Architecture

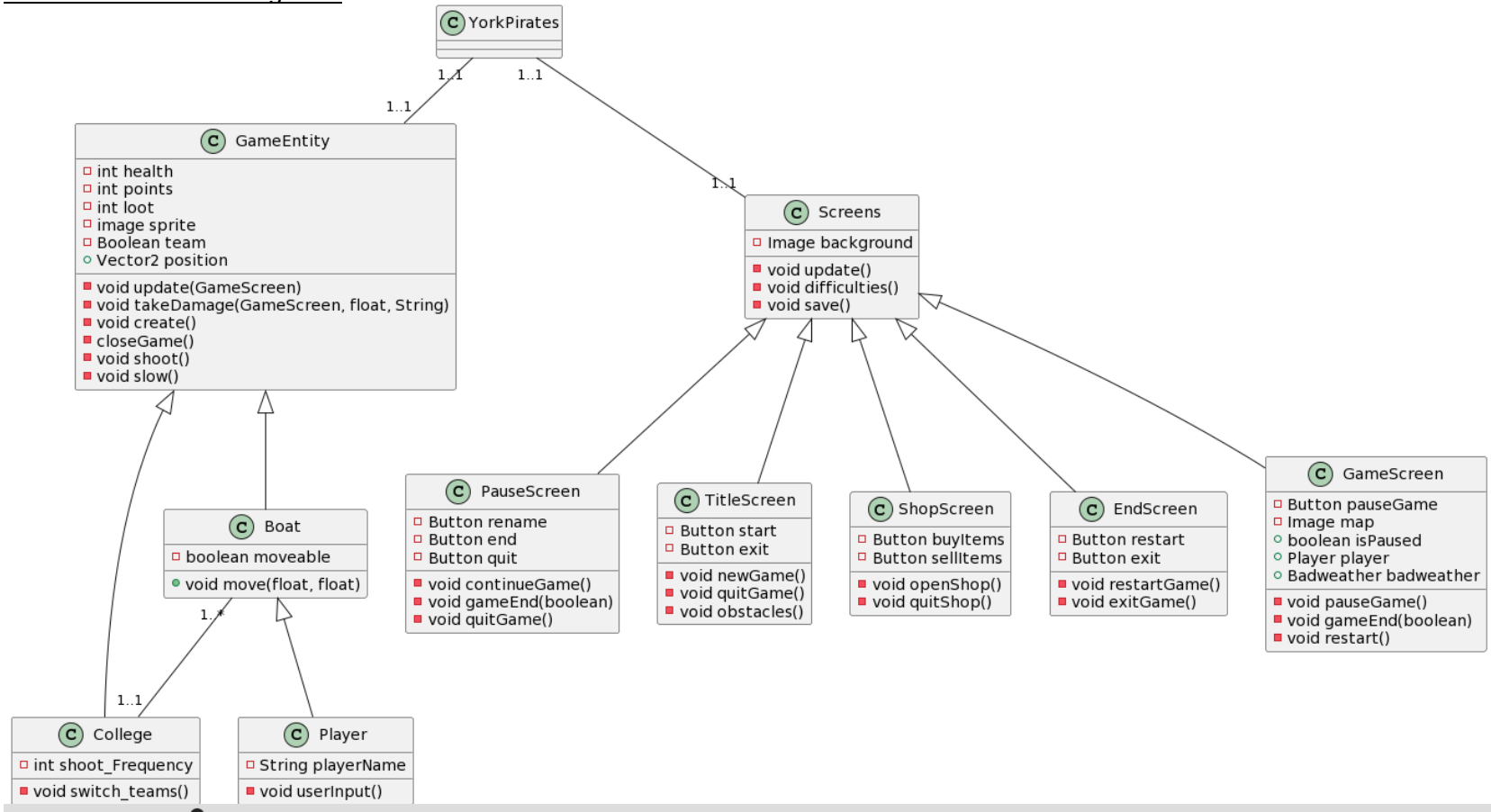
Team 14: Bass2

Katie Maison
Saud Kidwai
Jacob Poulton
Cody Spinks
Felix Rizzo French
Joachim Jones

