Architecture

Abstract Representation

The following diagram is an abstract representation of the path the game will follow, and the flow from one state of play to another.

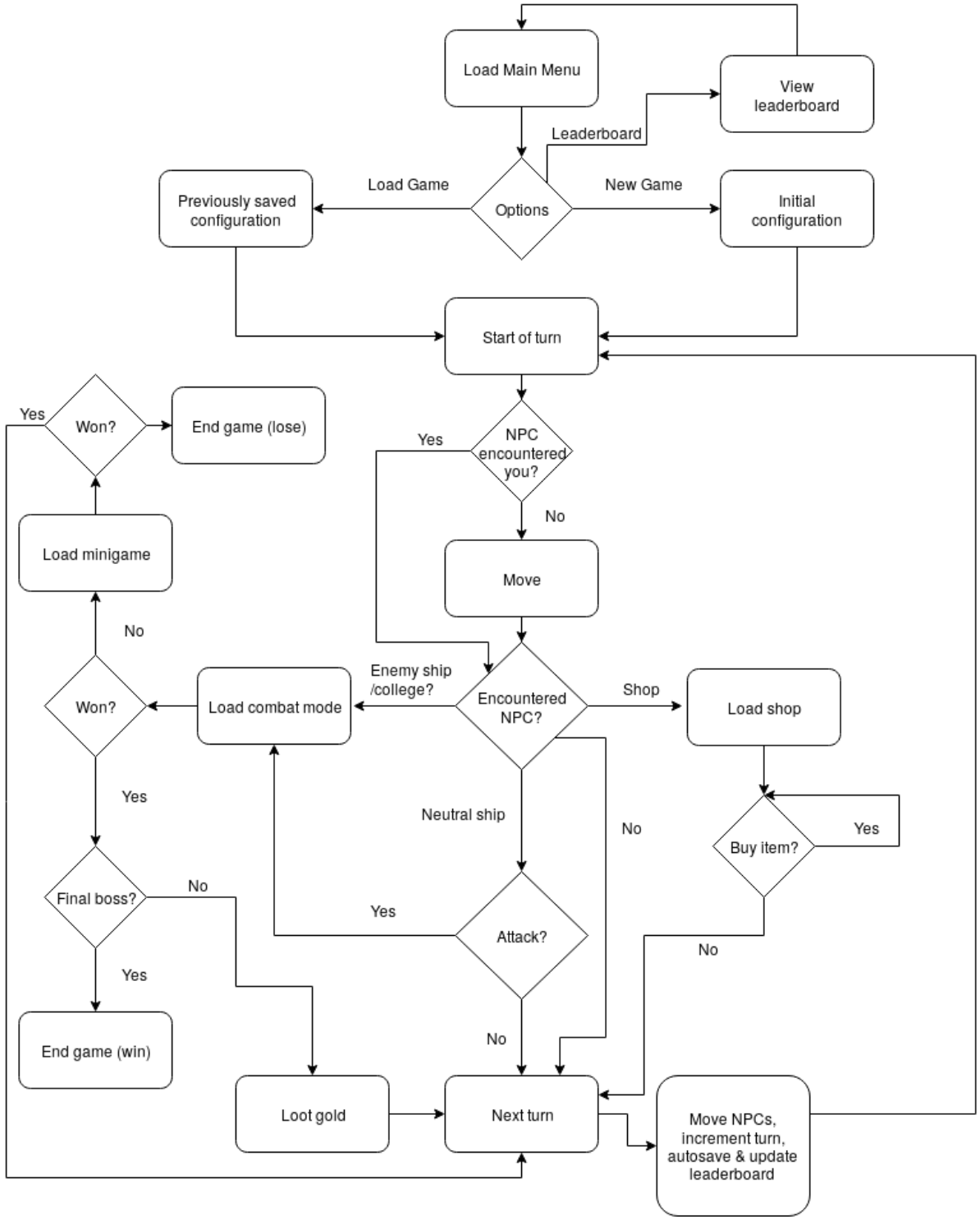

Figure 1. Game flow diagram

In order to implement the above design for the game, we propose the following abstract class structure:

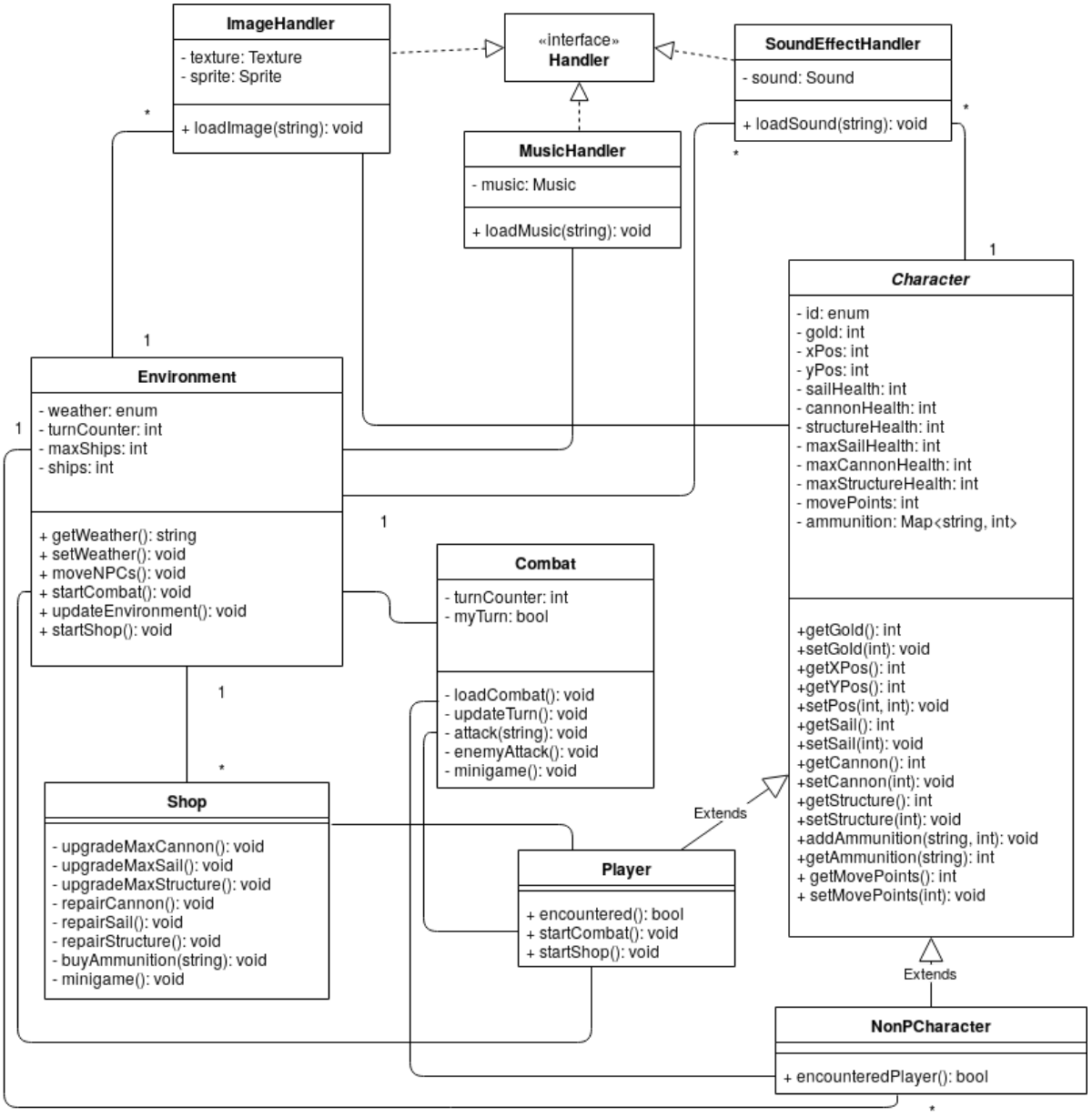

Figure 2. Class diagram

Justification

Game Flow Diagram

In reference to figure 1, we can identify three main modes of gameplay: “moving mode”, “combat mode”, and “shopping mode”. However, first we consider the main menu. The purpose of this is to load a configuration of the moving mode. In the case of **New Game**, there will be a default or initial configuration that is loaded; in the case of **Load Game**, a save file will be read and the resulting configuration loaded. There is also a **Leaderboard**, which will store the top ten scores (gold counts).

