Computer Science Coursework

Ben Elliot

Autumn 2020

Contents

Part I

Analysis

Chapter 1

Problem identification

1.1 Outline of the problem

I am going to create a game where a user plays against an AI where both have miners which mine gold which is used to train an army of soldiers to attack and destroy the oppositions castle. Towers will be able to be built in defence. The soldiers will be able to be upgraded in the castle and the towers built will also be able to be upgraded to be better. The AI should also get more difficult as you progress through the levels with different troop types being added in future levels. It is solvable by computational methods as there are several libraries that already exist to make an AI like this, and the mouse can be used as the cursor easily to move troops and build towers.

1.2 Stakeholders

The demographic for this game would mostly be people of all ages who like real-time strategy games, however the biggest age group will most likely be teenagers. This is because teenagers tend to play video games the most and are definitely the largest portion of the population when it comes to players of real time strategy type games. The game will be played on a PC using a mouse as a point and click type game making it easier to play. It should be a game where you can quickly do one level in some free time and then save the progress to come back to it again later. The game will start off very simple as to not overwhelm the user and gradually will start to add in more and more features so that the user has a chance to get used to everything.

1.3 Scope

I am a single student working on this project rather than with a whole team therefore:

- I will use simple 2D graphics rather than orthographic isometric graphics,
- There will only be a limited number of levels,

- I will use stock audio rather than custom audio,
- I will use stock graphics rather than making my own.

1.4 Research on similar games

1.4.1 Game 1: Medieval Empires RTS Strategy

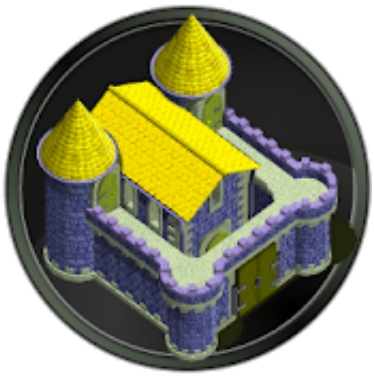
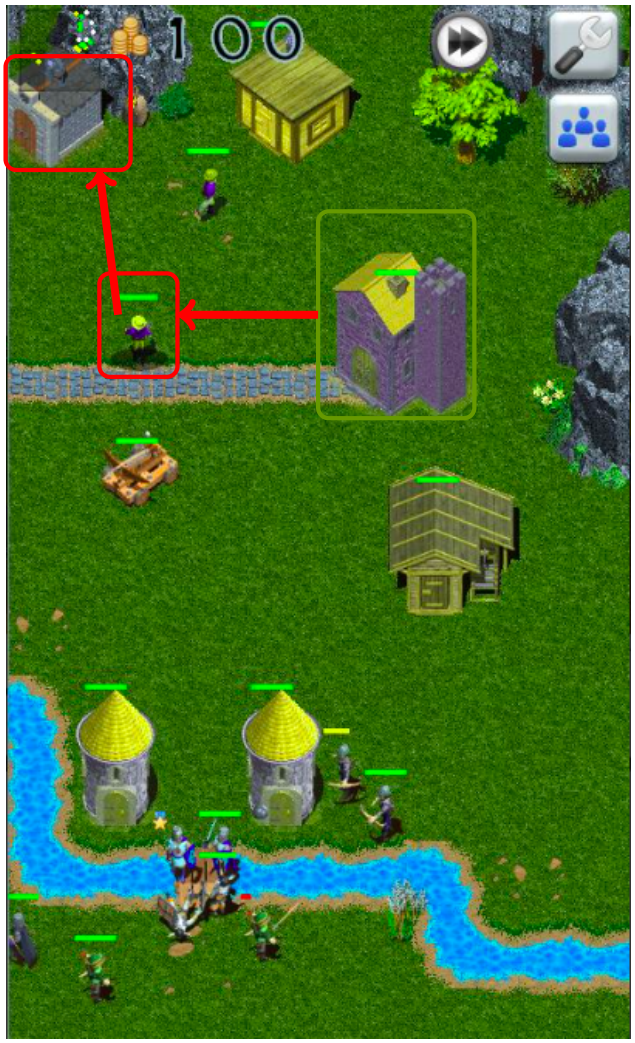


Figure 1.1: Logo of Medieval Empires



From this screenshot of the game you can see several different key features which I would like to include in my game. The first of these is a mine and miner system where you can send miners to collect gold and then return it to the castle. This is a feature I would love to include in my game as it gives a way for the player to slowly gain currency which they can do other things with. Another feature of this game which I like is ability to select multiple units of the same type and move them or set them to do the same task all at once - this I feel would be a greatly useful feature and make it much easier for the user. This game makes it so you can just select troops then tap to send them to a certain location while avoiding obstacles. To do this I will need to use some sort of pathfinding algorithm such as A* to make sure the troops do this properly and efficiently.

Figure 1.2: Screenshot from Medieval Empires

From this screenshot you can see that towers can be built and also upgraded, this gives another level of complexity and another option to the game which I would like to include. You can also see where the troops are trained in another building which can be built.

As this is only a small project I will probably not use an orthographic perspective and instead use a more simple 2D style perspective, this may decrease the user experience in some areas, however it needs to be done.

Figure 1.3: Screenshot from Medieval Empires

From this picture you can see an opposition AI with a different colour which the user is playing against. Making the AI for this or using other's libraries will be very difficult to manage and execute, so this will be one of the hardest parts of making my game.

1.4.2 Game 2: Kingdom rush

This is a tower defence type game which is useful to look at as a slightly different style of game that still features many of the same mechanics that I would like in my game, just implemented in a different way.

Figure 1.4: Screenshot from Kingdom Rush

From this figure (1.4) you can see that towers are built in order to defend something. This is similar to the way I would like to use towers, so that they can also be upgraded as well. In this game the currency is earn't by killing some troops of the opposition. This is an alternative way of gradually giving currency to the player, however could lead to stagnation and a lack of money in some stages of a more open world type game.

In this game there is also only one way the AI can travel so the troops only need to take one path. This greatly simplifies both the AI of the opponent and also the pathfinding algorithm of the best path for troops to take.

Figure 1.5: Screenshot from Kingdom Rush

You can also see that all the characters have a health bar which decreases when attacked - this is a function that I would like to introduce.

1.4.3 Game 3: Clash of Clans

This is a Real time strategy game where you build a base then attack other players bases in order to gain trophies and three different forms of currency.

Figure 1.6: Screenshot from Clash of Clans

In this screenshot you can see how a base is built up with walls and many other forms of defences. Nearly all things you place down in this game you can upgrade or change in some way at the expense of a kind of currency. You can also train troops to attack other people from within your base as well.

Figure 1.7: Screenshot from Clash of Clans

Clash of clans also has much more of a social element with friends and battling against other people instead of just against an AI, this can bring a much more competitive nature into the game and can push users to play the game more and enjoy the game more. However, online multiplayer is also hard to implement and carry out, so the most I would be able to do in the given time frame is to make a local multiplayer option instead of an AI.

1.5 Interview

There is a fairly wide range of stakeholders for my game as lots of different types of people enjoy strategy style games. This means that I will need to interview different types of people to see what they well enjoy in a game.

1.5.1 Interview 1: James - 17 years old

James is an avid gamer who plays games on his computer lots, including strategy games. This means that I will be able to ask him more in detail questions about specifics within the game as well as the mechanics of the game.

How much do you normally play strategy games?

Quite often, usually for around an hour a day.

On what device do you usually play these games?

It's a mix really between my phone if I am out and not at home where I like a quicker game that I can easily go onto wherever and my computer at home where I usually play bigger games.

Of the strategy games you play at home on a computer, what features do you like particularly about them?

When playing a strategy game I don't want the controls to be too complex or to have too much information displayed to me at once so that I can focus on what I'm doing more.

What do you think of using keyboard and mouse for key inputs and do you often find using them confusing?

Sometimes at the start they can be confusing, but with most good games there's a decent tutorial screen which means I can learn the controls fairly quickly and good games are designed to make sure you know the controls and give you specific challenges that mean you practice using those buttons.

Who do you think would be the main audience would be for this type of game?

Personally I think it will be mostly for people of a similar age to me of around 16 to 25 as people will need to have a computer and play video games actively.

What I learned from this interview:

1. Not to make the user interface or number of features too complex as this can take away from user experience.
2. Have a good tutorial for mouse and keyboard instructions that walks you through how to use every key and don't make the keys too complex.
3. The stakeholders for my game will most likely be teenagers or those of an age 16 to 25

1.5.2 Interview 2: Trudy - 50 years old

In this interview I hope to find out what a different age range of stakeholders thinks about these sorts of games.

How often do you normally play strategy games?

I don't play video games regularly at all, but of the ones I do play it is often logic type games.

On what device do you normally play video games?

I only ever play games on my phone as I don't have any games on my computer and it is much easier to quickly pick up games and put them down again.

What features do you like to have in a game?

I like a game that is easy to understand and play and addictive as well with some sort of competitive element to get me engaged in the game.

Are sound effects something you like in a game?

I play with the sound off most of the time, but often some sound effects are good for people.

What I learnt from this interview?

1. My stakeholders will largely not be for older people as they are much less likely to own a computer which they play games on. I could adapt a different version of my game that would also work on a phone, however I will keep it as a PC game to start with.
2. Some simple sound effects are good to include, even though a lot of people play with sound off and sound isn't particularly important in this style of game.

1.6 Computational Methods

1.6.1 Thinking Abstractly

My game will be abstracted as there will be lots of visual representation for more complex things so it is easier for the stakeholders to understand. Some examples of these are:

- Health bars: All the health of all characters and buildings in the game will be abstracted into a single health bar shown above the character.
- Miners: The miners are an easy abstraction to take care of how buildings are built and also how gold is mined.
- Simplified graphics and colours: This is an abstraction of more detailed precise graphics that would be too complicated to deal with.

1.6.2 Thinking ahead

- Controls: my game will mostly be a point and click type game using only a mouse, with some keyboard inputs as shortcuts to other functions on the screen such as select a troop type. This will lead to an easier user experience as there are less controls and buttons to try to remember.
- World background: The layout of the world in which the user and AI play on needs to be well thought out so that defences can be built in some sort of effective way, but also making sure there are no barriers stopping the player and AI from being able to meet.
- Currency collection: I need to make sure the amount of currency both the AI and the user receives is the same and that it is at the right rate so that enough defences and enough of an army can be built up without it getting too ridiculously big.

1.6.3 Thinking procedurally

The main game states are:

- Main menu
- Playing
- Paused
- Game over
- Instructions
- Game start

In the **main menu** game state there needs to be a screen where the user can select a level, picking up from where they left off last time, enter a settings menu or look at the instructions.

In the **Game start** menu, any new features specific to that level of the game need to be displayed along with stating up the AI and getting the map loaded with all the right pieces in place.

In the **Playing stage** the AI needs to be run and all aspects of the game taking care of.

In the **Paused** stage it should stop all current actions of the user and AI and show an options screen where settings can be accessed and also a resume button which caries on from what happened before the pause button was pressed.

In the **Game over** stage the winner is displayed and if the user wins the next level can also be unlocked and the unlocking of the level recorded.

1.6.4 Thinking Logically

There are several critical conditions which represent different stages of the game:

- User troops attacking AI troops
- User troops attacking AI buildings
- AI troops attacking user troops
- AI troops attacking user buildings
- The winning condition - one team's castle being destroyed

In the 'main game loop' there will be constant updates on the AI and what that is doing and also what the troops are doing and where they should be moving to.

1.6.5 Thinking concurrently

There will be lots of things happening at the same time while the game is running. For example there can be lots of different sets of troops moving around at the same time, while audio is also playing and AI is also being run in the background.

Chapter 2

Proposed Solution

2.1 Hardware and software requirements

Development requirements:

- A computer that can run unity and c# efficiently and well,
- It will need a fairly large amount of disk space so any PC with over 120GB storage should be sufficient,
- A computer with an 8GB memory should be sufficient.

End user requirements

- A PC able to run .exe game files,
- The game should be fairly low intensity on the CPU so a fairly slow PC should be able to run it - around 1.6GHz,
- Only a mouse and keyboard will be required to play this game, therefore no external devices needed,
- It should not take up a huge amount of disk space, hopefully under 1GB,
- The graphics will be fairly simplistic so there won't be a need for a computer that needs to handle complex graphics.

Software requirements

- Apple mac or macbook computer running OS X or later

Utilities and libraries

- I will be using Unity and all of their included libraries and features.

2.2 Proposed solution features

Feature	Justification
Graphics - simple 2D	From interviews and scope
Sound effects - some simple sound effects	From Interviews
Instruction menu - make sure it is clear and not too complicated	Interview and research
Setting/landscape - make sure the two players can reach each other and some strategy can be implemented	Research and Interviews
On screen information - keep it simple, not too much information at once	Interview and research

2.3 Solution of the game

The game will be a real time strategy game using simple 2D graphics with a multi level system where more levels are unlocked by beating previous levels. There is a simple instructions page. With some new levels more types of character or new features can be introduced. While playing you train troops from a central castle using gold. This gold is generated by telling miners to mine gold from a mine and it is brought back to the central castle. You build up and army and build defences also using gold - these defences include things like towers and can be repaired using the miners. The troops and tower defences can all be upgraded to be better. You will play against an AI that is also building up and army and defences and will occasionally attack you with their troops as well as defending from your attacks. The winning condition is that the castle is destroyed of one team. You can control the characters using shortcut keys to select all of a troop type and move them by pointing the mouse and clicking and the troop type will move to that location. Individual troops can also be selected and moved by clicking.

2.4 Success criteria

Requirement		User Requirement	Solves	Evidence	Justification	Success Criteria
		Main menu screen	Handles accessing of areas			
1	1.1	Buttons	Add buttons to access all other areas including the game itself and instruction pages and options.	Page 17	This is needed to make the game look nice with a good UI for the buttons and also usable so that people can actually access the game.	– Working buttons – Intuitive matching colour scheme
	1.2	Option preferences	Allows user to edit custom settings such as rebinding keys and turning sound on/off.	Page 17	To improver user experience I would want to make sure they can control certain things in game like volume and change keys to something they are more familiar with.	– Woking sound toggle
	1.3	Levels	Have multiple levels and access to later levels is unlocked after completing a certain amount of previous levels.	Page 13, 17	I want my game to have multiple levels to keep the user engaged and involved and make them feel as if they're working towards something.	– Multiple working levels – New levels opening up once old ones are complete
	1.4	Tutorial	A tutorial page to show the user how to play the game	Page 12	There needs to be a clear tutorial that the user can access to understand how to play the game.	– Working tutorial page – Tutorial that is simple and easy to understand.
		Playing state	Handle how the game plays			
2	2.1	Path Finding algorithm	Allows a user and AI to move troops and attack the other player.	Page 6	A pathfinding algorithm is needed in order to move troops otherwise the troops would hit obstacles and buildings and not be able to pass through.	– Successful efficient pathfinding algorithm that troops can follow.

Requirement		User Requirement	Solves	Evidence	Justification	Success Criteria
	2.2	AI	Have an AI to play against as the user.	Page 12	The user needs to play against someone to bring out a competitive element and keep the user engaged in the game and to give a purpose to paying the game as well.	– A working AI that can play the game – An AI that increases in intelligence for harder levels.
	2.3	Build menu	A build menu that allows you to build a castle and other towers and also upgrade everything as well.	Page 6, 12	In order for the user to actually be able to do anything they need to be made to build the main castle at the start of the game as the winning condition is dependant on this being destroyed. The towers are an extra strategic feature which can be used.	– User able to build towers at the expense of gold.
	2.4	Mine & Miners	Have miners which enable to user to slowly gain gold allowing them to build up defences and troops. More miners should be able to be trained in the castle.	Page 6, 14	We need a way of allowing the user to get currency.	– User able to train miners – User able to set miners to collect gold from a mine which is placed in a pre-determined location.
	2.5	Troops	Have different sorts of attacking troops which can attack the enemy structures and also enemy troops.	Page 10	There needs to be a way to interact with the opposition, this is the way. It also gives a way of accomplishing the winning condition.	– Train troops – Have multiple different types of troops – Give troops a visual health bar.
	2.6	Building repairs	Repair your structures by sending some miners away from mining and into fixing a building. This means buildings will also need a health bar.	Page 6	Gives a way of repairing buildings without having to build a completely new tower. Also gives another function of miners.	– Building have health bars – Building can be repaired by miners.

Requirement		User Requirement	Solves	Evidence	Justification	Success Criteria
		Saving progress	How to save data			
3	3.1	Levels complete	Handles saving progress of the user so they can come back when they left off after closing the game.	page 17	Not all levels are unlocked at the start of the game, so I do not want the user to have to re-complete levels which they have already done in order to get to where they left off.	– Have some way of saving the current level of that user even when the game is closed.
	3.2	Saving keybinds	Saves the keybinds of a particular user as it is customisable and can change.	Page 6	If the keybinds weren't saved the user would have to re-enter the changes they made to the keybinds every time they reloaded up the game.	– A working method that saves keybinds.

