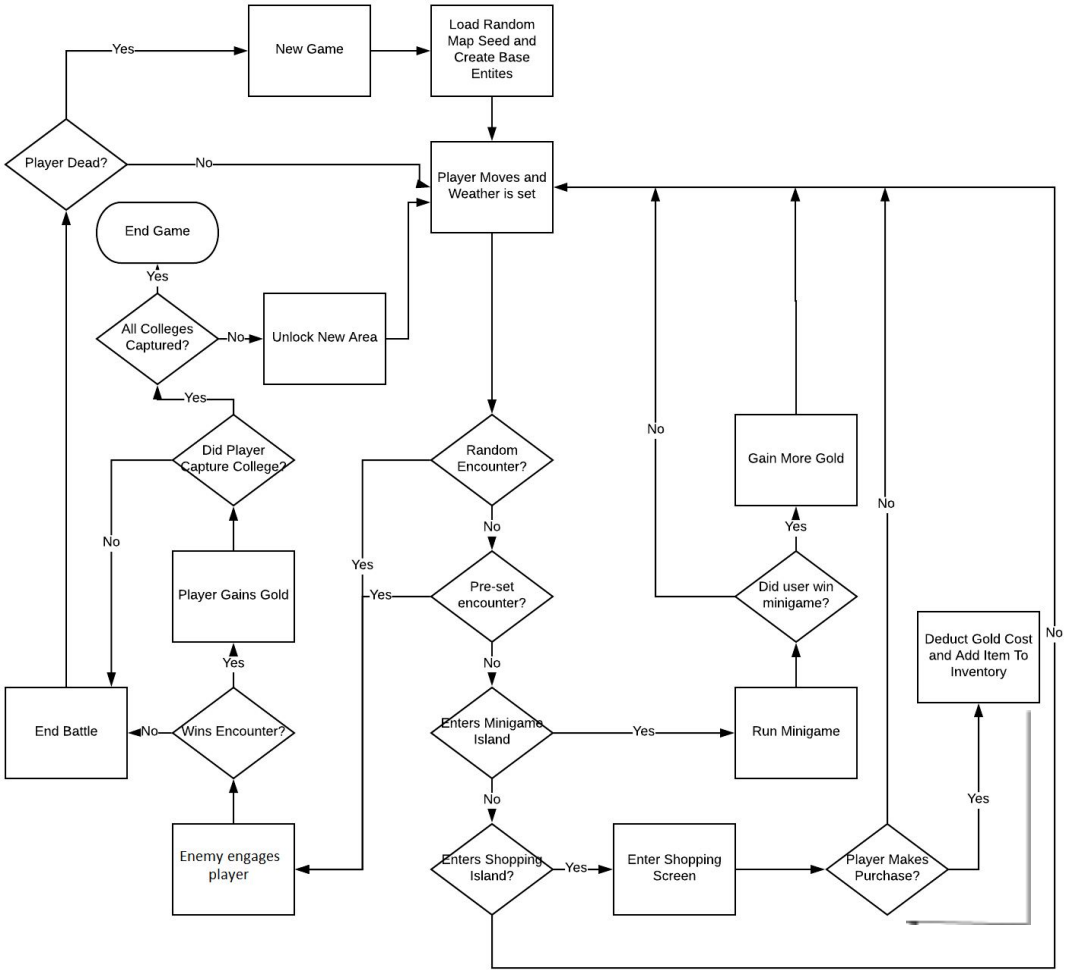
Team iPatch

Assessment 1: Architecture

Christian Pardillo Laursen, Filip Makosza, Joseph Leigh, Josh Wakefield, Tina Weng & Oliver Relph

Architecture

To begin in planning how we would structure our game, we made a behavioural model that simulates the game abstractly through a series of processes and user decisions which ultimately go from the start of the game to the end. This allowed us to see how the game should progress and elements that should be implemented as well as helping us to gain a better visualisation of possible game requirements. This chart was created with lucidchart.



Throughout our explanation of our simulated model, we will reference the requirements using the notation [Rx] to symbolise requirement number x.

Firstly, the game will open with the main menu showing, the user can then choose to start a new game when they wish to. Once a new game has been started, the software should initialise all base entities, including a UI showing gold, health and points [R3], and load a random map seed [R4] containing at least 5 colleges and 3 departments [R11], and an out of bounds area that the user cant enter in order to maintain linear gameplay [R13]. This allows for a different game scenario with every playthrough. After the game has been loaded and all entities are ready then the player is allowed to move their ship [R1]; everytime a movement is made the program checks if the user has

encountered a random enemy, a pre set encounter, the minigame island or the shopping island in that order.

