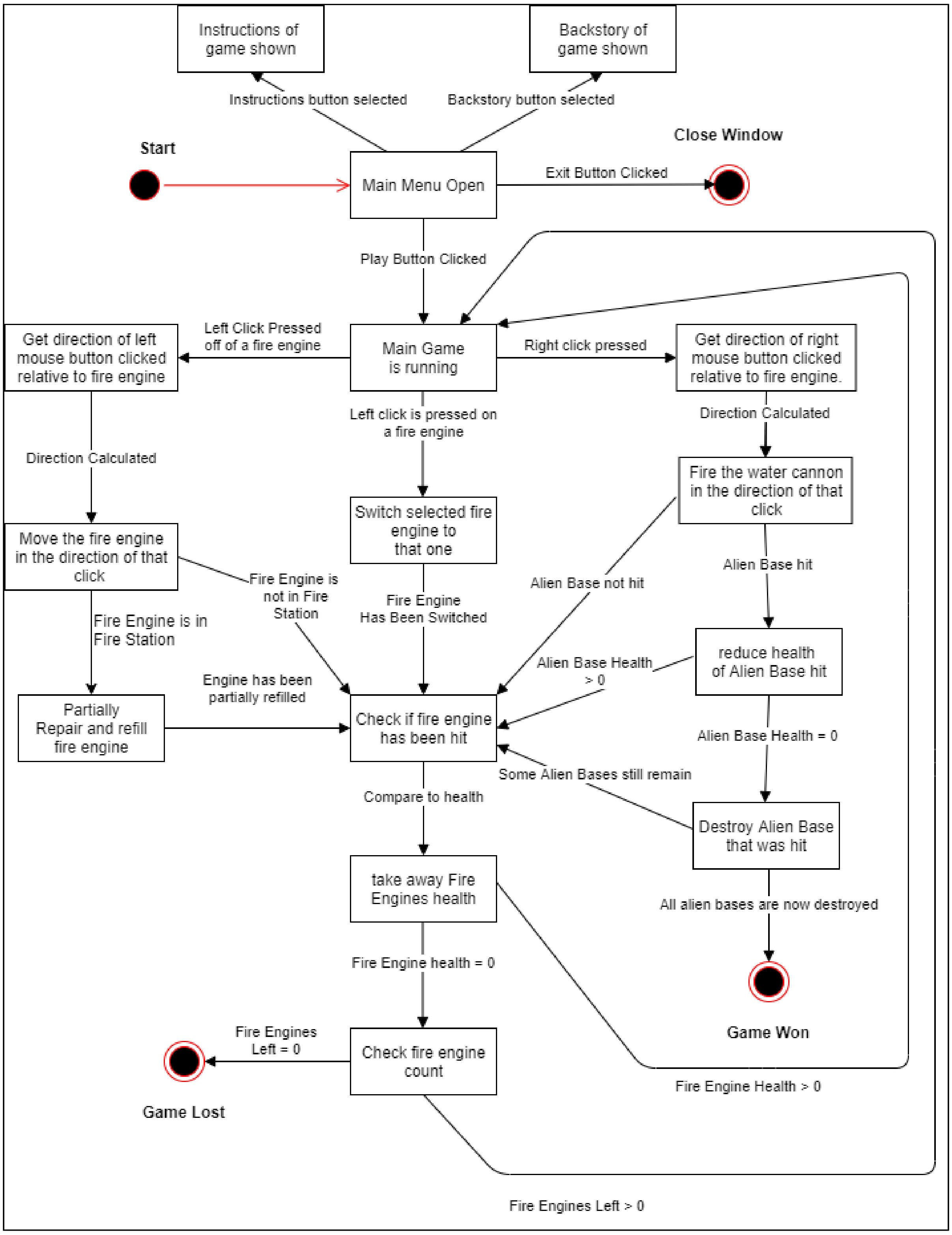
MiKroysoft

Module	SEPR
Year	2019/20
Assessme nt	2
Team	MiKroysoft
Members	Daniel Crooks, James Rand, Irene Sarigu, Alfie Jennings, Charlotte Clark, Jasper Law
Deliverable	Architecture 2
Website	https://mikroysoft.github.io/assessmentTwo.html

