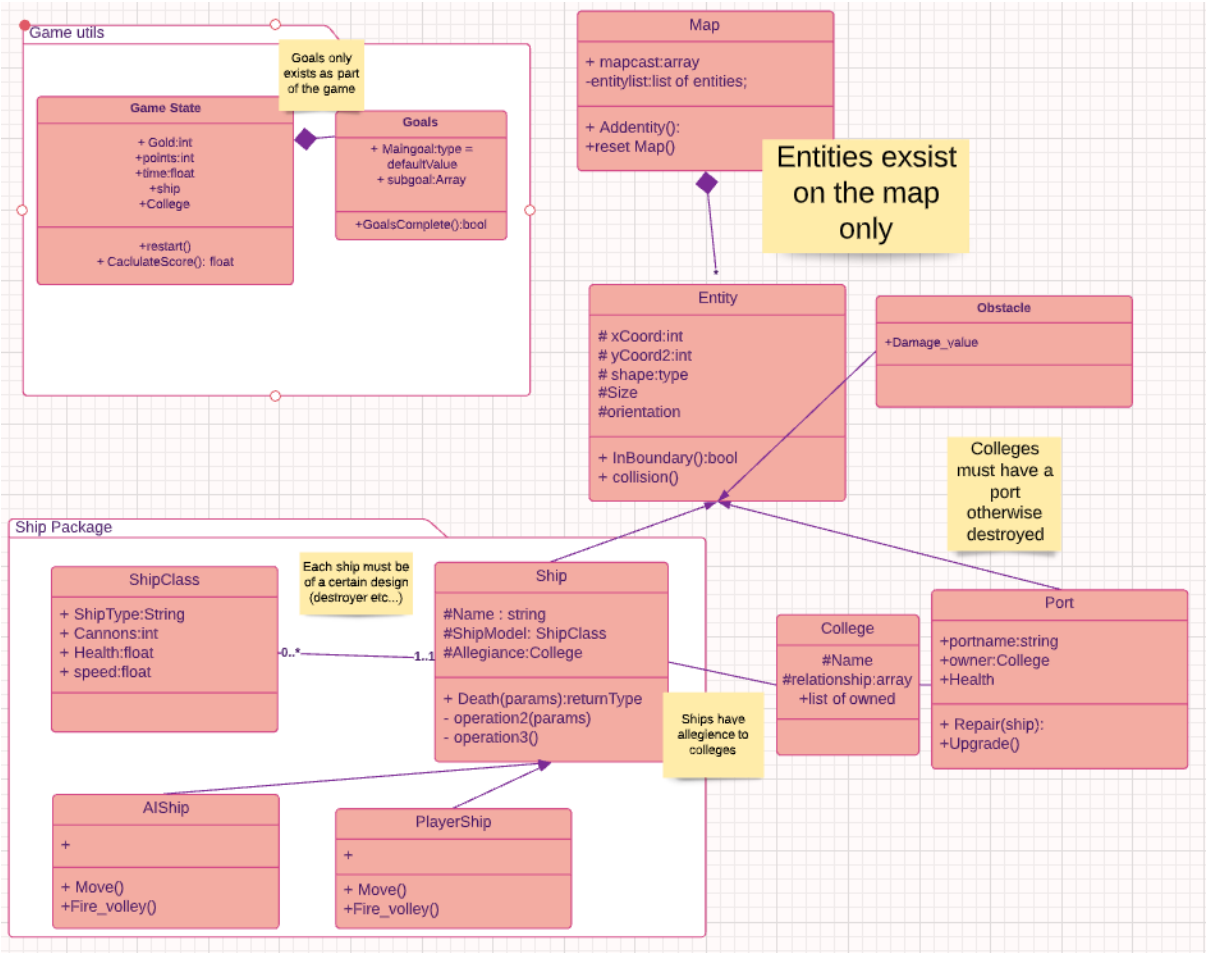
Architecture

Cohort 1 Team 9 - Team ‘ARRRGH’

Dominic Hall
Firas Marzouk
Ben Morrison
Harry Whittaker
Amelia Wigglesworth
Zehang Li

3. (a)

Below is the abstract representation of the architecture for the project. It is split into sections, as per different parts of the game development. Using diagram.net, we produced a rough outline on all the pieces that needed to be coded, with descriptors to indicate what factors are contained only within the class or accessible to instances of other classes (with '-' meaning private, '#' meaning protected, '+' meaning public), as well as how they relate to other parts of the code.