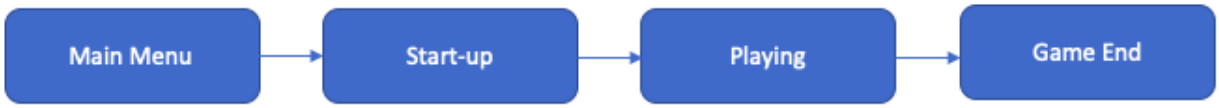
Part II

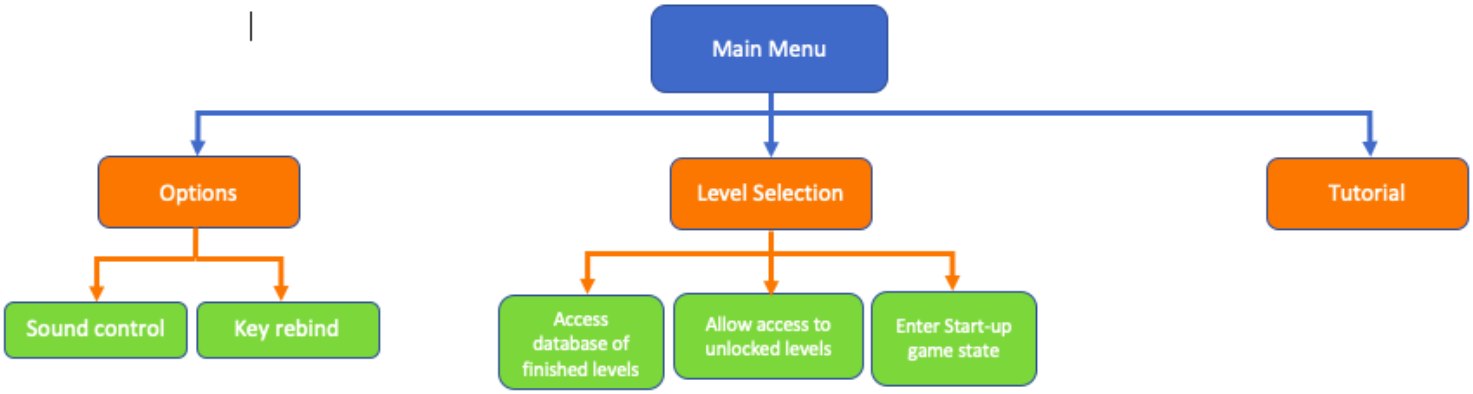
Design

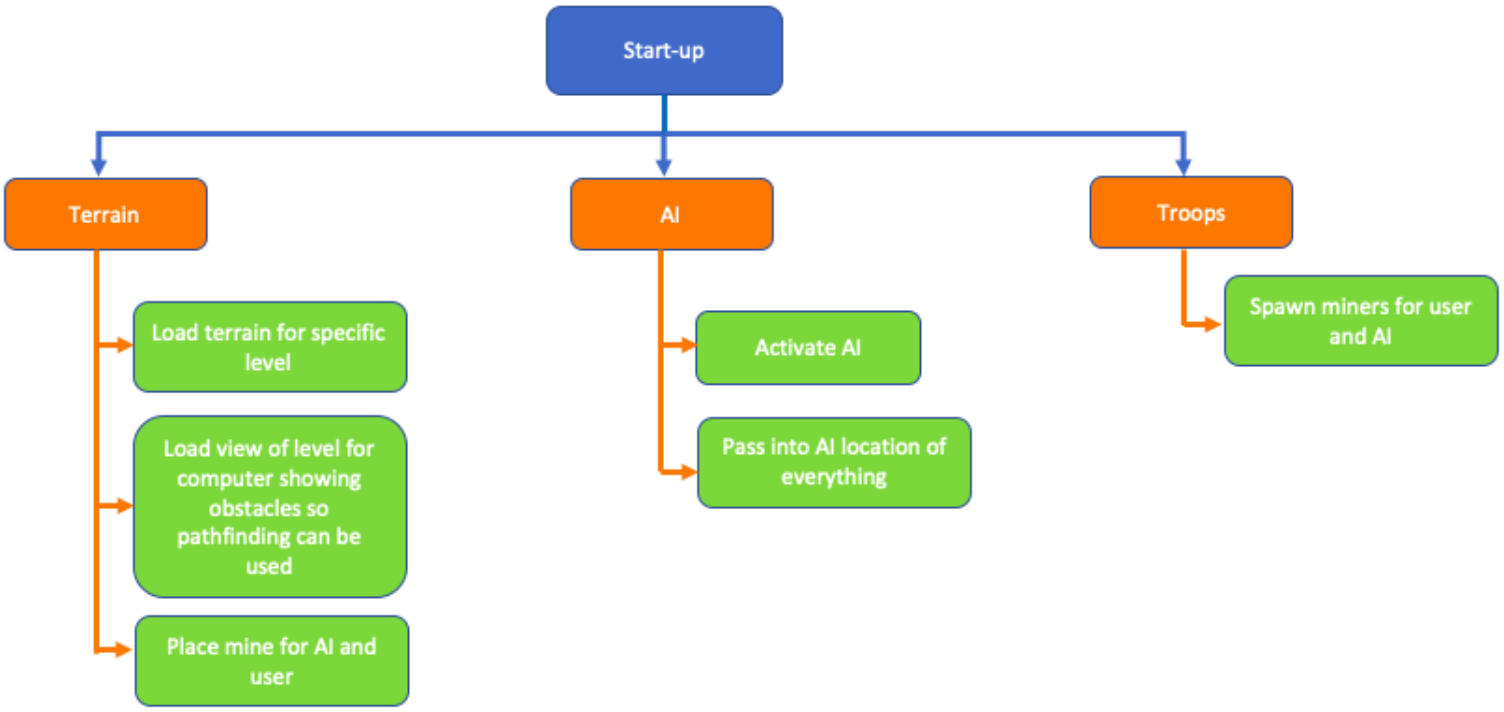
Chapter 3

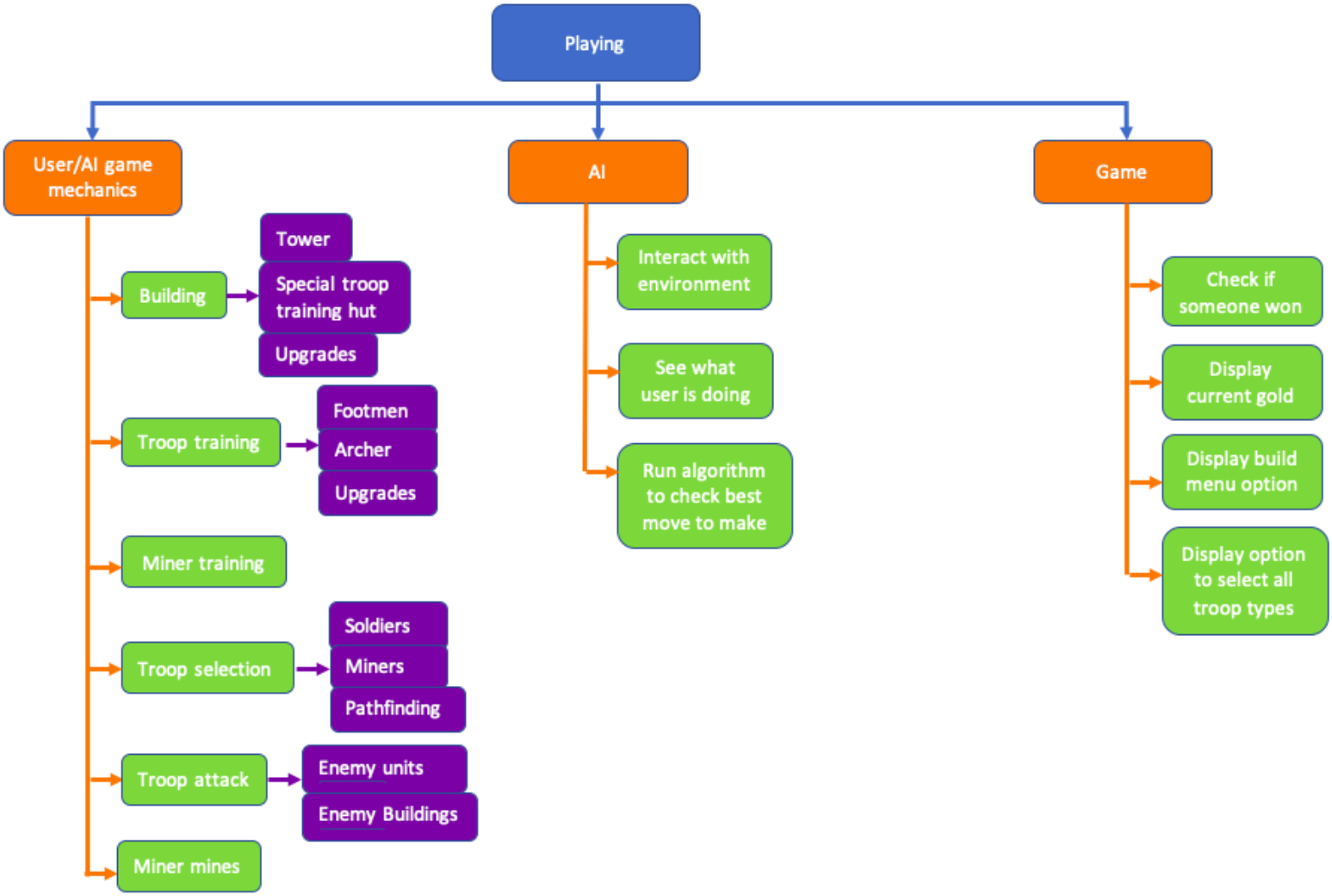
Design structure

3.1 Top down design



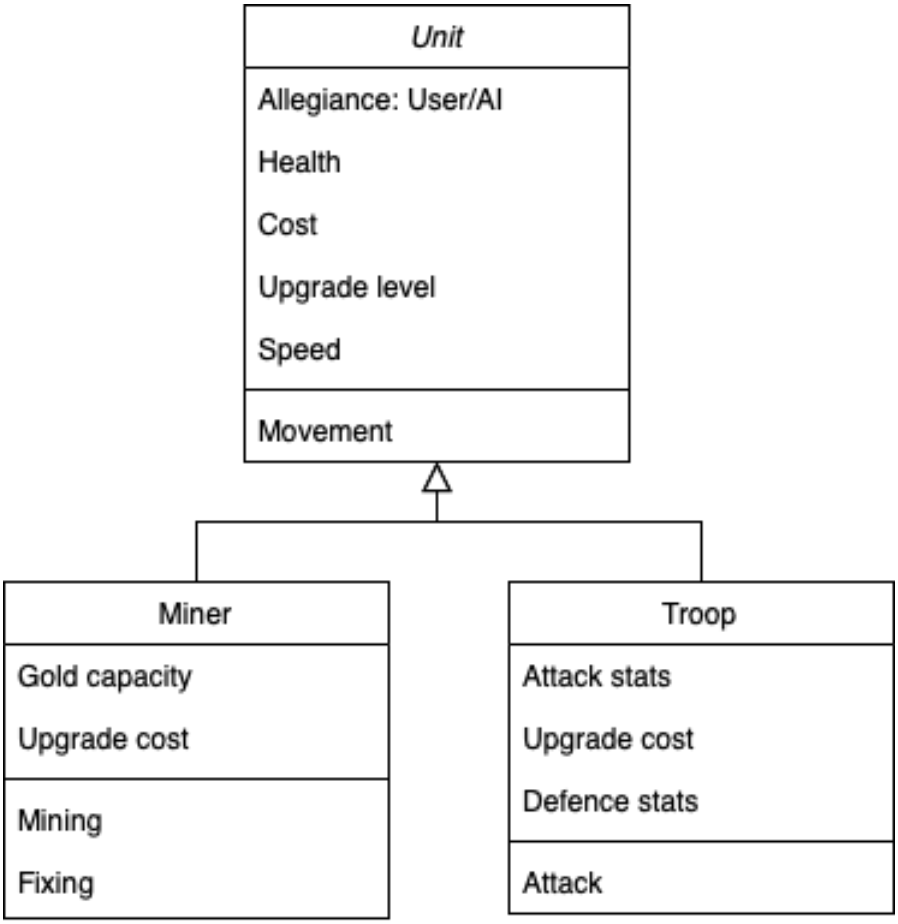


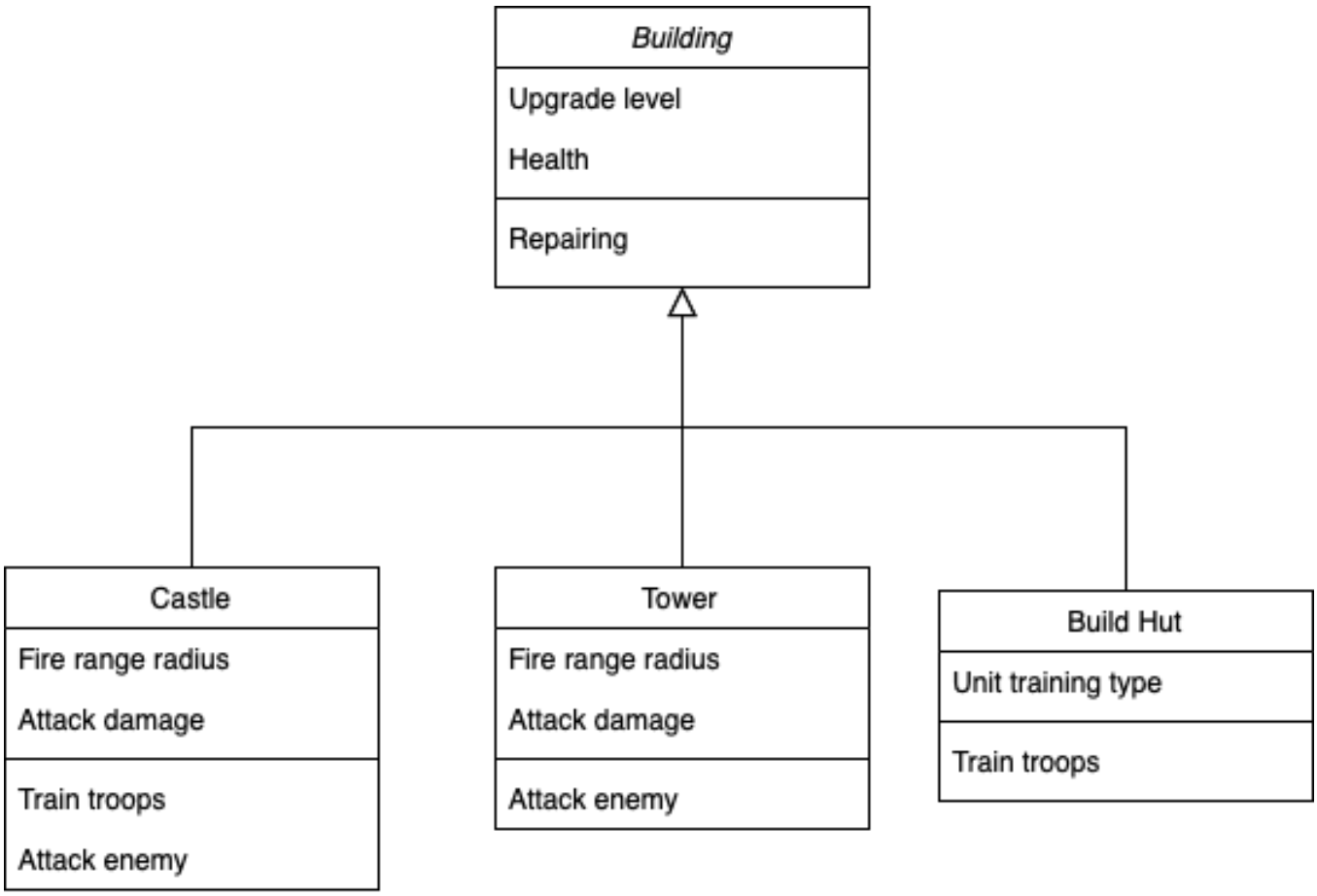

Start-up
Terrain
AI
Troops
Load terrain for specific level
Load view of level for computer showing obstacles so pathfinding can be used
Place mine for AI and user
Activate AI
Pass into AI location of everything
Spawn miners for user and AI

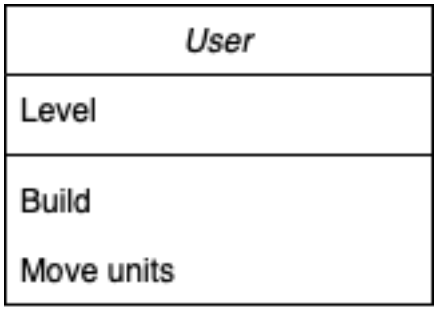

Playing
User/AI game mechanics
AI
Game
Building
Tower
Special troop training hut
Upgrades
Troop training
Footmen
Archer
Upgrades
Miner training
Troop selection
Soldiers
Miners
Pathfinding
Troop attack
Enemy units
Enemy Buildings
Miner mines
Interact with environment
See what user is doing
Run algorithm to check best move to make
Check if someone won
Display current gold
Display build menu option
Display option to select all troop types


Game End
Display message
Return to main menu
Save game result

3.2 Class Diagrams



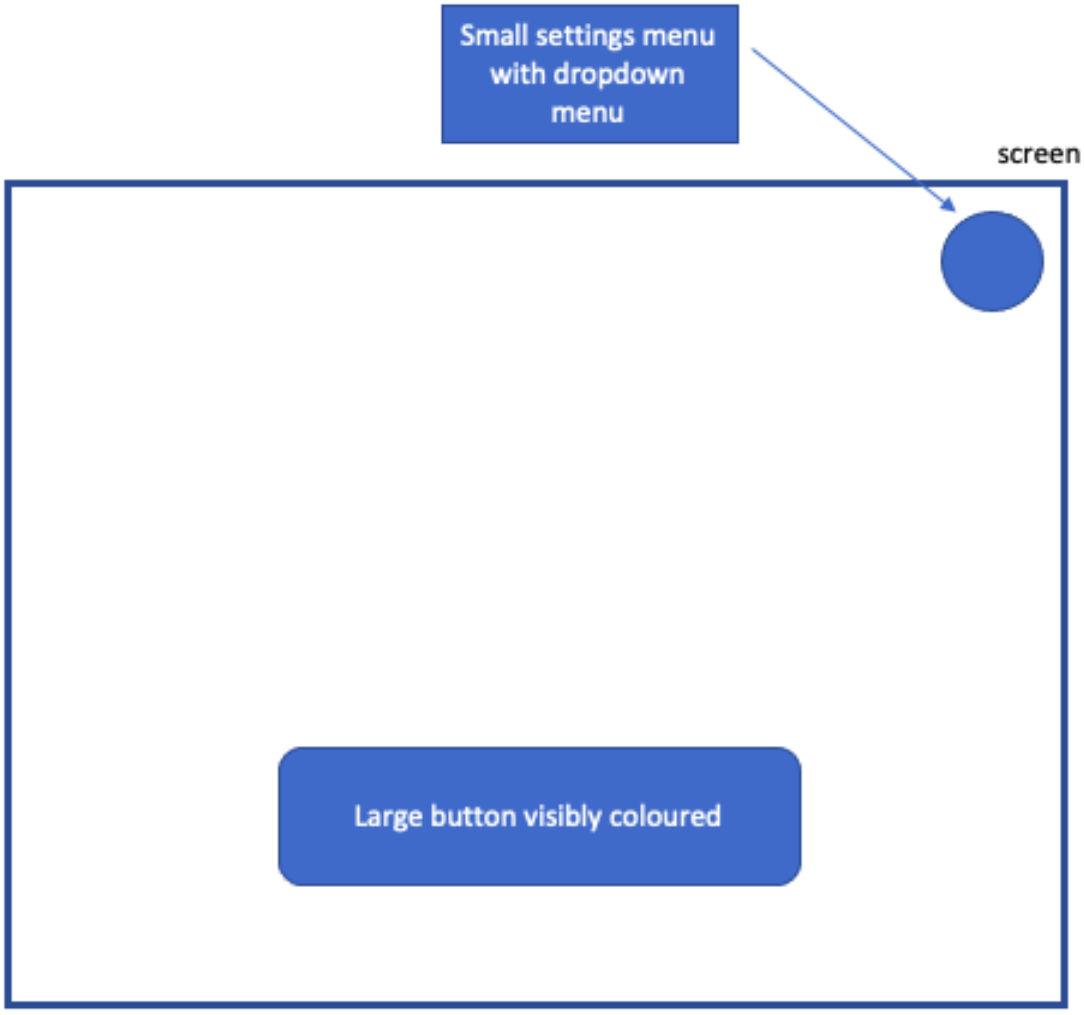




3.3 User interface design

Main menus

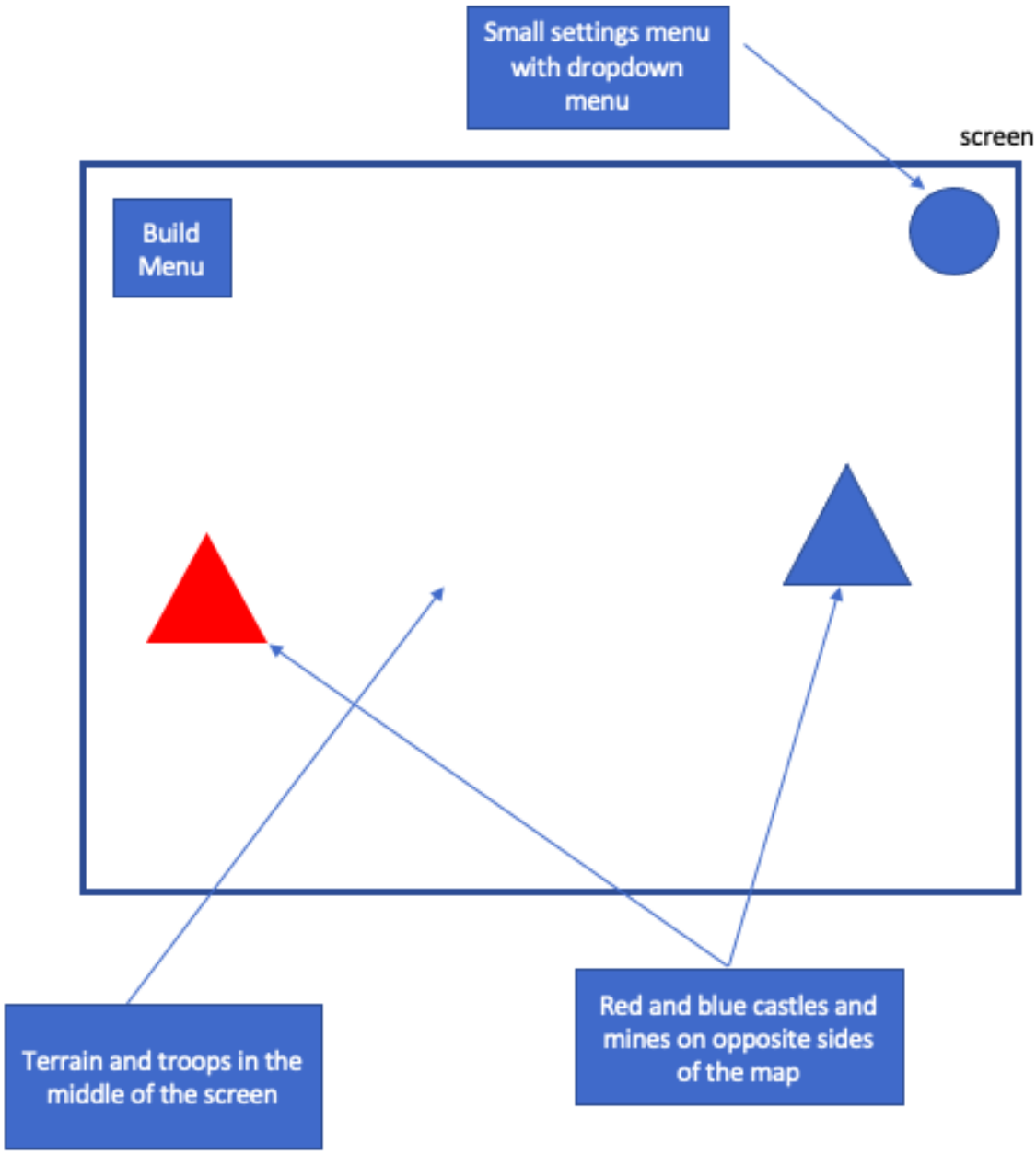
I want the design of the game to be simple and easy for the user to use. I also want the levels to be easy to access and change between. For this reason I will have a minimal amount of visible buttons in every menu, with dropdowns for things like setting menus and add a visible, stark colour to the button to go to the next menu screen so the user can easily find it.



The game itself

I want most of the game to be point and click game so there isn't too many buttons or keybinds that need to be pressed to keep it simple to the user. However there are certain menus that need to be on the screen; the settings menu, and the build menu (as per user requirements 2.3 & 1.2).

Also I want to place the castles and mines for each player on opposite sides of the screen and map, to give each player room and to make sure all the space on the map is used.



3.4 Algorithms

3.4.1 Algorithm 1: Attacking between troops

As part of the character class:

WHEN another object collides with this object THEN

```
    IF unit is not stationary
        GET the soldier script of enemy
        IF allegiance of enemy is not the same as this troops allegiance
            make object stationary
            START method attack another soldier passing in the enemy solider gameobject
```

The attack method:

```
WHILE enemy health > 0 AND this is alive AND enemy is alive AND is allowed to attack THEN
    WAIT 0.4s
    enemy health -= attack damage
    WAIT 0.4s
ENDWHILE
```

This algorithm makes sure soldiers can attack each other and can also be adapted with different checks to make soldiers able to attack miners and towers. It fulfils user requirement 2.5.

3.4.2 Algorithm 2: Mining cycle

This algorithm ensures that the user can get the miners to collect gold from the mine and deposit it in the castle, without the user actually having to do anything. This method uses the Movement function in the character class which takes in a position to move and moves the troop calling the method to that location via an A* pathfinding algorithm.

```
IF should mine
    IF close to the mine
        IF function not already called
            function called = true
            WHILE current gold + gold per cycle <= gold capacity THEN
                current gold += gold per cycle
                WAIT 1s
            ENDWHILE
    MOVEMENT(Castle_position)
    function called = false

    ELSE IF close to castle
        WAIT 0.2s
        IF allegiance == Blue
            Blue player gold += current gold
```

```
ELSE
    Red player gold += current gold
current gold = 0
WAIT 0.2s
MOVEMENT(Mine_position)
```

This algorithm fulfils user requirement 2.4 about the user able to get gold.

Testing for this function would need to check:

- The starting gold of the miner before reaching the mine,
- How quickly the miner gains gold to make sure it doesn't gain it too quickly or slowly due to the algorithm being flawed,
- Make sure the miner collects its full gold capacity
- Make sure the miner automatically reaches the castle
- Check that the miner deposits all gold to the castle so that the player gold goes up and the miners current gold goes to 0
- Make sure the miner then automatically returns to the castle

There would also need to be special case testing of this algorithm to check:

- What happens if a miner is attacked by an enemy while on the cycle
- What happens if the user tries to move a troop while in the cycle
- What happens if a miner already carrying gold at full or partial capacity is set on the mining cycle

3.5 Pre-development testing criteria

Before development of the game, I created a test table of features that needed to work in order for my game to work.

