Bermuda Digital Entertainment (Team 7)

Architecture

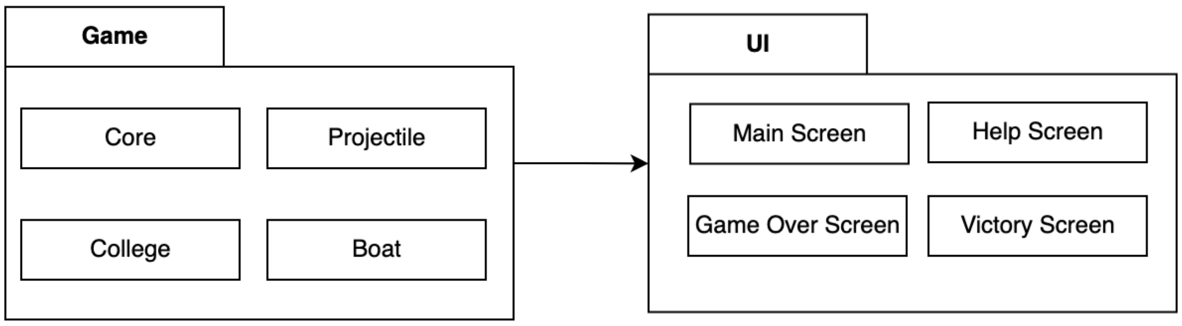
Sebastian Sobczyk, Aymeric Goransson, Joseph Sisson, David Ademola,
CJ Donoghue,, Prajwal Binnamangala,

Architecture

For all software architecture representations, draw.io was used to create and develop the diagrams.

