposed to work eventually.

Merging

When things came to merging I was trying to finish my code to completion as much as i could for that time to not have any major issues with merging and interacting with code of the others.

Finishing

Finishing was honestly a little bit stressful as I had to perfect everything for the demo and game demonstration and presentation, so me and the others had to fix as many bugs as we could and stop adding new features as otherwise there would be more bugs. As well as I had to record a perfect demo for our presentation that would show all aspects of our game.

But overall I must say that it turned out pretty well. Me and my team were working hard on our project and despite some arguments we had it went very well and my hard work wasn’t just for nothing.

2.2 Nae

As the team was too occupied with coding and other activities around the game project the team did not find any time to spare for writing weekly reports, so all information this report covers is after the team has finished the whole product. As a result, when describing what happened during the process, the author has to write it up from memory.

Part 1: the planning

After the Team had assembled, the first thing the team had to do was come up with a plan: how should the game look like, what should it feel like?

Which features should they include, and how many of these features COULD they include in the time they had?

After they had sorted out all the features that they wanted to add, they started on working out a Trello Board. Creating a system off categories, those being:

> “**System”**: everything which had to do with the program itself, eg Game loop, Menus, Frame Rates etc. (Matthew, Manny)
>
> “**Environment”**: everything which had to do with the world: background, undestroyable objects in the world, the dragon's lair, the world itself. (Nae, Manny)
>
> “**Dragon”**: everything that had to do with the dragon, firebreath, damage calculation, movement, camera, passive ability upgrades, sleep. (Manny)
>
> “**NPC”**: all entities the dragon could kill, or be killed by, and which did move around in the game world. Ranging from simple harmless sheeps, to dragonslayers, thieves and wizards, harming the dragon if approached too close. (Matthew)
>
> “**Buildings”**: all buildings and immovable but destroyable objects in the world, they where the core-object for the dragon to destroy, as they usually gave the greatest amount of money. Buildings include the Fortress, turrets, farms and houses, also the fear-system and Gold drop. (Nae)

"**Visuals and Audio"**: lastly, everything which had to do with the visuals of the game and how it sounded like. That being all the sprites for all the objects, the sounds they made and start/main and death music. (everyone)

After they made the category they again split these categories up in their smaller elements and then divided them evenly between each member, each member having one core category.

Then after that was done, the Team also started to decide when they wanted to do what. So they started to copy all the split-up categories and added them to a timeline. The most important features and objectives first, the more 'pleasing' but unnecessary ones later.

Part 2: the start

As they had now finished organising themselves, they started to code, at first each for themselves, working out their prototypes.

Building section worked on creating the core class off the buildings the "foundation", every other building class then should be extended from said foundation.

The first class extended being the Fortress, the main core off every Town in the game, the Fortress should keep track of all the buildings in the town, calculating their fear, their health, repairing them when necessary, attacking the dragon when comms up close and gathering the gold from the buildings around.

The second building extended from the foundation was the plain "house" class, for the moment it was just made to see how the fortress could handle building placement.

The third building was the farm, the farm being wider than the house, again. For the moment it was just made how the fortress could handle the building placement when different sized buildings were involved.

The issues involved during the Building Programming Phase one:

Extending a class from another one is more complicated than thought at first.

So was placing the buildings around the Fortress. As the buildings were kept in place and

checked only by four arrays, left and right, each one having arrays for the objects and the second being for the positions of the buildings.

He also tinkered with the idea of splitting the farm up into sections, the sections then being individually destroyable and randomly chosen, on creation, but also always limited to only 3 sections. At this time of development however it proved itself to be too difficult, also, taking into account that the farm had to be aware off it's own position.

*disclaimer, the author did not work on these sections so he can only write from what he did see, in which order.

*NPC section started with creating the first NPCs that being the sheep. Sheeps roamed around the area, and. Well they walked. Nothing more for now, later he made them into archers. Shooting at the dragon when in seight.

*Dragon started to create the dragon and the GameView and Game class. Being the Core Parts of the entire game, most other classes referred to the GameView for Y-positions, Spawn-pools, time/update functions and others.

The dragon at that time could already fly, breath fire and walk.