Part of game	Action to test	Working?
Main Menu	Load up game from start menu	
	Cauldrons passively producing smoke	
	Cauldrons produce increased smoke filling the screen when start game button pressed	
	Open up a settings menu	
	Open the level selection area once button pressed	
	Have some levels locked until earlier ones complete	
	When level selected, that level is loaded and shown on screen	
Level startup	Both AI and User start on same money, with the same buildings	
	Mine position and castle position found	
	Build menu displayed on screen	
	User gold displayed on screen	
	AI algorithm scanned walkable ground	
Castle	Castle can be clicked to open build menu	
	Soldiers can be trained by clicking on them	
	Miners can be trained by clicking on them	
	Training troops takes away gold from gold count	
	Visual gold count updated when changed	
	Waiting time for training troops	
	Castle can shoot at enemies approaching	

Miner	Can be set on mining cycle	
	Will pick up gold from the mine	
	Will deposit gold in the castle	
	Visual gold count changes	
	Can move to specified location	
	Can run away when being attacked	
	Changes colour based on allegience	
Soldier	Can move to any specified location	
	Can attack other soldier	
	Can attack other miners	
	Can attack towers and castes	
	Can run away from fights	
	Will stop attacking enemies when they are out of range	
Towers	Can be built using a build menu at a gold cost	
	Can shoot at enemy units	
	Will not shoot at friendly units	
	Can not be built on enemy side of the map	
	Cannot be built on top of existing buildings	
AI	AI can spawn in troops	
	AI can spawn in towers	
	AI can move units	
	AI can attack opposition units	
	AI can defend incoming attacks by moving units	

Part III

Development

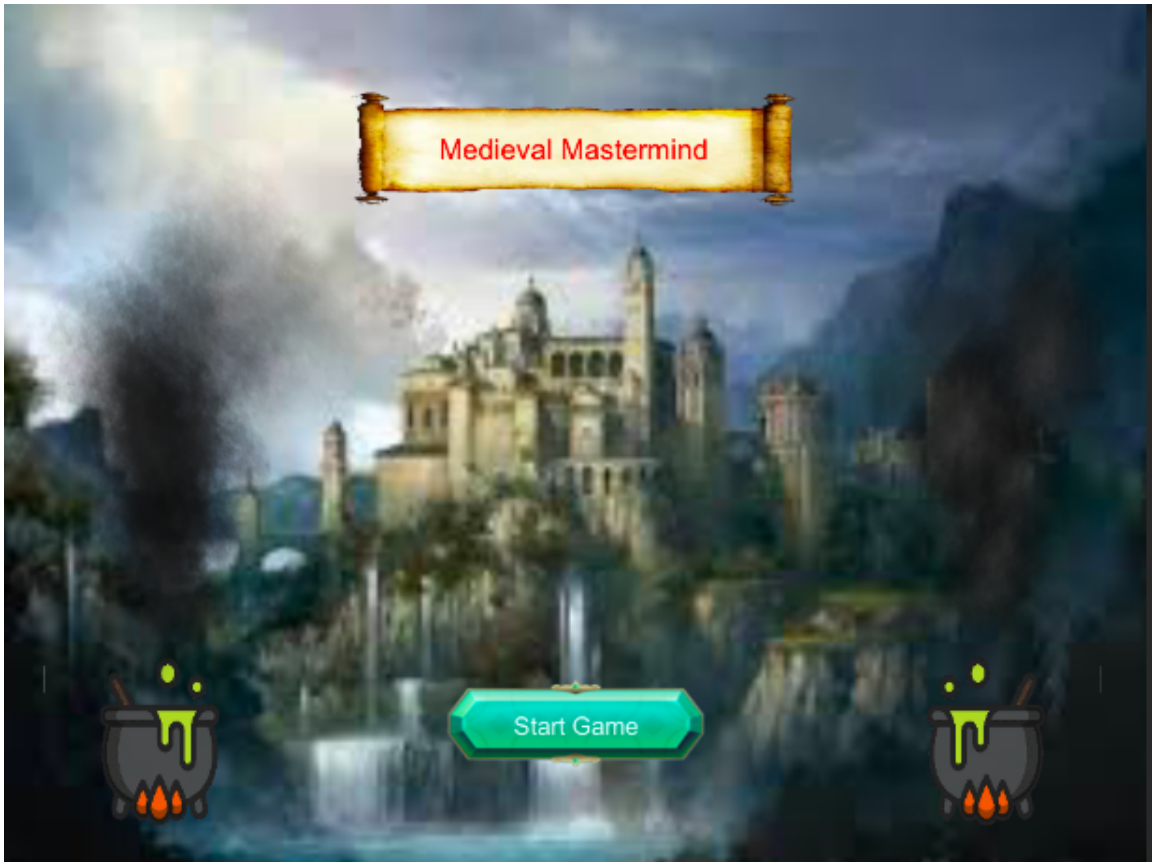
Chapter 4

The journey

4.1 Main Menu Screen

The first problem I tackled was the main menu scheme. Given the style of game I wanted to give it a medieval style look. This was achieved using things like custom font-types such as 'Medieval sharp'.

To add to the medieval look I also wanted to add some smoking cauldrons to the scene which I made using unity smoke particle system which just naturally released smoke over time. So the main game main menu now looks like this.



Where the start game icon is a button that should take you to a level select page.

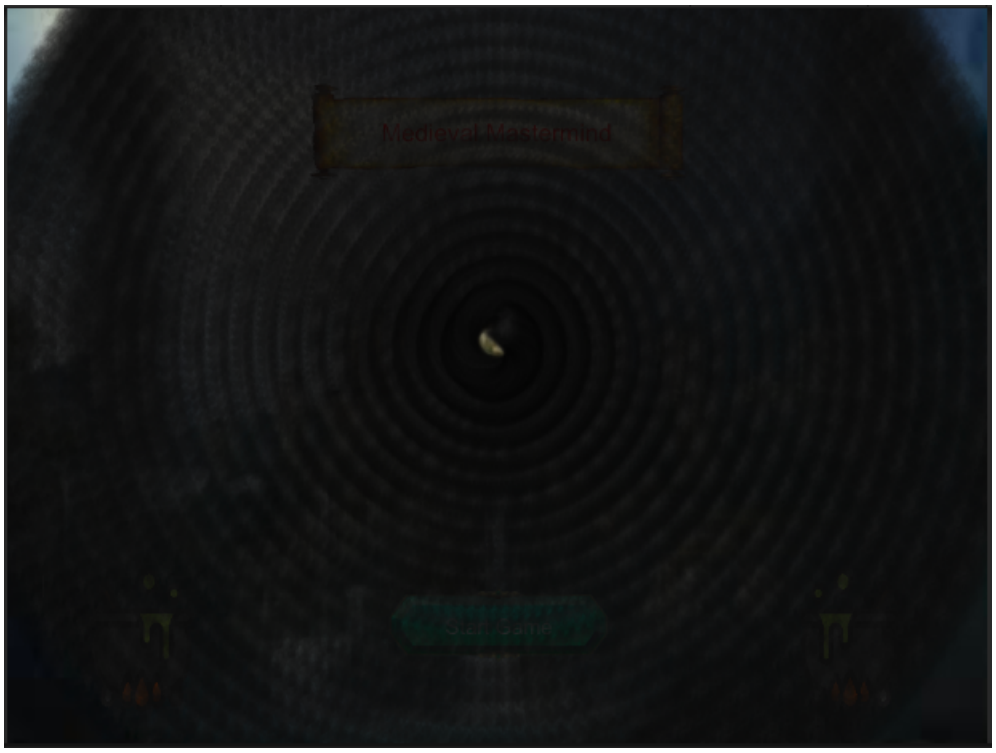
I wanted to add some animation to the smoke once the start game button was pressed to give some more animated graphics to the screen. This was done by changing the characteristics of the smoke particles being emitted, via the method 'change_particles()'.

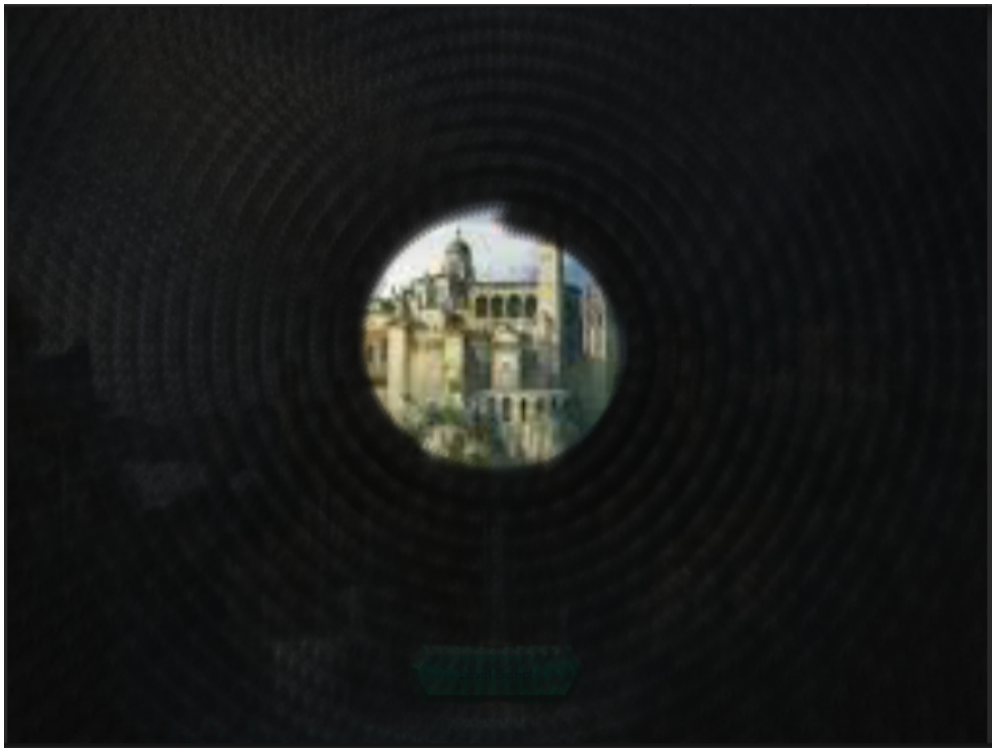
```
public void change_particles()
{

    ParticleSystem part = smoke.GetComponent<ParticleSystem>();

    var emission = part.emission;
    emission.rateOverTime = 15.0f;

    var sim_speed = part.main;
    sim_speed.simulationSpeed = 1f;
    sim_speed.startLifetime = 6.8f;
    sim_speed.startSize = 40f;
    sim_speed.gravityModifier = -3f;


    var shape = part.shape;
    shape.arc = 15.0f;

    StartCoroutine(change.stop());




    ParticleSystem part1 = smoke1.GetComponent<ParticleSystem>();

    var emission1 = part1.emission;
    emission1.rateOverTime = 15.0f;

    var sim_speed1 = part1.main;
    sim_speed1.simulationSpeed = 1f;
    sim_speed1.startLifetime = 6.8f;
    sim_speed1.startSize = 40f;
    sim_speed1.gravityModifier = -3f;


    var shape1 = part1.shape;
    shape1.arc = 15.0f;


}
```

This method does not take any arguments nor does it return any. First it fetches the particle system attached to each cauldron, then change the attributes about how the smoke is formed so that the cauldrons produce more smoke which travels faster and then starts the coroutine 'stop()'.

```
public IEnumerator stop()
{



    int children = transform.childCount;

    GameObject[] childs = new GameObject[children];

    for (int i = 0; i < children; i++)
    {
        if (i > 0)
        {
            childs[i] = transform.GetChild(i).gameObject;

        }
    }


    smoke3.SetActive(true);

    var start = smoke3.GetComponent<ParticleSystem>();
    var start1 = start.emission;
    start1.rateOverTime = 150f;

    yield return new WaitForSeconds(2.55f);

    for (int e = 1; e < children; e++)
    {
        childs[e].SetActive(false);
    }

    yield return new WaitForSeconds(2);
    smoke3.SetActive(false);

}
```

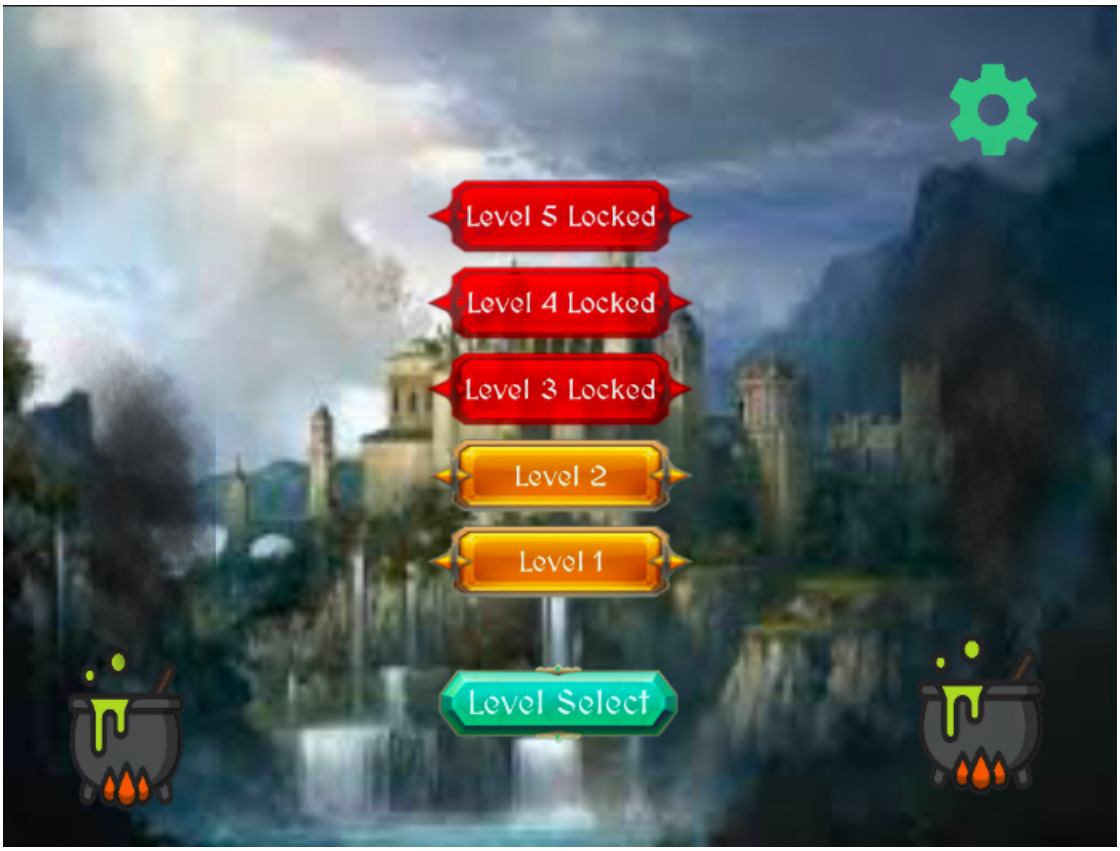
This script creates a middle burst of smoke which spirals outwards to fill the screen so that the next menu can be loaded underneath. This leaves the transition looking like:





I then imported and used the medieval sharp custom font as can be seen below.

The next menu pages followed a similar design using the same functions on the cauldrons and smoke. I also added a settings button which just toggles on and off to show the settings options.



In this you can see the level select menu. There are 5 different levels which can be unlocked, you have to complete the previous level to unlock the next level. I did this as the levels get harder the later on in the game, so I didn't want to make it too hard too early to ease the user into the game.

4.1.1 First Review

What has been done?

The menu screen has been built with buttons to start the game, see the settings menu and select a level to play. A smoke effect has also been added to the cauldrons to add to the look of the game.

How has it been tested?

It has been tested by visually checking that all the effects between screen changes work as intended and also that the buttons take you to the right place with the right menu's showing up.

How does it meet the user requirements and success criteria?

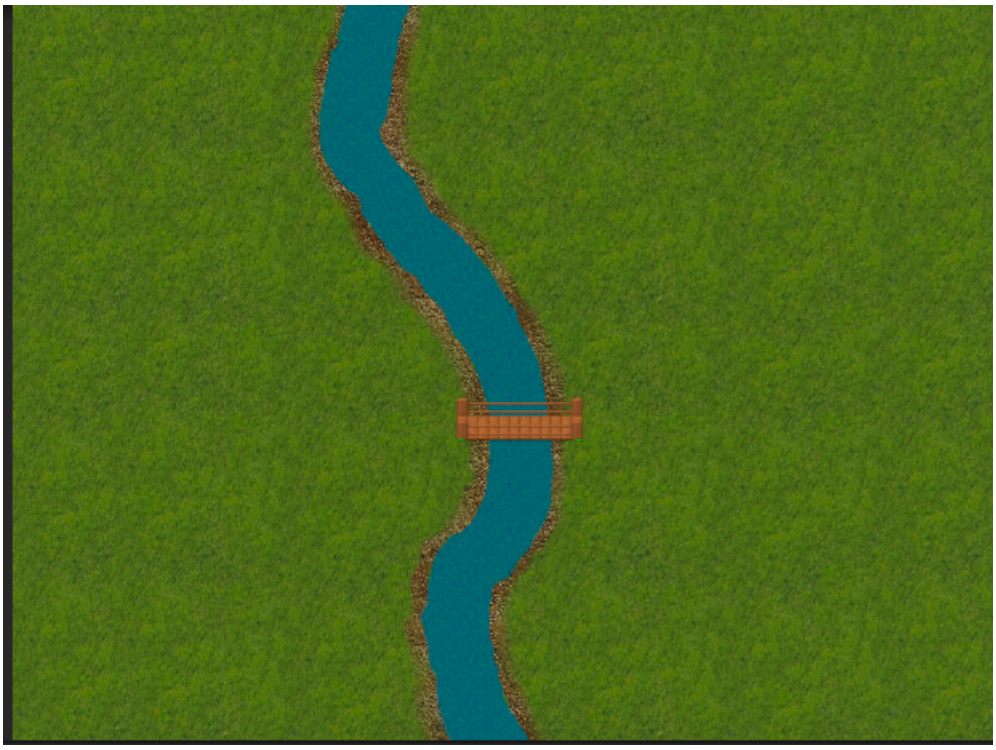
User requirements 1.1 (Buttons to access different areas of the game), 1.2 (Option preferences which is met by the setting menu) and 1.3 (Different levels), 3.1 (Unlocking different levels).

What user requirements were not met in this section?

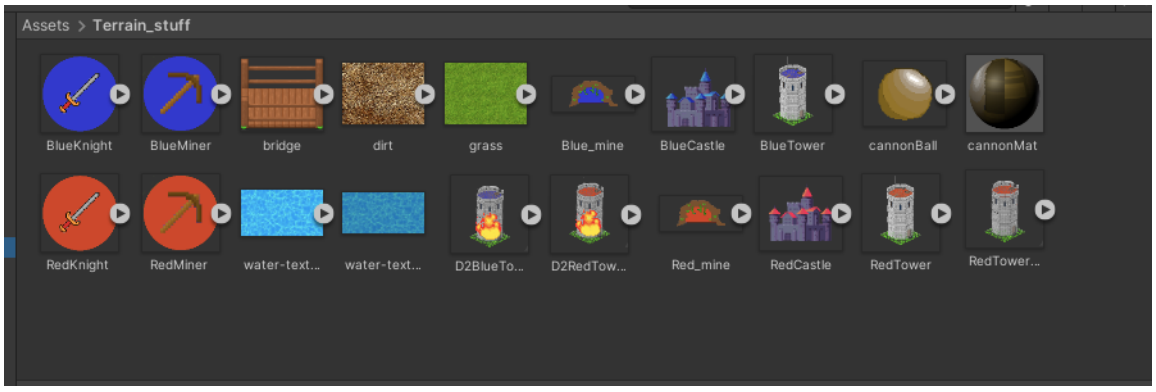
1.4 a tutorial menu was not added in this section as in order to make this the game itself would first have to be built.

4.2 The game itself

For the game, first some general design of how the background of the first level would look. I wanted to split up the AI and User areas, so to do this, I put a river flowing through the map breaking up the two. For the terrain the troops would be on, I used a unity terrain. I also added a bridge across the middle for the troops to cross.



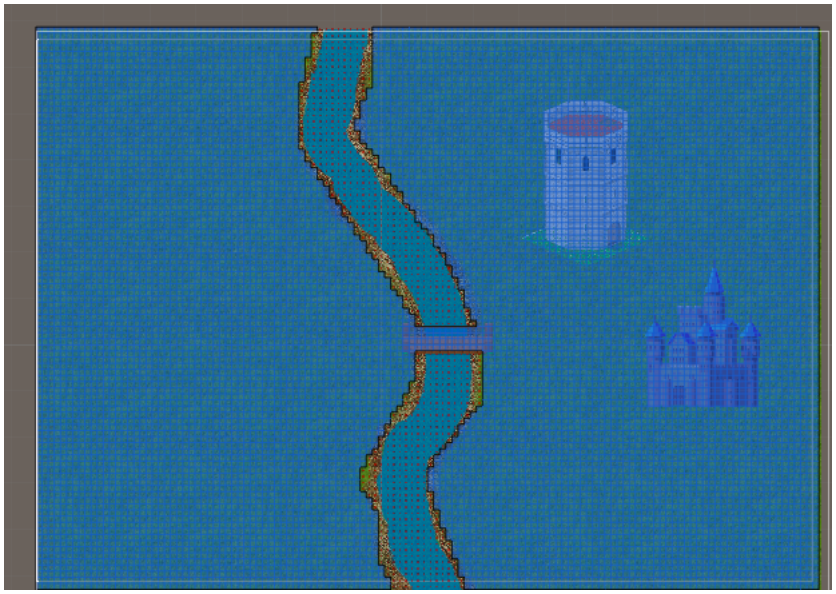
Next, for the UI elements I used a picture editing software called GIMP to make the images. All asset images can be seen below:



4.2.1 Movement of the troops

How to get the troops to move wherever the user or AI wanted to was something needed for the game, but also difficult. To do this, I used an imported A* algorithm from https://arongranberg.com/astar/ to help with my project and movement.

