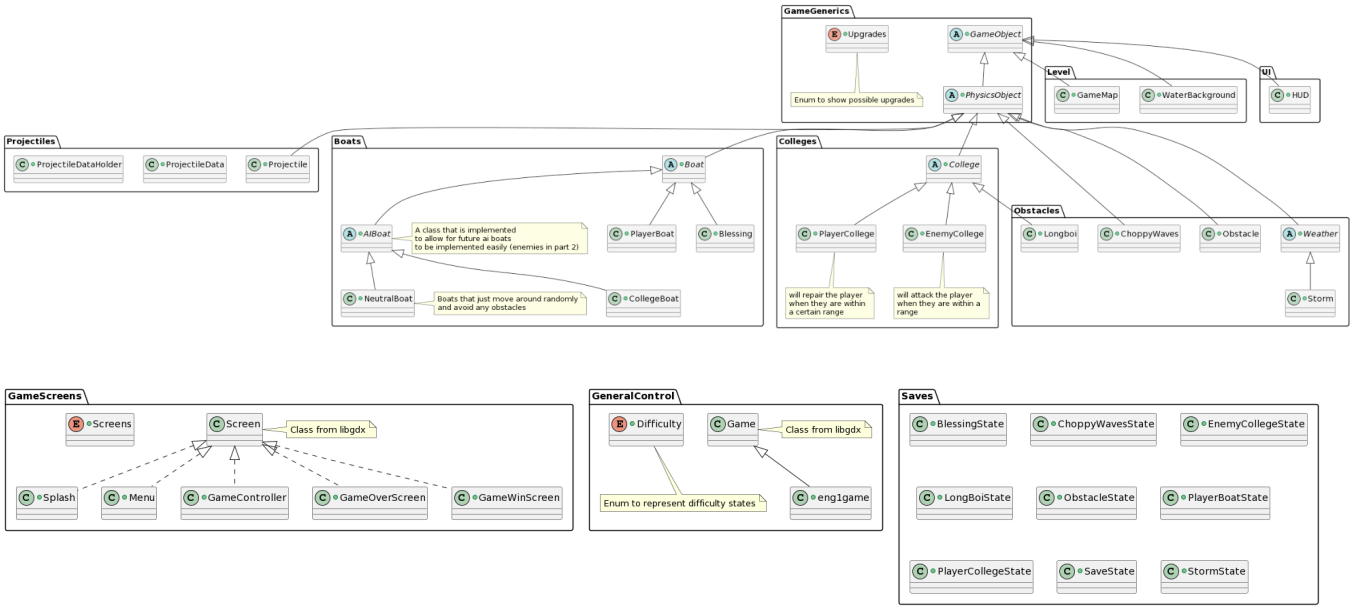
Architecture

Team 21
Zain Alshaikh
Corin Bertrand
Damien Heaton
Mandy Li
Brandon Oliver
Tom Tafijs

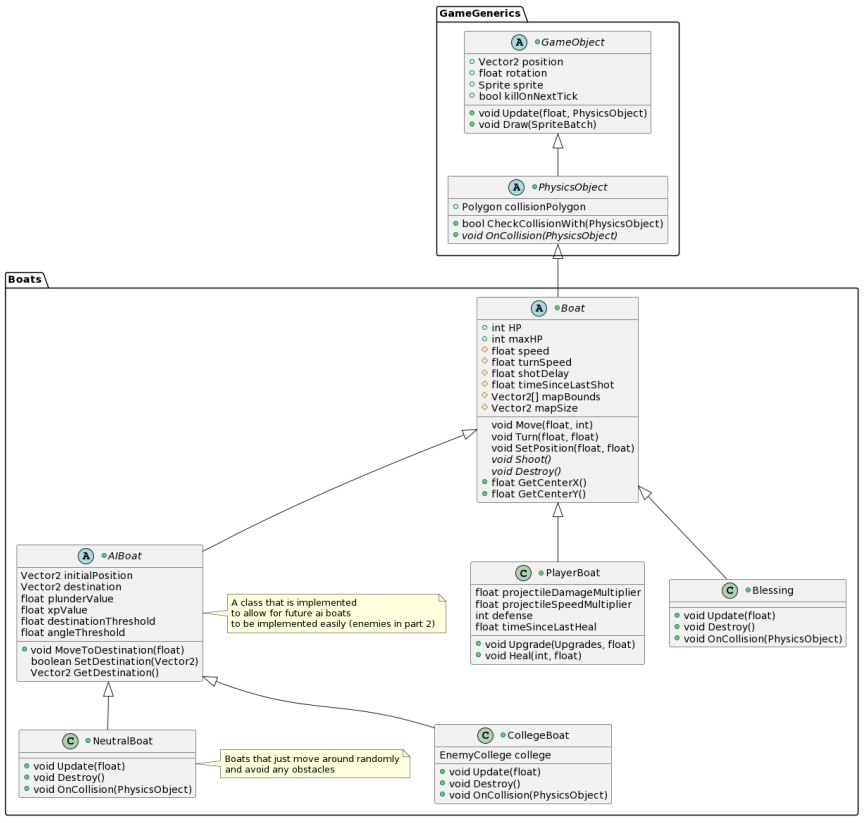
Architecture Specification

For the architecture development, we created UML diagrams with the use of PlantUML. We used a package diagram for the abstract architecture and a class diagram for the concrete architecture.