There is a UML package for game utilities, which contains the goals and the game state class, mainly involving the text shown in the game. Within the game state class, the attributes are the public values of gold, points and time, with the additional use of the public ship and college classes, which must be used to identify which college and ship are in use. The operations within game state include a restart operation and an operation to calculate the score. 'Goals' contains the main goal as well as subtasks, with a single operation for goal completion.

Ship package is the second package included, with ShipClass, Ship, AIShip and PlayerShip. ShipClass includes the attributes of ship class, cannons, health and speed, which would all be able to change based on an upgrading system. The main ship class includes the name of the ship, model based on 'ShipClass' and allegiance to a college. It includes an operation for death. AIShip inherits from Ship includes an AI for enemy ships to continue moving and attacking as if they were another player. PlayerShip inherits from Ship, as a class for the ship controlled by the player.

Additional classes exist for other sections of the game, including the Map class and Entity class defining positions and directions of entities, as well as collision effects. Entities are allowed to exist on the map only. All obstacles, the Port class and Ship class inherit from the Entity class. The College class exists to give ships allegiances, and colleges are assigned to Ports, which must allow upgrades and repairs.

In the concrete version of our architecture, some aspects of the game had to be built upon. We required some extra classes to ensure the game met the customer's requirements. One such example is a tutorials class, which exists to help the user learn the controls.

As per the customer requirements, we added the choice to select a college at the start of the game. Furthermore, we created a class for a title screen and end screen, as it created an authentic experience of starting and finishing the game.

A major change between the concrete and abstract representations is the need to work around a library. This can be primarily seen in Screens which we had to implement.

Most major classes have been separate files within the game folder, as shown in the image above.

- ‘Boat’ contains the boat class, which involves all the animations for movement and the camera positioning based on where the boat moves. There is also a sunk ship function in this file, to allow the game to end if the ship runs out of health.
- ‘AiBoat’ includes code to move ships around, as well as the addition of a class where if the ships collide into walls, the direction of travel reverses.
- ‘Castle’ class creates bases for the colleges using textures from entities saved elsewhere.
- ‘Entity’ ensures all entities are a certain size and if they need to, stay in certain positions across the map.
- ‘Enums’ involves the types of land across the map, as well as specifying types of entities and boats. This class also indicates relationships between ships and colleges within the game.
- ‘Hud’, as before, includes the timer, gold, points, and the added value of health. The Port and Neesa classes include the creation of bases across the map.
- ‘UserInputManager’ allows the player to control their ship, and controls the shape of the screen.

3. (b)
All items included in the abstract architecture have been developed into the concrete architecture, with some items being added exclusively to the concrete architecture.

The concrete architecture builds upon the abstract by adding helper classes and merging them together. For example we combined the game state primarily within the HUD as this is where the variables were needed to display to the user.

As with the abstract architecture, the concrete architecture includes game utilities, with added utilities for a better player experience. With the addition of health, players are able to view how much damage they have taken and decide upon whether they should continue battling or stop at a base and heal.

- CollageSelector, a file within the Screens folder, meets the requirement of UR_TEAM_SELECT, where upon starting the game the player is able to choose between 4 collages to align with.

- Campus_east, a file inside the map folder, meets the requirement of UR_MOVE_PLAYER in allowing the player to move on water, but not land, with hitbox registration ensuring no boat can move onto areas of land

- UR_REPAIR requirement is met within the Hud file, ensuring players are able to repair their ship when docked at a pier.

- UR_END, where the game ends when the player's ship sinks, is within the Screens folder with the EndScreen file. Ship sink mechanics are located in the boat class, additionally.

- Tutorials, found in the tutorials file, meet the UR_TIPS requirement, giving tips to players as they progress to guide their gameplay.

- A desired map layout was met, as per NF_MAP, which can be found in the map folder.