Part 3: the Merge

After all of them had managed to finalise their prototypes they had to merge them together into one project. That proved itself to be simple at first, but interaction with each of these individual parts proved itself a bit more difficult.

The buildings department started to add another building-type, twice the height of most other buildings, it was destined to become the turret, the only other fortification in the town to keep the dragon at bay. To make it shoot, one simply copy the code of the archer-NPC and paste it into the Turret class.

The idea for using arrays to keep track of the buildings and positions was thrown out the window and instead Arraylists where used, simply for the reason that it is much simpler to delete and add

items to said list, as they later decided that some of the builds, once destroyed, should be rebuild as something else. Using an array for them proved to be very complicated.

As the Dragon department also started to create collectible coins, the Buildings department started to create a “gold rate” and “current gold” variable which kept track of the gold in the town and used said gold to buy more buildings. The “current gold” being updated by a fixed amount, calculated from the “gold rate”, which again was calculated from the individual buildings currently being alive in the town. The buildings again calculated their gold rate by how many inhabitants they had + a fixed value. If the dragon so destroyed any building the buildings dropped a fixed + random amount of coins for the dragon to collect.

Issues the Buildings department run into during Phase two:

Merging a building with parts of the NPC-archer code proved itself to be much more complicated as one first has to understand what they are using and why they are using it. These being the creation-points the Action-controllers and divers other functions including the attack range and lockon.

After having figured that out, they wondered why the turret did not shoot, something was wrong with the action-controller, and sometimes it shoot to many arrows all at once like a shotgun, other times it shoot nothing, since they at first used a random-function for it to decide when to shoot. Later they opted for a simpler version using a if else block and a simple int counting up from 0 to 3, having the turret shoot 3 arrows after each other and then going into cooldown, giving the player time to attack and dodge the arrows. Also did they give all arrows a slight randomness so that the arrows were never exactly spot on.

By now the idea of splitting the farm up into different sections also had been thrown out the window as it did not give much to the game and as it only keeps spawning more and more problems, instead it was decided to make the farm one single object as all other buildings.

As the dragon now had an interactable breathattack, they found out that all buildings shared the same healthpool, that being, one could destroy one building, but all other buildings also lost health at the same time, causing the other buildings to become invincible.

*NPC: at that time the NPC-department started to develop stronger NPC-units, those being the dragonslayer, wizard and thief. The dragonslayer following the player until they died, also did they deal more damage then the archers. The wizards being slow but powerful units who shot projectiles that followed the player for a set amount of time until they disappeared, also did these projectiles deal a huge amount of damage. Lastly the thieves, which did not attack you in a sense as they harmed your HP but rather halted your lv-up progress by stealing gold from your lair.

*Dragon: the Dragon-department implemented damaging fire, collectable coins and simpletons in the code, making them easier to access for other classes that depended on them or needed them. Also did they finetune the mana-household the dragon could use.

Part 4: on the finishing line

As they did see the light at the end of the tunnel and the year soon came to an end, they started to give it their all once more. Building-department added the fear-mechanic, allowing towns to surrender and giving the dragon a daily income Also we finished up the level up-conditions for the fortress. And added a “fear-flag” a visual indicator for the player to see how much fear the town had gathered. Then they also added a visual indicator for the player to see if they damaged a building.

Work on the lair also finished up and the dragon now had a place to settle down, drop his coins and sleep. Once sleeping he gained XP according to how much money he had gathered. Once leveled-up he then could invest points in his abilities to help him take down the final town.

After most of the code had been finished most of the developers focused on drawing sprites for the game, again, each working on the sprites for their department.

<u>But first: the issues the Building-department run into during phase three:</u>

Fear calculation had to be adjusted so that it did not run out the moment it was generated, making it possible for the dragon to get a town into surrender mode. Once a town was in

surrender mode they did send out a villager, giving the dragon tribute, that tribute being a significant part of the towns income, also halting the towns development, keeping it on a smaller level.

Added a feature which morphed a destroyed house into a turret if fear was above a specified level, and vice versa (but town always kept at least 2 or 4 turrets, depending on the town level). This feature was at first a bit difficult to figure out, but this is where the arraylist came in handy.