This is the abstract architecture diagram:



This is a part of the concrete diagram, featuring 2 of the packages, for the full detailed diagram, visit https://lyrenhex.github.io/ENG1-Project-Part2/v2/architecture



Boat - Abstract Class
Abstract class that fulfils basic functionality that all boats must have (a position, rotation, HP value, the ability to move, the ability to shoot), collision handling, the ability to be destroyed. Inherited by PlayerBoat and AIBoat

PlayerBoat (Extends Boat)
Extends the boat's functionality to allow for player input to control them. Extends Destroy() to have a game over occur if the player is destroyed.

AIBoat - Abstract Class (Extends Boat)
Implements a simplistic pathfinding method utilising Move() and Turn() from Boat to allow the AI to rotate and move towards randomly generated destinations in the game space. Once close enough to its destination it will generate a new one and use Intersector to avoid setting paths that will cause collisions with Colleges, other Boats are ignored.

NeutralBoat (Extends AIBoat)
Passive AI, most of the behaviour is defined in AI Boat. This AI also does not engage in combat but drops a flat amount of XP and Plunder when destroyed.

CollegeBoat (Extends AIBoat)
This AI will follow the player boat once it is within a certain range and engages in combat. When destroyed, it drops a flat amount of XP and plunder.

Blessing (Extends Boat)
This is a stationary object that represents a magical powerup and placed in a random position. When the player boat collides with this object, it "collects" the powerup and becomes immune to damage for a set amount of time.

College - Abstract Class
Abstract class that has basic functionality that all colleges need. The range attribute will be used to either repair or attack the player depending on what kind of college it is.

PlayerCollege (Extends College)
If the player is in range, the player will have their boat repaired, and their HP will increase in a linear scale.

EnemyCollege (Extends College)
Attacks the player when they are in range, and can be destroyed when its health hits 0, at which point it will stop attacking the player, and change sprite.

Longboi (Extends College)
Attacks the player when they are in range. When defeated (health hits 0), it will stop attacking. The player will gain a damage upgrade and have the ability to fire ducks.

Obstacle (Extends PhysicsObject)
This is a concrete class which defines basic functionality for a stationary object in the game, e.g. a texture and a position. This was used for rocks and shipwrecks which deals damage to the player boat when collided with.

Weather (Extends PhysicsObject)
This is an abstract class which has a position and texture.

Storm (Inherits from Weather)
This class is for handling bad weather, and contains attributes for timeSinceLastDamage and timeBetweenDamage. When collided with, the player HP starts to decrease.

ChoppyWaves
This class is a special obstacle which when collided with, inverts the player controls. E.g. “A” normally turns left, so it would now turn right.

GameObject - Abstract Class
All objects extend this class. Contains information about the object: position, rotation, and sprite to allow for generics to update them all more easily, and contains the killOnNextTick flag for removal from the game.

PhysicsObject - Abstract Class
An interface extending GameObject which also adds the requirement of a collision polygon and a method for handling collision. Inherited by Boat, College, Projectile, and Obstacle.

GameController
The game’s behaviour is contained within this class. Every GameObject and PhysicsObject is contained in this class, as well as additional stats such as XP/Plunder and Projectile Data. Updates and draws all GameObjects, and draws all UI objects (done last to show UI on top)

GameMap
A class that contains boundaries so that no game objects can go off the map, while also controlling the camera movement.

Projectile
Physics Object which moves in a straight line until it hits something. Knows if it's player fired or not so it can correctly react when hitting enemies / players.

Projectile Data
Contains a constructor method for creating Projectiles, which are then referenced by each new instance of Projectile, saves resources by only creating and referencing 1 instance of each projectile data type.

Projectile Data Holder
This class creates each instance of ProjectileData that are used by the game, and is referenced by projectiles when inheriting stats.

HUD
A class that holds all UI objects (buttons, text, menu, shop, etc) and draws them on screen. Also deals with upgrading.

The Saves package consists of the following classes:

BlessingState - saves the position of the Blessing object
ChoppyWavesState - saves the position of the ChoppyWaves object
EnemyCollegeState - saves the number of remaining boats, current HP, current position
LongBoiState - saves the position and current HP of the LongBoi object
ObstacleState - saves the position of obstacle objects on the map
PlayerBoatState - saves information such as position, HP, projectile type
PlayerCollegeState - saves the position of the player college
StormState - saves the position of the storm "grid"

SaveState
When instantiated, this class creates a state for the above classes. The result of loading this state is identical to the saved state.

Difficulty
We used an Enum to represent the supported difficulty states. At the start screen, the key-presses e, Enter, and h are used to select the difficulty.

