CSC207 Design Document

Version 1.0 - 2023.11.04
Created 2023.11.04

Project Name

Ramy Mina
Abdullah Khan
Nazar Kolyasa
Nikola Jakovljevic

GitLab Repository:
https://mcsscm.utm.utoronto.ca/csc207_20239/group_17.git

Section 1: Project Identification

A turn-based strategy game (“TBS Game”) is a game that involves controlling elements on a game board to achieve a certain goal where the elements take turns to make their moves. Swashbuckler’s Gambit will be a spin-off on a turn-based game aiming to combine the strategic nature of TBS Games with the intrigue presented by an adventure game. Furthermore, we are all enthusiasts of sea warfare from the 1400s and would enjoy learning how to design a basic 2-D version of Sea of Thieves.

The primary element borrowed from the TBS Game format will be the turn-based movement. This movement will take place on an ASCII grid where each “element” will be one tile. The TBS game-style interactions will be augmented through “interactions” with elements on the game field that will occur in minigame-style popups. Swashbuckler’s Gambit will feature a story to complete as its main adventure aspect along with interactions with the game world to keep the player entertained.

Swashbuckler’s Gambit will have a Pirate theme and will be set in an archipelago.

Section 2: User Stories *(Acceptance criteria below table)*

Name	ID	Owner	Description	Implementation Details	Priority	Effort
Customization	1.1	Nazar	As a user who is controlling a ship, I want to be able to customize my ship so that I may personalize my experience.	Have the ship be composed of modular parts so that they can be interchanged with improved components.	3	2
Movement	1.2	Nazar	As a pirate of the ship, I want to be able to move my ship using a D-pad so that I may be able to go from one place to another.	Implement a grid pane where the ships can move North, South, East, and West.	1	3
Ship stats	1.3	Ramy	As a player, I want to be able to see the stats of my ship so that I may be able to determine how much damage the ship cannons can do.	Implement a hashmap with a key being the different types of ships and a value of the damage per bullet.	2	3

Upgrade Shop	1.4	Ramy	As a customer of an Island shop, I want to be able to see what items/upgrades are for sale and their respective prices so that I may know what I can afford to buy.	Update the screen by removing whatever is on it and add an image of the island that you are at as well as a new Gridpane with options of what can be bought and how much it costs.	2	3
Shoot at specific coordinate	1.5	Ramy	As a player fighting an NPC/Enemy, I want to be able to choose what coordinates to shoot at, so that I may be able to sink the ship.	Update the screen by removing whatever is on it, add a 4x4 grid as the display, and add a text field so that the user may be able to type the coordinate they want to shoot at.	1	3
Coordinate shot tracker	1.6	Ramy	As a player fighting an NPC/Enemy, I want to be able to see whether a shot landed on the ship or missed, so that I can make further strategic moves.	Update the screen each time the player shoots with their inputted coordinate and if the shot was successful.	1	3
Alt-Text	1.7	Abdullah	As a non-sighted player, I need the grid to contain alternative text for NPCs, Enemy, and Islands so that I know what is going on the grid before performing an action.	Every object contained within the grid should have alternate descriptions that are accessible to screen readers.	1	1
NPCs	1.8	Abdullah	As a player, I want there to be NPCs that I can interact with by fighting or communicating so that I can get loot or mystery hints.	Design an NPC interface featuring abstract methods like dialogue(), attacks(), and items(). Next, create two classes, shipNPC and islandNPC, that adhere to this interface. Depending on the specific type of NPC, you can introduce extra methods and attributes as needed.	1	3