Added a repair function so that destroyed buildings would repair over time if the fortress was still standing. At first the building-department tinkered with the idea of assigning a “repair-team” that was as big as the current amount of inhabitants and then became smaller if a building was to be repaired, and no buildings should have been repaired if this building-team was 0. This proved itself to be very complicated indeed, so that idea was thrown out the window. Instead they opted to give the fortress one building to repair on each side (left and right) and if one building was allread being repaired all other broken buildings had to wait until that building was finished with repairing. The speed at which it was repaired, depended on the amount of current town-inhabitants.

*The NPC-department did art by then as they had finished up most of their parts. After they had added the thief's gold-stealing ability.

Once NPC-department (and during) had finished up the Sprites, they also started on working with the UI-Menu, start, stop, music and volume changers. Credits-section, sound-effects, sound implementation, and other behind the scene objects. While also finding many of bugs in the meanwhile.

*Dragon-department, started to finish up the lair, and the level up function for the dragon.
Also was a gold pile added which stores all the gold the dragon had collected during the process. Once that was done, the dragon-department joined forces with the NPC-department and together they worked on UI-menus, sound-effects and other behind the scene objects. The rest of the time they invested in bug-fixing

The Sprites

Now onto the final items:

As the building-department also finished up the coding they opted to start on the Sprites for all the building and environment objects:

As they went through three different iterations until they found a somewhat cohesive style for the game, this report will spare you the first two iterations and only provide you with the final images used for the game.

The farms where made with breakability in mind, also as farms need agricultural space they are wide. The breakability is reflected by the small size of the buildings.

Houses are just that, houses. They are there for people to live inside. Again, they can be broken down quite easy so they are made out of wood and just a few rocks.

Turrets on the other hand are tall and sturdy, they need to be tall since after all, an arrow flies a longer distance when shot from a higher point. And they are made completely out of rock to reflect the sturdiness.

Figure 1. Farm sprites

Figure 2. House sprites

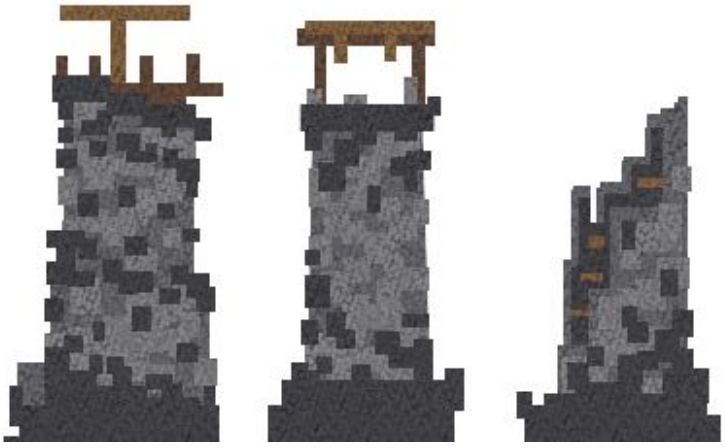

Figure 3. Turret Sprites

Lastly the Fortress, which combines the sturdiness of the Turrets, with the comfyness of the houses. It also symbolises the whole town which is why it has multiple smaller houses embedded into the walls.

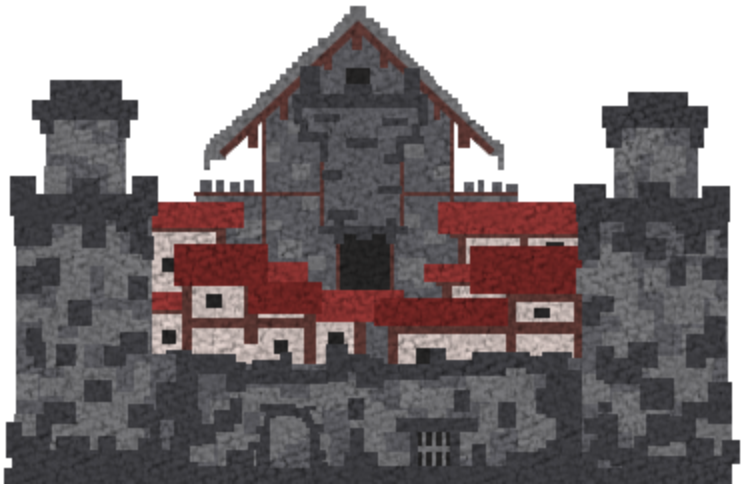

Figure 4. Fortress Level 3 Sprite

In the end I also was asked to do the lair, the main “hub” if one want to say so, for the dragon. As it is a cave, I again opted for the stone texture but also adding dirt and grass on the top to give it a bit of a natural feeling.

