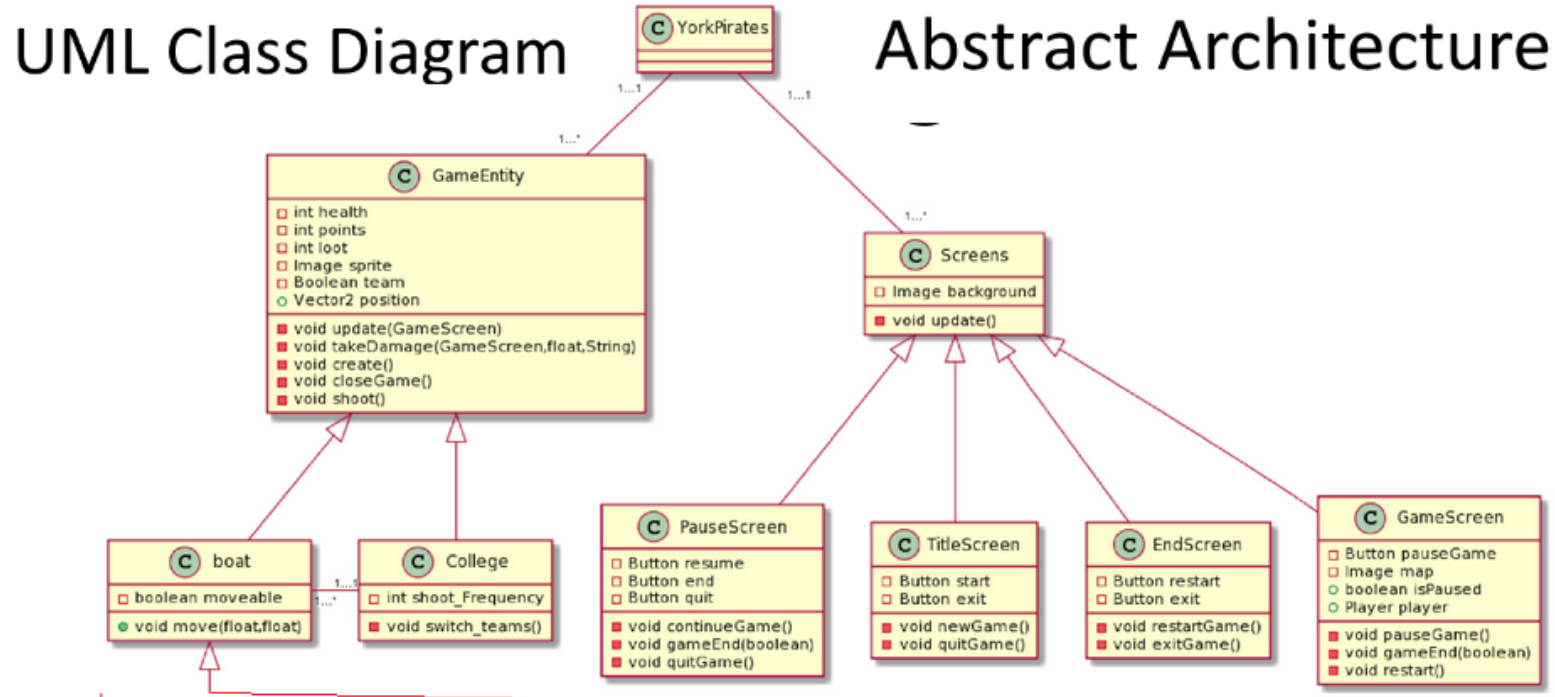
Architecture

Group 11 - 11 Musketeers

