Architecture

Group 26 - Spice Traders (prior team 22)

Dan Wade
Alice Cui
Robert Murphy
Charlie Crosley
James McNair
Marc Perales Salomo

3A Abstract architecture

The abstract representation of the game design consisted of a Unified Modelling Language (UML) diagram, and a file which listed the components and entities of the game to represent the structure of the software. By using these two methods of design, it was possible to obtain two different views of the abstract architectural design.

The basic UML diagram shown below (see Figure 1) was created using PlantUML and provides a "logical" architectural view of the game design [1], where abstract object classes are displayed to best represent the system requirements elicited.

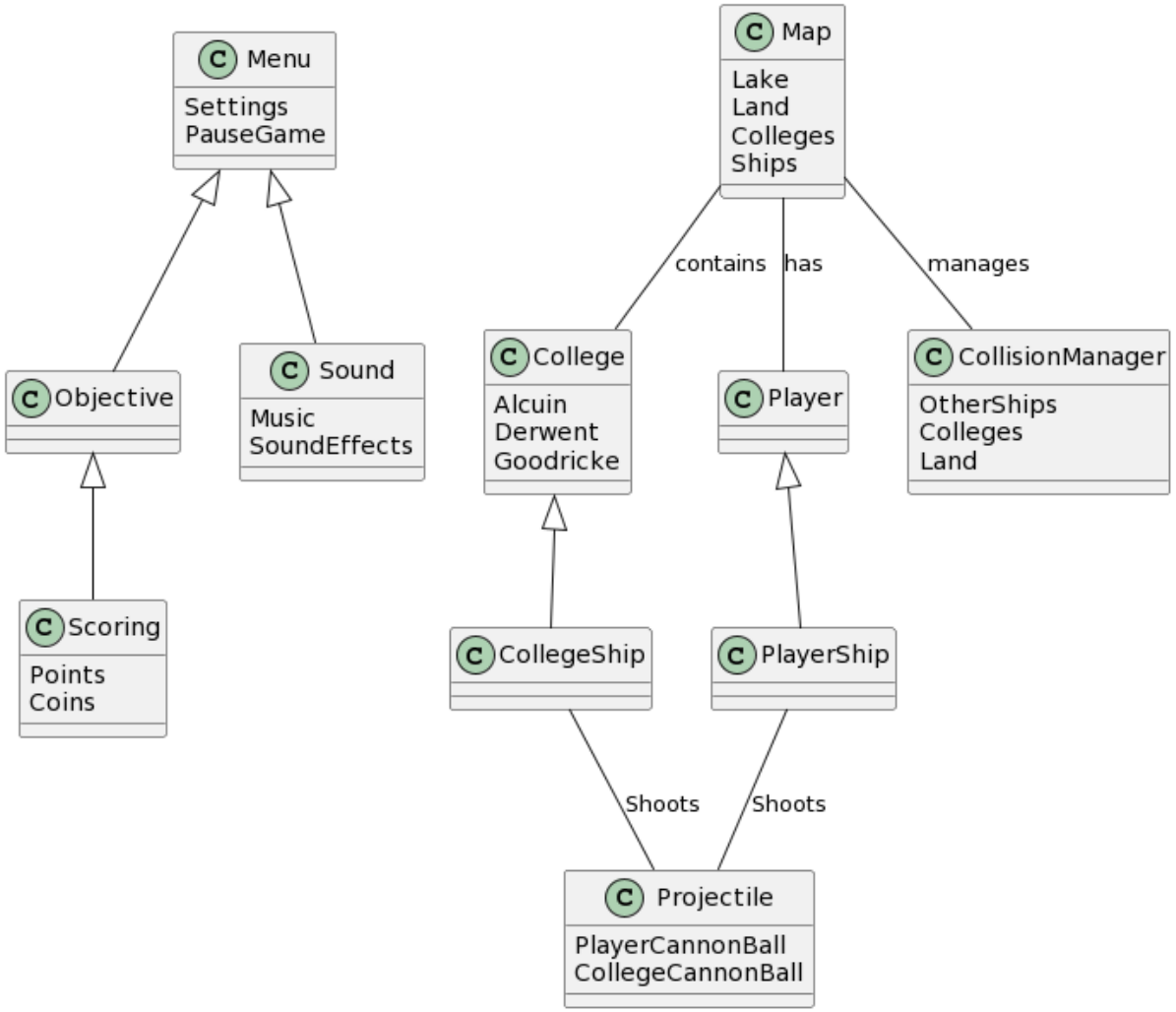

Figure 1: The initial UML diagram used at outset to obtain a logical view of the project.