Side-Islands	1.9	Abdullah	As a player, I want there to be islands that I will encounter while traversing the grid so that I can explore them and interact with the NCs on them to get loot.	Implement an island interface, where each island is on a separate grid, and has attributes for items/NPCs/minigames.	2	3
Map	1.10	Nikola	As a player, I want to see the last known location of enemy ships and alliance members.	Keep track of all ships' last firing locations & relation to the player and update the grid accordingly with red and green dots representing each.	2	3
Looting	1.11	Nikola	As a player, I want to be able to loot the remains of an enemy ship after it has been eliminated.	Keep track of eliminated ships' last location and a timer which is displayed on screen and reduces loot attainable as the ship goes fully underwater.	2	3
Inventory	1.12	Ramy	As a player, I want to be able to see my current items so that I make smart choices on when to use them.	Add a button in the grid pane where it has the player's items displayed in a list.	2	1
Money	1.13	Ramy	As a player, I want to be able to see how much money I currently have so that I can determine what I can/cannot afford in the shop.	Add a label in the grid pane that has a counter of how much money you have in total and how much it increases or decreases after every action.	1	1
Battling	1.14	Ramy	As a player, I want to be able to see the battles in action on a separate grid, so that I do not interfere with the main grid.	Update the screen with a size n by n sized grid when fighting the enemy by clearing the current state of the game and displaying the new grid with the battle being displayed. After the battle is done, update the screen with the state of the game after the	1	3

				battle.		
View Picture	1.15	Nazar	As a player, I want to see what's close to my ship so that I am more immersed in the game	Have a still picture representing what's in the distance with respect to the player ship in the gridpane above the map grid, which changes when the player gets close to Islands, NPCS.	2	3
Story	1.16	Abdullah	As a player I want a main goal to progress towards in the game, so I may be captivated in the game.	An adventure story with elements from classic seafarer/pirate tales that rewards players for completion along the way.	1	2
UI Buttons	1.17	Nazar	As a player, I want to be able to use UI buttons so that I may be able to save, pause, and load my game as well as check out the instructions.	Add 4 buttons in a grid pane; Save, Load, Pause, and Instructions. The save button will save the current state of the game. The load button will give you the option to load a specific file from the save files list. The pause button will freeze the game so that I can come back to my game later without saving it. The instructions button will show how to play the game by clearing the screen and displaying the instructions. Once clicked again, the screen should be updated back to the current state of the game.	1	3

Acceptance Criteria

Name	ID	Owner	Acceptance Criteria
Customization	1.1	Nazar	User can personalize the ship with decoration.
Movement	1.2	Nazar	User can move in 4 directions, North, South, East, or West by using a Dpad.
Ship stats	1.3	Ramy	User can press an info button that displays all of

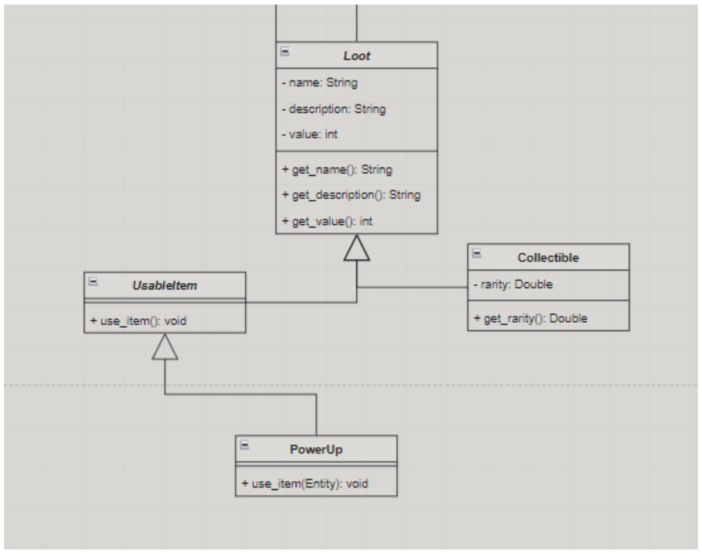
			the specifications according to the ship.
Upgrade shop	1.4	Ramy	User can visit upgrade shops which are scattered around the map and will have the option to interact with the vendor and purchase offered items or upgrades which will then be added to the user’s inventory or change ship stats/looks.
Shoot at specific coordinate	1.5	Ramy	User prompted with shoot action. User inputs coordinates to shoot at. User presses fire and a shot takes place.
Coordinate shot tracker	1.6	Ramy	Screen is updated after the shot has taken place. Damage is applied according to the stats of ships. User is informed on if the shot hit the other ship and if so how much damage was taken.
Alt-Text	1.7	Abdullah	User is able to spot items on the grid. User can press the item and a description will pop up. Text will be very visible and stand out as it is an accessibility feature.
NPCs	1.8	Abdullah	User can interact with NPCs. User can choose to attack or speak. User can also check NPC vendor items and purchase them. User’s inventory and vendor inventory will be adjusted accordingly.
Side-Islands	1.9	Abdullah	User can visit islands on the grid. User can see islands on the grid. Users can collect deserted items on the island. Users can play mini-games on the islands.
Map	1.10	Nikola	User sees red blips on the grid as the enemy’s last known location. User sees green blips on the grid as the team's location(s). If teammates/enemy relations change or get eliminated the grid is adjusted accordingly.
Looting	1.11	Nikola	User is notified that a ship near them is sinking by a blinking red dot on the grid. The timer starts at 5 minutes and once it reaches zero no loot can be retrieved. As the timer is running out less percent of the loot can be retrieved.
Inventory	1.12	Ramy	User has a button on the interface which when pressed once reveals all of the User’s items. When pressed again, it hides the user’s items.
Money	1.13	Ramy	User can see a label on their screen that keeps