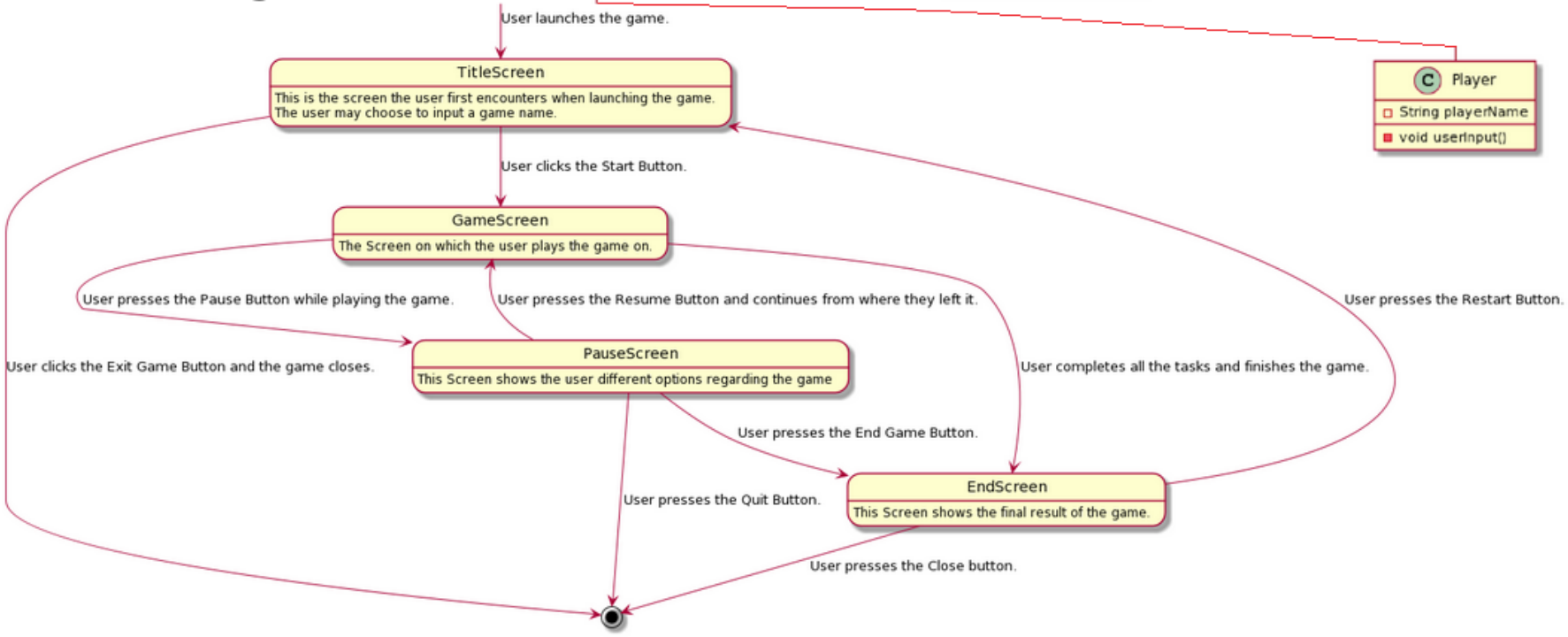
Osama Azaz
Adam Dawtry
Tom Jackson
Brendan Liew
Holly Reed
Harry Ryan