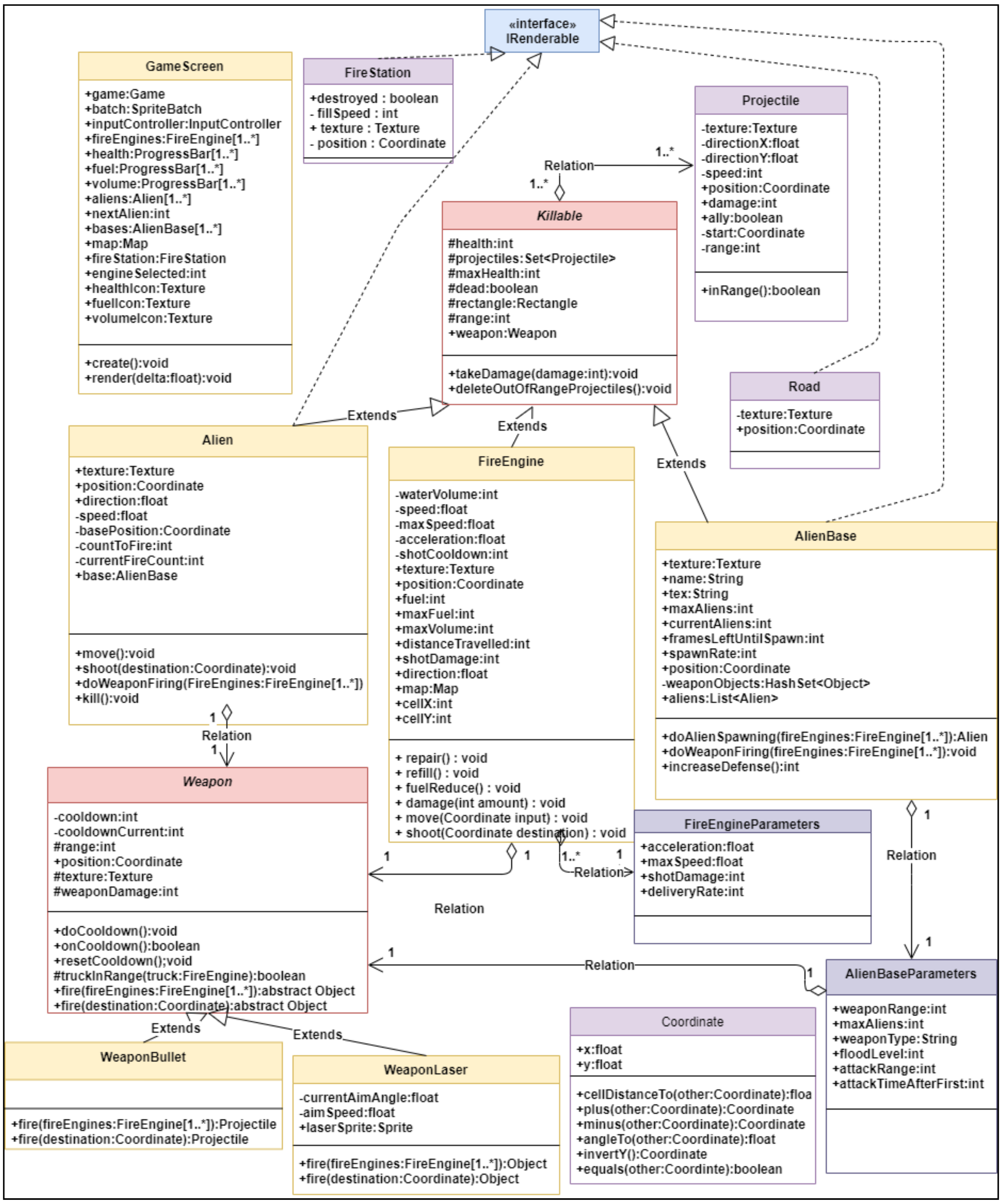
(a) The specific language that was used to describe the architectural diagrams was UML and tool used to create the diagrams was draw.io **[1]**. The state diagram **(Figure 1)** shows how processes in the game work. The UML class diagram **(Figure 2)** was created, as the structure of our code is based on an Object-orientated programming(OOP) paradigm **[2]**. Where a (UML) class diagram is a useful solution, in representing an OOP **[3]**.