The project has been broken up into several smaller components that are detailed below, and this in turn enabled a division of the classes into clusters.
These entities and components were grouped according to how they best fit together, and with each iteration of the project cycle this list could be checked and adjustments made where necessary.

- Map + Collision - Manage the tileMap structure for the fixed map (as discussed with stakeholder). And manage collisions
- Objective and Scoring - Manage the tracking of points and gold. Provide ways to upgrade ships with both
- Menu, General UI and Sound - Create a user interface to introduce the game.
- Ship - Control movement and combat of both player and AI ships.
- Projectiles - Manage the direction and distance of all launched projectiles
- Basic Entities - All other entities including the sea monster and power ups.

Concrete architecture

Throughout each iteration of the project a new UML diagram (also created using PlantUML) was updated so that the concrete architecture could be recorded. Due to the nature of the project expanding as the code base grew.
Below (Figure 2) is the UML for how the entity hierarchy is defined within the game. The 5 specific powerup classes have been omitted to make the UML more readable. Each of these classes simply extends the provided methods in the PowerUp class.

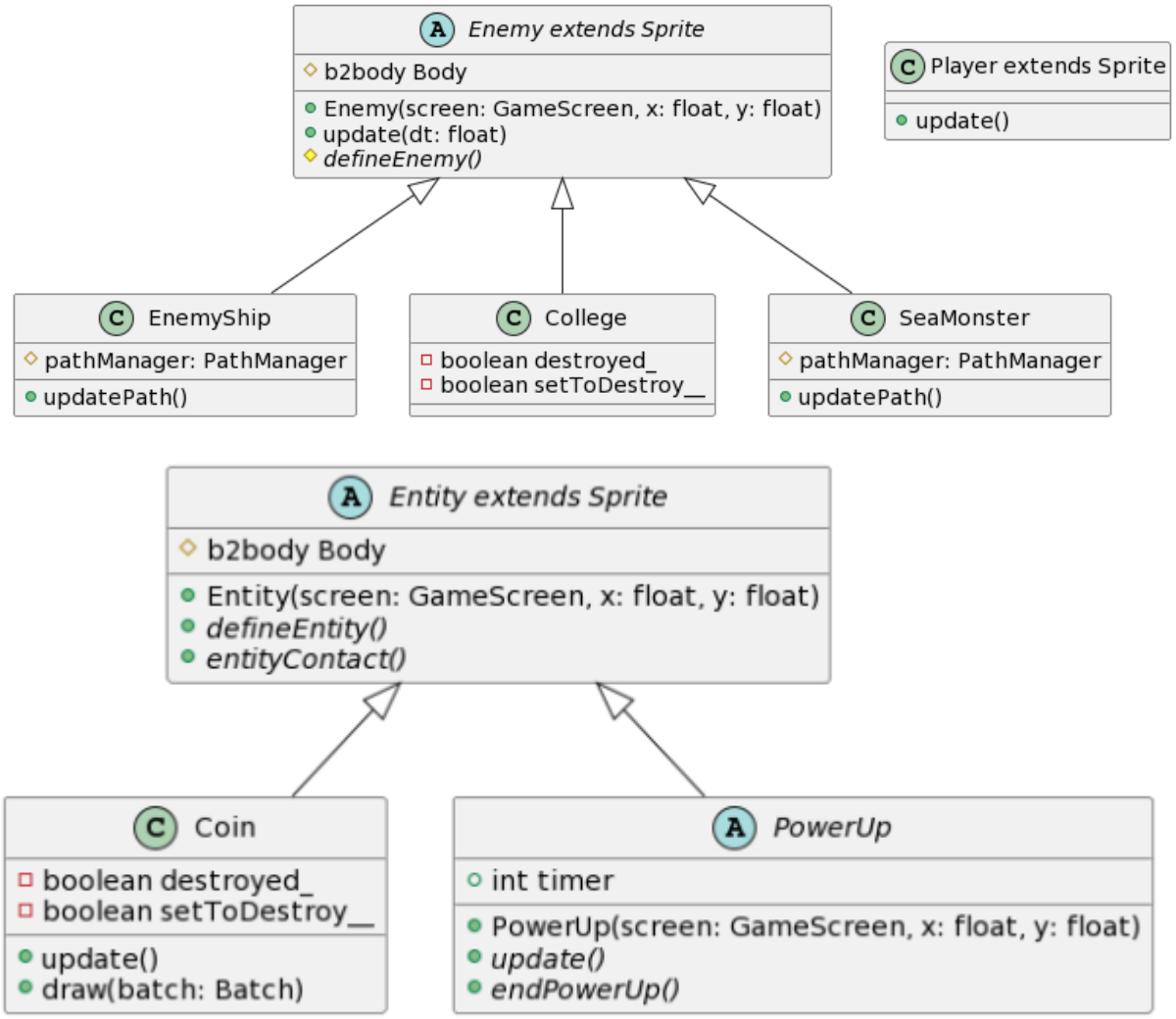

Figure 2: UML diagram for objects within game

The Java game-development framework LibGDX was chosen for development after considering the alternatives available, and the shift from abstract to concrete architecture was guided by this framework. The concrete architecture for this project uses a Component-Entity system. As demonstrated in the UML diagram, there are three main LibGDX classes extended by the core classes of the game: Sprite, Game and Screen.

- **Sprite**: As LibGDX offers sprite batches to be rendered with every update, the Sprite class is extended by every class of the game that requires textures to be rendered and animated. This includes the player ships, enemy ships, colleges and any other entities in the game world.
- **Game**: The PirateGame class is used as an "orchestrator" class to handle switching between screens of the game depending on the current game status. This class extends the LibGDX Game class which provides this functionality.
- **Screen**: Where the PirateGame class handles which screen to display at any point in time, each screen extends the LibGDX Screen class which allows for different screens to be displayed - e.g. main menu, game over and so on.

These three core LibGDX classes are entities from which the other component classes could be built upon under the Entity-Component system. Another entity, *InteractiveTileObject*, was created to store the map objects that would require collision detection in the event of a ship or projectile hitting them. This was originally planned to be included in the Map class, however during the programming stage this was removed to become its own entity and increase modularity.
Along with the entities, a hierarchical relationship has been established for different pathing strategies for ships, tornadoes and Sea Monsters. The UML for this relationship is shown in figure 3 below.