UML Class Diagram

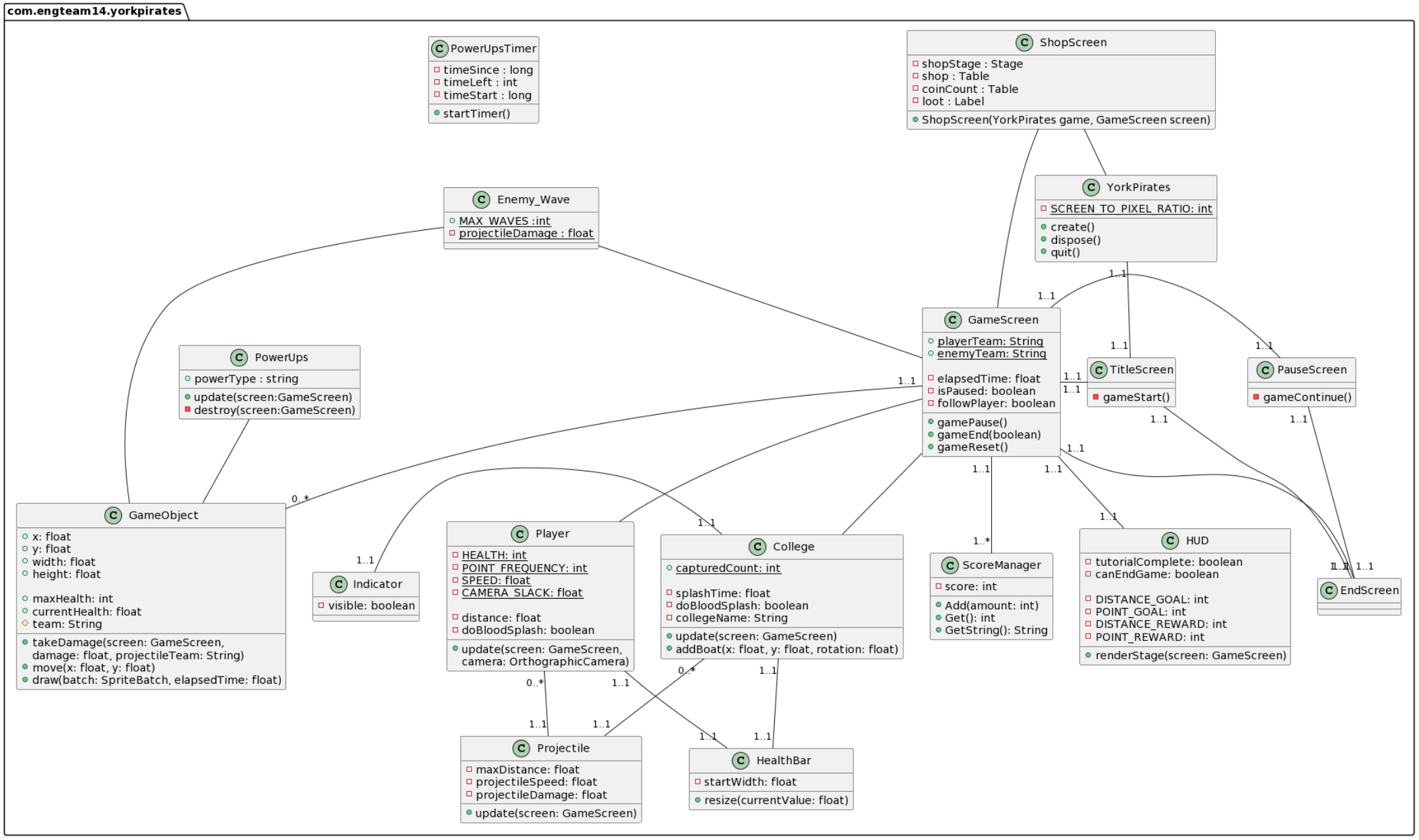


Abstract Architecture

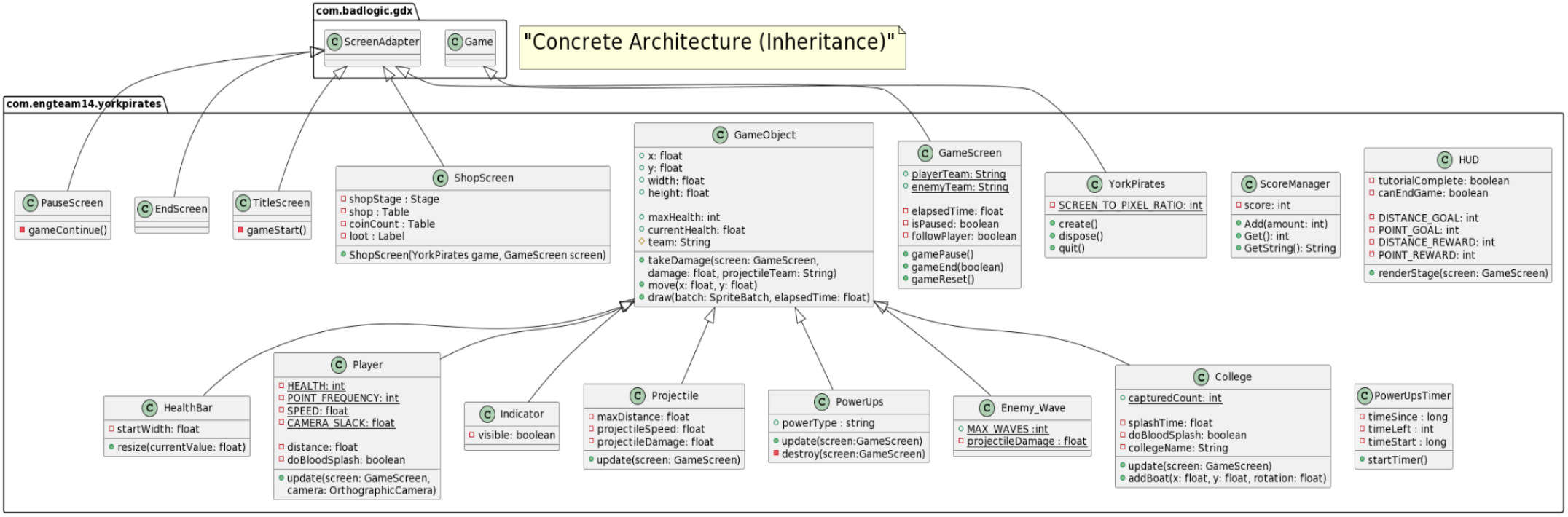
State Diagram



[2] https://engteam14.github.io/media/State_Diagram_4.png, [1] https://engteam14.github.io/media/Abstract%20Architecture.png



Concrete Architecture



We used the classes part of PlantUML to create class diagrams for the concrete and abstract structure of the project to show inheritance, in addition to state diagrams for the sequence of events that would occur throughout the course of our application's use using 'state' in PlantUML .We used intellij for syntax highlighting and for rendering the images. We simplified and removed getters, setters and some utility and instance attributes.

3.(b) Justification of Abstract Architecture

For our **abstract** architecture, we made two main classes, **GameEntity** and **Screens** which both have the method **update()** which is to perform calculations before each entity/screen is rendered.

- **YorkPirates -** The main class of the game
- **GameEntity**
  - The class that every object within the game scene is an instance of. Implements health, rendering, teams, and shooting projectiles. As these are features all objects use, having a base class implement them is important.
- **Boat** and **player**
  - **Boat** inherits all attributes and methods of **GameEntity** and **Player** inherits both. **Boat** additionally has a boolean attribute of **'movable'** which decides whether the boat can move or not. This is because for this stage, the enemy boats can't move but the friendly one can. However in future some Enemy boats may become movable but some could stay docked.
- **Colleges**
  - Inherits all attributes and methods of **GameEntity** but also has a method called **shootFrequency** to set how often it shoots on its own and a **switchTeams** method for switching the images and turning off shooting/being shot by the player, when the player captures it.
- **PauseScreen, TitleScreen** and **EndScreen**
  - Each has a different set of buttons needed for its screen and are child classes of screens to use its render method and all are child classes of screen to use the same background and update() method for calculations before rendering.
- **Game Screen**