Systematic justification for the abstract & concrete architecture

Boats
The boat object tree is constructed in this way to adhere to DRY (Don't Repeat Yourself) and it uses polymorphism (@Override) to achieve this goal. Adhering to DRY allows for the modification of a system's element to not require change in other unrelated elements. This then leads to logically related elements to sync as they change predictably and uniformly.

Using polymorphism has many benefits.
- It is time saving. Code can be reused as required.
- A variable can store different data types.
- Allows for easier debugging.

Colleges
The college object tree also adheres to DRY. This allows for easy instantiation of multiple colleges. It is also easy to change attributes of enemy colleges to balance or vary gameplay.

GameMap
This is a standalone class that only inherits from GameObject as required. It defines the layout of the map, and could also define the layout of obstacles or bad weather eventually (array of objects).

Projectiles
Our architecture allows us to create specific objects such as cannonballs by instantiating an object of this class, which allows for quick implementation of shooting upon implementation.

GameController
This class does not inherit from any other. It contains references to all of our other objects, and it orchestrates updating and drawing all of them. This allows for easy object control through update functions.

GameObject
Does not inherit from anything. This object is an abstract class where it contains only function declarations not definitions. This allows for the use of Java generics on different objects, e.g. we can easily draw all GameObjects. Since the GameObject class will have a draw() declaration, all game objects can be stored in a single array. This then means update() and draw() can be called on them easily, eliminating the need to call them on multiple seperate arrays. Updating and drawing the whole game has become easier. Deleting and dereferencing everything will also be easier as objects will be stored in a single array instead of multiple.

Other Implementation Details
LibGDX was chosen as it provides comprehensive collision functions, which allows for polygons to be checked against other polygons smoothly.

How the concrete architecture builds from the abstract architecture
The abstract architecture defines the entire structure of the project in basic detail, showing only the relationships between classes and the class' types. This provided a good base for us to work on the codebase and allowed us to structure our plan accordingly. As the project expanded in scope and modularity the abstract architecture was appropriately modified beforehand (such as the additions of the ProjectileData and ProjectileDataHolder classes, to allow easier implementation of new Projectile types) Modifying the abstract architecture beforehand and figuring out how the new additions relate and interact with everything else allowed us to easily implement these changes without causing clashes with the existing systems.

Relate the concrete architecture clearly to the requirements

- UR_NPC_BOATS: Gameplay includes other boats sailing around
    - NeutralBoat has been implemented. These boats just move around randomly and avoid any obstacles.
    - CollegeBoat is implemented for enemy boats. It is assigned to a specific college when instantiated and has functions defined for shooting.
- ● UR_WIN_TIME: Game is winnable in 5-10 mins with no experience
    - The game has a flat time cap of 10 minutes.
    - The final college is invulnerable until all of the other opposing colleges are defeated.
- UR_DEFEAT: Player can be defeated
    - A timer has been included in the GameController, resulting in a game over if it expires.
    - The player has a HP stat, which if it reaches 0 either by combat or collisions, will end the game.
- UR_COMBAT: Gameplay includes combat with colleges
    - Both PlayerCollege and EnemyCollege have been implemented. The player will be attacked by the EnemyCollege when they are within range.
- UR_EARN_XP: Player can earn XP for upgrades (done anywhere at any time)

    - An XP and Plunder stat is included in the GameController, as well as an XP tick multiplier stat to affect acquisition of both. They are also gained by destroyed colleges and boats.
- UR_SPEND_XP: Player can use XP for upgrades (upgrading stats) (done anywhere at any time)
    - Upgrades are included in the GameGenerics package.
- UR_SAIL: Player can sail around the lake
    - Movement and turning have been implemented in the boats
- UR_LEVELS
    - We used an Enum to represent the supported difficulty states. At the start screen, the keypresses `e`, `Enter`, and `h` are used to select the difficulty.
    - The GameController class has a `setDifficulty` function which assigns the difficulty to the player boat, as this is the only object affected when game difficulty is modified.
- UR_SAVE_GAME
    - The saves package includes classes which create save states for various aspects of the game.
    - The Menu has the render method to handle saving.
- UR_LOAD_GAME
    - The eng1game class has a create method which loads an existing save file if it exists in the saves directory. The loaded state is identical to the saved state.
- UR_OBSTACLES
    - The GameController spawns various obstacles such as rocks and shipwrecks, which deal damage upon collision.
- UR_WEATHER
    - There are rectangular "grids" within the game world that represent storms - whilst the player ship is inside this section, it will experience degraded health.
- UR_SHOP
    - The HUD class is modified to include the shop menu, which can be accessed via a keypress button.
- UR_SHOP_POWERUPS
    - The method RandomiseUpgrades in the HUD class provides two power-ups available to purchase with gold
- UR_MAGICAL_POWERUPS
    - There are two new classes Longboi and Blessing which have been implemented. Upon defeat or collision, they grant a special powerup.
- UR_SHIP_COMBAT
    - CollegeBoat has been implemented. When the player is within range, it will follow the player boat and shoot in its direction.