Architecture

SEPR Team: Berbils

Module	SEPR
Year	2019/20
Assessment	1
Team	Berbils
Members	Dylan Henley-Marshall Edward Demkowicz-Duffy Emily Wisher Samkeliso Kimbinyi Joshua Waha Peter Robinson
Deliverable	Arch1.pdf

Abstract Architecture

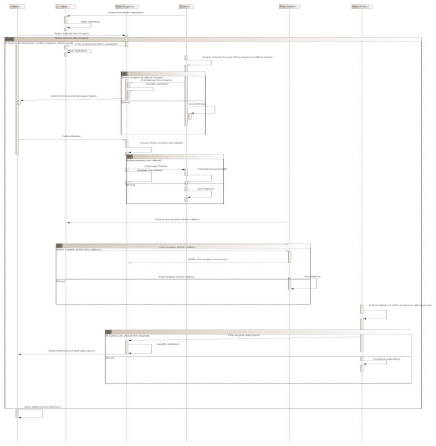
The team decided to express the architecture of our game via UML. We chose to use UML because it is the industry standard for modelling languages. We chose to express the architecture using 3 diagrams. A UML Interaction Diagram, UML State Diagram and UML Class Diagram.

Modelio was the software we decided to use to create the UML diagrams. We chose to use Modelio because it's open-source and runs on many platforms. Some members of the team used Windows and others used Linux.

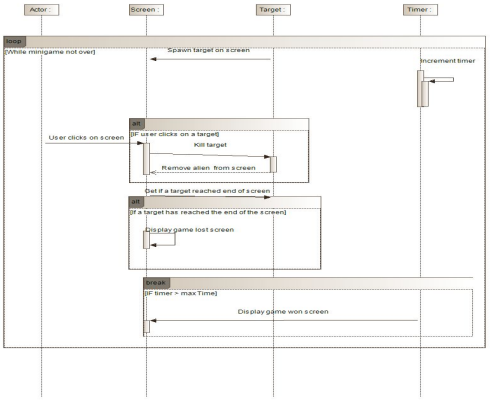
UML Interaction

We wanted a UML diagram that could show us how different parts of this game interact with each other without going into too specific a detail. For this, we had the choice between a sequence and a collaboration diagram. We chose a sequence diagram to show the sequential nature of which events, checks etc. Will occur and how they interact and depend on each other.

Diagram[Assignment1/ARCH/INT-Diagram-Game]



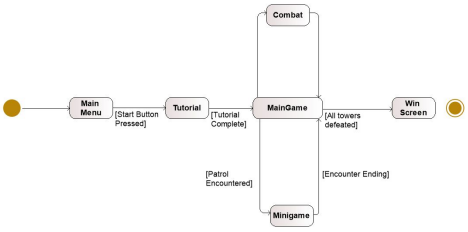
Diagram[Assignment1/ARCH/INT-Diagram-Minigame]



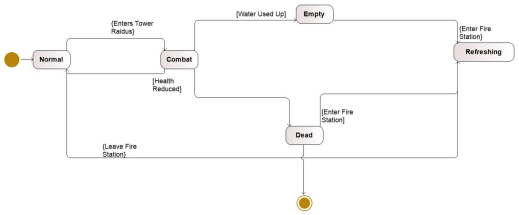
UML State Diagram

The State Diagram is used to describe the condition of part of the system. I describes the current condition and what needs to occur for it to transition into another condition. We chose to use this diagram because it's very abstract and shows the high-level attributes of game objects. The images can be viewed in larger resolution on the website.

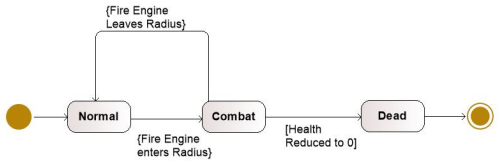
Game[Images/Assignment1/ARCH/State-Game.png]



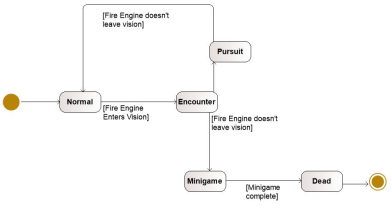
Fire Engines[Images/Assignment1/ARCH/State-FireEngine.png]



Tower[Images/Assignment1/ARCH/State-Tower.png]



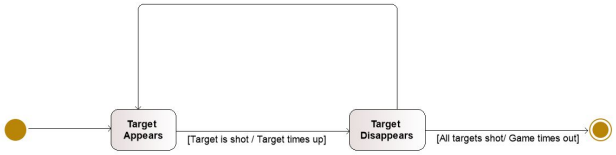
Alien Patrol[Images/Assignment1/ARCH/State-AlienPatrol.png]



Fire Station[Images/Assignment1/ARCH/State-FireStation.png]