  - When the game is restarted, a new instance of this is created so that the game doesn't have to be fully restarted and also has the methods for restarting, pausing and ending the game but is also a child of the screen.
- **Texture Handler**
  - When the game is started , this loads in the textures from the assets of our sprints such as map,cannonball,boats ,colleges and more.After the game is closed, the dispose() method helps clear off the textures that were used in the game .

Concrete Architecture

- **YorkPirates**
  - Due to the structure of LibGDX, we had to make a main Game class. This matches what we planned in our abstract architecture to an extent but screens are actually child classes of **ScreenAdapter** and YorkPirates instantiates TitleScreen and then switches between the others.
- **TitleScreen, EndScreen, PauseScreen**
  - These classes are extensions of the ScreenAdapter class and render their screens with Buttons and overlays on the paused instance of GameScreen.These classes fulfil the requirements: UR/FR.START_SCRN and due to TitleScreen, UR.SCRN_NAME / FR.START.NAME due to the ability to add a name on titleScreen, UR.RESTART_GAME due to being accessible by the pause and endscreen menu, FR.START.START and FR.START.EXIT due to the TitleScreen, FR.KILL_SCRN due to the EndScreen class and FR.GAME_SOUND due to the mute button on the PauseScreen.
- **GameScreen**
  - This class is the main gameplay environment, containing and rendering all instances of the objects within the game, which meets requirement UR.SEE_POS. Furthermore it has the methods for pausing the game with gamePause(), ending the game with gameEnd() and restarting the game with gameReset(). We put those methods in this class because every other class that needs these has access to an instance of this class.
- **HUD**
  - This new class was added for readability to avoid clutter in the main GameScreen class. This improved readability has made it much simpler to implement UI features such as tutorials, tasks and viewing points/loot, meeting UR.TUTORIAL, UR.SEE_TASKS, UR.VIEW_PNTS and

UR.VIEW_LOOT. Furthermore as it is separate it has given the ability to turn off rendering for it so that a different screen can be overlaid on the GameScreen without the HUD being visible.