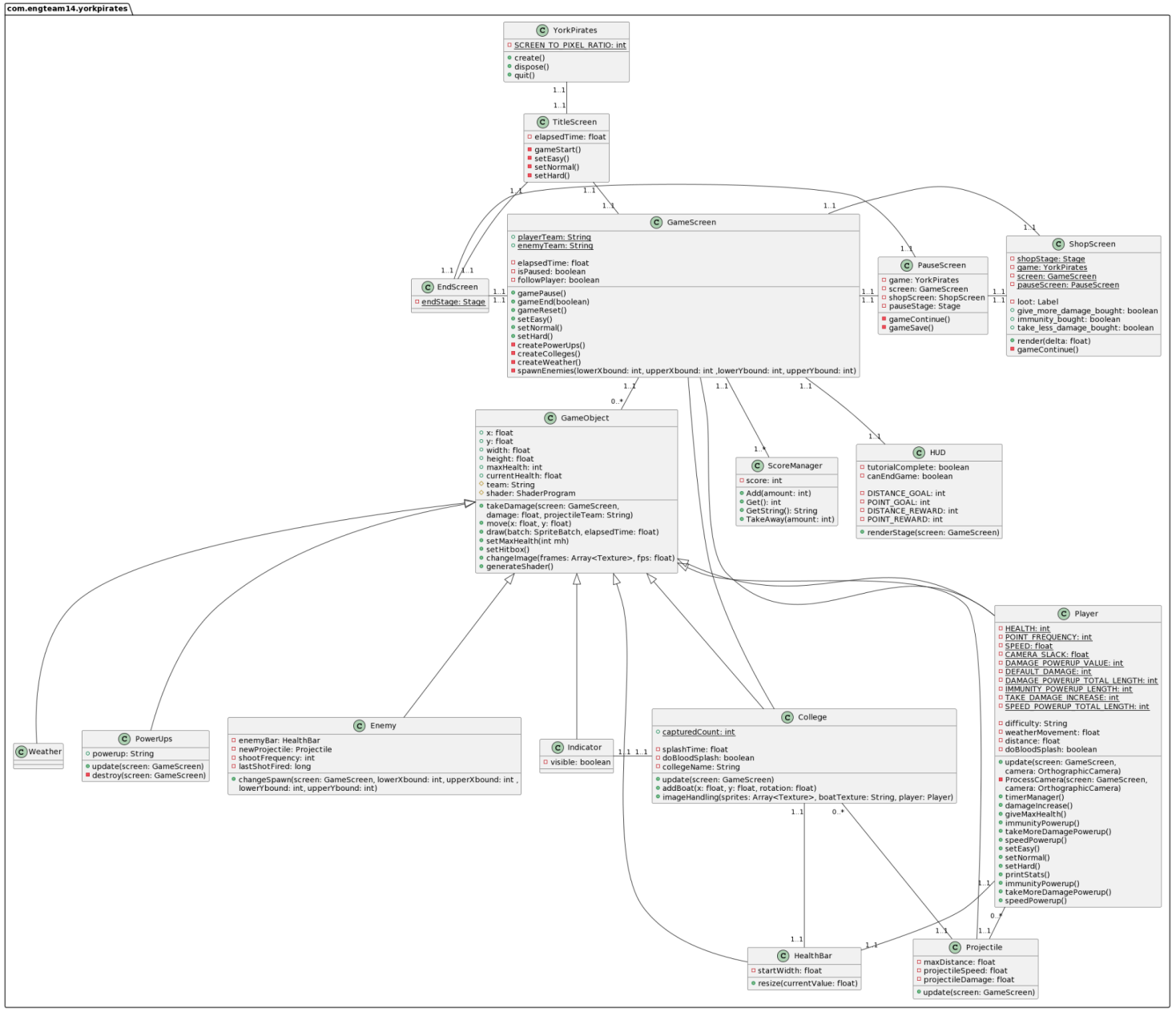
Team 12
Keira Lauder
Amy Mason
Dan Ho
Yibing Wang
Sina Khosravi