This is a 3D pathfinder so I had to trace round my river in SketchUp to make it a 3d model the pathfinder could recognise, so the walkable ground after doing this and importing the 3d model I now have:



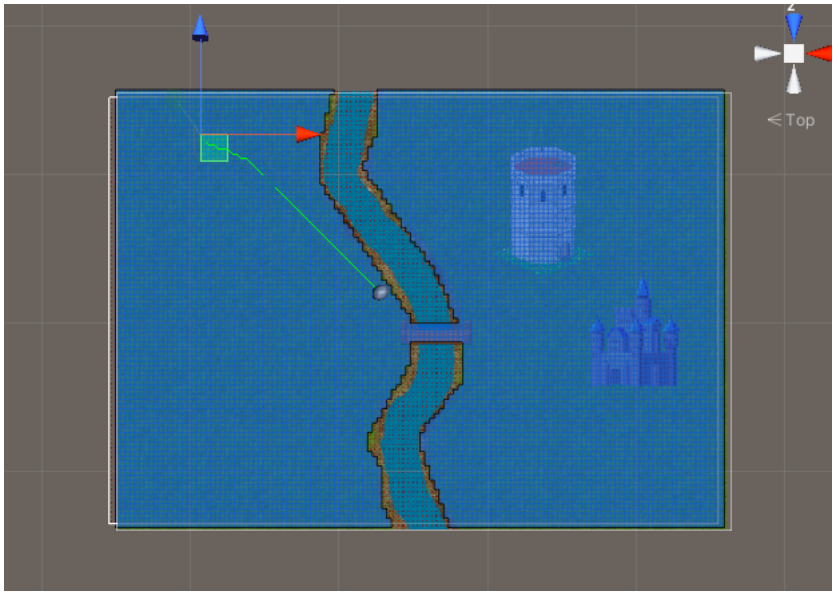
Where the red dots are unworkable ground.

How and A* algorithm works:

To roughly explain how an A* algorithm works:

- Like Djikstra, A* works by finding the lowest 'cost' of path between two nodes.
- However it uses things known as heuristics to take an educated guess at the fastest path so less computation time is needed.
- For the game, this means that there is a grid of dots or nodes on the plane of the terrain, and when a target location is specified, the A* algorithm connects up the nodes in one of the shortest ways possible, while avoiding obstacles.

The path the troop would take according to the A* algorithm can be visualised:



With the green line tracing out the path.

As can be seen from this green line, as you get closer to the target the movement becomes more jittery, this is due to a lack of 'smoothing'. This is not an issue I fixed in this version of the game as I was a one man team with limited time, and it barely negates from the user's playing experience.

```
    void Movement(GameObject character, Vector3 location_to_move)
    {
        //This script controls where the user wants to move a certain troop to and runs the A* algorithm to take it there.


        //instantiate at location_to_move location



        GameObject motion = Instantiate(Target, location_to_move, Quaternion.identity);
        AIDestinationSetter A = transform.GetComponent<AIDestinationSetter>();
        A.target = motion.transform;
        move_script.canSearch = true;
        move_script.canMove = true;
        Destroy(motion);
    }
}
```

This is the small snippet of code that is attached to the characters to control the movement.

4.2.2 The characters

The next main problem was the characters - ie the soldiers and miners. I handled the development of shared characteristics of the Miners and Soldiers by using a parent characters class.

One of the first features I implemented in the characters class was the movement of troops. This was handled using mouse clicks and sending out a raycast in the screen and finding what that raycast hit. If the raycast hit an object that the user could move, a shadow would appear around

the object, and if somewhere accessible was then pressed somewhere else on the terrain, the troop would move to that location.

This can be seen from this code:

```
// Update is called once per frame
void Update()
{
    //This is to check if the mouse button has clicked a game object, and wether the clicked game object wants to be moved.
    if (Input.GetMouseButton(0))
    {
        Ray ray = PlayView1.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        // Casts the ray and get the first game object hit
        Physics.Raycast(ray, out hit, 10000.0f);
        if (hit.collider)
        {
            if (hit.collider.gameObject.name == "Character")
            {
                //Stores the game object hit and activates the flag marking that a valid character has been selected
                shadow.SetActive(true);
                selected = hit.collider.gameObject;
                active_selection = true;



            }
            else if (active_selection)
            {
                active_selection = false;
                shadow.SetActive(false);
                //here a valid target has previously been selected so then we want to call the movement function to move the target for the A* script and make the character move
                Movement(selected, hit.point);

            }
            else
            {
                active_selection = false;
                shadow.SetActive(false);
            }

        }



    }
}
```

One of the errors I had with this script was when trying to spawn in multiple players at the same time, I had the error of the two objects moving together at the same time, and to the same place, even when only one of them was pressed.

This was fixed to change the name the game object needed to find is referenced to the game object the script is on, rather than the just a string:

```
void Update()
{
    //This is to check if the mouse button has clicked a game object, and wether t
    if (Input.GetMouseButton(0))
    {
        Ray ray = PlayView1.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        // Casts the ray and get the first game object hit
        Physics.Raycast(ray, out hit, 10000.0f);
        if (hit.collider)
        {
            if (hit.collider.gameObject.name == transform.gameObject.name)
```

After implementing the movement function, I made the child classes that would inherit from the character class. This was done using object oriented inheritance:

```
public class Miner : character_controll
```

```
public class Soldier : character_controll
```

For the miner script, the attributes gold_capacity and current_gold are defined:

```
private int gold_capacity = 10;
private int current_gold;
```

Because of how I ended up spawning in the troops, I needed to add a check to the miner class to check wether its allegiance (user is blue AI is red) has changed, and if it has to change the material to the other colours' material.

```
if (allegience1 == allegience)
{

}
else
{
    allegience1 = allegience;
    change_allegience();

}
```

```
void change_allegience()
{
    // Changes whether trigger is active or not for each different allegience
    if (allegience == 'R')
    {
        gameObject.GetComponent<CapsuleCollider>().isTrigger = false;
        gameObject.GetComponent<MeshRenderer>().material = matR;
    }
    else if (allegience == 'B')
    {
        gameObject.GetComponent<CapsuleCollider>().isTrigger = true;
        gameObject.GetComponent<MeshRenderer>().material = matB;
    }
}
```

4.2.3 Getting the mining cycle working

The next challenge was getting the mining cycle working - this is the automatic movement of the miners from the mine to the castle depositing money every time they get to the castle and filling up the gold of the miner while at the mine up to the maximum gold capacity. The gold counter showing how much gold you currently have should also be updated every time gold is deposited in the castle.

Most of the code for this section is in the coroutine get_gold().

```
private IEnumerator get_gold()
{

    if (mine_it) {

        //Need to empty the gold from the miner, need to update the text box for the amount of gold so its visible (and test to see if gold actually is updating)

        if (Vector3.Distance(transform.position, mine_position) < 10) //If it is within x units of the mine's position
        {

            if (!called)
            {
                called = true;
                while (current_gold + PlayView1.gameObject.GetComponentInChildren<Mining>().gold_per_second <= gold_capacity) //Fills up the gold until it has reached maximum ca
                {

                    current_gold += PlayView1.gameObject.GetComponentInChildren<Mining>().gold_per_second;
                    yield return new WaitForSeconds(1f);

                }
                Movement(gameObject, castle_position);
                called = false;
            }




        }


        else if (Vector3.Distance(transform.position, castle_position) < 10)
        {
            yield return new WaitForSeconds(0.2f);
            PlayView1.gameObject.GetComponentInChildren<Player_blue>().BroadcastMessage("update_gold", current_gold);
            current_gold = 0;
            yield return new WaitForSeconds(0.2f);
            Movement(gameObject, mine_position);

        }
        else
        {

        }

    }
}
```

```
if (mine_it) {

    //Need to empty the gold from the miner, need to update the text box for the amount of gold so its visible (and test to see if gold actually is updating)

    if (Vector3.Distance(transform.position, mine_position) < 10) //If it is within x units of the mine's position
    {

        if (!called)
        {
            called = true;
            while (current_gold + PlayView1.gameObject.GetComponentInChildren<Mining>().gold_per_second <= gold_capacity) //Fills up the gold until it has reached maximum ca
            {

                current_gold += PlayView1.gameObject.GetComponentInChildren<Mining>().gold_per_second;
                yield return new WaitForSeconds(1f);

            }
            Movement(gameObject, castle_position);
            called = false;
        }




    }
```

From code above: If mine it is true - this is a check in character_controll which checks a miner has been selected and the mine has been pressed. The troop will automatically move to the mine as part of the movement script. Once it is close (as get_gold() is called in fixed_update), if the function hasn't yet been called (this is to stop update calling the function many times over before it has moved, it says that it has now been called, then while the current gold is less than its maximum capacity, the gold is slowly filled into the miner's sack. Once its full, the movement function is called to move it back to the castle.

```
else if (Vector3.Distance(transform.position, castle_position) < 10)
{
    yield return new WaitForSeconds(0.2f);
    PlayView1.gameObject.GetComponentInChildren<Player_blue>().BroadcastMessage("update_gold", current_gold);
    current_gold = 0;
    yield return new WaitForSeconds(0.2f);
    Movement(gameObject, mine_position);

}
else
{

}
```

Once it is at the castle (or close to it) the update gold function on the main player script is called which updates the gold value seen on screen and the variable, empties the miners sack, then send the miner back to the mine.

Also tested it to make sure you can break the miner out of a constant loop of mining, with the amount of gold currently in the mining bag still saved for next time the mining cycle is started again.

4.2.4 The attacking between soldiers

The attacking between troops is handled in the character control parent script and the soldier child class itself. In character control, the method to detect if two soldier have collided is:

```
void OnTriggerEnter(Collider other)
{
```

To roughly describe how this works:

- If other thing is a soldier then check there not same allegience and check other conditions,
- THEN if this is a soldier call the attack class (if its not a soldier do nothing)
- Else if the other thing is a Miner check they are not the same allegience and other conditions (same as above) again,
- THEN if this is a soldier call the attack class again

```
if (other.gameObject.tag == gameObject.tag | other.gameObject.tag == "RedCastle" | other.gameObject.tag == "BlueCastle") //Make sure it will only attack and collide with
{

    if (other.gameObject.GetComponent<Soldier>())
    {

        Soldier scr = other.gameObject.GetComponent<Soldier>();
        if (scr.allegience != allegience)
        {

            if (other.gameObject != gameObject) // Work around for random bug when it collides with itself off spawn
            {

                stop_start(other);
                stationary = true;
                if (gameObject.GetComponent<Soldier>())
                {
                    StartCoroutine(transform.GetComponentInChildren<Soldier>().attack(other.gameObject));
                }
            }
        }
    }
    else if (other.gameObject.GetComponent<Miner>())
    {
```

```
else if (other.gameObject.GetComponent<Miner>())
{
    Miner scr = other.gameObject.GetComponent<Miner>();
    if (!gameObject.GetComponent<Miner>()) {

        if (scr.allegience != allegience)
        {

            if (other.gameObject != gameObject) // Work around for random bug when it collides with itself off spawn
            {


                stop_start(other);
                stationary = true;
                if (gameObject.GetComponent<Soldier>())
                {
                    StartCoroutine(transform.GetComponentInChildren<Soldier>().attack(other.gameObject));
                }
            }
        }
    }


}
```

The attack coroutine method can be seen below. It takes an argument of the gameobject it has collided with and if certain conditions (other's health is more than 0, it itself is alive and it is allowed to attack) then it takes the heath away from the opposition repeatedly at intervals. The condition of is allowed to attack is there so that the AI or user can run away with its troops and it will actually stop attacking the opposition if it is out of its fighting range. To change the value of canAttack to tell if its out of range, the method onTriggerExit was needed:

```
private void OnTriggerExit(Collider other)
{
    //stop attacking

    if (gameObject.GetComponent<Soldier>())
    {
        gameObject.GetComponent<Soldier>().canAttack = false;
    }
    stationary = false;
}
```

And the attack script:

```
public IEnumerator attack(GameObject other)
{

    //the actual attacking between soldiers
    canAttack = true;
    others = other.gameObject;


    if (other.gameObject.GetComponent<Soldier>()) //Checks if it is attacking a soldier
    {
        Soldier opp = other.gameObject.GetComponent<Soldier>();
        while (opp.health > 0 && isAlive && opp.isAlive && health > 0 && canAttack)

        {

            yield return new WaitForSeconds(0.4f);
            opp.health -= attack_damage;
            yield return new WaitForSeconds(0.4f);
        }

        |
        yield return new WaitForSeconds(0f);

    }
    else if (other.gameObject.GetComponent<Miner>())
    {
        print(canAttack);
        Miner opp = other.gameObject.GetComponent<Miner>();
        while (opp.health > 0 && isAlive && opp.isAlive && health > 0 && canAttack)

        {
            yield return new WaitForSeconds(0.4f);
            opp.health -= attack_damage;
            yield return new WaitForSeconds(0.4f);

        }



        yield return new WaitForSeconds(0f);

    }
}
```

4.2.5 Spawning the troops

The next problem to solve was the spawning of the troops. To do this, I created a drop down menu on the castle, so that when the castle is clicked the menu to train the different types of troops are shown. The troops should be able to be clicked and a countdown started to spawn in the troop so that the troops can't just be instantly spawned in to balance the game and the cost of the troops should also be subtracted from the player gold total.

The current visual representation of this menu is:

The script to control all of this is attached to the castle gameobject and called 'button_controll' as which can be seen below:

```
public class button_controll : MonoBehaviour
{
    private bool is_showing = false;
    public GameObject Canvas;
    private Button[] all_buttons;
    private Image loading_shadow_knight;
    private Image loading_shadow_miner;
    public GameObject soldier;
    public GameObject miner;
    // Start is called before the first frame update
    void Start()
    {

        all_buttons = gameObject.GetComponentsInChildren<Button>();

        //Adding listeners to all the buttons
        all_buttons[0].onClick.AddListener(toggle_onOff);
        all_buttons[1].onClick.AddListener(train_knight);
        all_buttons[2].onClick.AddListener(train_miner);



        //Knght shadow
        Image[] loading_shadow_knight_list = all_buttons[1].gameObject.GetComponentsInChildren<Image>();
        loading_shadow_knight = loading_shadow_knight_list[1];
        loading_shadow_knight.fillAmount = 0;


        //Miner shadow
        Image[] loading_shadow_miner_list = all_buttons[2].gameObject.GetComponentsInChildren<Image>();
        loading_shadow_miner = loading_shadow_miner_list[1];
        loading_shadow_miner.fillAmount = 0;



        Canvas = gameObject.GetComponentInChildren<Canvas>().gameObject;
        Canvas.SetActive(false);
    }
```

```
void toggle_onOff()
{
    if (is_showing)
    {
        Canvas.SetActive(false);
        is_showing = false;
    }
    else
    {
        Canvas.SetActive(true);
        is_showing = true;
    }
}

void train_knight()
{
    //This is a method to sort out the visuals on the knight button and then spawn in a knight after a wait
    //using a coroutine IEnumerator to do wait times easier and also makes it modular so that both miner and knight images can use it
    if (loading_shadow_knight.fillAmount <= 0)
    {
        loading_shadow_knight.fillAmount = 1;
        StartCoroutine(undo_fill(loading_shadow_knight));
    }


}

void train_miner()
{
    //This works the same was as the train_knight() method
    if (loading_shadow_miner.fillAmount <= 0)
    {
        loading_shadow_miner.fillAmount = 1;
        StartCoroutine(undo_fill(loading_shadow_miner));
    }
}
```

```
IEnumerator undo_fill(Image image)
{

    while (image.fillAmount > 0)
    {
        yield return new WaitForSeconds(0.05f);
        image.fillAmount -= 0.01f;
    }

    if (image == loading_shadow_knight)
    {
        GameObject spawned = Instantiate(soldier, transform.position + new Vector3(-70, 0, -150), Quaternion.identity);
        spawned.GetComponent<Soldier>().allegience = 'B'; //Need to make sure the allegience is set after spawning in the troop
    }

    else if (image == loading_shadow_miner)
    {
        //Haven't made the miner class yet, so need to go back and make sure miners spawn in once it is made. Added a simple error catch for now to check it works.
        try
        {
            GameObject spawned = Instantiate(miner, transform.position + new Vector3(8, 0, 8), Quaternion.identity);
            spawned.GetComponent<Soldier>().allegience = 'B';
        }
        catch
        {
            print("miner not assigned");
        }

    }

}
```

During the development of this section of the game, I spoke to people to see what they thought of the menu and they pointed out one main visual oversight which was the lack of actually visually representing how much each troop cost. This was easily fixed so the menu then looked like:

4.2.6 Problem with troop movement

During testing of the game of moving troops around repeatedly around the mines and towers, one thing I noticed was that troops could walk straight through the middle of the castle's and mine's which looked peculiar, so to fix this issue I added 3D cubes with a collider to the base of the castle that only the A* pathfinding would interact with the cubes and mark the area where they sit as unable to be walked on. So if the user clicked this area where the cube was, the AI would get as close as possible to target then stop. I tested this thoroughly to check wether it would get stuck constantly trying to get to the required destination, however as soon as it got as close as possible, the movement cycle would end instead meaning it didn't just get stuck.

4.2.7 Second Review

What has been done?

The movement of the troops has been handled using an A* pathfinding algorithm. The visual map elements were also made and the general character script was made which handled the movement and interaction of troops as well as the child classes of the miner which can be set on the mining cycle to collect gold and the soldier class which can attack enemy units. The spawning of the troops as part of the castle was also developed in this stage.

How has it been tested?

The movement of the troops was tested by first checking the troops actually moved to the intended position, then also testing boundary cases where it would try to move the troops to a location which they physically couldn't reach. The mining cycle was also tested to make sure that the user could set the miner on the mining cycle and then also stop the miner from continuing on this cycle whenever wanted. The attacking between units was tested by bringing every unit combination together and seeing if they reacted in the intended way and then also bringing in several different units at the same time and checking wether it still worked as intended.

How does it meet the user requirements and success criteria?

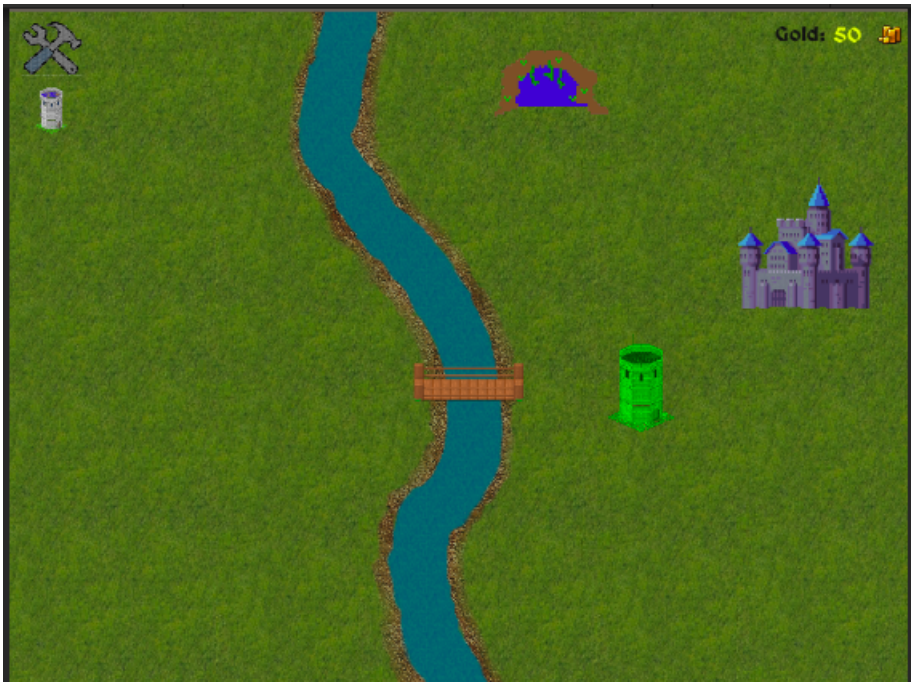
User requirements 2.1 (Pathfinding algorithm), 2.4 (The mine and miners) and 2.5 (Attacking of troops).

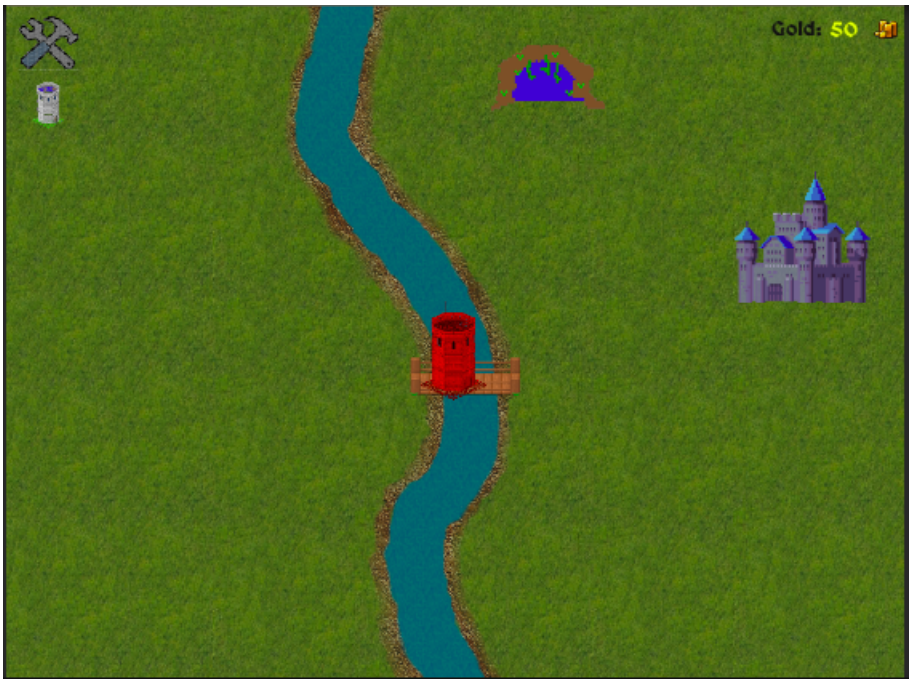
4.2.8 The build menu

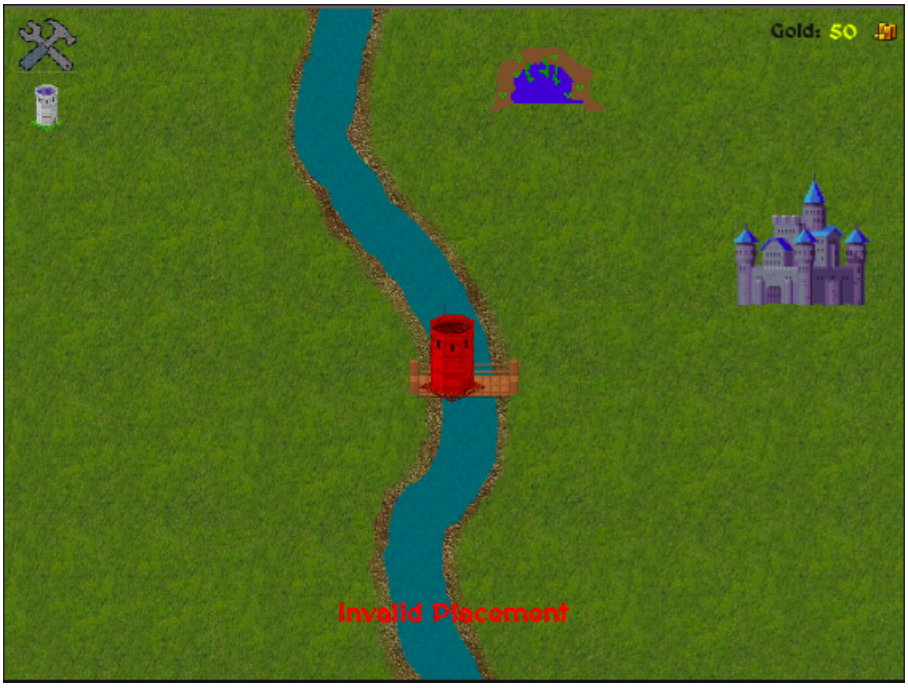
Next, a method was needed to spawn in towers and be able to place them.
The main features of this were:

- A clickable build menu button with a dropdown
- Dropdown to feature all buildings which can be built along with the cost of the building.
- One button clicked, the building needs to follow the mouse position of the user
- The building should not be able to be placed on the AI side of the playing arena,
- The building should not be able to be placed on top of each other or on top of existing buildings
- If the user does try to place something where it shouldn't be, it should flash red with an appropriate error message on the screen.
- The tower should be highlighted green when hovering over an area where it can be placed
- Once the user clicks the screen, if it is in a valid area, it should be placed and should be able to start shooting enemies.
- Needs to check wether the user has enough gold to actually buy the tower and if not flash red with appropriate error message.