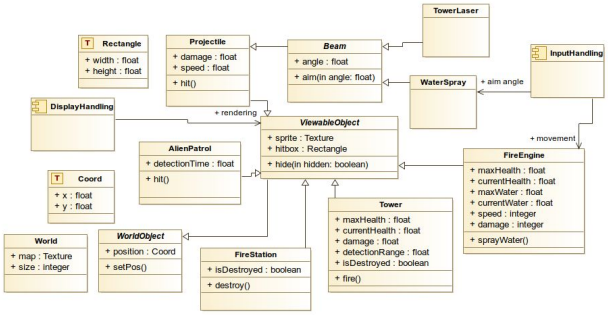
Minigame

Minigame[Images/Assignment1/ARCH/STATE-MiniGame]



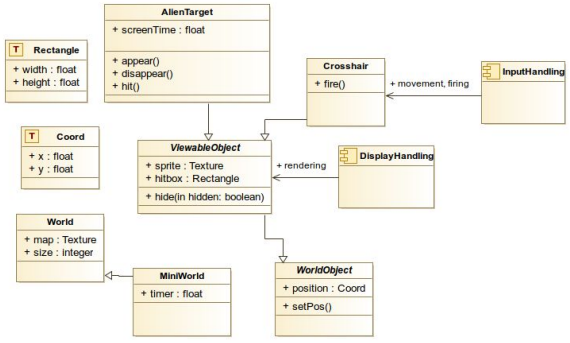
UML Class Diagram

Game[Images/Assignment1/ARCH/UML-mainGame.png]



Above is the abstract UML class diagram for Kroy. Before entering into a walkthrough of the decisions made, please note that primitive type choices are not fixed (particularly integers vs. floats) as the software used (Modelio) requires types to be specified.

Minigame[Assignment1/ARCH/UML-minigame.png]



Above is the abstract UML class diagram for the MiniGame.

Justification

For this project, we are targeting Windows Mac and Linux as stated in Requirement [NFR Operating]. This will be achieved by using Java 8, this also means that we have the option of running on a mobile device running the Android operating system. As stated in the Method and Planning document we will be using LibGDX. We chose to do this because it is a well-known game library written in java that leverages OpenGL. The use of OpenGL means that the graphics can also run across multiple operating systems.

Overall Structure

We plan an abstract class called “WorldObject” to represent anything that is contained within the world (not necessarily visible), which contains just a “Coord” object and a method to change it. This could also be safely implemented by just allowing the position attribute to be changed.

The next logical level of less abstraction is an actor in the world which also has an appearance - for this, we plan a “ViewableObject” abstract class which inherits from “WorldObject” and contains:

- A “Texture” object to indicate its appearance
- A “Rectangle” object to indicate the area in and around the actor that it is considered to be occupying, conventionally referred to as it’s “hitbox” or “hurtbox”.
- A “hidden” method to control whether or not the actor is visible at the moment - this is very useful for spawning actors in advance or removing them once they are destroyed or no longer relevant

Additionally, “ViewableObject” classes will have the ability to be displayed on screen by whatever method or system we implement for that - described here by the “DisplayHandling” component.

Components

Fire Station

FireStation, the stationary actor which represents the player’s base of operations. It can tell whether it is destroyed and can be told that it has been destroyed as in “FR_Destroy”.

Fire Engines

FireEngine, the actor that the player will spend most of the game controlling. All types of fire engine will have both maximum and current values for both health and water capacity, a value for their base speed and a value for their base damage. The input handling system will

control the movement of the actor (“FR_Move”), and whether/when it sprays water as in “FR_Attack”.

Alien Tower

Tower, the actor that forms one of the two primary enemies of the game. These are similar to the fire engines but have their own class planned so that they can be distinguished for purposes of targeting. They do not have a water capacity but do have a detection range value, within which they will fire upon the player’s fire engine (for which there is a fire method).

Alien Patrol

AlienPatrol, the actor that forms the other primary enemy of the game, roaming the world and initiating the minigame on contact with the fire engine. This has an attribute determining how long the fire engine must be in line of sight for the minigame to be triggered (as required by “FR_Minigame”) and a “hit” method which starts the minigame when it “spots” the fire engine and attacks it.

Projectile

Projectile, the class which represents any moving object which will strike another, like water sprays or tower missiles/lasers. It contains a damage attribute to indicate how much damage it will deal on impact and a speed attribute to indicate how fast it will move (if at all). It also has a “hit” method to handle the conditions under which it collides with an actor it considers hostile. “Projectile” has a further “Beam” class inheriting from it for the use of “TowerLaser” and “WaterSpray” (these are distinct for targeting purposes) which has an angle value that indicates where it is pointing and an “aim” method which changes that. Note that the InputHandling system changes this for the fire engine’s WaterSpray actor.

Minigame

In the minigame, there will be several aliens “popping up” on screen, either from the sides or from objects on the background; the player will need to move their cursor over them and click on them within a certain time limit or they lose.
“MiniWorld” inherits from the World class and adds a timer to control the length of the minigame. Within the minigame, there are only two extra actors; the crosshair (which is essentially a texture mapped to the cursor) and the targets. The targets take the form of “AlienTarget”, with a float telling the game how long they should stay on screen and method which make them appear, disappear and get hit. The “Crosshair” class simply has a fire method which will be activated by clicking the mouse or tapping the screen. It will be moved by the movement of the mouse, should there be one.