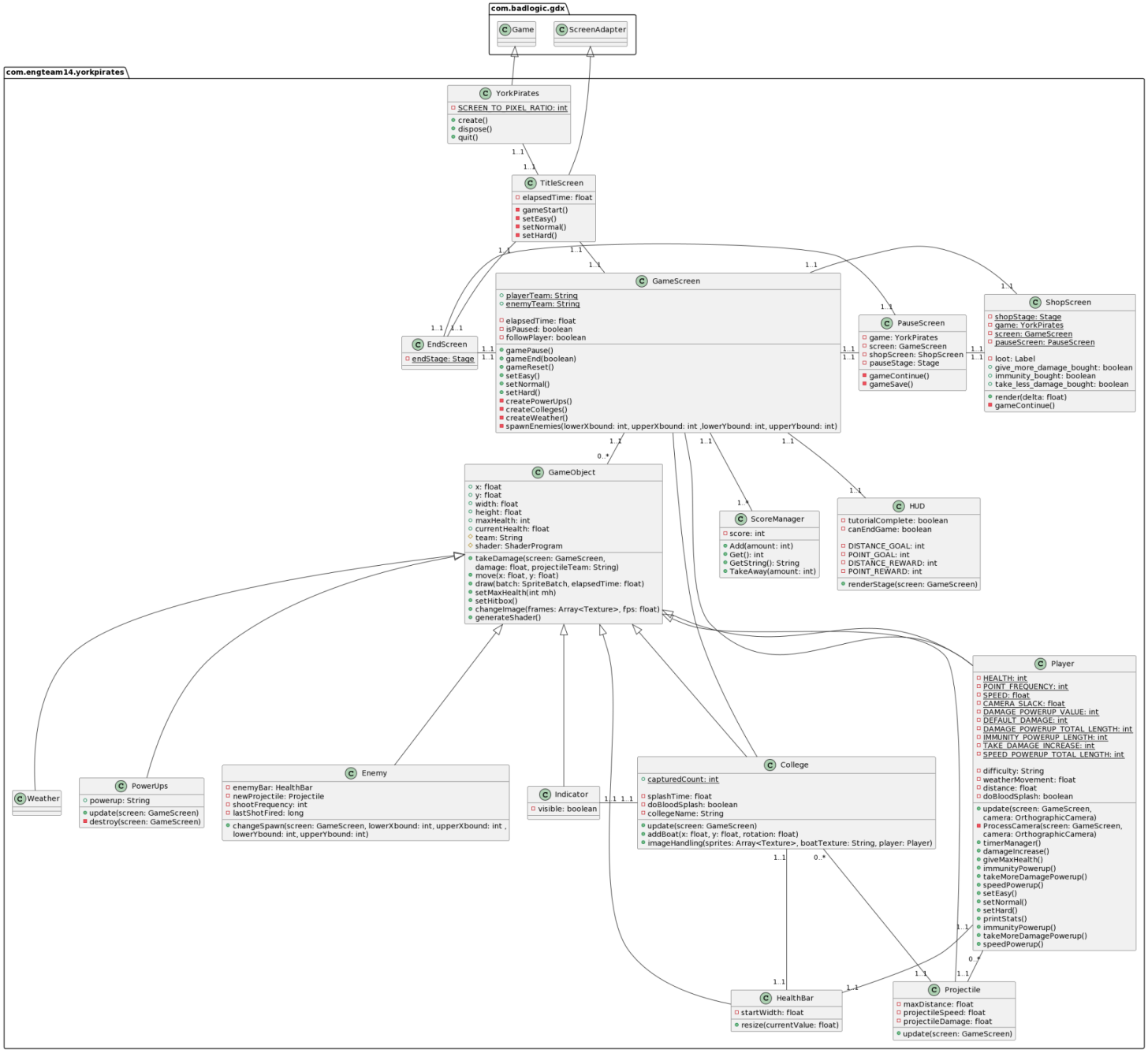
[2] State Diagram Assessment 1
[1] Abstract Diagram Assessment 1
[5] Abstract Diagram Assessment 2
[6] State Diagram Assessment 2

Abstract and State Diagrams



[3] Assessment 1 Version Concrete Diagram
[7] Assessment 2 Version Concrete Diagram





Inheritance

[4] Assessment 1 Version Inheritance
[8] Assessment 2 Version Inheritance

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 have focused on how we could structure it so that adding more entities and screens would be simple in future, and how we could reduce the code that would be duplicated in our code base. To do this, 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. Enemy boats can move and have combat now.
- **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, EndScreen and Shop Screen**
  - 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.