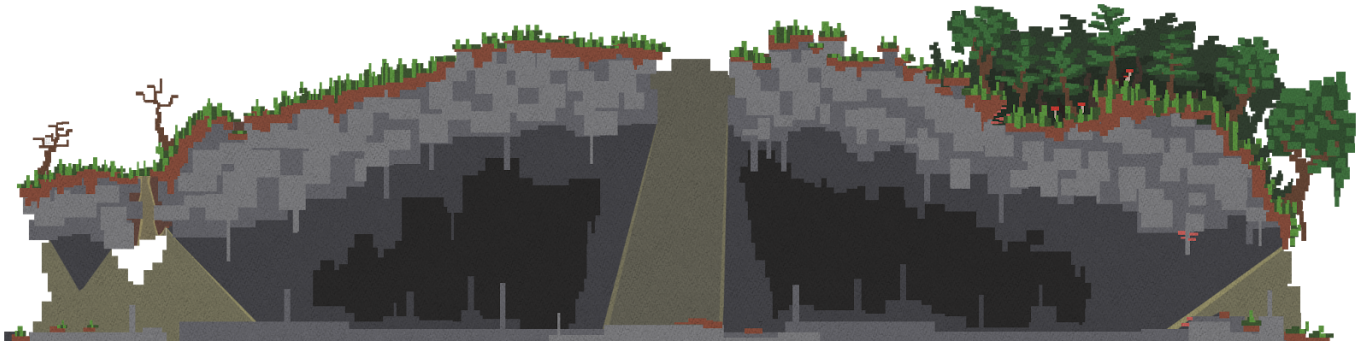

Figure 5. Lair Sprite

And because our world was pretty much empty in between the Towns i was also asked to draw something for said emptiness. So I drew some trees and stones.

Figure 6. Tree, mushrooms and stone sprites

These are almost all the sprites that have been made during the process, the only sprites not listed here are the fortress lv 1 and 2, the backgrounds for the forest and town, the foreground of the lair, the ruins of the farm, house and fortress, some UI-elements and the flag and flagpole.

2.3 Manny

Dragon

The fluid snake-like movement for the dragon's body was done by using a chain of bitmaps/segments. The first segment follows the head and every other segment follows the segment before while keeping a distance of ¼ of the width of the segment. Each bitmap was rotated to face the segment before it and scaled to match the body line of the dragon giving the illusion of a neck, main body and tail. A waving animation was added to give more life to the dragon by displacing the draw Rect of a segment by a cosine function perpendicular to the direction with a phase proportional to how far along the body the segment is. The frequency of the wave is proportional to the movement speed of the dragon.

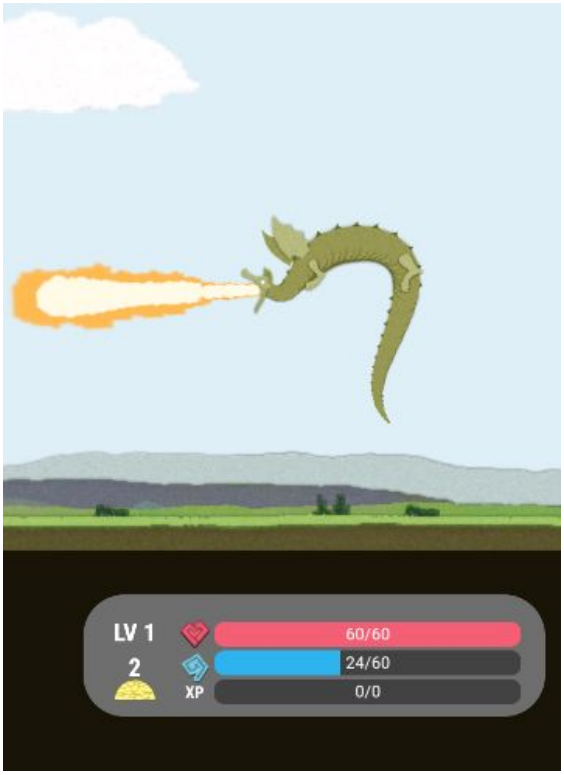

Figure 7. Dragon breathing fire

The flapping of dragon's wings was done by scaling in the y direction from 1 to -1 while rotating the wing's sprite along with the segment it is attached to.