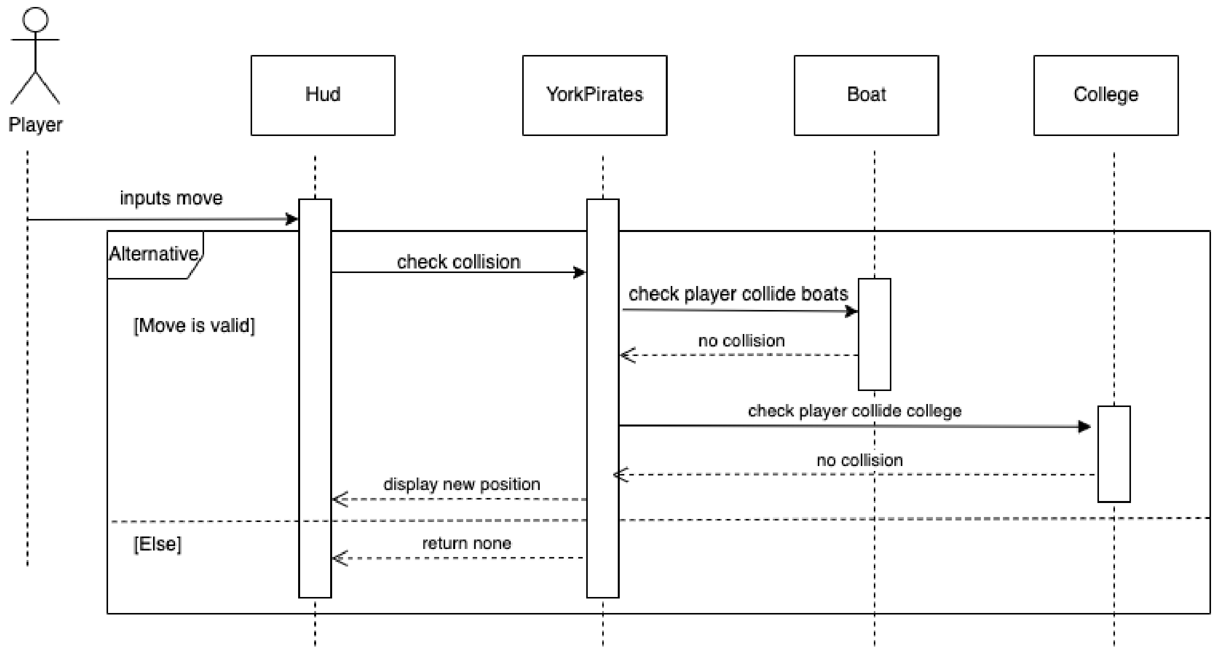
Abstract Software Structure

Package Diagram



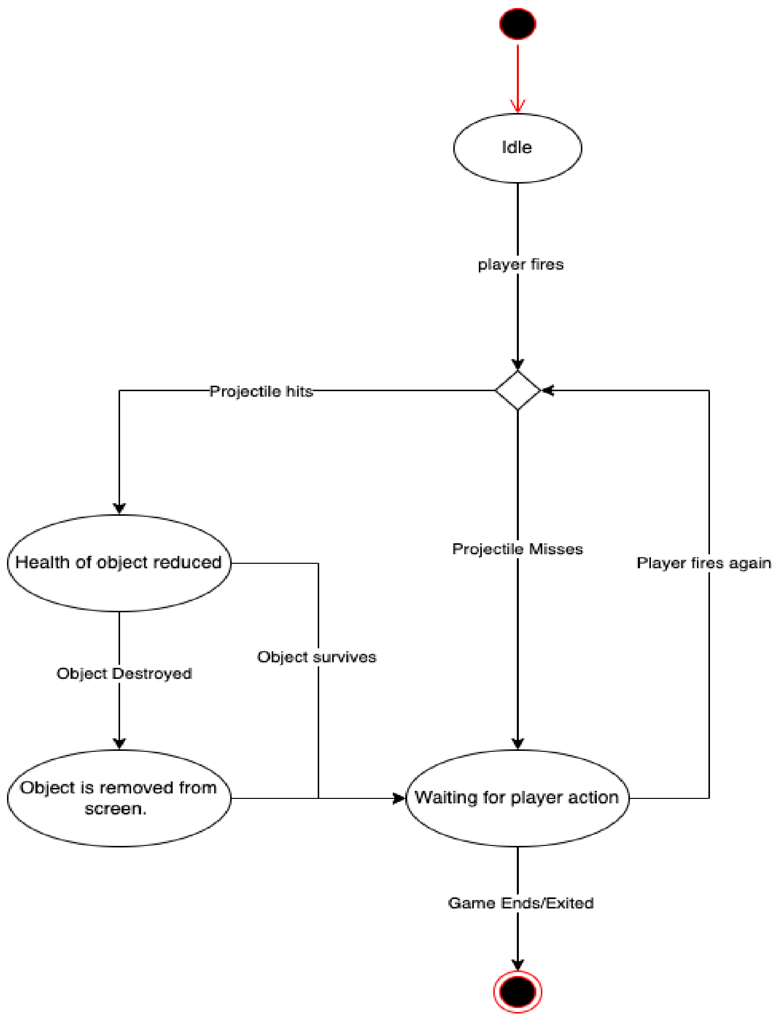
This is a package diagram of our initially planned software structure where there is a dependency relationship of a game package and a user interface package and the classes within the packages.

Sequence Diagram of Player Movement



A sequence diagram of a simple move command input by the player just to give a general understanding of the communication between the UI and game core when there is a player input.

