# Architecture Report

## Architecture

| Package | Class | Description |
|---|---|---|
| **Attacks** | **Attack** | This class holds the base information for an attack. Everything else in the Attacks package is an instance of attack (apart from flee) as they are all different variations of the default attack. This contains the variables to be inherited by other instances to create different attacks with differing damages. |
| **Combat** | **Flee**<br><br>**Ram**<br><br>**GrapeShot** | Inherits from attack. Random number generated to decide if the flee is successful, if the number is not high enough, the user will stay in combat.<br>This attack inherits the default damage from the attack class and adds a multiplier to it, making it a variation on the basic attack.<br>This attack inherits the default damage from the attack class and adds a multiplier to it, making it a variation on the basic attack. |
| | **MyShip** | This class holds all the information about a player's ship. It contains information such as the type of ship, what college the ship is aligned to, the current upgrades the ship owns and more. |
| | **ShipCombat** | This is called whenever there is combat in the game between two ships. This creates the new screen containing the two ships ready to battle, their health bars and all the different attack options the user has. |
| | **College** | The three colleges made are Vanbrugh, Derwent and James. This holds the name of the college and who it's allies and enemies are.This also controls whether or not the boss of this college has been defeated. |
| | **GameObject** | This class is just the generic object class, that all other objects extend from in order to make a new object, this contains all the building blocks needed to set up any time of object inside of InteliJ. |
| | **MainMenu** | Main menu is the first class called when the game is initialised. It gives the user the option to either enter the sailing mode of the game or the combat mode. This has been done to easily show clients what the game is capable of but eventually we hope to have it changed to "load game" and "new game" options in order to allow users to save their progress or to start the game new again. |
| | **MoveableObject** | This class extends from GameObject but allows the object to move also. This is necessary for the player's ship in sailing mode. |
| | **PirateGame** | PirateGame is the main class for this program. It calls the MainMenu and initialises it. This class holds all the necessary variables and calls to start the game up from the beginning. |

| | | |
|---|---|---|
| | **Player** | Player stores all the player's information for a single playthrough of the game. This includes their current points and how they will be incremented. |
| | **Ship** | Ship creates a new instance of a ship.This is used to create new ships and enemy ships. There is the groundwork in here to allow the user to select a specific boat type at the beginning of a playthrough, but for now there is just the standard "brig" type of ship that all enemies are also. |
| | **ShipSailing** | This class contains the entire sailing mode. This creates the landscape used when the user is sailing, including creating islands and loading in the graphics of the ships. This holds all the control options used to steer the ship and eventually call the combat mode when an enemy ship is met. |
| | **ShipType** | Despite not currently being in use, ShipType holds the 3 different ship types the user can choose between at the start of the game. However, this was not necessary to program just yet but it was important to keep the option there for further improvements to the game. |
| | **Department** | Chemistry and physics are the two chosen departments in our game. They have a name, a product (which the user can buy to upgrade their ship) and the base price of the product that they sell. |
| | **AbstractScreen** | This class controls the changing of screens inside the game. This dictates what screen is present during each class. |

**Attacks** - This package holds all the classes of available attacks. This is not required but is a good programming habit as it kept our code readable and tidy for the future when other people may be using our code.
**Combat** - This package contains all the information needed when the player is in combat. Again, this is not required however we found it good practice in order to keep our code readable and reusable.

## UML Diagram
Please find our collection of UML diagrams referenced at [1], [2], [3], [4] and [5]. These in turn show how the entire program links, and then how each separate package works.

## Tools used to create concrete architecture
To create the most realistic yet readable concrete architecture, our group opted to use a plugin for IntelliJ (The IDE we used for our project) called Sketchit. This plugin generated UML for each package inside the plantUML syntax. Using a plugin from our source code meant that all classes and functions were named and represented at 100% accuracy which is required when developing a concrete architecture.

## Description of languages used to describe architecture
The languages used to describe our architecture were strictly the standard UML 2.0 notation. We chose such a modelling language as we knew that in software development, this is the most popular notation, meaning that other groups can easily understand our program if only given our UML diagrams.

