Architecture Report

Architecture

Package	Class	Description
attacks	**Attack**	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.
	Flee	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.
	Ram	This attack inherits the default damage from the attack class and adds a multiplier to it, making it a variation on the basic attack.
	GrapeShot	
combat	**CombatScreen**	Called whenever there is combat in the game between two ships. This creates a new screen containing the two ships ready to battle, their health bars and all the different attack options the user has.
	CombatShip	Contains the texture of the ship actors drawn in CombatScreen.
	BattleEvent	Contains the different events that could happen during combat.
	AttackButton	Inherits from TextButton to create buttons used for CombatScreen.
screen	**SailingScreen**	Contains the entire sailing mode. This creates the landscape used when the user is sailing, including islands and loading in the graphics of the ships. All other screens apart from MainMenu can be accessed from this screen by interacting with corresponding objects.
	CollegeScreen	Called whenever player interacts with the island of an ally college. Provides healing service.
	DepartmentScreen	Called whenever player interacts with the island of a department. Provides different upgrades and healing service.
	MainMenu	The first class called when the game is initialised. It gives user the option to enter SailingScreen, CombatScreen, DepartmentScreen or CollegeScreen. 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” in order to allow users to save their progress or to start the game new again.
base	**BaseScreen**	Implements the Screen interface. This class provides the base code for every screen in the game.
	BaseActor	Extends the Actor class. This class provides the base code for all objects in the game.

	PhysicsActor	Extends BaseActor class. This class provides the base code for all objects that require movement and collision.
	Ship	Used to create new instance of a ship. This class provides the sailing feature of the player ship and allows user to select a specific boat type at the beginning of a playthrough, but for now all ships are of Brig type.
	College	The current colleges are Vanbrugh, Derwent and James. This holds the name of the college, its allies and enemies as well as controls whether or not the boss of this college has been defeated.
	PirateGame	The instance of the game that is run in the launcher. It initialises MainMenu and holds all the necessary variables and calls to start the game up from the beginning.
	Player	Stores all the player's information for a single playthrough of the game.
	ShipType	Holds the different ship types that the user can choose between at the start. However, due to it being unnecessary for this assessment currently only Brig is used for all ships, but it is important to keep the option for further improvements to the game.
	Department	The current departments are Chemistry and Physics. This holds the name of the department, its product (which the user can buy to upgrade their ship) and the base price of the product that they sell.

Different packages are not required, however, we found it good practice in order to keep our code readable and reusable:
base - Contains all the base codes for screens and objects(actors) so all children classes don't have to rewrite the same code.
attacks - Holds all the classes of available attacks.
combat - Contains all the information needed when the player is in combat.
screen - Holds all the different screens in the game.

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])
Attack	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.
CombatScreen	Having a CombatScreen 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.
BaseActor	Creating objects (actor in this case) is one of the most important and common operation in the game. This class makes this process much easier as similar code don't need to be rewritten, thus helps massively when it comes to future expansion **NF3**.
MainMenu	Main menu is implemented to easily present the features required at the current stage to the clients.
College	College allows us to meet requirement **F8**, of capturable colleges once a player defeats the college boss in a battle. The class also allows us to meet the requirement **F5** (game must have at least five colleges) by its inclusion. 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 contains an instance of the game at the current playthrough, which can be passed to the launcher to run. This is necessary when creating any game using LibGdx.
PhysicsActor	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. Also it is necessary to have some form of collision detection for **F1 (**the game must feature the University as islands on a huge lake).
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. This class's 'money' attribute also allow requirements **F7** and **F11** to be met via the 'addGold' method.
Ship	This class allows us to meet requirement **F1** (ships as transportation) as the object has methods (''playerMove'') that allow ship movement on the map. The ship class is also needed for sailing and combat as without a ship for the player, it would be impossible to implement these.

SailingScreen	This allows us to meet the functional requirement **F3** (sailing mode). It also helps meeting **F4.2** where there should be a clear difference between sailing and combat.
Department	Department class allows requirement **F11.1** (upgrade ships) to be met as there is a 'purchase' method. The class also allows us to meet the requirement **F5** (game must have at least three departments) by its inclusion. 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).
BaseScreen	BaseScreen makes it easier to create screens as the same code don't need to be rewritten, thus helps greatly when it comes to future expansion **NF3**.

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 and Minigame were no longer needed in this version of the game.
5. Sailing was implemented into the game which was not planned in the abstract architecture. After consulting with our client, we decided it was necessary to add this feature as it was impossible to use tiled map without such feature.
6. We also found out that the process of creating new objects and screens involved a lot of repetitive code, which motivated us to create the classes "BaseActor", "BaseScreen" and "PhysicsActor" that stores all the necessary code and can be inherited for use off the shelf.

References:

[1] SEPR “Concrete UML for York Pirates” 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 “Updated Assessment 1 Requirements” Rear Admirals [Online] Available: https://therandomnessguy.github.io/SEPR/Assessment/2/Updates/Upd2Req1.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]