These show these requirements working:







The code to do all this is under the class "build_menu" which has attributes:

```
public class build_menu : MonoBehaviour
{
    /* In this i need to:
     * Get tower to follow mouse around once tower pressed at right size,
     * add cube to tower for health and other things to tower script,
     * Make sure tower can actually be placed where the user chooses,
     * make sure if there is a character underneath the placement of the tow
     * give meaningful messages to user if building cant be placed somewhere
     */

    private Button[] all_buttons;

    public GameObject Canvas;

    public GameObject Screen_canvas;

    private bool is_showing = false;

    private bool placed = true;

    public GameObject tower;

    private GameObject placed_tower;

    private Vector3 position_to_move;

    private Camera PlayView1;

    public int tower_cost = 50;

    public GameObject ErrorText;
```

The methods of this class are:

```
// Start is called before the first frame update
void Start()
{
    all_buttons = gameObject.GetComponentsInChildren<Button>();

    all_buttons[0].onClick.AddListener(toggle_showing);
    all_buttons[1].onClick.AddListener(build_tower);


    Canvas = gameObject.GetComponentInChildren<Canvas>().gameObject;
    Canvas.SetActive(false);


    Screen_canvas = GameObject.FindGameObjectWithTag("Screen_canvas");

    PlayView1 = GameObject.FindGameObjectWithTag("Camera1").GetComponent<Camera>();

    ErrorText = GameObject.Find("ErrorText");
}
```

```
// Update is called once per frame
void FixedUpdate()
{

    if (!placed) //Running in update as a while loop doesnt work properly in unity
    {

        Vector3 mousePos = Input.mousePosition;
        position_to_move = PlayView1.ScreenToWorldPoint(new Vector3(mousePos.x, mousePos.y, 2));
        placed_tower.transform.position = position_to_move;

        bool check = check_place();

        if (Input.GetMouseButtonDown(0)) //if the mouse has been clicked
        {
            if (check)
            {
                placed_tower.GetComponent<Image>().color = Color.white;
                Screen_canvas.GetComponentInChildren<Player_blue>().BroadcastMessage("update_gold", -tower_cost);
                placed = true;
            }
            else
            {
                StartCoroutine(error_text("Invalid Placement"));
            }
        }




    }

}
```

```
void build_tower()
{
    if(Screen_canvas.GetComponentInChildren<Player_blue>().player_gold >= tower_cost)
    {
        placed_tower = Instantiate(tower, new Vector3(0, 0, 0), Quaternion.identity);
        placed_tower.transform.SetParent(Screen_canvas.transform, false);
        placed = false;
    }
    else
    {
        StartCoroutine(flash_red(Canvas.gameObject));
        StartCoroutine(error_text("Not enough gold"));
    }
```

```
bool check_place()
{
    //This checks wether the position that has been selected can actually be placed there or not.
    RaycastHit hit;
    Physics.Raycast(placed_tower.transform.position, new Vector3(0, -1, 0), out hit, 1000);
    if (hit.transform != null)
    {
        if (hit.transform.gameObject.layer == 11)
        {
            //A valid placement - place gameobject here
            placed_tower.GetComponent<Image>().color = Color.green;
            return true;
        }
        else
        {
            //Not a valid placement
            placed_tower.GetComponent<Image>().color = Color.red;
            return false;

        }
    }
    else
    {
        //Not tried to place in the screen;
        placed_tower.GetComponent<Image>().color = Color.red;
        return false;
    }



}
```

```
private IEnumerator flash_red(GameObject obj)
{
    obj.GetComponentInChildren<Image>().color = Color.red;
    yield return new WaitForSeconds(0.15f);
    obj.GetComponentInChildren<Image>().color = Color.white;
    //get text to flash sayibg what went wrong as well

}
IEnumerator error_text(string text)
{
    ErrorText.GetComponent<Text>().text = text;
    yield return new WaitForSeconds(0.9f);
    ErrorText.GetComponent<Text>().text = "";
}
```

One of the errors I found during testing of this method was that once you had pressed to build the tower there was no way of cancelling the build and refunding the money. So to fix this, I added a cancel build button.

4.2.9 Shooting towers

The next problem to solve was the shooting of the towers at the enemy.

The way I handled this was to first find the nearest enemy to the tower, then separately take damage away from the soldier and just visually fire a cannonball towards the soldier. The reason I approached it in this way was so that the cannonball didn't need to be made into a physics object with a collider and there was no need to detect wether the cannonball had actually hit the soldier or not as the castles should have 100% accuracy at shooting anyway.

To find the closest gameObject I used the following algorithm:

1. Add all enemy soldier within the shooting radius to a list;
2. At the beginning of every shoot cycle, iterate through the list to find the closest enemy and return the gameObject of that enemy;
3. Fire the cannonball towards the enemy;
4. Subtract the cannonball damage away from the closest enemy soldier;
5. Add a wait time for the cannonball to recharge;
6. Repeat

However after testing this algorithm, certain issues arose:

- Enemy soldiers were never removed from the list when they either died, or ran away from the fight;
- The cannonballs were never destroyed once they were fired and just pilled up on top of the enemy;
- The friendly towers would sometimes shoot at each other;

The first error was fixed by adding an OnTriggerExit function to find when enemies were no longer within the shooting radius. I could have also solved this problem by finding all enemies

near a tower every time it needed to shoot and just clearing the list at the end of every shoot cycle, however when testing with a lot of troops nearby this slowed down the game a fair amount and made it more computationally difficult, particularly on a computer with lower specs.

The second error was fixed by just destroying the cannonball gameObject once it reached the enemy's position

The third error was fixed by adding an allegiance check to all the gameObjects found nearby.

The actual code for this is:

```
private IEnumerator fireCannon(float delayTime, GameObject cannonClone)
{
    yield return new WaitForSeconds(delayTime); // start at time X
    float startTime = Time.time; // Time.time contains current frame time, so remember starting point
    while (Time.time - startTime <= 1)
    { // until one second passed

        if (!closestObject)
        {
            Destroy(cannonClone);
        }
        cannonClone.transform.position = Vector3.Lerp(new Vector3(transform.position.x, transform.position.y, transform.position.z + 10), closestObject.transform.position, Time.time - startTime);


        yield return 1; // wait for next frame

    }
    if (closestObject.GetComponent<Soldier>())
    {
        //if its a soldier

        closestObject.GetComponent<Soldier>().health -= cannonballDamage; // reduces health of opposition
        if(closestObject.GetComponent<Soldier>().health <= 0)
        {
            nearby.Remove(closestObject);
            nearby.RemoveAll(x => x == null);
        }

    }
    else if (closestObject.GetComponent<Miner>())
    {
        //if its a miner

        closestObject.GetComponent<Miner>().health -= cannonballDamage; // reduces health of opposition
        if (closestObject.GetComponent<Miner>().health <= 0)
        {
            nearby.Remove(closestObject);
            nearby.RemoveAll(x => x == null);
        }

    }
    Destroy(cannonClone);


}
```

```
private GameObject findNearestEnemy()
{

    var closestDistance = 1000000f;


    foreach (GameObject obj in nearby)
    {

        if (obj)
        {

            if (Vector3.Distance(transform.position, obj.transform.position) < closestDistance)
            {

                closestDistance = Vector3.Distance(transform.position, obj.transform.position);
                closestObject = obj;
                //need to remove the closest object the righht one in the list ->first element turning to null rather than being removed from the list
            }
        }
        else
        {

            closestDistance = 1000000f;
            closestObject = null;

        }

    }
    if (allegience == 'R')
    {
        if (closestObject)
        {
            playerRed.GetComponent<Player_red>().BroadcastMessage("defend_towers", closestObject);
        }
        else
        {
            if (health < maxHealth)
            {
                StartCoroutine(playerRed.GetComponent<Player_red>().heal_towers(gameObject));
            }
        }


    }
```

```
private void shoot()
{
    if (shouldShoot)
    {
        closestObject = findNearestEnemy();
        //Need to fire the cannonball here:
        if (closestObject)
        {
            GameObject cannonClone = Instantiate(cannonBall);

            StartCoroutine(fireCannon(0f, cannonClone));
        }

    }
```

The next problem to solve is to get the soldiers to attack the enemy towers. As there is already an attacking method on the soldiers, so to the attacking script a check of wether the soldier was near an attackable tower was added as so:

```
else if (other.gameObject.GetComponentInParent<BlueTowerScript>())
{
    if (gameObject.GetComponent<Soldier>())
    {
        StartCoroutine(gameObject.GetComponent<Soldier>().attack(other.GetComponentInParent<BlueTowerScript>().gameObject));
        gameObject.GetComponent<Soldier>().checkIfCloseToTower = true;
    }
    else if (gameObject.GetComponent<Miner>())
    {
        StartCoroutine(gameObject.GetComponent<Miner>().heal_tower(other.GetComponentInParent<BlueTowerScript>().gameObject));

    }
}
```

I also wanted to get it so that the miners could also be sent to heal their own towers which can also be seen as part of the above script.

The castles should also be able to shoot opposition towers, this was easily done by adding the tower component to the castles.

4.2.10 Third Review

What has been done?

The building and placement of troops in a valid location, the shooting of the towers at enemy units and the attacking of the enemy towers by friendly units, the repairing of towers by friendly miners.

How has it been tested?

The placement of troops was tested by trying to place troops in odd, invalid locations on the map, to check there was no place where a tower could be spawned that wasn't meant to happen. The shooting of cannonballs from the tower was tested by adding a plethora of different friendly and enemy units within the shooting radius and checking who the tower chose to shoot at and also checking that the tower found a new target once its old target was dead. The attacking of a tower from a troop was tested by moving troops near a tower and checking the towers health only went down when the troop was near the tower and it was also tested that the tower was stronger than a troop and would kill it before the soldier destroyed the tower. The repairing of towers by miners was tested by first damaging a tower then moving a miner nearby and checking only if the miner was close would the health of the tower go up.

How does it meet the user requirements and success criteria?

User requirements: 2.3 (Build menu), 2.6 (Building repairs)

4.3 The AI

As the game is single player, the user needs to play against an AI. Given the short time frame for this project the AI had to follow a fairly simple algorithm which was not the smartest, however with more time and a team of developers the AI could be improved using something more complex like deep learning and neural networks.

However my AI followed the following algorithm:

At the start of the game:

- Spawn in a miner and send him to mine to earn gold
- Spawn a soldier and move them to a defensive position

For the main part of the game:

- Check if any opposition soldiers are near any built towers, soldiers or miners; if they are, send any soldiers to attack them.
- If the AI has enough money, periodically have a chance of training more troops or building a tower.
- If the AI has a troop advantage over the opposition, have a chance of sending AI troops over to the attack some opposition troops or buildings.

At the end of the game:

- If the opposition has no soldiers and no towers left, the AI will send all troops in to attack the enemy castle
- Once the castle for either team is destroyed, one team wins and the game is over.

4.3.1 Fourth Review

What has been done?

The enemy AI has been built which gives the user someone to actually play against.

How has it been tested?

The AI was tested by checking the each case where the AI could take some action. These include: when a unit is being attacked, when a tower is being attacked, when the AI has gold over the price of a troop, when the AI has gold over the price of a tower, when the AI has enough units to attack the user.

How does it meet the user requirements and success criteria?

User requirements: 3.1 (The AI)

4.4 Testing methods

Now that the game has been built and partially tested, the test table created before the building of the game can now be used again to test each feature to see if they are working. The approach to the development was iterative so each feature was built and tested individually so these tests should pass easily.

Part of game	Action to test	Working?
Main Menu	Load up game from start menu	Yes
	Cauldrons passively producing smoke	Yes
	Cauldrons produce increased smoke filling the screen when start game button pressed	Yes
	Open up a settings menu	Yes
	Open the level selection area once button pressed	Yes
	Have some levels locked until earlier ones complete	Yes
	When level selected, that level is loaded and shown on screen	Yes
Level startup	Both AI and User start on same money, with the same buildings	Yes
	Mine position and castle position found	Yes
	Build menu displayed on screen	Yes
	User gold displayed on screen	Yes
	AI algorithm scanned walkable ground	Yes
Castle	Castle can be clicked to open build menu	Yes
	Soldiers can be trained by clicking on them	Yes
	Miners can be trained by clicking on them	Yes
	Training troops takes away gold from gold count	Yes
	Visual gold count updated when changed	Yes
	Waiting time for training troops	Yes
	Castle can shoot at enemies approaching	Yes

Miner	Can be set on mining cycle	Yes
	Will pick up gold from the mine	Yes
	Will deposit gold in the castle	Yes
	Visual gold count changes	Yes
	Can move to specified location	Yes
	Can run away when being attacked	Yes
	Changes colour based on allegience	Yes
Soldier	Can move to any specified location	Yes
	Can attack other soldier	Yes
	Can attack other miners	Yes
	Can attack towers and castes	Yes
	Can run away from fights	Yes
	Will stop attacking enemies when they are out of range	Yes
Towers	Can be built using a build menu at a gold cost	Yes
	Can shoot at enemy units	Yes
	Will not shoot at friendly units	Yes
	Can not be built on enemy side of the map	Yes
	Cannot be built on top of existing buildings	Yes
AI	AI can spawn in troops	Yes
	AI can spawn in towers	Yes
	AI can move units	Yes
	AI can attack opposition units	Yes
	AI can defend incoming attacks by moving units	Yes

There were also some extreme case testing that needed to be done on the game, to try to check how the game would respond if the user did something unexpected. These included:

- Check what happens if a user tries to move a troop of screen
- Check what happens if the user doesn't have enough money to purchase a troop or building
- Check what happens if a user tries to place a tower in an invalid place
- Check what happens if a user tries to enter a level that is not yet unlocked.

Now that the main game has been built and tested, it can be passed onto the stakeholders so they can test it and see if it meets their requirements and if there are any missed bugs.

4.5 Stakeholder testing

After giving a copy of the game to the stakeholders I will then ask them these questions:

- How easy was it to navigate through the windows?
- Once playing the game, how easy was it to work out what you needed to do?
- How difficult was the AI to beat?
- Did you use the settings to change anything?
- Would you play the game again?

How easy was it to navigate through the windows?

James: The menu screens were fairly self explanatory and I was able to navigate to the main game fairly easily.

Trudy: I was able to navigate to the level select fairly easily, however at first I was unsure as to why some of the levels were locked and how I would unlock them as it was not made clear.

Once playing the game, how easy was it to work out what you needed to do?

James: As I have played many similar games before it was fairly intuitive to me and I was able to work out what I was meant to do easily.

Trudy: Buying troops was easy, however it was sometimes annoying working out why I couldn't place towers in certain areas and also the soldiers would not always attack the intended target or get stuck fighting something else.

How difficult was the AI to beat?

James: The AI was quite easy to beat as it would never have as many troops as me and wouldn't place towers in the most strategic positions.

Trudy: Even though I have not played many of these types of game before, the AI was still moderately easy and I could work out how it would react to certain actions, so I could draw its troops away with one of mine, then send in the rest of my troops to take out their towers and castle.

Did you use the settings to change anything?

James: No I never even entered the settings menu as I didn't think there would be any controls that really needed changing.

Trudy: I had a look at the settings menu, but the options to change weren't particularly good so I never changed anything.

Would you play the game again?

James: I enjoyed playing the game for the first time, but it would need more levels with a harder AI for me to play it again.

Trudy: I would definitely play it again if more levels were added with more features unlocked in later levels.

What I learnt from these interviews

From these interviews, I learnt that the general concept for the game was good however it needs some refinements for instance a better tutorial to properly explain the game and also a better AI with more levels with an increasing difficulty in the AI. All these things I would've liked to have included in the original iteration of the game, however due to time constraints not helped by the pandemic I was unable to built these before the deadline.

Part IV

Evaluation

4.6 Success criteria

Success Criteria	Met?
Working buttons	Yes
Intuitive colour scheme	Yes
Working sound toggle	Yes
Working levels	Yes
New levels openeing when old ones complete	Yes
Tutorial page	**Partially**
Efficient working pathfinding algorithm	Yes
A working AI	Yes
An AI that increases in intelligence	**No**
Users able to build towers for a cost	Yes
Users able to train miners	Yes
Users able to set miners to collect gold	Yes
Users able to train soldiers	Yes
Have health bars on units	**No**
Buildings can be repaired by miners	Yes
Have a way of saving the levels already completed	**No**

Success criteria not met

For the success criteria not met, this was mostly extra features that could not originally be implemented due to time pressure, however given more time and a team of developers all the extra features could easily be added as the legacy code has been built in order to support these features.

4.7 Evidence of success criteria

Figure 4.1: Buttons between menu screens working and the medieval font and colour scheme

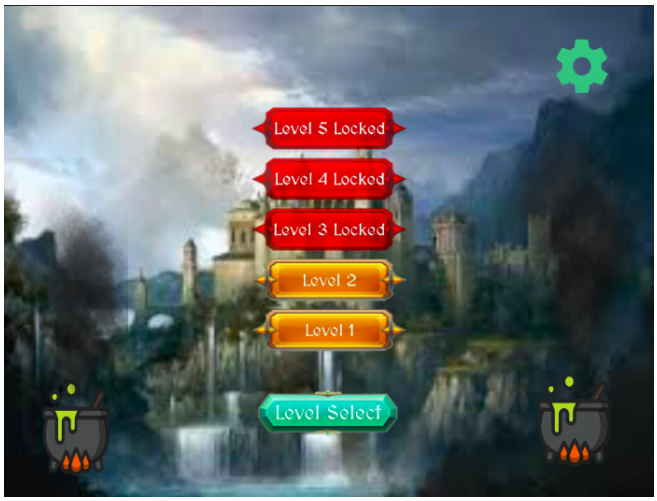


Figure 4.2: Buttons between menu screens working and the medieval font and colour scheme. Also shows levels being unlocked and locked with only the unlocked ones being clickable

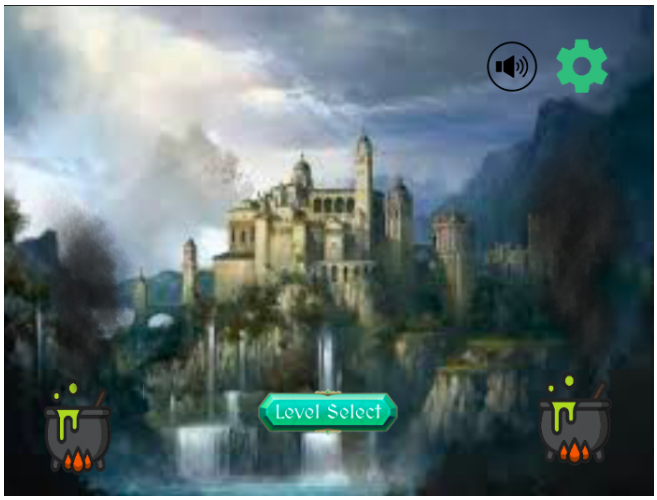


Figure 4.3: Working sound toggle

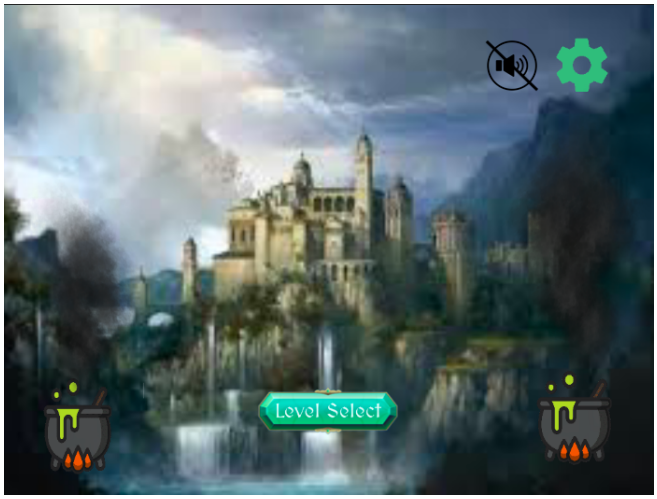


Figure 4.4: Working sound toggle

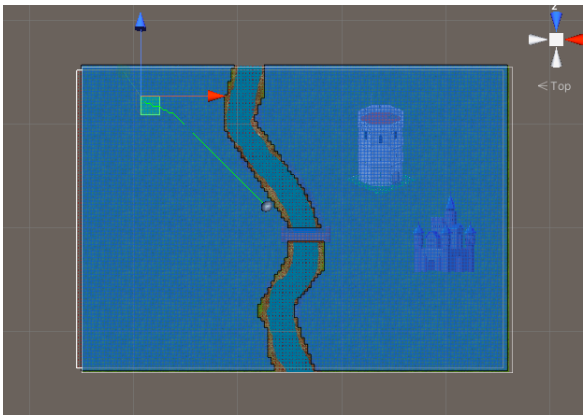


Figure 4.5: Working pathfinding algorithm

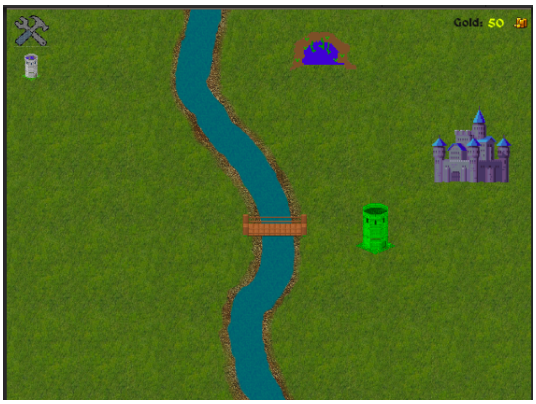


Figure 4.6: Users able to build towers

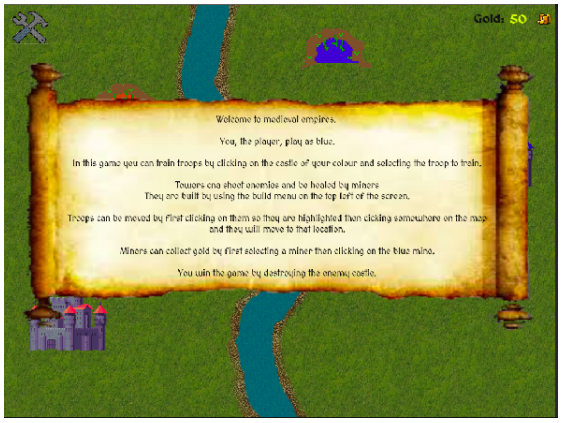


Figure 4.7: Tutorial page

Figure 4.8: Users able to train miners and soldiers

4.8 Limitations

The biggest limitation for this game is the AI. An in depth, improving AI is hard to implement even with many frameworks like TensorFlow the time and know-how needed taken to develop a

more complex AI that was not predictable.