If the user encounters an enemy they are engaged [R2], and the player has a battle with the enemy [R16], the same occurs with the pre set encounters however instead of a random enemy, we have pre chosen the enemy and this encounter often plays into the progression of the game. If the user wins the encounter then they acquire spendable gold [R7] and the program checks if the encounter was in fact the player taking over a college [R11], if not then the battle ends and the program checks if the player is dead [R8]. If the player did take over the college then the program checks if the user has control over all colleges, if so then the game terminates [R14], else the next playable area is unlocked.

If the user instead enters the minigame island, then our minigame window pops up and the user plays the minigame [R17], if the user ends up winning then they are awarded more gold which can be spent, if not however then they gain nothing and lose time which in turn reduces their score at the end [R6].

If the user enters the shopping island then the shopping pop up opens. Depending on which shop is entered the loot will differ however only certain items will be specialised, if the user decides to purchase an item then the gold cost for the item is taken away from the users inventory of gold and the item is added to the users inventory and any stat changes are applied [R9]. If the user does not make a purchase then the player is given the ability to move again.

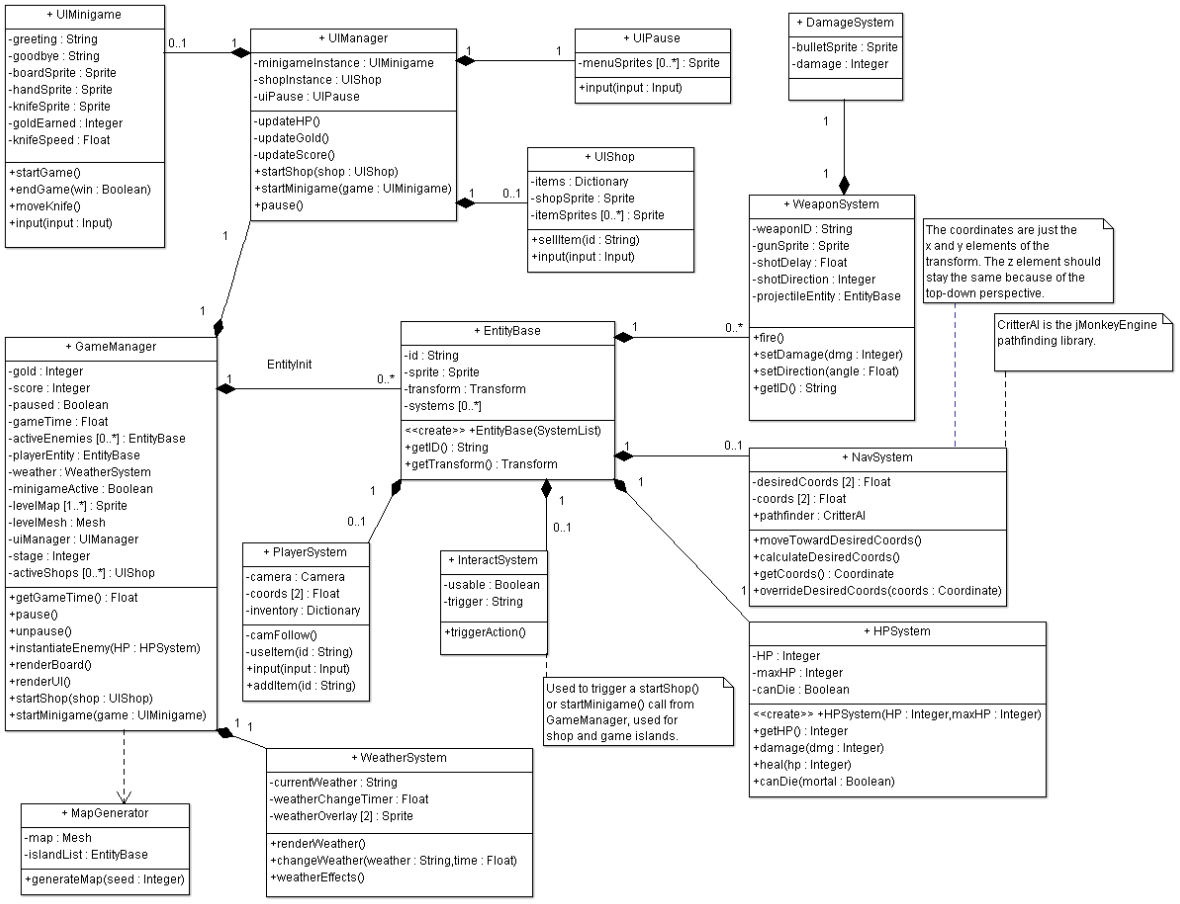
If there is no random encounter, nor pre set encounter and the user does not enter the minigame or shopping island then regular sailing continues.

In addition to this, every time the player moves, a weather parameter is checked and the weather can change accordingly based on this. The change in weather can alter the point multiplier for that area, for example stormy weather can generate waves which might damage the ship, in exchange for receiving more points during the storm [R14].