The animation of the dragon's limbs was done using cosine and sine functions to move the arms and legs in arcs proportional to the movement speed to create the illusion of taking steps while walking.

The head and jaw open and shut by rotating their respective bitmaps and displaced by a small random value to make it shake when breathing fire to make it look powerful.

For performance each segment consisted of 2 alternating 64x64 bitmaps, with the total sprite sheet being 256x384. Each bitmap was prescaled, with rotations and positions calculated outside the drawing function.

Figure 8. Dragon sprite sheet

I initially had trouble applying the matrix calculations while using a sprite sheet as it would rotate the whole sprite sheet without cropping the selected sprite. The solution was to pre-crop the sprite into individual bitmaps during initialization for later use.

To make the dragon grow on level up I tried inserting more segments and resizing the segments, limbs and head of the existing dragon. However this proved too difficult to get right and the final solution was to scrap the existing dragon and reconstruct the dragon from scratch with more segments and larger sprites.

The body was made up of up 65 segments (for the fully grown dragon) with ⅛ of the segments being used for collision. These were spaced evenly along the front ⅔ of the body, excluding the end of the tail. To make the transitions between the segments smooth while the dragon moves the turning of the dragon per frame was limited by having the current direction being the sum of one-tenth of the target direction (from the controls) and the previous direction.

Though such animation was not essential to the game I think it is a nice feature and I found moving the dragon around to be satisfying to look at. Overall developing the dragon was challenging and fun and I am pleased with the result.

Parallax and Scenery

In order to create the illusion of horizontal movement for the dragon while keeping it on the screen the world has to move around the dragon. However, this made some of the physics difficult as the frame of reference was constantly moving. The solution was to keep all positions, speed, directions and colliders in world coordinates while drawing the non-dragon sprites displaced from their positions by the horizontal position of the dragon's head.

The ground, hills and mountain backgrounds each used a single screen spanning sprite.The sprites were displaced in proportion to the players speed with the hills and mountains in the back moving slower to give the illusion of perspective. As the dragon could move left and right, each sprite was drawn 3 times, (left, center and right) to prevent any gaps. As one of the sprites moves completely off screen it moves to the other end. Performance issues were fixed by minimizing the number of transparent and translucent pixels in the sprite.

The sky is a canvas.drawRect with a color filter applied so it brightens during the day and darkens during the night. There was originally a sprite for the sky but this was scrapped due to performance issues. This was probably because it was a moving bitmap that covered over half the screen with many other bitmaps drawn in front of it as well as having a color filter applied to it.

The level wraps around such that if the dragon travels the length of the level in either direction it is teleported to the other side. Each part of the dragon had to be individually displaced by the island length when the head passes the designated edge and no other objects was to be placed within one screen width of the edge to give a smooth transition,

The horizontal positions of the trees were randomly placed in 3 zones, right of the lair, between the kingdoms to the left and between the kingdom and the edge on the right. Each position was randomly given one of the 8 tree sprites and would be called to be drawn only when the dragon is in one of these zones for performance. The mushrooms and rocks were handled in a similar fashion. Each of these sprites were pre-downscaled to half its original resolution then resized in on game initialization.

Projectiles and gold

Figure 9. Gold from a surrendered kingdom and the dragon pierced with projectiles

Object pooling was used with timed removal of objects to fix performance issues. An ArrayList was used for active objects and another for inactive objects. For each frame the game loops through the array of active objects to apply physics (eg. translations, collisions, gravity), drawing and logic.