Another limitation is the lack of variety of troops and towers that are included in the game. Ideally in later levels there would be different types of troops that could be trained, for example archers which could shoot at range and also different sorts of towers. This could easily be implemented in future versions of the game with some further development and new classes based of the parent character and building classes.

A further limitation is the inability to upgrade troops, this could have been implemented by just adding a button and adjusting the attack damage and health stats, however it would be more difficult to get an AI to upgrade the right troops at the right time.

Health bars on all units would also have been useful as currently the user has no idea what health each troop and tower is on and how close they are to dying.

Another limitation is the inability to select and move multiple troops at the same time. Trying to move troops in a coordinated attack could get frustrating and difficult without this feature as each troop has to be moved individually.

4.9 Maintenance

Updates in future games should improve the AI system, and also add more levels with more features such as different types of troops and the ability to upgrade troops.

For upkeep of the code and to improve the code base, a better way detecting nearby enemies and firing at them would be useful as this feature is the least modular part of the program and each time a new troop is added another conditional check needs to be added to the character class to make sure it identifies it.

If the stakeholders wish to add other features that they recommend this could easily be implemented due to the general modularity of the program (other than the afore mentioned issue). A way of collecting this information and any uncovered bugs would be to create a forum page online.

The code is mostly commented so that a team of new developers should hopefully be able to understand what each part of the code is doing and new features would also need to given more documentation.

Part V

Appendix

Chapter 5

Code appendix

5.1 BuildMenu.cs

build_menu ▸ FixedUpdate()

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.PackageManager;
using UnityEngine;
using UnityEngine.UI;


public class build_menu : MonoBehaviour
{
    /* In this i need to:
     * Get tower to follow mouse around once tower pressed at right size,
     * add cube to tower for health and other things to tower script,
     * Make sure tower can actually be placed where the user chooses,
     * make sure if there is a character underneath the placement of the tower they do not get trapped there
     * give meaningful messages to user if building cant be placed somewhere.
     */

    private Button[] all_buttons;

    public GameObject Canvas;

    public GameObject Screen_canvas;

    private bool is_showing = false;

    private bool placed = true;

    public GameObject tower;

    private GameObject placed_tower;

    private Vector3 position_to_move;

    private Camera PlayView1;

    public int tower_cost = 50;

    public GameObject ErrorText;

    private GameObject towerButton;

    private GameObject cancelCross;

    private bool cancelled = false;


    // Start is called before the first frame update
    void Start()
    {
        all_buttons = gameObject.GetComponentsInChildren<Button>();

        all_buttons[0].onClick.AddListener(toggle_showing);
        all_buttons[1].onClick.AddListener(build_tower);

        Screen_canvas = GameObject.FindGameObjectWithTag("Screen_canvas");

        PlayView1 = GameObject.FindGameObjectWithTag("Camera1").GetComponent<Camera>();

        ErrorText = GameObject.Find("ErrorText");

        towerButton = GameObject.Find("BlueTower");
```

build_menu ▸ FixedUpdate()

```
        cancelCross = GameObject.FindGameObjectWithTag("Cancel");
        cancelCross.SetActive(false);

        towerButton.GetComponentInChildren<Text>().text = tower_cost.ToString();

        Canvas = gameObject.GetComponentInChildren<Canvas>().gameObject;
        Canvas.SetActive(false);



    }

    // Update is called once per frame
    void FixedUpdate()
    {

        if (!placed) //Running in update as a while loop doesnt work properly in unity
        {

            Vector3 mousePos = Input.mousePosition;
            position_to_move = PlayView1.ScreenToWorldPoint(new Vector3(mousePos.x, mousePos.y, 720));
            placed_tower.transform.position = position_to_move;

            bool check = check_place();

            if (Input.GetMouseButtonDown(0)) //if the mouse has been clicked
            {
                if (check)
                {
                    placed_tower.GetComponent<Image>().color = Color.white;
                    Screen_canvas.GetComponentInChildren<Player_blue>().BroadcastMessage("update_gold", -tower_cost);
                    placed = true;
                    placed_tower.GetComponent<BlueTowerScript>().shouldShoot = true;
                    towerButton.SetActive(true);
                    cancelCross.SetActive(false);
                }

                else if (cancelled)
                {
                    cancel_build();
                }

                else
                {
                    StartCoroutine(error_text("Invalid Placement"));
                }
            }

        }

    }

    void toggle_showing()
    {
        if (is_showing)
        {
            Canvas.SetActive(false);
            is_showing = false;
        }
        else
        {
```

build_menu ▸ FixedUpdate()

```
            Canvas.SetActive(true);
            is_showing = true;
        }

    }

    void build_tower()
    {
        if(Screen_canvas.GetComponentInChildren<Player_blue>().player_gold >= tower_cost)
        {
            placed_tower = Instantiate(tower, new Vector3(0, 0, 0), Quaternion.identity);
            placed_tower.transform.SetParent(Screen_canvas.transform, false);
            //placed_tower.transform.position = new Vector3(placed_tower.transform.position.x, 0, placed_tower.transform.position.z)
            placed = false;
            cancel_placement_button();
        }
        else
        {
            StartCoroutine(flash_red(Canvas.gameObject));
            StartCoroutine(error_text("Not enough gold"));
        }


    }
    bool check_place()
    {
        //This checks wether the position that has been selected can actually be placed there or not.
        RaycastHit hit;
        Physics.Raycast(new Vector3(placed_tower.transform.position.x, placed_tower.transform.position.y + 50, placed_tower.transfor
        if (hit.transform != null)
        {
            if (hit.transform.gameObject.layer == 11)
            {
                //A valid placement - place gameobject here
                placed_tower.GetComponent<Image>().color = Color.green;
                return true;
            }
            else if (hit.transform.gameObject == cancelCross)
            {
                placed_tower.GetComponent<Image>().color = new Color(1, 1, 1, 0.6f);
                cancelled = true;
                return false;
            }
            else
            {
                //Not a valid placement
                placed_tower.GetComponent<Image>().color = Color.red;
                cancelled = false;
                return false;

            }
        }
        else
        {
            //Not tried to place in the screen;
            placed_tower.GetComponent<Image>().color = Color.red;
            cancelled = false;
            return false;
        }
```

build_menu ▸ FixedUpdate()

```
184
185
186
187         }
188
189         public IEnumerator flash_red(GameObject obj)
190         {
191             obj.GetComponentInChildren<Image>().color = Color.red;
192             yield return new WaitForSeconds(0.15f);
193             obj.GetComponentInChildren<Image>().color = Color.white;
194             //get text to flash sayibg what went wrong as well
195
196         }
197         public IEnumerator error_text(string text)
198         {
199             ErrorText.GetComponent<Text>().text = text;
200             yield return new WaitForSeconds(0.9f);
201             ErrorText.GetComponent<Text>().text = "";
202         }
203
204         void cancel_placement_button()
205         {
206             towerButton.SetActive(false);
207             cancelCross.SetActive(true);
208         }
209
210         void cancel_build()
211         {
212             Destroy(placed_tower);
213             towerButton.SetActive(true);
214             cancelCross.SetActive(false);
215             placed = true;
216         }
217     }
218
```

5.2 button_controll.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class button_controll : MonoBehaviour
{
    private bool is_showing = false;
    public GameObject Canvas;
    private Button[] all_buttons;
    private Image loading_shadow_knight;
    private Image loading_shadow_miner;
    public GameObject soldier;
    public GameObject miners;
    public int soldierCost = 25;
    public int minerCost = 20;
    private GameObject buildMenu;
    public List<GameObject> BlueKnights = new List<GameObject>();
    public List<GameObject> BlueMiners = new List<GameObject>();

    // Start is called before the first frame update
    void Start()
    {

        all_buttons = gameObject.GetComponentsInChildren<Button>();

        //Adding listeners to all the buttons
        all_buttons[0].onClick.AddListener(toggle_onOff);
        all_buttons[1].onClick.AddListener(train_knight);
        all_buttons[2].onClick.AddListener(train_miner);



        //Knght shadow
        Image[] loading_shadow_knight_list = all_buttons[1].gameObject.GetComponentsInChildren<Image>();
        loading_shadow_knight = loading_shadow_knight_list[1];
        loading_shadow_knight.fillAmount = 0;


        //Miner shadow
        Image[] loading_shadow_miner_list = all_buttons[2].gameObject.GetComponentsInChildren<Image>();
        loading_shadow_miner = loading_shadow_miner_list[1];
        loading_shadow_miner.fillAmount = 0;



        Canvas = gameObject.GetComponentInChildren<Canvas>().gameObject;
        Canvas.SetActive(false);

        buildMenu = GameObject.Find("BuildMenu");
    }

    // Update is called once per frame
    void Update()
    {

    }

    void toggle_onOff()
    {
        if (is_showing)
        {
```

No selection

```
58
59          void toggle_onOff()
60          {
61              if (is_showing)
62              {
63                  Canvas.SetActive(false);
64                  is_showing = false;
65              }
66              else
67              {
68                  Canvas.SetActive(true);
69                  is_showing = true;
70              }
71          }
72
73          void train_knight()
74          {
75              //This is a method to sort out the visuals on the knight button and then spawn in a knight after a wait
76              //using a coroutine IEnumerator to do wait times easier and also makes it modular so that both miner and knight images can u
77              if (gameObject.GetComponent<Player_blue>().player_gold >= soldierCost)
78              {
79                  if (loading_shadow_knight.fillAmount <= 0)
80                  {
81                      loading_shadow_knight.fillAmount = 1;
82                      StartCoroutine(undo_fill(loading_shadow_knight));
83                      gameObject.GetComponent<Player_blue>().BroadcastMessage("update_gold", -soldierCost);
84                  }
85              }
86              else
87              {
88                  StartCoroutine(buildMenu.GetComponent<build_menu>().flash_red(all_buttons[1].gameObject));
89                  StartCoroutine(buildMenu.GetComponent<build_menu>().error_text("Not enough gold"));
90              }
91
92
93
94          }
95
96          void train_miner()
97          {
98              //This works the same was as the train_knight() method
99              if (gameObject.GetComponent<Player_blue>().player_gold >= soldierCost)
100             {
101                 if (loading_shadow_miner.fillAmount <= 0)
102                 {
103                     loading_shadow_miner.fillAmount = 1;
104                     StartCoroutine(undo_fill(loading_shadow_miner));
105                     gameObject.GetComponent<Player_blue>().BroadcastMessage("update_gold", -minerCost);
106                 }
107             }
108             else
109             {
110                 StartCoroutine(buildMenu.GetComponent<build_menu>().flash_red(all_buttons[2].gameObject));
111                 StartCoroutine(buildMenu.GetComponent<build_menu>().error_text("Not enough gold"));
112             }
113
114         }
115
116         IEnumerator undo_fill(Image image)
117         {
118
119             while (image.fillAmount > 0)
```

No selection

```
    IEnumerator undo_fill(Image image)
    {

        while (image.fillAmount > 0)
        {
            yield return new WaitForSeconds(0.05f);
            image.fillAmount -= 0.01f;
        }

        if (image == loading_shadow_knight)
        {
            GameObject spawned = Instantiate(soldier, transform.position + new Vector3(-70, 0, -150), Quaternion.identity);
            spawned.GetComponent<Soldier>().allegience = 'B'; //Need to make sure the allegience is set after spawning in the troop
            BlueKnights.Add(spawned);
        }

        else if (image == loading_shadow_miner)
        {
            GameObject spawned = Instantiate(miners, transform.position + new Vector3(-90, 0, -180), Quaternion.identity);
            spawned.GetComponent<Miner>().allegience = 'B';
            BlueMiners.Add(spawned);



        }

    }
}
```

5.3 Character_controll.cs

character_controll ▸ No selection

```
using System.Collections;
using System.Collections.Generic;
using Pathfinding;
using UnityEngine;

public class character_controll : MonoBehaviour
{

    public char allegience;
    public int health = 100;
    public AILerp move_script;
    public Camera PlayView1;
    //private int layer_mask;
    public GameObject selected;
    public bool active_selection = false;
    public GameObject Target;
    public GameObject shadow;
    private Color alphaColor = new Color(1, 1, 1, 0);
    public bool isAlive = true;
    public bool fade = false;
    private float lower = 0f;
    private bool destroyed_called = false;
    public float maxDistance = 5;
    public bool stationary = false;
    public bool mine_it;
    public Vector3 mine_position;
    private GameObject homeTower;



    // Start is called before the first frame update
    virtual public void Start()
    {

        move_script.canSearch = false;
        move_script.canMove = false;

        shadow = transform.GetChild(0).gameObject;
        shadow.SetActive(false);

        PlayView1 = GameObject.FindGameObjectWithTag("Camera1").GetComponentInChildren<Camera>();

        if (allegience == 'B')
        {
            mine_position = GameObject.FindGameObjectWithTag("BlueMine").gameObject.GetComponentInChildren<Cube_miner>().transform.position + new Vector3(0, 0, -45);
            homeTower = GameObject.Find("BlueCastle");
        }
        else
        {
            mine_position = GameObject.FindGameObjectWithTag("RedMine").gameObject.GetComponentInChildren<Cube_miner>().transform.position + new Vector3(0, 0, -45);
            homeTower = GameObject.Find("RedCastle");
        }


    }

    // Update is called once per frame
    virtual public void FixedUpdate()
    {
        //This is to check if the mouse button has clicked a game object, and wether the clicked game object wants to be moved.
        if (Input.GetMouseButton(0))
```

character_controll ▸ No selection

```
    // Update is called once per frame
    virtual public void FixedUpdate()
    {
        //This is to check if the mouse button has clicked a game object, and wether the clicked game object wants to be moved.
        if (Input.GetMouseButton(0))
        {
            Ray ray = PlayView1.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;

            // Casts the ray and get the first game object hit
            Physics.Raycast(ray, out hit, 10000.0f, 1 << LayerMask.NameToLayer("GROUND/obstacle") | 1 << LayerMask.NameToLayer("GROUND/ground") | 1 << LayerMask.NameToLayer("Characters"));



            if (hit.collider)
            {


                if (hit.collider.gameObject == gameObject)
                {
                    if (hit.collider.gameObject.GetComponent<Soldier>())
                    {
                        if (hit.collider.gameObject.GetComponent<Soldier>().allegience == 'B')
                        {
                            shadow.SetActive(true);
                            selected = hit.collider.gameObject;
                            active_selection = true;
                        }

                    }
                    else if (hit.collider.gameObject.GetComponent<Miner>())
                    {
                        if (hit.collider.gameObject.GetComponent<Miner>().allegience == 'B')
                        {
                            shadow.SetActive(true);
                            selected = hit.collider.gameObject;
                            active_selection = true;
                        }
                    }
                        //Stores the game object hit and activates the flag marking that a valid character has been selected




                }
                else if (active_selection)
                {

                    if((hit.collider.gameObject != PlayView1.gameObject.GetComponentInChildren<Mining>().gameObject) && (hit.collider.gameObject != PlayView1.gameObject.GetComponentInChildren<Mining>().gameObject.GetComponentInChildren<Cube_miner>().gameObject))
                    {

                        shadow.SetActive(false);
                        active_selection = false;
                        mine_it = false;
                        //here a valid target has previously been selected so then we want to call the movement function to move the target for the A* script and make the character move

                        Movement(selected, hit.point);
                    }
                    else
                    {
                        mine_it = true;
                        active_selection = false;
                        shadow.SetActive(false);
                        //here a valid target has previously been selected so then we want to call the movement function to move the target for the A* script and make the character move
                        Movement(selected, mine_position);
                    }


                }
                else
```

character_controll ▸ No selection

```
136
137             }
138             if (health <= 0)
139             {
140                 isAlive = false;
141                 fade = true;
142                 if (allegience == 'B')
143                 {
144                     if (gameObject.GetComponent<Soldier>())
145                     {
146                         homeTower.GetComponent<button_controll>().BlueKnights.Remove(gameObject);
147                     }
148                     else
149                     {
150                         homeTower.GetComponent<button_controll>().BlueMiners.Remove(gameObject);
151                     }
152
153                 }
154                 else
155                 {
156                     if (gameObject.GetComponent<Soldier>())
157                     {
158                         homeTower.GetComponent<Player_red>().Soldier_list.Remove(gameObject);
159                     }
160                     else
161                     {
162                         homeTower.GetComponent<Player_red>().Miner_list.Remove(gameObject);
163                     }
164
165                 }
166
167             }
168             if (fade)
169             {
170                 StartCoroutine(fade_destroy());
171
172             }
173
174
175
176
177
178
179         }
180
181         public void destroy()
182         {
183             Destroy(gameObject);
184         }
185
186
187
188         void OnTriggerEnter(Collider other)
189         {
190             // To use this function one set of troops needs to be a trigger and the other side cant be a trigger with the collider -- B:on R:off
191             if (!stationary)
192             {
193
194
195                 if (other.gameObject.tag == gameObject.tag | other.gameObject.tag == "RedCastle" | other.gameObject.tag == "BlueCastle") //Make sure it will only attack and collide with things on the same tag as it
196                 {
```

character_controll ▸ No selection

```
187
188         void OnTriggerEnter(Collider other)
189         {
190             // To use this function one set of troops needs to be a trigger and the other side cant be a trigger with the collider -- B:on R:off
191             if (!stationary)
192             {
193
194
195                 if (other.gameObject.tag == gameObject.tag | other.gameObject.tag == "RedCastle" | other.gameObject.tag == "BlueCastle") //Make sure it will only attack and collide with things on the same tag as it
196                 {
197
198                     if (other.gameObject.GetComponent<Soldier>())
199                     {
200
201                         Soldier scr = other.gameObject.GetComponent<Soldier>();
202                         if (scr.allegience != allegience)
203                         {
204
205                             if (other.gameObject != gameObject) // Work around for random bug when it collides with itself off spawn
206                             {
207
208                                 stop_start(other);
209                                 stationary = true;
210                                 if (gameObject.GetComponent<Soldier>())
211                                 {
212                                     StartCoroutine(transform.GetComponentInChildren<Soldier>().attack(other.gameObject));
213                                 }
214                             }
215                         }
216                     }
217                     else if (other.gameObject.GetComponent<Miner>())
218                     {
219                         Miner scr = other.gameObject.GetComponent<Miner>();
220                         if (!gameObject.GetComponent<Miner>()) {
221
222                             if (scr.allegience != allegience)
223                             {
224
225                                 if (other.gameObject != gameObject) // Work around for random bug when it collides with itself off spawn
226                                 {
227
228
229                                     stop_start(other);
230                                     stationary = true;
231                                     if (gameObject.GetComponent<Soldier>())
232                                     {
233                                         StartCoroutine(transform.GetComponentInChildren<Soldier>().attack(other.gameObject));
234                                     }
235                                 }
236                             }
237                         }
238
239
240
241                     }
242                     else if (other.gameObject.GetComponent<BlueTowerScript>())
243                     {
244                         if (gameObject.GetComponent<Soldier>())
245                         {
246                             StartCoroutine(gameObject.GetComponent<Soldier>().attack(other.gameObject));
247                             gameObject.GetComponent<Soldier>().checkIfCloseToTower = true;
```

character_controll ▸ No selection

```
                }
                else if (other.gameObject.GetComponent<BlueTowerScript>())
                {
                    if (gameObject.GetComponent<Soldier>())
                    {
                        StartCoroutine(gameObject.GetComponent<Soldier>().attack(other.gameObject));
                        gameObject.GetComponent<Soldier>().checkIfCloseToTower = true;
                    }
                    else if (gameObject.GetComponent<Miner>())
                    {
                        StartCoroutine(gameObject.GetComponent<Miner>().heal_tower(other.gameObject));
                    }


                }
                else if (other.gameObject.GetComponentInParent<BlueTowerScript>())
                {
                    if (gameObject.GetComponent<Soldier>())
                    {
                        StartCoroutine(gameObject.GetComponent<Soldier>().attack(other.GetComponentInParent<BlueTowerScript>().gameObject));
                        gameObject.GetComponent<Soldier>().checkIfCloseToTower = true;
                    }
                    else if (gameObject.GetComponent<Miner>())
                    {
                        StartCoroutine(gameObject.GetComponent<Miner>().heal_tower(other.GetComponentInParent<BlueTowerScript>().gameObject));

                    }
                }

                //raycast in all direction

                /*
                 * HOW THIS WORKS:
                 *
                 * If other thing is a soldier then check there not same allegience and check other conditions,
                 * THEN if this is a soldier call the attack class (if its not a soldier do nothing)
                 *
                 * Else if the other thing is a Miner check they are not the same allegience and other conditions (same as above) again,
                 * THEN if this is a soldier call the attack class again
                 *
                 * Cannot combine the two as they are different data types and c# is strongly typed.
                 */

            }

        }
        else
        {
            stationary = false;
        }


    }

    private void OnTriggerExit(Collider other)
    {
        //stop attacking

        if (gameObject.GetComponent<Soldier>())
        {
            gameObject.GetComponent<Soldier>().canAttack = false;
```