Class Diagram

The class diagram represents the general overall structure of the game, and was created using ArgoUML following UML conventions. The GameManager class is the primary class which is responsible for controlling the flow of the game. It is used to keep track of various globally relevant pieces of data, such as score, active entities, and the state of the game. It also stores an instance of the WeatherSystem and UIManager, and uses the MapGenerator class to procedurally generate game levels.

World elements are built around the basic EntityBase class. An EntityBase can be initialised with different Systems to change what it does within the game world, with the EntityBase class itself serving to provide a generic container for the entity's ID, graphics, and transform within the game space. These systems give the entity functionality like movement or a limited supply of health.



The WeatherSystem is instantiated inside the GameManager and is used to handle the weather visuals and effects within the program.

The UIManager handles everything on the game's UI layer, displaying health and gold when sailing and changing to shop and minigame overlays when necessary. It is also responsible for dealing with the pause screen.

Justification

The class diagram uses several nonstandard types, like Mesh and Sprite. These represent abstractions of classes and libraries provided by the jMonkeyEngine framework.

The class structure of the project was focused around composition, influenced by the entity-component-system architecture design philosophy often used in the modern game development industry.[1] ECS is very powerful and flexible, but it isn't necessary for a project of limited scope like this one. Because of the relative simplicity of the game, we chose to focus on composition of (mostly) self-contained blocks of functionality, with each such block being referred to as a system. Game entities are created using the EntityBase class and a set of Systems to describe its functionality. For example, a stationary indestructible island with a single cannon which fires at the player would be represented by an EntityBase with a WeaponSystem and a specialised AI system that implements its firing behaviour. An enemy ship with a single cannon would be represented by an EntityBase with a WeaponSystem, HPSystem, NavSystem, and a specialised AI system that implements its firing behaviour. Focusing on composition from self-contained blocks allows for greater flexibility compared to traditional inheritance, as new systems can be implemented freely without having to be concerned about it breaking existing entities. It is also easier to implement

functionally distinct entities, as enabling a piece of functionality comes down to adding a relevant system to the entity's System List. This makes it easy to avoid messy and complex inheritance trees caused by having many similar but not identical entities, even if they're implemented relatively late in development.

While most of the composition takes place inside entities, some game functionality is divided into the WeatherSystem, MapGenerator, and UIManager. It wouldn't make sense to attach any of these systems to a regular game entity, so they are used directly within the root GameManager class. The WeatherSystem controls environmental weather effects and the UIManager handles GUI rendering and input. This has the same benefits as composition for game entities, where new systems can be added relatively easily without breaking how the other systems interact with the GameManager. The UIManager can be extended in turn to implement new features on the UI layer. The MapGenerator is only used to generate a level at the beginning of the stage and it does not store any data, so there is no need to keep it instantiated. The GameManager simply calls the static map generation method when necessary.

The Behavioural Diagram uses industry standard notation for processes, decisions and termination and follows a set structure which describes the general simulation of the game.

The game must start at the main menu as that's what is shown when the game is first launched, after opening a new game the only logical process is to then load in all the beginning entities as well as the map as these are essential for the game to be playable. After the game is in a playable state the player is given free rein to make a move as otherwise there is no game progression. Once a player moves the game should offer 4 options: a random encounter, a pre-set encounter, a shopping window or the minigame, the behavioural diagram supports this and goes through a series of decisions asking whether any of those criteria have been met. This is required as otherwise a random encounter might not occur when it is supposed to etc. After an encounter has been triggered the player must enter battle and either win or lose as the 2 outcomes, if the player loses then they must lose health as per game rules and if their health drops below 0 then they are dead and the game is over also as per game rules. If they win an encounter then there are 3 possibilities, they defeated an enemy, a college or the final boss (final college), all 3 options give the player gold as shown in the diagram however, defeating a college unlocks a new area and defeating all colleges completes the game. All these scenarios are shown in the diagram as natural game progression however all but completing the game loop back around to the player turn as that is how we wish for our game to play. Everytime the loop is complete, i.e. a new player turn, the weather parameter is checked and the weather possibly updated in order to satisfy our requirement for different weather giving a different point multiplier.

If the decision for an encounter returns no, then the behaviour for the game should also check if the user has entered a shopping area or a minigame. If the user did enter a shopping area then the game must give the user the option to purchase an item or do nothing before returning to the player move process. The process is the same for the minigame which also opens and gives the player the option to play before returning to the player move process.

If none of the decisions return a yes result then nothing happened after the player moved and so the diagram reverts back round and allows the player to move again. In general the diagram shows the

abstract flow of the game from start to finish without describing too much detail and allows for us to follow in order to make sure our game meets all the requirements and also runs as its supposed to.

[1] M. Fox, “Game Engines 101: The Entity/Component Model,” *Gamasutra*, 12-Aug-2010. [Online]. Available: https://www.gamasutra.com/blogs/MeganFox/20101208/88590/Game_Engines_101_The_EntityComponent_Model.php. [Accessed: 04-Nov-2018]