The gold is given the illusion of spinning by fluctuating the width (by changing the Rect used for drawing) and lighting (using LightingColorFilter) with a cosine function. For each collision between the ground and the gold coin it would multiply the vertical speed by -0.3 stop them from bouncing forever. Additionally the horizontal speed was multiplied by a friction coefficient to prevent the coin from moving forever in the horizontal direction.

The projectiles turn towards the direction of movement for a smooth arc by rotating the bitmap about a point close to the leading end of the sprite by using matrices. The projectiles stick to the dragon by storing the relative position and rotations on impact, setting the dragon collider hit as a parent and rotating/translating the projectile relative to the collider using matrices. There was a bug where the game crashes when the dragon grows while having projectiles still stuck to it. This was because the dragon was reconstructed with larger segments and the projectile had lost the segment it was attached to. To avoid this, attached projectiles were removed from the world quickly and not drawn when the dragon is sleeping.

Fire

The fire breath was implemented similar to projectiles except they are shot rapidly in succession for a stream of flames, have no gravity and are instantly removed after travelling the range of the fire attack. The size of the flame projectile is proportional to the distance travelled to create a cone of fire. The flickering flame animation is done by randomly switching between 6 flame 32x32 bitmaps. There are up to 40 flames in the pool but only one of the active flames is used for collision for performance. When the collider flame has travelled the attack range and deactivated the next flame shot becomes the collider.

Controls

The dragon needed to be able to move and breath fire at the same time so multitouch was needed. Initially I attempted to use the pointer index from the MotionEvent class to retrieve multiple motion event actions but could not get it to work properly. I ended up looping through all the pointers, finding their positions and checking if the actions occurred within the fire button's radius or the movement joystick radius to distinguish the actions as well as using the deprecated ACTION_POINTER_2_UP to find if either movement or fire breathing needed to be stopped. This is not ideal but it works so well enough.