			count of their money. The amount changes based on the actions the user takes.
Battling	1.14	Ramy	When battling, User sees a second smaller grid appear on the side of the screen so that it doesn't cover the main grid. After the battle is finished the smaller grid goes away and the main grid is updated according to the outcome of the battle.
View Picture	1.15	Nazar	User sees a picture of what's close to them according to their location. User sees islands/NPCs appear once they get closer to them. User stops seeing certain aspects as they get further away.
Story	1.16	Abdullah	User has storyline to follow. User has objectives to complete which reward them once done.
UI Buttons	1.17	Nazar	User has a save button that saves their current progress in the game. User has a load button that can load a previous game save. User has a pause button which will pause the game when pressed once and unpause the game when pressed a second time User has an instructions button that describes some controls, hits and current main objectives of the game.

Section 3: Software Design

Design Pattern #1: Template Method

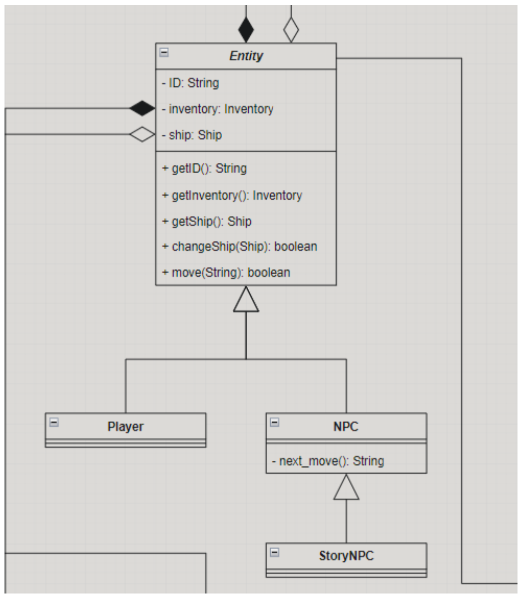
Overview: This feature will be used to represent rewards and items.



Implementation Details: The UML diagram outlines these main components:

- The **Loot** abstract class, which includes three private attributes and three public getter methods for each attribute. Attributes: **name, description,** and **value**, which will be utilized in a barter system.
- Two subclasses of Loot, where each subclass adds a layer of functionality. The **Collectible** class is meant to represent an item with no practical use but with value, thus the additional **rarity** attribute and respective getter. The **UsableItem** abstract subclass represents an item with a function, thus adding a method to represent the item being used. The **PowerUp** subclass modifies the use_item method to be applied to an **Entity**.

- **Overview:** This feature will be used to represent moveable characters.