Figure 1:



Instructions of game shown
Backstory of game shown
Instructions button selected
Backstory button selected
Start
Close Window
Main Menu Open
Exit Button Clicked
Play Button Clicked
Left Click Pressed off of a fire engine
Main Game is running
Right click pressed
Get direction of left mouse button clicked relative to fire engine
Get direction of right mouse button clicked relative to fire engine.
Direction Calculated
Left click is pressed on a fire engine
Direction Calculated
Fire the water cannon in the direction of that click
Switch selected fire engine to that one
Move the fire engine in the direction of that click
Fire Engine is not in Fire Station
Alien Base not hit
Alien Base hit
Fire Engine is in Fire Station
Fire Engine Has Been Switched
Alien Base Health > 0
reduce health of Alien Base hit
Engine has been partially refilled
Partially Repair and refill fire engine
Check if fire engine has been hit
Alien Base Health = 0
Some Alien Bases still remain
Compare to health
Destroy Alien Base that was hit
take away Fire Engines health
All alien bases are now destroyed
Fire Engine health = 0
Game Won
Fire Engines Left = 0
Check fire engine count
Fire Engine Health > 0
Game Lost
Fire Engines Left > 0

Figure 2:

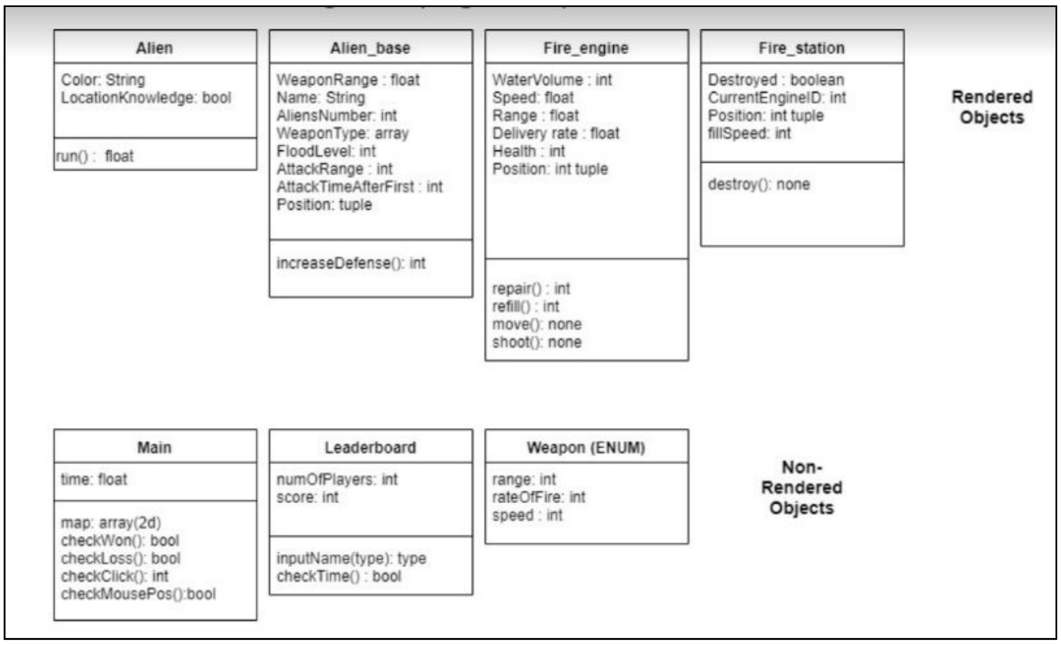


To see a more in depth class diagram, with all classes please go to:
https://mikroysoft.github.io/assessment2-architecture.png

(b) The concrete architecture builds from the abstract architecture in many different ways.

Class Diagram Changes

Figure 3:



The following changes were made from the original class diagram ***(Figure 3)*** in ***Figure 2***:

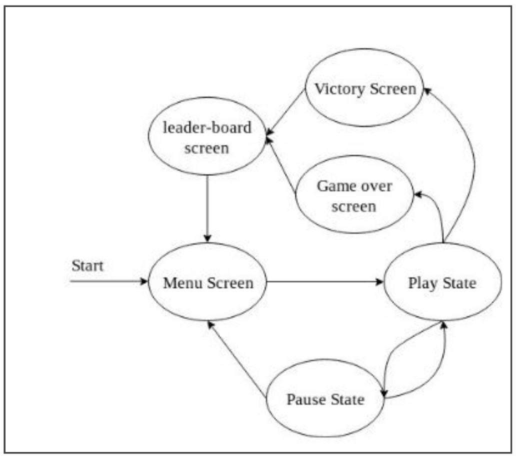
- AlienBase → Originally called Alien_base, a few changes were made to this class in the new architecture, as the team originally unanticipated the amount of variables and classes that were needed to meet the user and system requirements. Firstly, the updated functional requirement FR_EXTRA_TERRESTRIAL requires that the bases have different spawn rates of aliens - therefore the integer variable spawnRate was added to AlienBase. Where FR_EXTRA_TERRESTRIAL is also needed to satisfy the user requirement of UR_THE.
- FireEngine → Originally called Fire_engine, a lot of changes were made to this class in the new architecture, as we realised that not all required attributes of the fire engine were catered for in the old one. In the original architecture the user requirements UR_ENGI and UR_REF were breached, since they require FR_ENGINE_VOLUME to be satisfied, which would only be satisfied if each fire engine had a unique maximum volume of water that it can store - thus the public integer of maxVolume was added. UR_ENGI was also breached for the following: FR_ENGINE_SPEED so maxSpeed was added, FR_ENGINE_ACCELERATION so acceleration was added and FR_ENGINE_DAMAGE so a maxHealth was added.
- Weapon → This whole class was edited to make game objects with the type of FireEngine, Alien or AlienBase to possess weapons more easily. Weapon is also a superclass (parent), with WeaponBullet and WeaponLaser being subclasses(its children) which inherit from it, and are two different variations of weapon.
- FireStation → Stayed similar to the original architecture planned.
- GameScreen → This is a class that has been added that stores the main parts of the game create() and render(delta:float), which are used to first create the main game

screen and then update the screen as the game goes on, frame by frame. So the game ends correctly (UR_END) GameScreen stores valuable variables and information, which can be used to decide whether the game has been won or not. It is also where entities such as fire engines, alien bases and aliens are added to the game. Therefore satisfying the user requirement that the game follows the theme given by the client, of Kroy (UR_THE).

Additionally, a class called Coordinate has been added to the architecture, replacing the previously planned system to store entities positions using a tuple. We decided to do this to make a more understandable system for positioning (using x and y). We also made this class to be a good representation of code to show potential Computer Science on open days, which satisfies the new user requirement of UR_UOD.

On the class diagram from the website, map generation is done in the class of Map, which uses a main asset of a text file with map information. UR_STRY is also satisfied in the architecture in BackStoryScreen.

State Diagram Changes
Figure 4:



The main changes in the state diagram from ***Figure 4*** to ***Figure 1*** was to make the diagram more UML - friendly. Additionally, the diagram in ***Figure 1*** shows more easily how the main parts of the game work and link onto one another. Furthermore, the state diagram shows how the Fire Engines are controlled by left - clicking to move and right - clicking to shoot, clicking is deemed fairly similar to tapping on mobile phones. The decision on tapping was made on purpose to satisfy the user requirement of UR_MOB. As the navigation used in the method touchDown, in the class InputController *(on the websites class diagram)*, is fairly easy to replicate in modern smartphones.

How decisions were made
We noticed that the structure of variable, method and class names didn't follow a structure, therefore we decided that in our implementation and architecture in camel case.
Also, the overall structure of the UML diagrams was changed based on lecture slides, and other sources**[2 - 4]**, which gave a great insight on how UML state diagrams and class diagrams should be used when representing a systems architecture. It was decided in a MiKroySoft team meeting, after more research into UML diagrams**[4]**, that we would use a state diagram as a diagram to show the behaviour of the system (since a state diagram is a Behaviour diagram**[4]**) and a class diagram to show the structure of the system (since a class diagram is a Structure diagram**[4]**).
Lastly, the feedback given to the team after assessment 1 was used to produce better representations of our code, in better formatted UML representations.

References:

[1] Draw.io. (2020). *Flowchart Maker & Online Diagram Software*. [online] Available at: https://www.draw.io/ [Accessed 8 Jan. 2020].
[2] development, W. and Language, U. (2020). *UML: Unified Modeling Language*. [online] IONOS Digitalguide. Available at: https://www.ionos.com/digitalguide/websites/web-development/uml-unified-modeling-language/ [Accessed 19 Jan. 2020].
[3] Noel Ceta, Tallyfy. (2020). *All You Need to Know About UML Diagrams: Types and 5+ Examples*. [online] Available at: https://tallyfy.com/uml-diagram/#class-diagram [Accessed 17 Jan. 2020].
[4] GeeksforGeeks. (2020). *Unified Modeling Language (UML) | State Diagrams - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/ [Accessed 18 Jan. 2020].