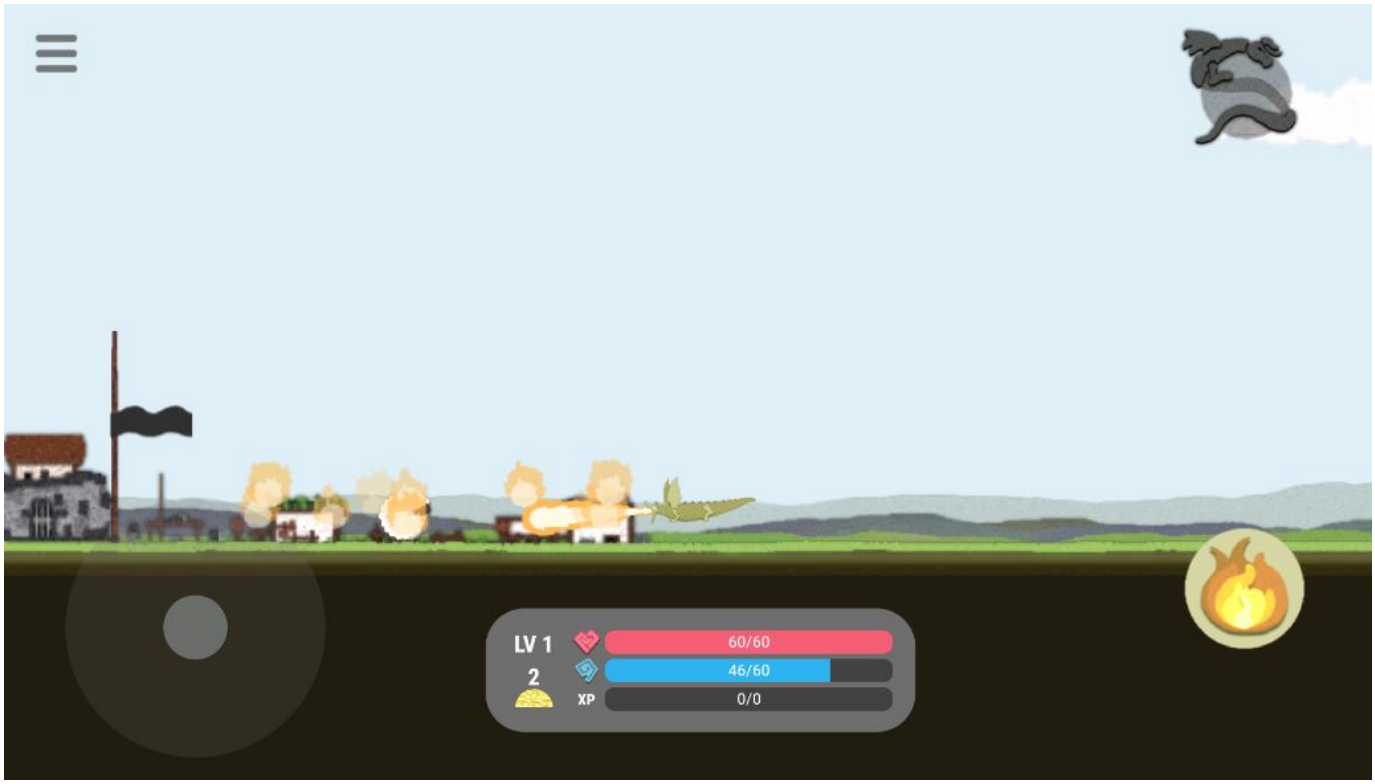

Figure 10. Screenshot showing controls and fire

Lair

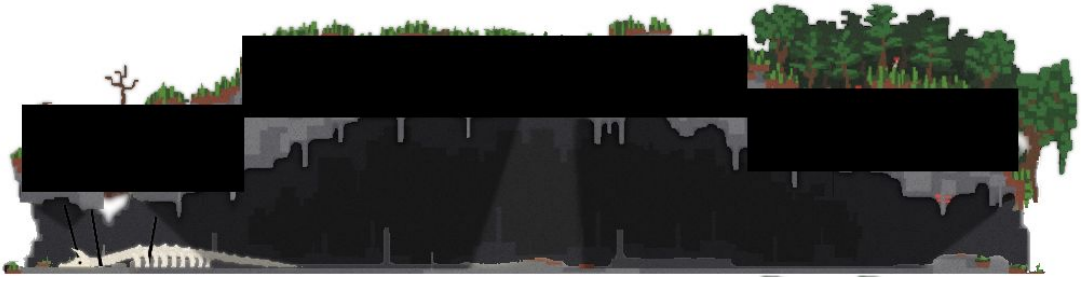

Figure 10. The lair and its colliders

The lair needed to be entered only through the left and right entrances and fade into a cross section when the dragon was inside. 3 Rects were used for the top of the lair with the position of the dragon set to 0 on collision. The speed was set to 0 if the dragon collided with the roof of the lair and -1 with the ceiling to make it appear that it was hitting its head with the stalactites above.

Collision with the gold pile was done by setting the maximum y position of the dragon (ground level) to be the perimeter of a circle collider (y = sqrt(dR*dR - dx*dx), R = radius) that is centered on the sprite such that it matches the curvature of the gold pile. The gold pile was raised or lowered depending on the amount of gold with the height being in proportion to the inverse cubed of the amount of gold to match the volume of a hemi-sphere and to apply a soft limit to the height of the gold pile so the dragon could still pass through the lair.

To speed up the time while sleeping we divided the frame period (fixedDeltaTime) by the time scale (1 for normal speed, x8 while sped up) in the code responsible for frame skipping and thread sleeping. Similarly for the slow motion during the game over animation, with the time scale being x0.2.

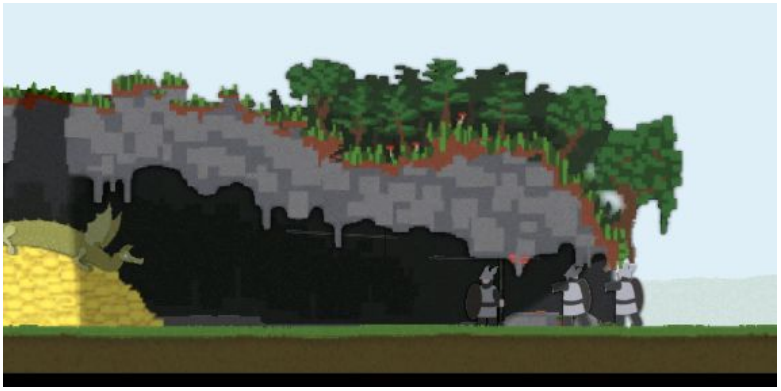

Figure 11. The dragon on top of its gold with 3 invading dragon slayers

Sound

For the majority of the sounds we used the SoundPool class. We had problems with sound effects not stopping when they needed too (such as when paused, on game over or when far away from the source). To solve the pause issue we found the SoundPool.autoPause() method. For the game over we released all sounds and then scrapped the current SoundEffects class and replaced it with a new one when the game restarted. For away sounds we stored the id of the sound effect and then set the volume inversely proportional to the distance between the source and the dragon and then switching it off when it reaches a certain distance.