In either case, the game will now be in **moving mode**. If an NPC has *not* encountered the player, the player can move to an available position on the map grid. If an NPC *has* encountered the player, then this is skipped. At this point, if an NPC and the player occupy the same position, we have a choice of how to proceed. If the NPC is an enemy (either a ship or a college), then **combat mode** will be triggered. If the NPC is neutral, then the player may choose whether to attack and trigger combat mode or not. If the shop (which counts as an NPC) is encountered, the **shopping mode** will begin.

Assuming that no combat or shopping has occurred, the next turn will then begin. In the process of incrementing the turn, movable NPCs will be moved, and the game will be autosaved. **Moving mode** will then begin again.

In **combat mode**, either the player will win or lose. If the player loses, then the minigame will load. If the player wins this, they can proceed to the next turn of the game. If they lose, it is game over. If the player has won, the game will check if it was the final boss that has been defeated - if so, the player has won the overall game. If not, the gold of the defeated NPC will be looted, and the turn will be incremented (see above).

In **shopping mode**, the player will have the option to buy ammunition, ship upgrades, or ship repairs with the gold they have accumulated. Once the player exits the shop, the turn will be incremented (see above).

Class Diagram

Character

The Character class is the main parent class for both Player and NonPCharacter classes. Within the Character class, we identify:

- **id**: an enum which can be “player”, “enemy”, “friendly”, or “neutral”
- **gold**:
    - For Player, this will be the amount of gold they have collected
    - For NonPCharacter, this will be the amount of gold that Player can loot if Player wins in combat
- **xPos** & **yPos**: the x- and y-coordinates of the ship
- **sailHealth**, **cannonHealth**, **structureHealth** - health count for different parts of the ship
- **maxSailHealth**, **maxCannonHealth**, **maxStructureHealth** - maximum health counts for different parts - can be upgraded in the **Shop**.
- **movePoints**: the movement points currently available to the ship

- **ammunition**: A mapping between types of ammunition and the quantities currently held by the ship

Player and NonPCharacter Classes

Both classes are the children of the **Character** class, but have their own unique methods.

For **Player**:

- the **encountered()** method will identify whether the player has encountered other interactable objects:
    - if the **id** of the object is “enemy”, it will call the **startCombat()** method immediately, which will begin **Combat** mode
    - if the object is a shop, it will call the **startShop()** method

For **NonPCharacter**:

- the **encounteredPlayer()** function will identify if the NPCs have met the player. Methods of **Player** will be called as appropriate (see above).

Environment

The Environment is the class we use to describe the world map in detail.

- **updateEnvironment()** will be called at the start of each turn. This will update the **weather** and **turnCounter** attributes. If **ships** is significantly below **maxShips**, then more instances of **NonPCharacter** will be created. Positions of NPCs will be changed.

Shop

The shop class allows the player to buy items with their accumulated **gold**.

The shop class basically just contains functions with straightforward names. For instance, upgradeXX() function will upgrade player’s ship and repairXX() functions will repair the corresponding part of the ship. In addition, player can buy ammo with buyAmmunition(string) function.

- **upgradeMaxXX()** methods will increase the corresponding maximum health count of the player.
- **repairXX()** will increase the corresponding health count of the player.
- **buyAmmunition(string)** will increase the player’s ammunition of the specified type.

Combat

As we plan to make a turn-based battle system, within the Combat class, there is a attribute called **turnCounter**. This attribute is used to keep track of which turn is current in progress. In addition, the **myTurn** variable is used to identify if it is player’s turn or not.

The **updateTurn()** method is associated with **myTurn** attribute to update the turn state. The **attack()** and **enemyAttack()** are used by player and enemy to respectively attack each other.

Handler Interface

The Handler interface allows various assets to be loaded into the game. For each, the **loadXX(string)** method will allow us to quickly and efficiently render an asset instead of writing a lengthy block of code each time.