character_controll ▸ No selection

```
    private void OnTriggerExit(Collider other)
    {
        //stop attacking

        if (gameObject.GetComponent<Soldier>())
        {
            gameObject.GetComponent<Soldier>().canAttack = false;
            if (other.gameObject.GetComponent<BlueTowerScript>())
            {
                gameObject.GetComponent<Soldier>().checkIfCloseToTower = false;

            }
        }


        stationary = false;
    }

    void stop_start(Collider other)
    {
        Vector3 move_to = new Vector3(0, 0, 0);
        Movement(gameObject, transform.position - move_to);
        character_controll cs = other.gameObject.GetComponentInParent<character_controll>();
        cs.Movement(other.gameObject, transform.position + move_to);

    }
    private IEnumerator fade_destroy()
    {

        yield return new WaitForSeconds(0f);
        lower += 0.033f;
        gameObject.GetComponent<MeshRenderer>().material.color = Color.Lerp(gameObject.GetComponent<MeshRenderer>().material.color, alphaColor, lower);

        if (!destroyed_called)
        {
            Invoke("destroy", 1.5f);
            destroyed_called = true;
        }

    }




    public void Movement(GameObject character, Vector3 location_to_move)
    {
        //This script controls where the user wants to move a certain troop to and runs the A* algorithm to take it there.


        //instantiate at location_to_move location



        GameObject motion = Instantiate(Target, location_to_move, Quaternion.identity);
        AIDestinationSetter A = transform.GetComponent<AIDestinationSetter>();
        A.target = motion.transform;
        move_script.canSearch = true;
        move_script.canMove = true;
        //Destroy(motion);
    }
```

character_controll ▸ No selection

```
    private void OnTriggerExit(Collider other)
    {
        //stop attacking

        if (gameObject.GetComponent<Soldier>())
        {
            gameObject.GetComponent<Soldier>().canAttack = false;
            if (other.gameObject.GetComponent<BlueTowerScript>())
            {
                gameObject.GetComponent<Soldier>().checkIfCloseToTower = false;

            }
        }


        stationary = false;
    }

    void stop_start(Collider other)
    {
        Vector3 move_to = new Vector3(0, 0, 8);
        Movement(gameObject, transform.position - move_to);
        character_controll cs = other.gameObject.GetComponentInParent<character_controll>();
        cs.Movement(other.gameObject, transform.position + move_to);

    }
    private IEnumerator fade_destroy()
    {

        yield return new WaitForSeconds(0f);
        lower += 0.033f;
        gameObject.GetComponent<MeshRenderer>().material.color = Color.Lerp(gameObject.GetComponent<MeshRenderer>().material.color, alphaColor, lower);

        if (!destroyed_called)
        {
            Invoke("destroy", 1.5f);
            destroyed_called = true;
        }

    }




    public void Movement(GameObject character, Vector3 location_to_move)
    {
        //This script controls where the user wants to move a certain troop to and runs the A* algorithm to take it there.


        //instantiate at location_to_move location



        GameObject motion = Instantiate(Target, location_to_move, Quaternion.identity);
        AIDestinationSetter A = transform.GetComponent<AIDestinationSetter>();
        A.target = motion.transform;
        move_script.canSearch = true;
        move_script.canMove = true;
        //Destroy(motion);
    }
```

character_controll ▸ No selection

```
        }

    }




    public void Movement(GameObject character, Vector3 location_to_move)
    {
        //This script controls where the user wants to move a certain troop to and runs the A* algorithm to take it there.


        //instantiate at location_to_move location



        GameObject motion = Instantiate(Target, location_to_move, Quaternion.identity);
        AIDestinationSetter A = transform.GetComponent<AIDestinationSetter>();
        A.target = motion.transform;
        move_script.canSearch = true;
        move_script.canMove = true;
        //Destroy(motion);
    }
}
```

5.4 Change_screen.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Change_screen : MonoBehaviour
{

    public GameObject title_screen;
    public GameObject menu_screen;
    public GameObject smoke3;
    public GameObject[] childs;


    // Start is called before the first frame update
    void Start()
    {
        title_screen = GameObject.FindGameObjectWithTag("Title_screen") as GameObject;
        menu_screen = GameObject.FindGameObjectWithTag("Menu_screen") as GameObject;

    }

    // Update is called once per frame
    void Update()
    {

    }

    public IEnumerator stop()
    {



        int children = transform.childCount;

        GameObject[] childs = new GameObject[children];

        for (int i = 0; i < children; i++)
        {
            if (i > 0)
            {
                childs[i] = transform.GetChild(i).gameObject;

            }
        }


        smoke3.SetActive(true);

        var start = smoke3.GetComponent<ParticleSystem>();
        var start1 = start.emission;
        start1.rateOverTime = 150f;

        yield return new WaitForSeconds(2.55f);

        for (int e = 1; e < children; e++)
        {
            childs[e].SetActive(false);
        }

        yield return new WaitForSeconds(2);
        smoke3.SetActive(false);
```


No selection
{
}
public IEnumerator stop()
{
int children = transform.childCount;
GameObject[] childs = new GameObject[children];
for (int i = 0; i < children; i++)
{
if (i > 0)
{
childs[i] = transform.GetChild(i).gameObject;
}
}
smoke3.SetActive(true);
var start = smoke3.GetComponent<ParticleSystem>();
var start1 = start.emission;
start1.rateOverTime = 150f;
yield return new WaitForSeconds(2.55f);
for (int e = 1; e < children; e++)
{
childs[e].SetActive(false);
}
yield return new WaitForSeconds(2);
smoke3.SetActive(false);
}
}

5.5 Mining.cs

No selection

```
    {

    }

    public IEnumerator stop()
    {



        int children = transform.childCount;

        GameObject[] childs = new GameObject[children];

        for (int i = 0; i < children; i++)
        {
            if (i > 0)
            {
                childs[i] = transform.GetChild(i).gameObject;

            }
        }


        smoke3.SetActive(true);

        var start = smoke3.GetComponent<ParticleSystem>();
        var start1 = start.emission;
        start1.rateOverTime = 150f;

        yield return new WaitForSeconds(2.55f);

        for (int e = 1; e < children; e++)
        {
            childs[e].SetActive(false);
        }

        yield return new WaitForSeconds(2);
        smoke3.SetActive(false);

    }
}

```

5.6 Soldier.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Mining : MonoBehaviour
{
    public int gold_per_second;

    // Start is called before the first frame update
    void Start()
    {
        gold_per_second = 4;
    }

    // Update is called once per frame
    void Update()
    {

    }

}

```

Soldier ▸ attack(GameObject other)

```
using System.Collections;
using System.Collections.Generic;
using Pathfinding;
using UnityEngine;

public class Soldier : character_controll
{
    public int attack_damage;
    public int upgrade_level;
    public Material matR;
    public Material matB;
    private char allegience1;
    public bool canAttack;
    private bool closeToTower = false;
    public bool checkIfCloseToTower = false; //add check to update....
    private GameObject others;
    private bool notAlreadyRunning = true;

    // Start is called before the first frame update
    new void Start()
    {
        base.Start();
        attack_damage = 20;
        upgrade_level = 0;


    }

    new void FixedUpdate()
    {
        base.FixedUpdate();
        // Update is called once per frame
        if (allegience1 == allegience)
        {

        }
        else
        {
            allegience1 = allegience;
            change_allegience();

        }
        if (checkIfCloseToTower)
        {
            try
            {
                if (Vector3.Distance(transform.position, others.transform.position) < 80)
                {
                    closeToTower = true;

                }
                else
                {
                    closeToTower = false;

                }
            }
            catch
            {

            }
```


Soldier
attack(GameObject other)
}
if (closeToTower && others.GetComponent<BlueTowerScript>())
{
StartCoroutine(attack_tower());
}
}
void attack_upgrade()
{
//Script to hande the attack stats upgrades of the soldiers.
if (upgrade_level == 0)
{
attack_damage = 5;
}
else if (upgrade_level == 1)
{
attack_damage = 10;
}
else if (upgrade_level == 2)
{
attack_damage = 15;
}
}
void change_allegience()
{
// Changes whether trigger is active or not for each different allegience
if (allegience == 'R')
{
gameObject.GetComponent<CapsuleCollider>().isTrigger = false;
gameObject.GetComponent<MeshRenderer>().material = matR;
}
else if (allegience == 'B')
{
gameObject.GetComponent<CapsuleCollider>().isTrigger = true;
gameObject.GetComponent<MeshRenderer>().material = matB;
}
}
public IEnumerator attack(GameObject other)
{
//the actual attacking between soldiers
canAttack = true;
others = other.gameObject;
if (other.gameObject.GetComponent<Soldier>()) //Checks if it is attacking a soldier


Soldier ▸ attack(GameObject other)
if (other.gameObject.GetComponent<Soldier>()) //Checks if it is attacking a soldier
{
Soldier opp = other.gameObject.GetComponent<Soldier>();
while (opp.health > 0 && isAlive && opp.isAlive && health > 0 && canAttack)
{
yield return new WaitForSeconds(0.4f);
opp.health -= attack_damage;
yield return new WaitForSeconds(0.4f);
}
yield return new WaitForSeconds(0f);
}
else if (other.gameObject.GetComponent<Miner>())
{
print(canAttack);
Miner opp = other.gameObject.GetComponent<Miner>();
while (opp.health > 0 && isAlive && opp.isAlive && health > 0 && canAttack)
{
yield return new WaitForSeconds(0.4f);
opp.health -= attack_damage;
yield return new WaitForSeconds(0.4f);
}
yield return new WaitForSeconds(0f);
}
}
private IEnumerator attack_tower()
{
BlueTowerScript opp = others.gameObject.GetComponent<BlueTowerScript>();
if (notAlreadyRunning)
{
notAlreadyRunning = false;
while (opp.health > 0 && isAlive && opp.isAlive && health > 0 && canAttack)
{
yield return new WaitForSeconds(1.3f);
opp.health -= attack_damage;
yield return new WaitForSeconds(2.5f);
}
notAlreadyRunning = true;
}
}

5.7 Start_Game.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Events;
using System.Security.Cryptography.X509Certificates;
using System.Threading;
using System.Runtime.InteropServices;
using System.Runtime.InteropServices.ComTypes;
using System;



public class Start_Game : MonoBehaviour
{

    public Button yourButton;
    public GameObject smoke;
    public GameObject title_screen;
    public GameObject menu_screen;
    public GameObject smoke1;
    public Change_screen change;
    public Camera PlayView1;


    void Start()
    {
        Button yourButton = gameObject.GetComponent<Button>();
        yourButton.onClick.AddListener(change_particles);

        title_screen = GameObject.FindGameObjectWithTag("Title_screen") as GameObject;
        menu_screen = GameObject.FindGameObjectWithTag("Menu_screen") as GameObject;

        GameObject smoke = GameObject.Find("smoke (1)");
        GameObject smoke1 = GameObject.Find("smoke (2)");



        PlayView1 = GameObject.FindGameObjectWithTag("Camera1").GetComponent<Camera>();
        PlayView1.gameObject.SetActive(false);
    }


    public void change_particles()
    {

        ParticleSystem part = smoke.GetComponent<ParticleSystem>();

        var emission = part.emission;
        emission.rateOverTime = 15.0f;

        var sim_speed = part.main;
        sim_speed.simulationSpeed = 1f;
        sim_speed.startLifetime = 6.8f;
        sim_speed.startSize = 40f;
        sim_speed.gravityModifier = -3f;


        var shape = part.shape;
        shape.arc = 15.0f;

        StartCoroutine(change.stop());
```


No selection
sim_speed.gravityModifier = -3f;
var shape = part.shape;
shape.arc = 15.0f;
StartCoroutine(change.stop());
ParticleSystem part1 = smoke1.GetComponent<ParticleSystem>();
var emission1 = part1.emission;
emission1.rateOverTime = 15.0f;
var sim_speed1 = part1.main;
sim_speed1.simulationSpeed = 1f;
sim_speed1.startLifetime = 6.8f;
sim_speed1.startSize = 40f;
sim_speed1.gravityModifier = -3f;
var shape1 = part1.shape;
shape1.arc = 15.0f;
}
}

5.8 BlueTowerScript.cs

BlueTowerScript ▸ fireCannon(float delayTime, GameObject cannonClone)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BlueTowerScript : MonoBehaviour
{

    private Button towerButton;
    private GameObject shootingRadius;
    private bool isShowing = false;
    private List<GameObject> nearby = new List<GameObject>();
    public int health = 200;
    public int maxHealth = 200;
    public int cannonballDamage = 20;
    private GameObject closestObject;
    public bool shouldShoot = false;
    private Color alphaColor = new Color(1, 1, 1, 0);
    public bool isAlive = true;
    public bool fade = false;
    private float lower = 0f;
    private bool destroyed_called = false;
    public Sprite Undamaged;
    public Sprite BlueDamaged;
    public Sprite BlueVeryDamaged;
    public GameObject cannonBall;
    public char allegience = 'B';
    private GameObject playerRed;


    /*
     * TO DO on this script:
     *
     * ********* Need to get tower shooting cannonballs visually at the target:
     * ********* -- image of cannonball
     * ********* -- location of where cannonball needs to fire
     * ********* -- spawn and fire the cannonball to that location DONE
     *
     * ********* need to reduce enemy health DONE
     *
     * ********* need to get oppoisiton fighting the tower and reducing its health DONE
     *
     * ********* need to get the differing healths to give differnt images of the tower DONE
     *
     * ********* need to get the tower able to be healed by miners DONE
     *
     * ********* need to fix bug where clicking on where tower can shoot means troops cant go there. DONE
     *
     * ********* BUG: Cancel build button: DONE -> UNDONE!!! -> DONE (watch out for preinstantiated prefab)
     *
     * ********* BUG: Moving troops while firing DONE
     *
     * ********* BUG: troops still getting fired at even when moved away DONE
     *
     * ********* BUG: troops can currently fight tower from anywhere inside the shooting radius -- cant just fix by putting if statement within the while loop DONE
     *
     * ********* need to see which one the tower is actually firing at!! -> DOESNT GET SECOND OBJECT DONE
     *
     *
     * ********* Is removed from the list but doesnt start shootin at the new object DONE
     *
     */
```

BlueTowerScript ▸ fireCannon(float delayTime, GameObject cannonClone)

```
61          *
62          */
63
64          // Start is called before the first frame update
65          void Start()
66          {
67              if (gameObject.name != "BlueCastle")
68              {
69                  towerButton = gameObject.GetComponent<Button>();
70                  towerButton.onClick.AddListener(TowerButton);
71                  try
72                  {
73
74
75                      shootingRadius = GameObject.Find("ShootingRadius");
76                      shootingRadius.SetActive(false);
77                  }
78                  catch
79                  {
80
81                  }
82              }
83
84              playerRed = GameObject.Find("RedCastle");
85
86              InvokeRepeating("shoot", 0f, 2f);
87          }
88
89          // Update is called once per frame
90          void FixedUpdate()
91          {
92              nearby.RemoveAll(x => x == null); //For some reason needs to be in update... but now works anyway
93
94
95              if (health <= 0)
96              {
97                  isAlive = false;
98                  fade = true;
99              }
100
101             else if (health <= 50 && health > 0 && gameObject.GetComponent<Image>().sprite != BlueVeryDamaged)
102             {
103                 gameObject.GetComponent<Image>().sprite = BlueVeryDamaged;
104             }
105             else if( health <= 100 && health > 50 && gameObject.GetComponent<Image>().sprite != BlueDamaged)
106             {
107                 gameObject.GetComponent<Image>().sprite = BlueDamaged;
108
109             }
110             else if (health > 100 && gameObject.GetComponent<Image>().sprite != Undamaged)
111             {
112                 gameObject.GetComponent<Image>().sprite = Undamaged;
113             }
114
115             if (fade)
116             {
117                 StartCoroutine(fade_destroy());
118
119             }
120         }
121
122         void TowerButton()
```

BlueTowerScript ▸ fireCannon(float delayTime, GameObject cannonClone)

```
121
122         void TowerButton()
123         {
124             if (isShowing)
125             {
126                 shootingRadius.SetActive(false);
127                 isShowing = false;
128             }
129             else if (!isShowing)
130             {
131                 shootingRadius.SetActive(true);
132                 isShowing = true;
133             }
134
135         }
136
137         private void OnTriggerEnter(Collider other)
138         {
139
140             //this should control the shooting
141             if (other.gameObject.GetComponent<Miner>())
142             {
143                 if (other.gameObject.GetComponent<Miner>().allegience == allegience)
144                 {
145
146                 }
147                 else
148                 {
149                     nearby.Add(other.gameObject);
150                 }
151
152             }
153             else if (other.gameObject.GetComponent<Soldier>())
154             {
155                 if (other.gameObject.GetComponent<Soldier>().allegience == allegience)
156                 {
157
158                 }
159                 else
160                 {
161
162                     nearby.Add(other.gameObject);
163                     shouldShoot = true;
164                 }
165             }
166             else
167             {
168                 nearby.Add(other.gameObject);
169                 shouldShoot = true;
170             }
171
172
173
174
175
176         }
177
178         private void OnTriggerExit(Collider other)
179         {
180
181             if (other.gameObject.GetComponent<Miner>())
182             {
```

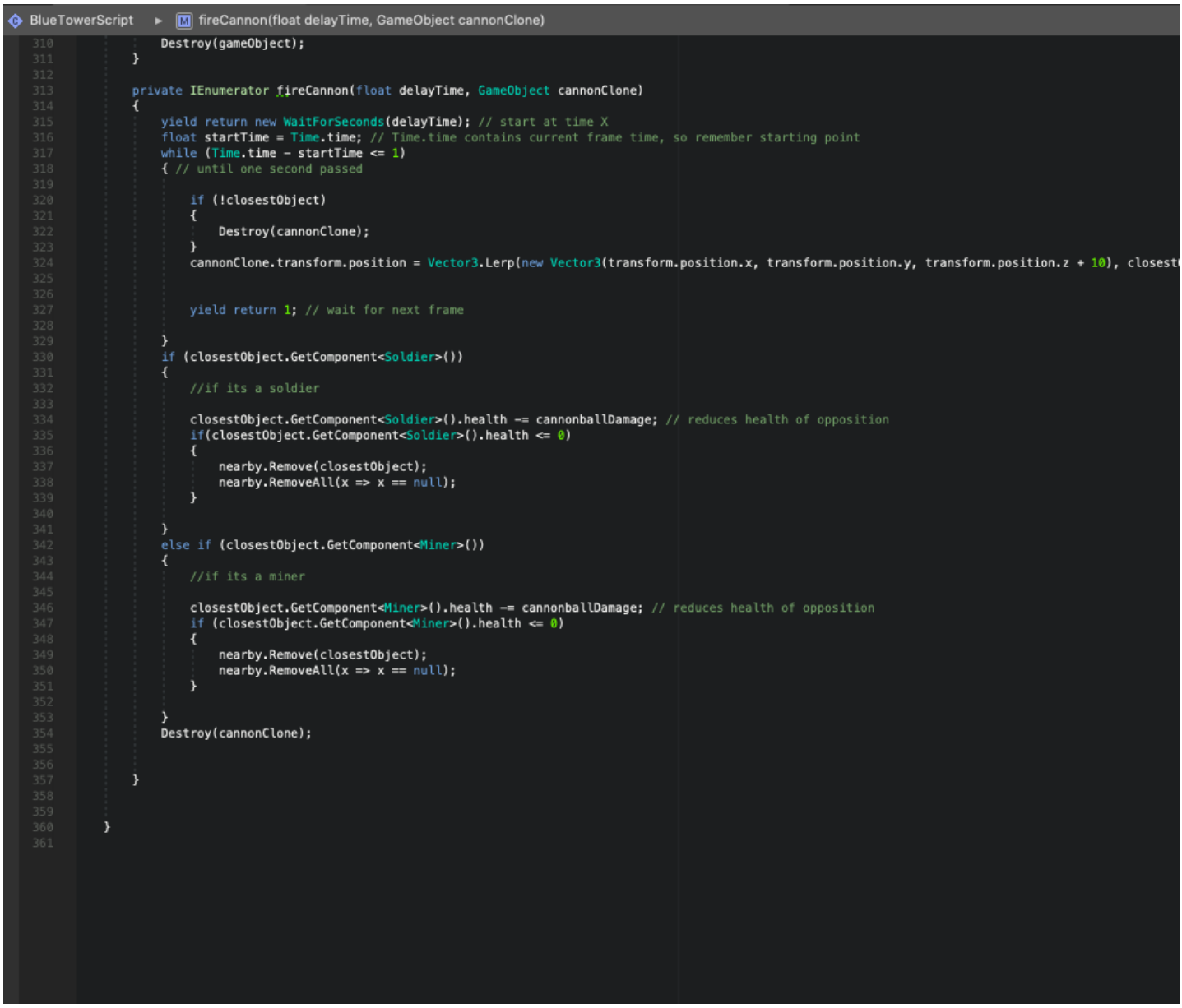
BlueTowerScript ▸ fireCannon(float delayTime, GameObject cannonClone)

```
178         private void OnTriggerExit(Collider other)
179         {
180 
181             if (other.gameObject.GetComponent<Miner>())
182             {
183                 if (other.gameObject.GetComponent<Miner>().allegience == allegience)
184                 {
185                     other.gameObject.GetComponent<Miner>().shouldHeal = false;
186                 }
187                 else
188                 {
189                     nearby.Remove(other.gameObject);
190                     nearby.RemoveAll(x => x == null);
191 
192                     closestObject = null;
193                 }
194 
195             }
196             else if (other.gameObject.GetComponent<Soldier>())
197             {
198                 if (other.gameObject.GetComponent<Soldier>().allegience == allegience)
199                 {
200 
201                 }
202                 else
203                 {
204                     nearby.Remove(other.gameObject);
205                     nearby.RemoveAll(x => x == null);
206 
207                     closestObject = null;
208                 }
209             }
210             else
211             {
212                 nearby.Remove(other.gameObject);
213                 nearby.RemoveAll(x => x == null);
214 
215                 closestObject = null;
216             }
217 
218 
219 
220         }
221 
222         private void shoot()
223         {
224             if (shouldShoot)
225             {
226                 closestObject = findNearestEnemy();
227                 //Need to fire the cannonball here:
228                 if (closestObject)
229                 {
230                     GameObject cannonClone = Instantiate(cannonBall);
231 
232                     StartCoroutine(fireCannon(0f, cannonClone));
233                 }
234 
235             }
236 
237 
238 
239 
```