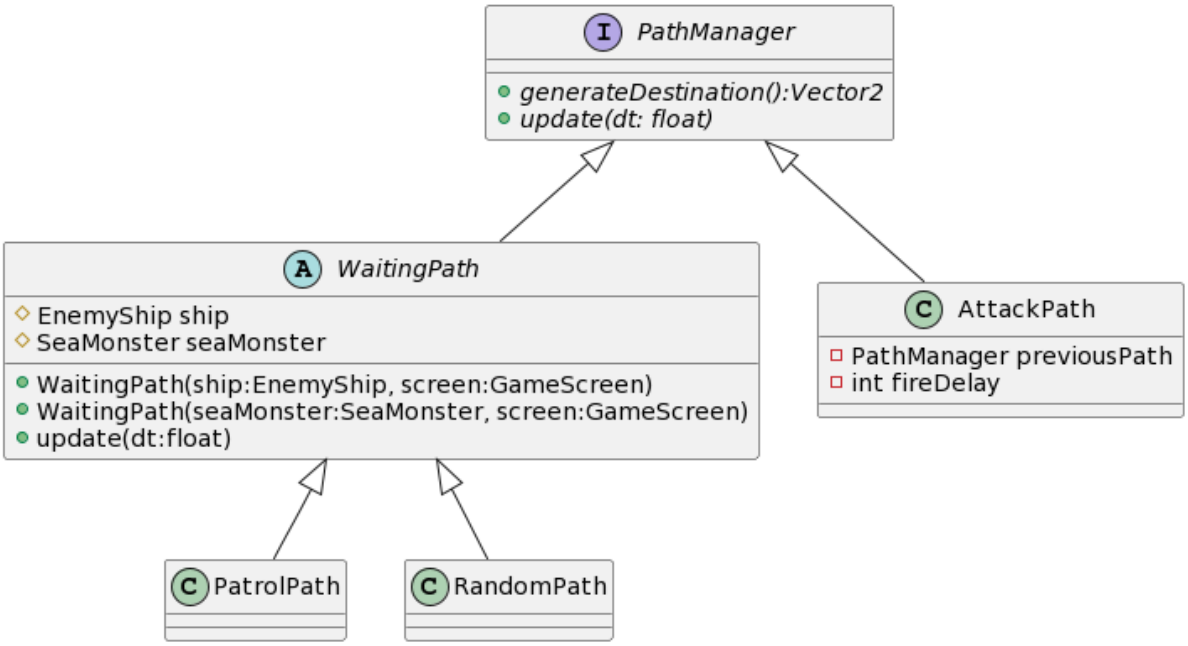

Figure 3: Structure of the pathfinding module

3B Transition from abstract to concrete

The concrete architecture builds from the abstract architecture using LibGDX to influence the evolution of the project. Component classes are constructed using certain pre-loaded LibGDX entity classes as described above, therefore a Entity-Component system architecture is employed for the project.

The Map class from the abstract representation is split into two entities for the concrete architecture, Game and Screen. Game has one component, PirateGame, which is used as an orchestrator to switch between screens depending on certain events occurring. Screen has multiple components, with the main gameplay screen class, GameScreen, handling the loading of the map via the Tiled map creation framework.

Collision detection is separated from the Map class to become its own entity. This entity was created during the programming part of the project and was not included with LibGDX. This allowed for collision detection to become modular, as any number of components could be added to this entity should the need arise.

The Menu class from the abstract representation became its own screen component under the Screen entity within the concrete architecture. This was required due to the screen feature of LibGDX, which allows for multiple screens to be switched between during runtime depending on certain triggers.

Relationship between concrete architecture and requirements

The requirements elicited via the brief document and during the stakeholder meeting are documented within the “Requirements” documentation provided within this project. Below are tables that demonstrate how the entities and components of the concrete architecture relate to those requirements.

Requirements ID	Entity/Component(s)	Relationship description
UR_PLAYER_MOVEMENT FR_PLAYER_MOVEMENT	GameScreen	The player’s keyboard input is read within the GameScreen class to move the ship
UR_PLAYER_SHOOTING FR_SHOOTING	GameScreen/Player	When the player presses the fire key, the input is read by the GameScreen class which calls the fire() method from the Player class, which fires a cannonball from the ship