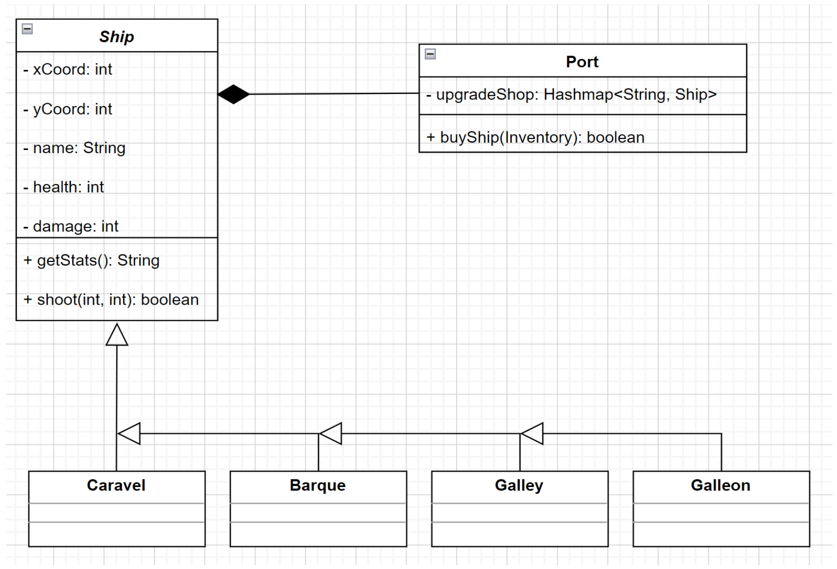
Implementation Details: The UML diagram outlines these main components:

- The **Entity** abstract class, which includes three private attributes, three public getter methods for each attribute, and two public methods. Attributes: **ID, inventory,** and **ship**. The method **changeShip** changes the currently active Ship with one from the inventory. The method **move** attempts to move the entity on the map.
- Two subclasses of **Player** and **NPC** will each provide a separate implementation of the **move** method, where the **Player** will have moves determined by user input while **NPC** will have their movements determined by the computer. **StoryNPC** will have functionality relating to the storytelling aspects of the game.

For these UML diagrams, the template method pattern was chosen, which is a behavioral design pattern where for certain classes a large "template" superclass is extended, so that all the basic functions of each subclass already exist and only have to be extended for specific needs of each of the subclasses.

Design Pattern #2: Observer Pattern

Overview: This pattern will be used to implement the update grid and ship functionality.



UML Diagram:

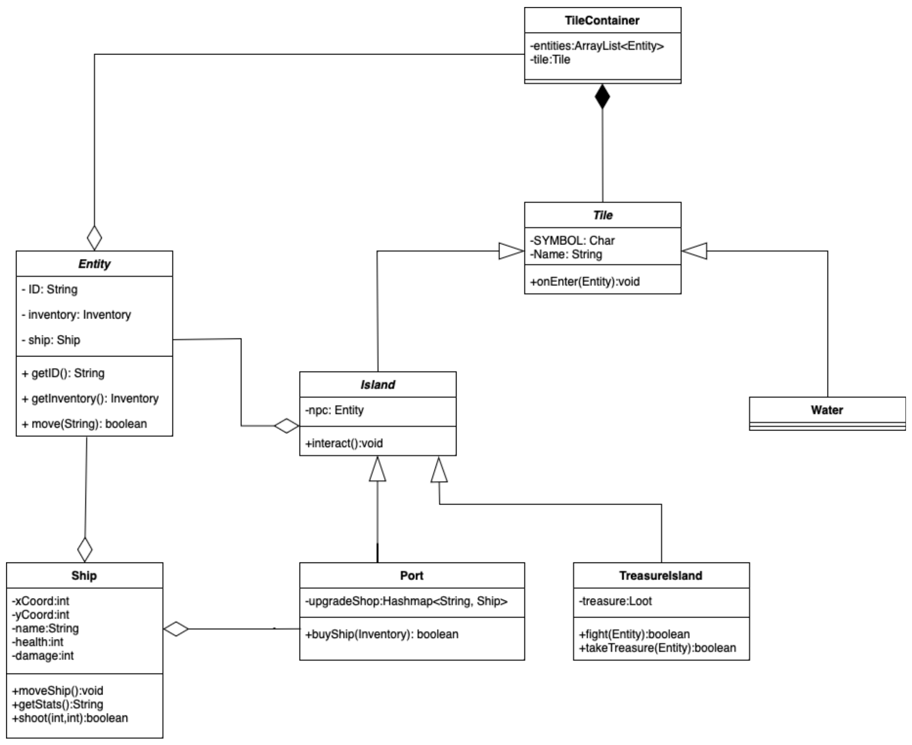
Implementation Details: Overview: The UML diagram outlines these main components:

- The **Ship** abstract class, which includes five private attributes and two public methods. Attributes: **xCoord**, **yCoord**, **name**, **health**, and **damage**. Methods: **getStats()** and **shoot()**.
- Four subclasses of **Ship** with the names: **Caravel**, **Barque**, **Galley**, and **Galleon** which inherit all the properties and methods from the **Ship** class.
- The **Port** class which is composed of the **Ship** class. This class contains one private attribute and one public method. Attribute: **upgradeShop** which is a HashMap of key: String and value: Ship. Method: **buyShip()** which returns a boolean indicating whether a purchase was successful or not.

For this UML diagram, the observer pattern was chosen. According to the observer pattern, it is a behavioral design pattern that enables some objects to inform other objects when their states change. This UML diagram shows how the grid changes its current state to indicate whether or not the enemy ship was hit when the ship fires at a given coordinate. When the player purchases a new ship, the ship's stats are another item that modifies its current state.

Design Pattern #3: Strategy Pattern

The strategy pattern will be used to implement the functionality of the grids tiles *(TileContainer and Tile classes, with subclasses Island and Water. Island also has subclasses Port, and TreasureIsland):*



Implementation Details: The UML diagram outlines these main components:

- The **TileContainer** class, which has attributes: **entities**, and **tile**. This class keeps track of what is happening at each individual tile on the grid.

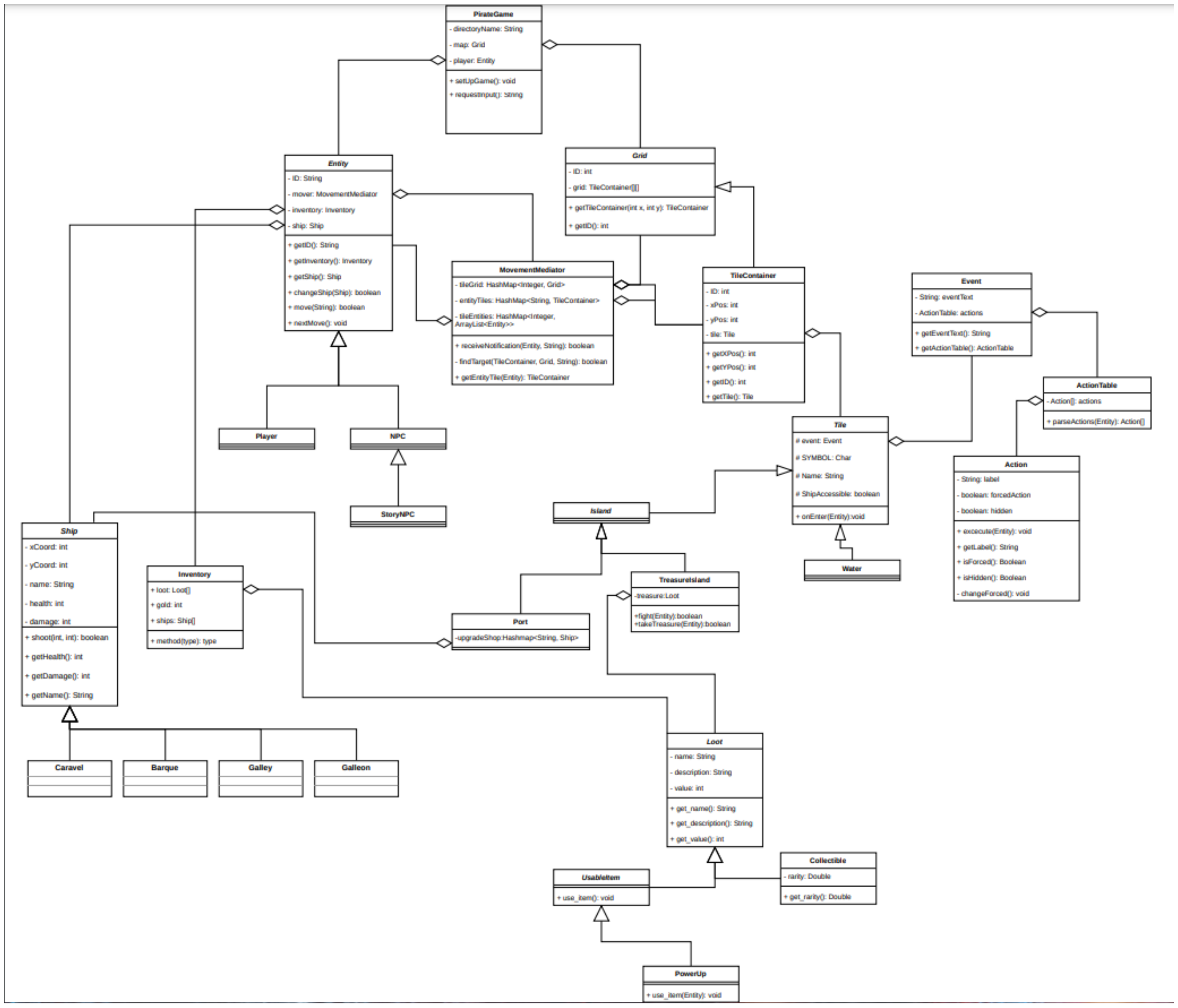
- The ***Tile*** abstract class, which has attributes: **SYMBOL**, which is an ASCII character, and **Name**, which is a String. **Tile** also has the method **onEnter(Entity),** which takes the Entity representing the player as its argument.
    - Subclasses ***Island*** (abstract class) and **Water**, which specifically define the behavior of each ***Tile*** on the grid. (Strategy Pattern)