Performance problems arose when there were too many sounds playing at the same time because of the many gold coins. To solve this I used an array of MediaPlayers instead which would only start playing if it had finished playing the previous sound with a maximum of 4 coin sounds playing at the same time.

For the music we originally created and started the media player when needed during the game but would run into problems when too many were instantiated and not released which clogged up the garbage collection. Releasing them also caused problems with the media players not being prepared to play or not being in the state to start or stop. In the end a single media player was used for the theme and game over music that was instantiated on start up. Care was made that only one of these was playing at any time and the music was stopped by pausing the media player and then reset to the beginning with the seekTo method.

Other Bugs and Challenges

Around the time of implementing sound there was a bug where the game ran twice as fast for the first playthrough. We then found that pausing and resuming the game on starting a game solved this problem after Matthew noticed that it became normal after restarting. We still don’t know why this occurs or why the solution works.

We found that on different devices the game will run at different speeds. This was due to different screen sizes and framerates (we wanted the frame rate to be the refresh rate of the device). To solve this we had to use world coordinates with the unit of distance not in pixels but in screen widths. We also used the frame period (fixedDeltaTime) for speed calculations.

At high frame rates we found that slow movement was not being shown. This was because we initially used int for the positions and speed of the objects and the distance moved per frame at high frame rates (and hence low frame period) was below 1 so were truncated to 0. This was solved by switching to using float for positions and speed.

Then when objects were moving towards targets we would get them oscillating about the target position. This was because the object would overshoot the target and then go back in the other direction as the distance from the target would never be exactly 0. This was solved by having the object stop if it was within a close enough distance from the target (typically within 10 pixels).

3 Conclusion

3.1 Conclusion

Despite an initial delay, slow start and deviations from the original schedule we managed to finish the project with 99% of the features present in the proposal, with the scrapped features being the archer (which was merged with the dragon slayer) and the cow (which we deemed necessary) and met all functional requirements. We also extended and added some extra features such as physics and animations that we hope made the game more enjoyable.

Implementation was very challenging with every feature causing bugs and compatibility issues with each other's code. Each member's part was dependent on each other and merging the parts was difficult. In implementing these features we made full use of the knowledge and skills gained in class as well as from self study.

The team worked well together despite Manny and Nae (to put it in a nice way) being quite blunt and assertive which sometimes clashed with Matthew's more gentlemanly demeanor. In the end everyone worked hard and pulled through in their respective roles.

3.2 Future Work

Though we implemented the features in the proposal we still feel that it is still lacking as a game, though it might serve as an okay demo. The visuals and sound can definitely be improved. Further playtesting with people outside of the team is needed for balancing and polishing the gameplay.

The features we feel will make this game more complete include:

- Story segments in the form of flashbacks to when the dragon's mother was alive and the event that caused her death. These can include gameplay hints/tutorials in the form of memories of the mother teaching the dragon about the world and how to survive.
- More abilities such as a melee attack (so the dragon can attack NPC without mana) and a dragon roar that can stun enemies.
- Goals outside of defeating the final kingdom (responsible for your mothers death) and amassing gold. These can include attracting a mate and producing eggs (that can function as extra lives on death), titles for achieving conditions like conquering all of kingdoms at the same time or surviving a certain number of days.
- More buildings and NPC such as mines, churches, priests, travelling merchants and animals.
- Going back and forth between the lair and gold sources can be time consuming and repetitive so perhaps adding more lairs or caves that are connected via tunnels that allow fast travel will solve this.
- Adding more islands so that the dragon can conquer another set of kingdoms while still being able to go back to previous ones.