Concrete Architecture

We started our implementation by creating the classes seen in the abstract architecture. While doing this, we came to find other more efficient, in-built features of LibGDX, such as **ScreenAdapter** or **TiledMap**, which provided better solutions than the ones in the abstract architecture. Additionally to this, as we developed more of the game we found ourselves needing new classes as well as to change old ones. The bullet points following discuss this.

- We have focused on the same features as abstract however have added some aspects for the ease of adding extra functionality in future. One of the outcomes of this is the addition of **TiledMap**s to the game. This allows us to rapidly draft prototype and final levels for the game using the software Tiled, which greatly improves development times, furthermore the **TiledMap** allows for the implementation of a co-ordinate based collision system which is largely more efficient than the previous **Rectangle** based one we used.
- **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 and ShopScreen**
  - These classes are extensions of the ScreenAdapter class and render their screens with Buttons and overlays on the paused instance of GameScreen. (note: the attributes for these classes are omitted for clarity). These inheriting ScreenAdapter is different to the abstract architecture as we were not fully familiar with the structure of libGDX. These classes fulfill 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 the pause 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, and UR.SPEND and FR.SPEND due to the ShopScreen as a way to spend the loot/plunder.
- **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**
  - We did not have this class in the abstract architecture but we added it 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 separate 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** was created to lay the groundwork for future possible implementations of a more complex loot and points system. 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. We have also added some more constant values that is initialised in this class so that it can help with altering the player's agility and health for example. We have added some more methods to support other classes when they have been instantiated.
- **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.
- **PowerUps**
  - PowerUps is a child class of **GameObject** that inherits all its methods. When the player sails into the powerup, the update() method's job is to look at which powerup on screen the player has sailed into. This is because it will gather the name of the powerup and then call from the **Player** class, the powerup method. This fully meets the requirements of UR.POWER_UP and FR.DISPLAY_POWER_UP.

- Enemy
  - **Enemy** is a child class of **GameObject** that inherits all of its methods. Enemy boats can now shoot and move by themselves which fully meets the functional requirements of FR.ENEMY_ATK.
- Weather
  - **Weather** is a child class of **GameObject** that inherits all of its methods. Instances of **Weather** are made in the **GameScreen** class, so that the images of the bad weather can appear on the screen. Changes have been made in **Player** class as when the player's hitbox overlaps() the weather's hitbox, then the speed of the player is halved. This fully meets the functional requirement of FR.BAD_WTHR.

Bibliography

[1] “York Pirates! Abstract Architecture Class Diagram” *York Pirates!* https://keiral11.github.io/Team12Website/media/Abstract-Architecture.png

[2] “York Pirates! State Diagram” *York Pirates!* https://keiral11.github.io/Team12Website/media/State_Diagram_4.png

[3] “York Pirates! Concrete Architecture Class Diagram” *York Pirates!* https://keiral11.github.io/Team12Website/media/concrete.png

[4] “York Pirates! Concrete Architecture Class Diagram (Inheritance)” *York Pirates!* https://keiral11.github.io/Team12Website/media/inheritence.png

[5] “York Pirates! Abstract Architecture Class Diagram” *York Pirates!* https://keiral11.github.io/Team12Website/media/a2Abstract.png

[6] “York Pirates! State Architecture Diagram” *York Pirates!* https://keiral11.github.io/Team12Website/media/a2State.png

[7] “York Pirates! Concrete Architecture Class Diagram (Inheritance)” *York Pirates!* https://keiral11.github.io/Team12Website/media/a2Inheritance.png