BlueTowerScript ▸ fireCannon(float delayTime, GameObject cannonClone)

```
    }

    private GameObject findNearestEnemy()
    {

        var closestDistance = 1000000f;


        foreach (GameObject obj in nearby)
        {

            if (obj)
            {

                if (Vector3.Distance(transform.position, obj.transform.position) < closestDistance)
                {

                    closestDistance = Vector3.Distance(transform.position, obj.transform.position);
                    closestObject = obj;
                    //need to remove the closest object the righht one in the list ->first element turning to null rather than being removed from the list
                }
            }
            else
            {

                closestDistance = 1000000f;
                closestObject = null;

            }

        }
        if (allegience == 'R')
        {
            if (closestObject)
            {
                playerRed.GetComponent<Player_red>().BroadcastMessage("defend_towers", closestObject);
            }
            else
            {
                if (health < maxHealth)
                {
                    StartCoroutine(playerRed.GetComponent<Player_red>().heal_towers(gameObject));
                }
            }



        }


        return closestObject;
    }

    private IEnumerator fade_destroy()
    {

        yield return new WaitForSeconds(0f);
        lower += 0.3f;
        gameObject.GetComponent<MeshRenderer>().material.color = Color.Lerp(gameObject.GetComponent<MeshRenderer>().material.color, alphaColor, lower);

        if (!destroyed_called)
```

```
▶ M fireCannon(float delayTime, GameObject cannonClone)

vate IEnumerator fade_destroy()


yield return new WaitForSeconds(0f);
lower += 0.3f;
gameObject.GetComponent<MeshRenderer>().material.color = Color.Lerp(gameObject.GetComponent<MeshRenderer>().material.color, alphaColor, lower);

if (!destroyed_called)
{
    Invoke("destroy", 1.5f);
    destroyed_called = true;
}



lic void destroy()

Destroy(gameObject);


vate IEnumerator fireCannon(float delayTime, GameObject cannonClone)

yield return new WaitForSeconds(delayTime); // start at time X
float startTime = Time.time; // Time.time contains current frame time, so remember starting point
while (Time.time - startTime <= 1)
{ // until one second passed

    if (!closestObject)
    {
        Destroy(cannonClone);
    }
    cannonClone.transform.position = Vector3.Lerp(new Vector3(transform.position.x, transform.position.y, transform.position.z + 10), closestObject.transform.position, Time.time - startTime); // lerp from


    yield return 1; // wait for next frame

}
if (closestObject.GetComponent<Soldier>())
{
    //if its a soldier

    closestObject.GetComponent<Soldier>().health -= cannonballDamage; // reduces health of opposition
    if(closestObject.GetComponent<Soldier>().health <= 0)
    {
        nearby.Remove(closestObject);
        nearby.RemoveAll(x => x == null);
    }

}
else if (closestObject.GetComponent<Miner>())
{
    //if its a miner

    closestObject.GetComponent<Miner>().health -= cannonballDamage; // reduces health of opposition
    if (closestObject.GetComponent<Miner>().health <= 0)
    {
        nearby.Remove(closestObject);
        nearby.RemoveAll(x => x == null);
    }

}
Destroy(cannonClone);
```


BlueTowerScript
fireCannon(float delayTime, GameObject cannonClone)
Destroy(gameObject);
}
private IEnumerator fireCannon(float delayTime, GameObject cannonClone)
{
yield return new WaitForSeconds(delayTime); // start at time X
float startTime = Time.time; // Time.time contains current frame time, so remember starting point
while (Time.time - startTime <= 1)
{ // until one second passed
if (!closestObject)
{
Destroy(cannonClone);
}
cannonClone.transform.position = Vector3.Lerp(new Vector3(transform.position.x, transform.position.y, transform.position.z + 10), closest
yield return 1; // wait for next frame
}
if (closestObject.GetComponent<Soldier>())
{
//if its a soldier
closestObject.GetComponent<Soldier>().health -= cannonballDamage; // reduces health of opposition
if(closestObject.GetComponent<Soldier>().health <= 0)
{
nearby.Remove(closestObject);
nearby.RemoveAll(x => x == null);
}
}
else if (closestObject.GetComponent<Miner>())
{
//if its a miner
closestObject.GetComponent<Miner>().health -= cannonballDamage; // reduces health of opposition
if (closestObject.GetComponent<Miner>().health <= 0)
{
nearby.Remove(closestObject);
nearby.RemoveAll(x => x == null);
}
}
Destroy(cannonClone);
}
}

5.9 Menu_screen.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class menu_screen : MonoBehaviour
{

    public GameObject[] childs;
    public GameObject levels;
    public GameObject settings;
    bool is_set_showing;
    public bool[] lvl_unlocked; //Eventually change lvl_unlocked to be read/write to file so progress saves;
    public int lvl_number;
    public GameObject levels_object;
    bool active;
    GameObject[] clones;
    ParticleSystem[] part;
    Camera mainCamera;
    Camera PlayView1;



    // Start is called before the first frame update
    void Start()
    {
        int children = transform.childCount;
        int lvl_number = 5;

        lvl_unlocked = new bool[] { true, true, false, false, false };

        childs = new GameObject[children];
        clones = new GameObject[lvl_number];
        GameObject levels = new GameObject();

        part = GetComponentsInChildren<ParticleSystem>();

        part[2].gameObject.SetActive(false);

        mainCamera = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();
        PlayView1 = GameObject.FindGameObjectWithTag("Camera1").GetComponent<Camera>();




        for (int i = 0; i < children; i++)
        {

            childs[i] = transform.GetChild(i).gameObject;
            if (childs[i].name == "lvl_select")
            {
                levels = childs[i];
            }

        }

        settings = GameObject.FindGameObjectWithTag("Settings") as GameObject;

        Button Set_but = settings.GetComponentInChildren<Button>();
        Set_but.onClick.AddListener(toggle_set);

```

No selection

```
57
58          settings = GameObject.FindGameObjectWithTag("Settings") as GameObject;
59
60          Button Set_but = settings.GetComponentInChildren<Button>();
61          Set_but.onClick.AddListener(toggle_set);
62
63          Button lvl_but = levels.GetComponentInChildren<Button>();
64          lvl_but.onClick.AddListener(level_sorter);
65
66          bool is_set_showing = false;
67          settings.transform.GetChild(1).gameObject.SetActive(false);
68
69      }
70
71
72      // Update is called once per frame
73      void Update()
74      {
75
76      }
77
78      void toggle_set()
79      {
80
81          if (is_set_showing)
82          {
83              settings.transform.GetChild(1).gameObject.SetActive(false);
84              is_set_showing = false;
85          }
86          else if (!is_set_showing)
87          {
88              settings.transform.GetChild(1).gameObject.SetActive(true);
89              is_set_showing = true;
90          }
91
92
93
94      }
95
96      void level_sorter()
97      {
98          //Add a vec3 initial start location & set scale size
99
100         Vector3 location = new Vector3(0f, -30f, 100f);
101
102         Vector3 scale = new Vector3(1.3f, 0.8f, 1f);
103
104         if (!active)
105         {
106             active = true;
107             for (int i = 0; i < lvl_number; i++)
108             {
109                 var gap = 12;
110                 if (lvl_unlocked[i])
111                 {
112                     GameObject clone = Instantiate(levels_object) as GameObject;
113                     clone.transform.SetParent(transform);
114                     location.Set(location.x, location.y + gap, location.z);
115                     clone.transform.position = location;
116                     clone.transform.localScale = scale;
117                     clone.GetComponentInChildren<Button>().interactable = true;
118                     clone.GetComponentInChildren<Text>().text = "Level " + (i + 1).ToString();
```

No selection

```
    void level_sorter()
    {
        //Add a vec3 initial start location & set scale size

        Vector3 location = new Vector3(0f, -30f, 100f);

        Vector3 scale = new Vector3(1.3f, 0.8f, 1f);

        if (!active)
        {
            active = true;
            for (int i = 0; i < lvl_number; i++)
            {
                var gap = 12;
                if (lvl_unlocked[i])
                {
                    GameObject clone = Instantiate(levels_object) as GameObject;
                    clone.transform.SetParent(transform);
                    location.Set(location.x, location.y + gap, location.z);
                    clone.transform.position = location;
                    clone.transform.localScale = scale;
                    clone.GetComponentInChildren<Button>().interactable = true;
                    clone.GetComponentInChildren<Text>().text = "Level " + (i + 1).ToString();
                    Button a = clone.GetComponentInChildren<Button>();
                    a.onClick.AddListener(clear_screen);
                    a.onClick.AddListener(load_level_1);

                    clones[i] = clone;
                }
                else if (!lvl_unlocked[i])
                {
                    GameObject clone = Instantiate(levels_object) as GameObject;
                    clone.transform.SetParent(transform);
                    location.Set(location.x, location.y + gap, location.z);
                    clone.transform.position = location;
                    clone.transform.localScale = scale;
                    clone.GetComponentInChildren<Button>().interactable = false;
                    clone.GetComponentInChildren<Text>().text = "Level " + (i + 1).ToString() + " Locked";
                    clone.GetComponent<Image>().color = new Color(1, 0, 0, 1);
                    clones[i] = clone;
                }
            }


        }
        else if (active)
        {
            active = false;
            foreach (GameObject a in clones)
            {
                Destroy(a);
            }

            location.Set(0f, 50f, 100f);

        }
```

No selection

```
    void clear_screen()
    {
        var emission = part[1].emission;
        emission.rateOverTime = 15.0f;

        var sim_speed = part[1].main;
        sim_speed.simulationSpeed = 1.2f;
        sim_speed.startLifetime = 6.8f;
        sim_speed.startSize = 40f;
        sim_speed.gravityModifier = -3f;


        var shape = part[1].shape;
        shape.arc = 15.0f;

        //StartCoroutine(change.stop());

        var emission1 = part[0].emission;
        emission1.rateOverTime = 15.0f;

        var sim_speed1 = part[0].main;
        sim_speed1.simulationSpeed = 1.2f;
        sim_speed1.startLifetime = 6.8f;
        sim_speed1.startSize = 40f;
        sim_speed1.gravityModifier = -3f;


        var shape1 = part[0].shape;
        shape1.arc = 15.0f;



        part[2].gameObject.SetActive(true);

        var start = part[2].GetComponent<ParticleSystem>();
        var start1 = start.emission;
        start1.rateOverTime = 150f;



        Invoke("end", 2.734f);
        Invoke("stop_smoke", 5.5f);




    }
    void stop_smoke()
    {
        part[2].gameObject.SetActive(false);
    }
```

5.10 Sorting_Order.cs

Sorting_Order ▸ Start()

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Sorting_Order : MonoBehaviour
{

    //this changes the sorting order of the different objects to make get them rendering and interacting in the right order

    public string layer;

    public bool change_children;
    // Start is called before the first frame update
    void Start()
    {
        GetComponent<Renderer>().sortingLayerName = layer;
        if (change_children)
        {
            int children = transform.childCount;

            for (int i = 0; i < children; i++)
            {
                transform.GetChild(i).GetComponent<Renderer>().sortingLayerName = layer;

            }
        }

    }

}
```

5.11 Player_blue.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.Animations;
using UnityEngine;
using UnityEngine.UI;

public class Player_blue : MonoBehaviour
{
    public int player_gold = 50;
    public GameObject gold_text;
    // Start is called before the first frame update
    void Start()
    {
        gold_text = GameObject.FindGameObjectWithTag("Gold");
        update_gold(50);
    }

    // Update is called once per frame
    void Update()
    {

    }

    void update_gold(int gold)
    {
        player_gold += gold;
        gold_text.GetComponent<Text>().text = player_gold.ToString();
    }
}

```

5.12 Miner.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Miner : character_controll
{
    private int gold_capacity = 12;
    public int current_gold;
    private char allegience1;
    public Material matR;
    public Material matB;
    private bool called = false;
    private Vector3 castle_position;
    public int minerRepair = 10;
    public bool shouldHeal = true;
    private bool justSpawned = true;


    // Start is called before the first frame update
    new void Start()
    {

        base.Start();
        if(allegience == 'B')
        {
            castle_position = GameObject.FindGameObjectWithTag("BlueCastle").GetComponentInChildren<Cube_castle>().transform.position + new Vector3(-34, 0, -32);
        }
        else
        {
           castle_position = GameObject.FindGameObjectWithTag("RedCastle").GetComponentInChildren<Cube_castle>().transform.position + new Vector3(-34, 0, -32);

        }




    }

    // Update is called once per frame
    new void FixedUpdate()
    {

        base.FixedUpdate();

        if (allegience1 == allegience)
        {

        }
        else
        {
            allegience1 = allegience;
            change_allegience();

        }
        StartCoroutine(get_gold());


    }


    void change_allegience()
```

```
void change_allegience()
{
    // Changes whether trigger is active or not for each different allegience
    if (allegience == 'R')
    {
        gameObject.GetComponent<CapsuleCollider>().isTrigger = false;
        gameObject.GetComponent<MeshRenderer>().material = matR;
    }
    else if (allegience == 'B')
    {
        gameObject.GetComponent<CapsuleCollider>().isTrigger = true;
        gameObject.GetComponent<MeshRenderer>().material = matB;
    }
}


private IEnumerator get_gold()
{

    if (mine_it) {


        //Need to empty the gold from the miner, need to update the text box for the amount of gold so its visible (and test to see if gold actually is updating)

        if (Vector3.Distance(transform.position, mine_position) < 10) //If it is within x units of the mine's position
        {

            if (!called)
            {
                called = true;
                while (current_gold + PlayView1.gameObject.GetComponentInChildren<Mining>().gold_per_second <= gold_capacity) //Fills up the gold until it has reached maximum ca
                {

                    current_gold += PlayView1.gameObject.GetComponentInChildren<Mining>().gold_per_second;
                    yield return new WaitForSeconds(1f);

                }
                Movement(gameObject, castle_position);
                called = false;
            }




        }


        else if (Vector3.Distance(transform.position, castle_position) < 10)
        {
            yield return new WaitForSeconds(0.2f);
            if (allegience == 'B')
            {
                PlayView1.gameObject.GetComponentInChildren<Player_blue>().BroadcastMessage("update_gold", current_gold);
            }
            else
            {
                GameObject.FindGameObjectWithTag("RedCastle").GetComponentInChildren<Player_red>().playerGold += current_gold;
            }

            current_gold = 0;
```

No selection

```
            else if (Vector3.Distance(transform.position, castle_position) < 10)
            {
                yield return new WaitForSeconds(0.2f);
                if (allegience == 'B')
                {
                    PlayView1.gameObject.GetComponentInChildren<Player_blue>().BroadcastMessage("update_gold", current_gold);
                }
                else
                {
                    GameObject.FindGameObjectWithTag("RedCastle").GetComponentInChildren<Player_red>().playerGold += current_gold;
                }

                current_gold = 0;
                yield return new WaitForSeconds(0.2f);
                Movement(gameObject, mine_position);

            }
            else if (justSpawned)
            {
                Movement(gameObject, mine_position);
                justSpawned = false;
            }

        }

    }
    public IEnumerator heal_tower(GameObject tower)
    {
        if (shouldHeal)
        {
            if (tower.gameObject.GetComponent<BlueTowerScript>().health < tower.gameObject.GetComponent<BlueTowerScript>().maxHealth - minerRepair)
            {
                yield return new WaitForSeconds(2f);
                tower.gameObject.GetComponent<BlueTowerScript>().health += minerRepair;
                StartCoroutine(heal_tower(tower));
            }
            else
            {
                yield return new WaitForSeconds(2f);
                tower.gameObject.GetComponent<BlueTowerScript>().health = tower.gameObject.GetComponent<BlueTowerScript>().maxHealth;
            }
        }
        else
        {
            yield return new WaitForSeconds(1f);
            shouldHeal = true;
        }

    }
}
```

5.13 Player_red.cs

No selection

```
using System.Collections;
using System.Collections.Generic;
using System.Net.Sockets;
using UnityEngine;
using UnityEngine.UI;

public class Player_red : MonoBehaviour
{
    public int playerGold;
    public int soldierCost = 25;
    public int minerCost = 20;
    public GameObject Canvas;
    private Button[] all_buttons;
    private Image loading_shadow_knight;
    private Image loading_shadow_miner;
    public GameObject soldier;
    public GameObject miners;
    private GameObject buildMenu;
    public GameObject TowerObj;
    public Camera PlayView1;
    public GameObject Screen_canvas;
    public List<GameObject> Miner_list = new List<GameObject>();
    public List<GameObject> Soldier_list = new List<GameObject>();
    public List<GameObject> Tower_list = new List<GameObject>();
    public bool isOnStartUp = true;
    private GameObject mine;
    public int towerCost = 50;
    private bool minerHealing = false;




    // Start is called before the first frame update
    void Start()
    {
        playerGold = 100;
        all_buttons = gameObject.GetComponentsInChildren<Button>();

        //Adding listeners to all the buttons



        //Knght shadow
        Image[] loading_shadow_knight_list = all_buttons[1].gameObject.GetComponentsInChildren<Image>();
        loading_shadow_knight = loading_shadow_knight_list[1];
        loading_shadow_knight.fillAmount = 0;


        //Miner shadow
        Image[] loading_shadow_miner_list = all_buttons[2].gameObject.GetComponentsInChildren<Image>();
        loading_shadow_miner = loading_shadow_miner_list[1];
        loading_shadow_miner.fillAmount = 0;



        Canvas = gameObject.GetComponentInChildren<Canvas>().gameObject;
        Canvas.SetActive(false);

        buildMenu = GameObject.Find("BuildMenu");
        mine = GameObject.Find("RedMine");
    }
```

No selection

```
    // Update is called once per frame
    void Update()
    {

        if (gameObject.activeSelf && isOnStartUp)
        {
            StartCoroutine(AIScript());
            isOnStartUp = false;
        }


    }



    private string train_soldier()
    {
        if (playerGold >= soldierCost)
        {
            playerGold -= soldierCost;
            GameObject spawned = Instantiate(soldier, transform.position + new Vector3(-70, 0, -150), Quaternion.identity);
            spawned.GetComponent<Soldier>().allegience = 'R'; //Need to make sure the allegience is set after spawning in the troop
            Soldier_list.Add(spawned);
            return null;

        }
        else
        {
            return "NoGold";
        }
    }

    private string train_miner()
    {
        if (playerGold >= minerCost)
        {

            playerGold -= minerCost;
            GameObject spawned = Instantiate(miners, transform.position + new Vector3(-90, 0, -180), Quaternion.identity);
            spawned.GetComponent<Miner>().allegience = 'R';
            Miner_list.Add(spawned);
            return null;

        }
        else
        {
            return "NoGold";
        }

    }



    private GameObject instantiate_tower(Vector3 location)
    {
        if (playerGold >= towerCost)
        {
            GameObject placed_tower = Instantiate(TowerObj, new Vector3(0, 0, 0), Quaternion.identity);
            placed_tower.transform.SetParent(Screen_canvas.transform, false);
            placed_tower.transform.position = PlayView1.ScreenToWorldPoint(new Vector3(location.x, location.y, 720));
            placed_tower.GetComponent<BlueTowerScript>().shouldShoot = true;
```

No selection

```
115
116         private GameObject instantiate_tower(Vector3 location)
117         {
118             if (playerGold >= towerCost)
119             {
120                 GameObject placed_tower = Instantiate(TowerObj, new Vector3(0, 0, 0), Quaternion.identity);
121                 placed_tower.transform.SetParent(Screen_canvas.transform, false);
122                 placed_tower.transform.position = PlayView1.ScreenToWorldPoint(new Vector3(location.x, location.y, 720));
123                 placed_tower.GetComponent<BlueTowerScript>().shouldShoot = true;
124                 placed_tower.GetComponent<BlueTowerScript>().allegience = 'R';
125                 Tower_list.Add(placed_tower);
126                 return placed_tower;
127             }
128             else
129             {
130                 return null;
131             }
132
133         }
134
135
136         private IEnumerator AIScript()
137         {
138
139             yield return new WaitForSeconds(0.5f);
140             StartCoroutine(train_miner_to_mine());
141             yield return new WaitForSeconds(0.5f);
142             train_soldier();
143             yield return new WaitForSeconds(0.5f);
144             //yield return new WaitUntil(() => playerGold >= towerCost);
145
146             instantiate_tower(new Vector3(250, 190, 90));
147
148
149
150         }
151
152         private IEnumerator train_miner_to_mine()
153         {
154             yield return new WaitForSeconds(0f);
155             int len = Miner_list.Count;
156             train_miner();
157             yield return new WaitUntil(() => len + 1 == Miner_list.Count);
158
159             Miner_list[Miner_list.Count - 1].GetComponent<Miner>().mine_it = true;
160
161         }
162
163         public void defend_towers(GameObject closestAttacker)
164         {
165             foreach (GameObject soldier in Soldier_list)
166             {
167
168                 soldier.GetComponent<Soldier>().Movement(soldier, closestAttacker.transform.position);
169             }
170         }
171         public IEnumerator heal_towers(GameObject towerToHeal)
172         {
173             if (!minerHealing)
174             {
175
176                 foreach (GameObject miner in Miner_list)
```

No selection

```
158
159             Miner_list[Miner_list.Count - 1].GetComponent<Miner>().mine_it = true;
160
161         }
162
163         public void defend_towers(GameObject closestAttacker)
164         {
165             foreach (GameObject soldier in Soldier_list)
166             {
167
168                 soldier.GetComponent<Soldier>().Movement(soldier, closestAttacker.transform.position);
169             }
170         }
171         public IEnumerator heal_towers(GameObject towerToHeal)
172         {
173             if (!minerHealing)
174             {
175
176                 foreach (GameObject miner in Miner_list)
177                 {
178                     miner.GetComponent<Miner>().mine_it = false;
179                     miner.GetComponent<Miner>().Movement(miner, towerToHeal.transform.position);
180                     StartCoroutine(miner.GetComponent<Miner>().heal_tower(towerToHeal));
181
182                 }
183                 minerHealing = true;
184             }
185             //currenty heals while walking to the tower still
186             yield return new WaitUntil(() => towerToHeal.GetComponent<BlueTowerScript>().health == towerToHeal.GetComponent<BlueTowerScript>().maxHealth);
187
188             minerHealing = false;
189
190             foreach (GameObject miner in Miner_list)
191             {
192                 miner.GetComponent<Miner>().mine_it = true;
193                 miner.GetComponent<Miner>().Movement(miner, miner.GetComponent<Miner>().mine_position);
194
195             }
196
197
198         }
199
200     }
201
```