## Justification

| Class | Justification (Requirements referencing can be found at [6]) |
|---|---|
| **MyShip** | This class allows us to meet requirement **F1** (ships as transportation) as the object has methods ('setMoving' and 'turn') that allow ship movement on the map. |
| **Attacks** | Attacks is necessary in helping build the combat system required in **F4.** This is a simple way to move attacks into their own class, making it easier to add and remove new attacks. |
| **ShipCombat** | Having an ShipCombat class allows us to meet all of criteria **F4**, more specifically **F4.2** as this dictates that there should be a difference between sailing mode and combat mode. |
| **GameObject** | Creating an object is necessary for eventually making moveable ones, such as the ships. |
| **MainMenu** | Main menu is needed at the start of the game so the user can choose between creating a new game or loading a previously saved one. This is not required for the current assessment however it is ready to be implemented when needed. |
| **College** | College allows us to meet requirement **F8**, of capturable colleges once a player defeats the college boss in a battle.<br><br>The class also allows us to meet the requirement **F5** (game must have at least five colleges) by its inclusion.<br><br>Having a college allows the player to repair their ship if it has been damaged, this meets the requirement **F11.2** (plunder can be spent on healing). |
| **PirateGame** | This class loads in the game and initialises it. This is necessary when creating any game. |
| **MoveableObject** | Making objects moveable (such as the players ship) was vital in meeting **F3** (the game must have a sailing mode) as it would be impossible to create a sailing mode without the ship object being able to move. |
| **Player** | This class allows us to meet the requirement **F6** (point system), as the method 'addPoints' allows the points attribute in the Player instance to be modified whenever necessary.<br><br>This classes 'money' attribute also allow requirements **F7** and **F11** to be met via the 'addMoney' method. |
| **Ship** | The ship class is needed for sailing and combat. Without there being a ship made for the player, it would be impossible to implement sailing or combat. |
| **ShipType** | Despite not being needed yet, ShipType is ready to offer the user different types of ships to play as. |
| **ShipSailing** | This allows us to meet the functional requirement **F3** (sailing mode). |

| | |
|---|---|
| **Department** | Department class allows requirement **F11.1** (upgrade ships) to be met as there is a ‘purchase’ method.<br><br>The class also allows us to meet the requirement **F5** (game must have at least three departments) by its inclusion.<br><br>Having a college allows the player to repair their ship if it has been damaged, this meets the requirement **F11.2** (plunder can be spent on healing). |
| **AbstractScreen** | AbstractScreen allows requirements **F4.2** which states that there should be a clear difference between sailing mode and combat mode. This class switches screen completely during the two different modes. |

## Changes From Abstract Architecture

Since writing our abstract architecture [7] , we found changes where we could slim our code down in order to make it more readable and traceable. These changes were then implemented to create our final concrete architecture. Here are the changes we made:

1. Because of how tiled and libgdx works, we decided to hard code islands into the map, rather than having an island class that department and college can inherit from. Also department and college have nothing in common meaning that having them inherit from the same class is almost pointless as there is no shared code between the two.
2. Got rid of enemy. Enemy was a redundant class as it could just be made from a single ship. This meant that the entire enemy class was useless and could be replaced with creating a new instance of ship, which is a much better programming technique. This allowed us to trim down our code and make it more receptive to new changes if more enemies need to be made.
3. We initially planned the ship to have a function of attack. Now we have an attack class with a “DoAttack” function which calls the specific attack the user selects. This gives a lot more flexibility to programming new attacks and allowing the user to unlock new attack styles.
4. Previously, we had planned for there to be just 8 classes; Player, Ship, Enemy, Island, Department, College, Minigame and weather. However, the specification said specifically to not go over the stated requirements, meaning the weather was no longer needed in this version of the game.
5. Sailing was implemented into the game which was not planned in the abstract architecture. We felt it necessary to add this feature as it would be too difficult to have colleges, combat and department without an easy way to get between them. The only other alternative would have been to make a menu based game but our group found that to be too basic and unoriginal.

## References:

[1] SEPR “Concrete UML for Core” Rear Admirals [Online] Available:
https://therandomnessguy.github.io/SEPR/Images/Ass2UML/Core.png
[Accessed: Jan. 20 2019]

[2] SEPR “Concrete UML for York Pirates” Rear Admirals [Online] Available
https://therandomnessguy.github.io/SEPR/Images/Ass2UML/york_pirates.png
[Accessed: Jan. 20 2019]

[3] SEPR “Concrete UML for Attacks” Rear Admirals [Online] Available:
https://therandomnessguy.github.io/SEPR/Images/Ass2UML/Attacks.png
[Accessed: Jan. 20 2019]

[4] SEPR “Concrete UML for Combat” Rear Admirals [Online] Available:
https://therandomnessguy.github.io/SEPR/Images/Ass2UML/Combat.png
[Accessed: Jan. 20 2019]

[5] SEPR “Concrete UML for Screen” Rear Admirals [Online] Available:
https://therandomnessguy.github.io/SEPR/Images/Ass2UML/Screen.png
[Accessed: Jan. 20 2019]

[6] SEPR “Assessment 1 Requirements” Rear Admirals [Online] Available:
https://therandomnessguy.github.io/SEPR/Assessment/1/Req1.pdf
[Accessed: Jan. 20 2019]

[7] SEPR “Assessment 1 Architecture” Rear Admirals [Online] Available:
https://therandomnessguy.github.io/SEPR/Assessment/1/Arch1.pdf
[Accessed: Jan. 20 2019]