Implementation report

Module	SEPR
Year	2019/20
Assessment	3
Team	York Fire Marshalls
Members	Samuel Whitehead, Andrew Connor, Alex Dawson, Finn Jackson, Fred Dodd, Phuong Kha
Deliverable	Implementation Report

Overview of changes made

In this version of the game, we have used the requirements documentation made by the previous team, and implemented them into the game. Whilst we have added many new features, we have kept all of the previous architecture, and only changed a few small elements of the code. In this document we will explain how our code implements the requirements, and explain the new features.

Changes to the map

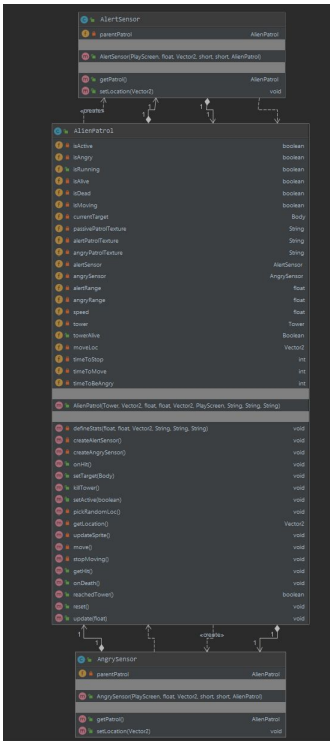
As we had to fit more alien bases into the game we had decided to expand the map to allow for this. The way the previous teams map was set up was awkward to expand and also did not resemble york, which did not allow it to meet the product brief. We decided to do a full overhaul in response to this, creating a new tiled map but still implementing the object features in order to spawn towers as well as create collision objects. The main difference other than the size is the visual side, in which we feel gives the game a more fun and vibrant look while also actually resembling york in an abstract way.


```
renderer = new OrthoCachedTiledMapRenderer(maploader.map, 1 / Kroy.PPM, 4000);
```

We also edited the above line, which caches the map so it can be loaded quickly on every game tick. The edit involves the end of the line, the ‘,4000’ previously this was not required and the cache size was defaulted to 2000 but as the new map is larger it required more space to be cached.

In order to meet the required number of towers in the product brief we had to add more towers to the map. This did not require any major changes to the code as the way it was implemented by the previous group mainly used the tiled map editor to do this, so we added more ellipse objects but with different property values in order to give them different health

values. There were also some changes to the code to make the weapons slightly different, they fire in the same way but their fire rates are increased for the more difficult bases.



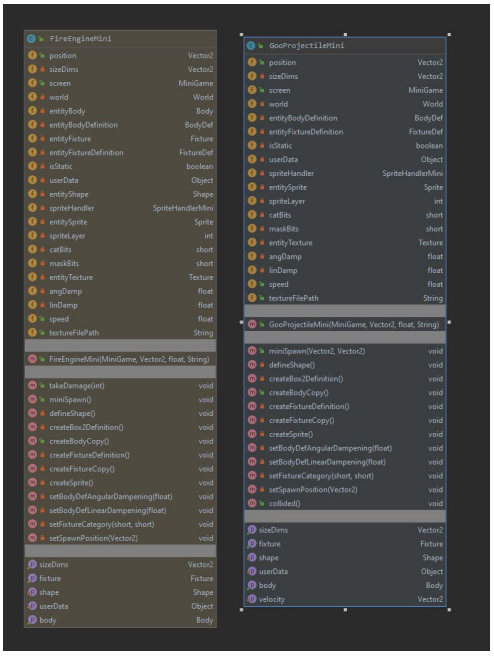
Patrols

To fulfil the product briefing, we had to create alien patrols that would roam the map and trigger a mini game. We achieved this by creating the object AlienPatrol object, which extends from the box2D entity BoxGameEntity. The patrols each have two different sensors associated with them - one called an AlertSensor and another called an AngrySensor. When a fire engine enters the Alert range of a patrol, the patrol stops and the sprite changes to show that it has noticed a truck approaching. If the player comes even closer, into the Angry range, the patrol charges for the player, moving a set distance in a straight line before stopping and resetting itself. If the truck collides with the patrol, the minigame is triggered and the patrol runs back to the AlienTower it belongs to.
Each patrol is spawned from an AlienTower, which spawns one every couple of seconds. When one gets hit by water from an engine, it runs back to the tower it belongs to and resets itself. If the tower is already destroyed, instead the patrol simply vanishes, dissolved into the water.

The UML for the patrol class, as well as the two Sensor classes belonging to it, is shown next to here. Each of the two sensors use different classes in order for the ContactListener to be able to differentiate between the two.

Minigame

The minigame, triggered whenever a fire engine collides with an alien patrol, is a 4 second long game where the player dodges between alien orbs approaching from all sides of the screen. As well as the MiniGame class, which is an alternate Screen like PlayScreen, we define two new types of Entities: FireEngineMini and GooProjectileMini. When called, MiniGame creates a FireEngineMini based off of the FireEngine object of the player. When damaged, this also damages the original truck. The game then spawns GooProjectileMinis using HandleGooShots(), which waits half a second between each shot from a random edge of the screen, with a velocity aimed at the players current position.



The UML for the FireEngineMini class and the GooProjectileMini is shown here, including important methods and parameters.

FR_ Minigame	Triggered when the fire engine collides with an alien patrol	UR_Progress

Fire Station destroyed

As a functional requirement(FR_Destroy) set by the previous group, we had to make sure the fire station would be destroyed 8 minutes after the first fortress was destroyed, stopping the user from repairing the fire engines

FR_Destroy	The fire station is destroyed 8mins after the first fortress is destroyed, taking away the ability to repair fire engines	UR_Progress

We did this by calling a timer from the tower class after the first tower was destroyed. This timer is a method within the play screen class, which calls a method StationDestroyed, which can be found in the FireStation object and stops the fire truck from being repaired after 8 minutes has passed. We found this important as both in the requirements(NFR_Length) and the product brief, it clearly states that the length of the game should be suitable for open days, and we found that by destroying the fire station after a particular amount of time, it would stop the game from lasting for a long period of time.

NFR_Length	The Game length should be suitable for Open day	UR_Time	The Game lasts 10-15 minutes

Increased number of fire engines

As the product briefing stated that we needed at least four unique fire engines, we had to create two more fire engines to be added to the game. This also supports the requirement that the user should be able to change the fire engine on arrival of the fire station (FR_Change). The new small fire engine is faster, with a smaller hit box, but also has lower health and water capacity. The new alien fire engine is similar to the regular fire engine but fires in a trident pattern. This was done by adding two new constructors which contain the new images of the firetrucks and new changes to the health, water levels and the weapon that is used.

FR_Change	When entering the fire station the user can change and repair/refill fire engines	UR_Player

Increasing difficulty over time

In both the product brief and requirements(FR_Difficulty) it mentions the need for the game to get more difficult as the user plays. We have done this by making the alien bases increase health linearly as the game progresses. We used trial and error to find both a time and incrementation that would make the game difficult to complete. We did this by adding a timer

in the constructor of the towers which is called every sixty seconds, this then calls another method within tower which increases all the towers that are alive by 100 health and also the range by 1.1x

FR_ Difficulty	The System shall always make the game will get more difficult in a linear fashion, more alien patrols and harder fortresses	UR_Progress