- **GameObject**
  - Every object in the game is an instance of **GameObject** where ones with seperate functionality are a child class of **GameObject**. This is so that common attributes and methods such as currentHealth, takeDamage and position within the world (x, y) are shared among all objects. This class is similar to how we described in the abstract architecture however we encapsulated loot and points in **ScoreManager**.
- **ScoreManager**
  - **ScoreManager** is used to keep track of loot and points.It also encapsulates the values, which in the case of points makes it easier to update the points value from the **Player** when they move() or the loot value from the **College** when it is defeated, meeting UR.COLLECT_POINTS and UR.COLLECT_LOOT.
- **College**
  - College is a child class of GameObject with the further features that it has **Projectile**s and a **HealthBar** and **Indicator**. This is in a separate class as it shoots automatically rather than through user input like Player. This functionality was extended between abstract and concrete architecture by the addition of instances of **HealthBar**, **Indicator** and **Projectile**.
- **Player**
  - In the abstract architecture, **Player** was a child of **Boat** because **Boat** allowed movement. However we decided to put the movement method into GameObject because **Projectile**, **HealthBar** and **Indicator,** also needed to be able to move and so therefore we could use the move() method for all of these, as well as in future, moveable enemy boats. This ensures we still meet the requirement UR.UPDATE_POS.
- **HealthBar**
  - **HealthBar** was not in our abstract, however we realised the **HealthBar** was needed for both the **Player** and the **College** and so to save us from code repetition we made **HealthBar** into its own class. This will also make implementing enemy boats in the future easier.
- **Projectile**
  - In our abstract architecture, shooting was implemented as part of **GameObject**, however as we now have more objects in the game and not all of them shoot. Having all objects do this would be inefficient so we moved it into its own class, which **Player** and **College** both use, allowing UR.ATK_CLG to be met.
- **Indicator**
  - In our abstract implementation we did not have a method which allows the user to see where they are relative to the colleges (UR.CLG_POS). This is why we added **Indicator**s, these draw arrows showing the player which direction each college is, fulfilling the requirement UR.CLG_POS.
- **Sound Manager**
  - The class contains all the sound effects for the game such as cannons fired by the Boat,the death sounds when the player dies and more, which fits the requirements for FR.GAME_SOUNDS.Under TitleScreen,there is a function which includes the button to able to toggle on and off the sounds of the game on the menu when the game is paused.
- **Power Ups, Power Ups Timer AND Consumables**
  - This class contains the power ups that can be collected by the player around the map to aid the ship such as regenerating health , increasing damage of the cannons and more totalling up to 5 and also generate a countdown timer to ensure the power up collected by the player is only limited and dissipates when the timer reaches 0. This fits the requirement UR.POWER_UPS .This class has the methods to remove the power up on the screen once it has been picked up by the player ship.
- **ShopScreen**
  - This class contains the shop on the screen where player can buy upgrades for their ship using the loot and plunder during their gameplay.The ScoreManager lay the groundwork for this class as it encapsulates the loot value which is the currency needed to buy ship upgrades in the shop.This fits the requirement for UR.SPEND_LOOT.
- **Enemy_Wave AND WeatherManager**
  - This class generates a wave obstacle which spawns on the map randomly to disrupt the advancement of the player.Weather Manager also updates the waves and the weather. This fits the requirement of UR.WEATHER and UR.OBSTACLES

Bibliography

[1] “York Pirates! Abstract Architecture Class Diagram” *York Pirates!* https://engteam14.github.io/media/Abstract%20Architecture.png.

[2] “York Pirates! State Diagram” *York Pirates!* https://engteam14.github.io/media/State_Diagram_4.png.