- **Water** inherits all attributes and methods from ***Tile***. It represents water on the grid.

- ***Island*** inherits all attributes and methods from ***Tile.*** ***Island*** also has the attribute **npc**, of type Entity, which is the Entity representing the NPC of that island, and a method **interact()**.
    - Subclasses **Port** and **TreasureIsland**, which specifically define the behavior of each ***Island***. (Strategy Pattern)

- **Port** inherits all attributes and methods from ***Island,*** and has the attribute **upgradeShop**, which is a Hashmap<String, Ship> that maps ship names (String) to their corresponding upgraded Ships (**Ship**). **Port** also has the method **buyShip()**, which checks the player's inventory to see if they can afford to upgrade their ship, and returns a boolean indicating whether the purchase was successful or not.
- **TreasureIsland** inherits all attributes and methods from **Island**, and has the attribute treasure, which is the Loot the player obtains from the island. **TreasureIsland** also has the methods **fight()** and **takeTreasure(). fight()** has the Player fight the island's NPC in a minigame and returns a boolean indicating whether the player won or lost the fight. **takeTreasure()** returns a boolean indicating whether the player took the treasure or not.

The strategy pattern was chosen for this UML diagram as the ***Tile*** and ***Island*** objects have different behaviors depending on which ***Tile,*** or by extension, which ***Island*** the player's ship position is at. The behavior of a ***Tile*** is that a ***Tile*** is either **Water** or an ***Island.*** Similarly, an ***Island*** has 2 different behaviors: Either its a **Port**, or **TreasureIsland.** This application of the strategy pattern is particularly useful in covering the user stories: “Side-Islands” and “Upgrade Shop”



Design Pattern #4: Composite Pattern

Implementation Details: The UML diagram outlines these main components:

- The **PirateGame** class, which has attributes: **directoryName**, **map**, and **player**.
- The **Grid** class, which contains a 2D array for storing **TileContainer**s
- The other features with different design patterns interact with this structure.
  See the complete UML Diagram Below

The **PirateGame** class is the main game controller and can be regarded as the trunk of a tree in a composite structure. It breaks down and distributes actions to each composing class to control the game loop. Tasks are delegated to composing classes and are run down the command loop to be processed and back up the command loop to be executed and presented to the user.