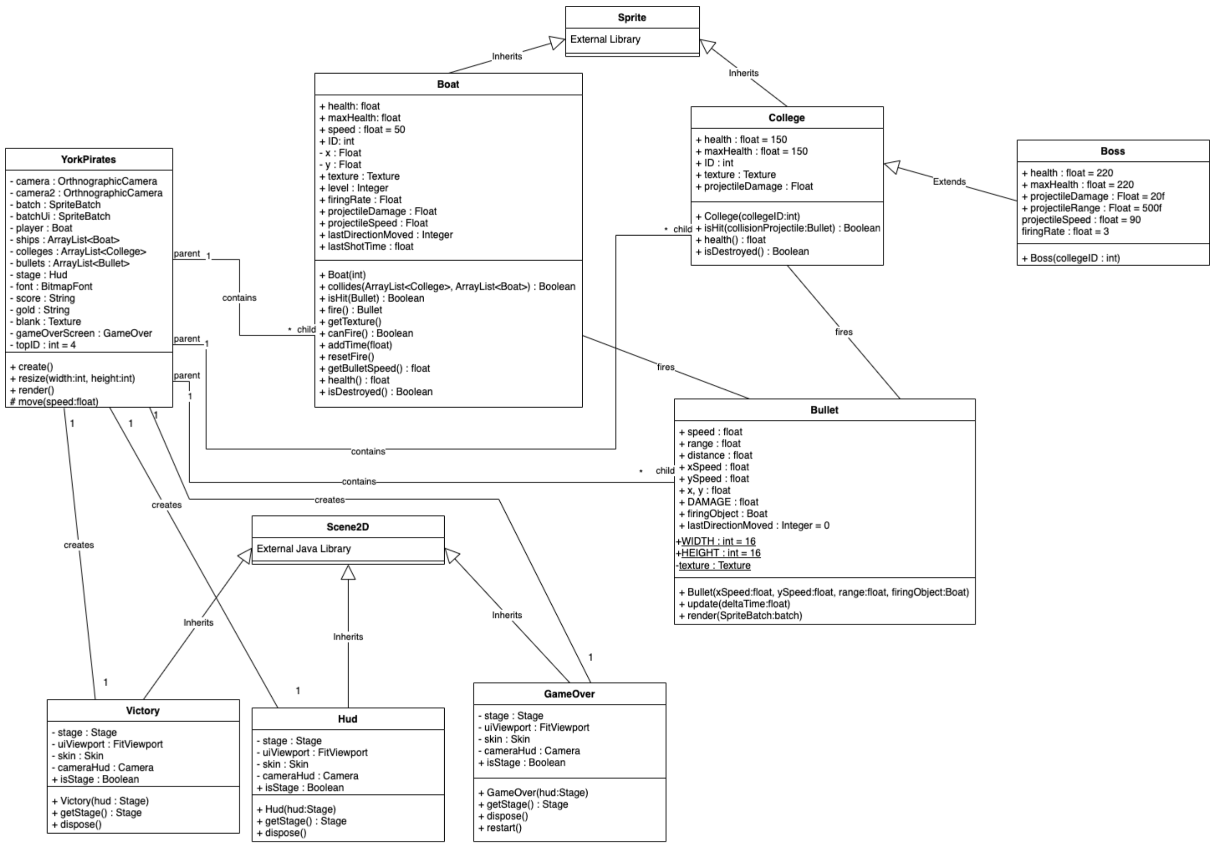
State Diagram of Player Firing



A UML state diagram of a player firing during the game and the possible states the system may enter during this process. A general structure for all actions the player can do such as if the player can move to a certain position or collides with an object.

Concrete Software Structure

Class Diagram



A UML class diagram was created to represent the relations of the classes in the YorkPirates package.

Software Architecture Development

The original package structure was going to be 2 main packages called UI and game core to separate the back end and front end of the game more clearly for implementations sake. Upon implementation it was then decided to place all elements of both packages into a singular package core (com.yorkpirates.game) to allow for easier compiling and testing with gradle. However, as seen in the concrete class diagram the UI classes are still disjointed from the game classes and only connected through the YorkPirate class so the structure of the package diagram was still kept to a certain degree. This was to keep all instances of the classes within one class instead of having the UI needing to request data from the YorkPirate class in order to get data of another class since the game is built to be around 5-10 minutes the YorkPirate class will not be overloaded with class instances. The same classes were kept in the concrete architecture and the package diagram with a few more additional classes that were external or added later on such as the Controls class.

In terms of the abstract behavioural models, the ingame system process kept it similarly to the sequence diagram but with slight changes as the class YorkPirates contains the instances of the Boat, College and Bullet classes and directly communicates to them when the player inputs a control. As mentioned before it was to make it easier for communication between the player actions and game classes as all instances of any class is kept in one main class rather than having instances of classes separated removing the need of a UI class.

Additionally in the state diagram the process is kept the same as the Boat class will use the fire() method when the player fires and classes Boat and College share a similar method isHit() to check whether the object has been hit and will remove health from them. It will then use the isDestroyed() method to return if the object has no more health and sequentially removing the object.

Software Requirements

In the 1.0 requirements, it is clear that there is a fully functional “York Pirates” game as the player is capable of moving around the map. Overall the game should only take 5-10 minutes to complete unless the player does not attack any of the colleges and plays passively (1.1). Capable of seeing the entire map when maximised and takes less than 10 seconds to travel across the map (1.2). There will be objectives of destroying all colleges and a boss college in order to win the game (1.3). There is no story (1.4) and no progress saving so once the game is exitted there is no returning to the same game (1.5).

For requirement 2.0, there is real-time combat against colleges as there are 3 colleges that are created on the map in the create() method in YorkPirate class (2.1) and the player is able to use the fire() method in class Boat to create a Bullet instance to attack colleges in which the College class will use the isHit() method to reduce their health if hit (2.2). The colleges are then able to return fire with the fire() method to fight back the player (2.3). The fired cannonballs are visible to the player and are given a set speed so that they are not too fast so that the player can dodge them (2.4).

There is a scoring system so that when a college has been defeated the points are added to the player’s score and shown to them on screen 3.0.

As soon as the game begins, in the top left corner of the screen the objective of the game is displayed in solid pixel white text it would say “Destroy 3 colleges” as one of the objectives (4.0).

The art style is an old arcade game with bright visual colours and pixelated art it was chosen so that the player feels more visually attracted to the game due to nostalgia(5.0).

The screen is able to scale from 13 inches to 27 inches however it might make it harder to play with a smaller screen (6.0). The game allows screen/camera scrolling (6.1) and is only compatible with PC/Laptops (6.2).

Simple keyboard controls the player uses WASD to move around and SPACEBAR to fire a cannonball (7.0).