UR_PLAYER_PLUNDER UR_PLAYER_POWERUP FR_PLAYER_PLUNDER FR_PLAYER_SHOP	Coin/Entity/Hud	A coin entity is generated and spawned. Upon a collision between the ship and the coin being detected, a “coin collected” sound plays and

FR_PLAYER_POWERUP		the player's HUD is update to confirm they have collected the coin. Players should also be able to spend plunder in the shop
UR_PLAYER_EXP FR_PLAYER_EXP_TIME FR_PLAYER_EXP_COLLEGE	Hud/College	As the player moves around the map, their score increases as time passes, handled by the Hud class. Defeating a College increases the player's point total by 100 and gives a randomised amount of coins between 0 and 9
UR_PLAYER_UPGRADE FR_PLAYER_UPGRADE	SkillTree	The SkillTree class offers various options for the player to upgrade using points, and the option selected increases the player's stats
UR_ENTITY_GENERATION FR_ENTITY_GENERATION FR_SHIP_GENERATION FR_COLLEGE_GENERATION	Sprite/Entity/AvailableSpawn/College/EnemyShip	The various texture sprites used by the game are spawned within their class
UR_MAP_GENERATION FR_MAP_GENERATE FR_MAP_COLLISION	GameScreen	GameScreen is the screen that handles the gameplay, and generates the map at the start of every game
UR_UI_INTERFACE FR_UI_SOUNDTOGGLE FR_UI_STATISTICS	Hud	The Hud class displays the health, coin and experience point totals on the screen for the player to view
UR_UI_MAINMENU FR_MAINMENU_START FR_MAINMENU_EXIT	PirateGame/MainMenu	The create() method within the PirateGame class initially sets the screen to MainMenu, which allows players to start a new game, enter the options or help menu, or exit
UR_UI_PAUSE FR_PAUSE_HALT FR_PAUSE_RESUME	GameScreen/Options	Pressing the escape key pauses the game and allows the player to either resume the game, look at skill tree, visit the options menu, or exit the game

Figure 4: A table demonstrating how each requirement is accounted for within the concrete architecture of the project.

[1] I. Somerville, "Architectural design," in *Software Engineering*, 10th ed: Pearson, 2015, ch. 6, pp. 167-195