Design Document

Version 2.0 – 2023.11.10

Created 2023.10.24

Project Name

Andrew Ding
Alan Wang
Wenjun He (William)
Yingfan Hu (Daniel)

GitLab Repository:

https://mcsscm.utm.utoronto.ca/csc207_20239/group_59

SECTION 1: PROJECT IDENTIFICATION

- The motivation behind this project is that having tons of experiences in gaming encourages us to use our advantage as well as programming skills to create our own game. Furthermore, as we studied knowledge related to design concepts in this course, it would be a perfect opportunity to use our advantage in the project, using design methods such as waterfall and sprints.
- This will improve and add functionality in the game by enhancing existing features such as objects and images to make the game more immersive. It will add functionality by adding trolls, map, start/ setting menu to add difficulty, more interaction, and accessibility.

SECTION 2: USER STORIES

User Stories

Name	ID	Owner	Description	Implementation Details	Priority	Effort
Map	1.1	Alan	As an adventure gamer, I want to be able to move around the map without losing track of where I am, so that I can backtrack. I am also able to see a godview representation of the map.	There will be two layers, one of which is the map and the other which is completely black, with the black part on top. Parts of the black frame will be toggled transparent to show the map underneath as the player traverses through rooms.	1	3
Brightness	1.3	Andrew	As a user, I want to change the brightness of the Game.	A setting class tracking the variable and modifications in Adventure review	1	1

Shop	1.4	Daniel	As a user, I want to use the coins I collected from games/defeating trolls to purchase gear for my character such as armor to increase defense, or power ups to increase health/attack, so that I can counteract the difficult challenges I face.	GUI that can be universally accessed where users can buy/sell items to obtain objects that provide user status effects. There will be a hashmap that maps the objects to their quantity in the shop.	1	3
Homepage and Visual Design	1.6	William (shared contribution with Daniel)	As a user who plays games for leisure, I do not want to start the game immediately, so that I can choose to continue the game, start a new one, or change my settings with well-designed pictures to enhance my satisfaction when playing games.	Initialise UI with a homepage that can start a new game, settings menu to change settings, load game, etc. Redesigned the homepage as well as the entire structure of a game and how they should be displayed.	1	4
Difficulty	1.7	William	As a user who plays games for leisure, I want to choose a level of the game that suits my level.	Different level of games are designed with different types of rooms and trolls. The Initialised UI will contain difficulty options that will load a certain level. Several game design changes include: Use WASD key to control directions, design of the gridpane, etc.	1	4
Background Music	1.8	Andrew	As a user who plays games for leisure, I want to be able to hear the background music rather than either nothing or robot commentary	A separate class for Music, will play suitable music based on the current view or player's location, linked to AdventureGameView	2	2
Button Effect	1.9	Alan	As a user who plays games, I want to feel like I am actually interacting with the game visually.	Adding mouse-hover events to enlarge the buttons when they are hovered over.	2	1

Colour Contrast	1.12	Andrew	As a user, i want to Change contrast level for low vision player	A setting class tracking the variable and modifications in Adventure review	1	1

Acceptance Criteria

Name	ID	Description
Map	1.1	The location of the player should be displayed accurately. The player will always be positioned in the centre of a room. The range of the map should encompass every room. The map should be oriented with north facing up. The map should have layers if there exist layers.
Brightness	1.3	Given that I clicked the settings, I could tune the brightness of the game. The setting pages consist of multiple buttons that have different functionalities
Shop	1.4	Given that I click the SHOP button or shop command, I can access the shop. I am able to sell an object the player has in their inventory for funds. I am able to buy an object if and only if the player has enough funds and there is stock available for that object. If the shop is out of stock, there will be an error message that notifies the player.
Homepage and Visual Design	1.6	Given I run the program, the GUI first starts with a homepage, Then I can choose to start a new game/change settings, etc.
Difficulty	1.7	Player can choose the difficulty to start a new game in the homepage. Then, the game with that certain difficulty would be loaded.
Background Music	1.8	Given I play the game I should be able to hear different background music. I should be able to change the volume of the music on the setting page.
Button Effect	1.9	The buttons should change colours when I hover over it. The capabilities of the button should not change.

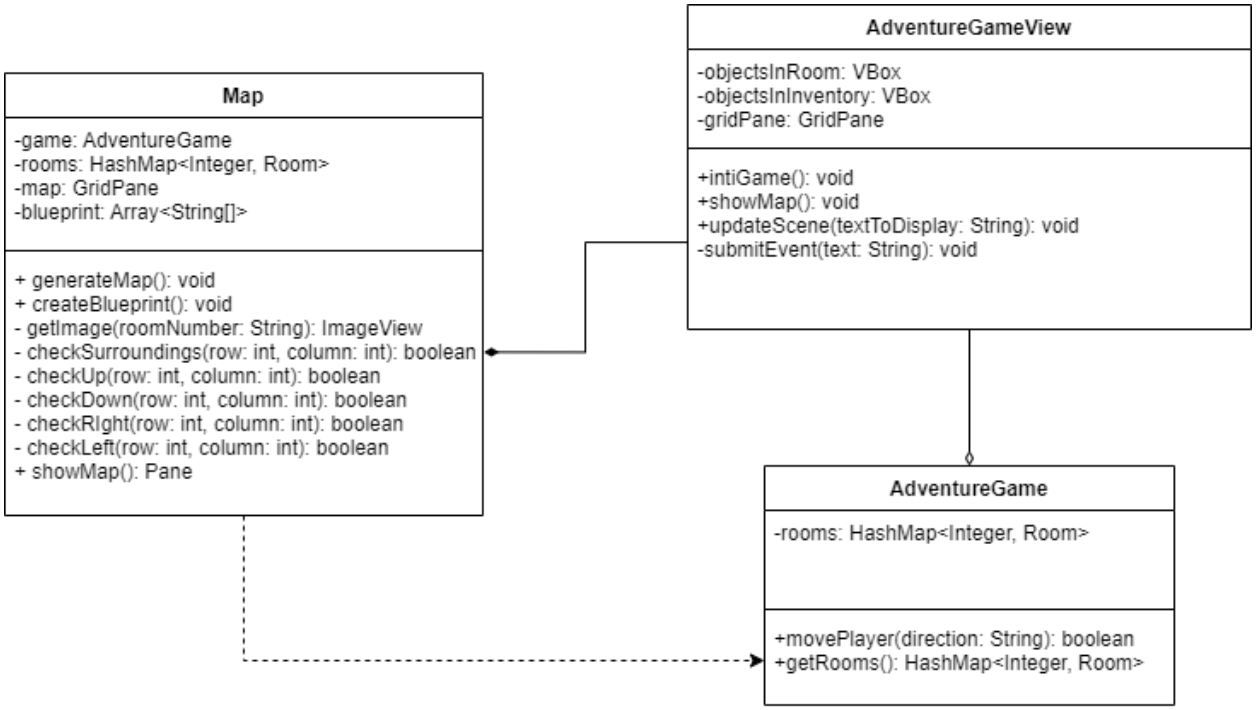
Colour Contrast	1.12	Given that I have restricted vision, I can toggle contrast settings to increase the contrast of pictures so I can see more easily.

SECTION 3: SOFTWARE DESIGN

NOTE: The UML only depicts the additional methods, classes, interfaces, and attributes that will be used or relevant.

Design Pattern #1: Observer Pattern

Map Overview: Map will observe AdventureGame and update itself accordingly and return the result to AdventureGameView.



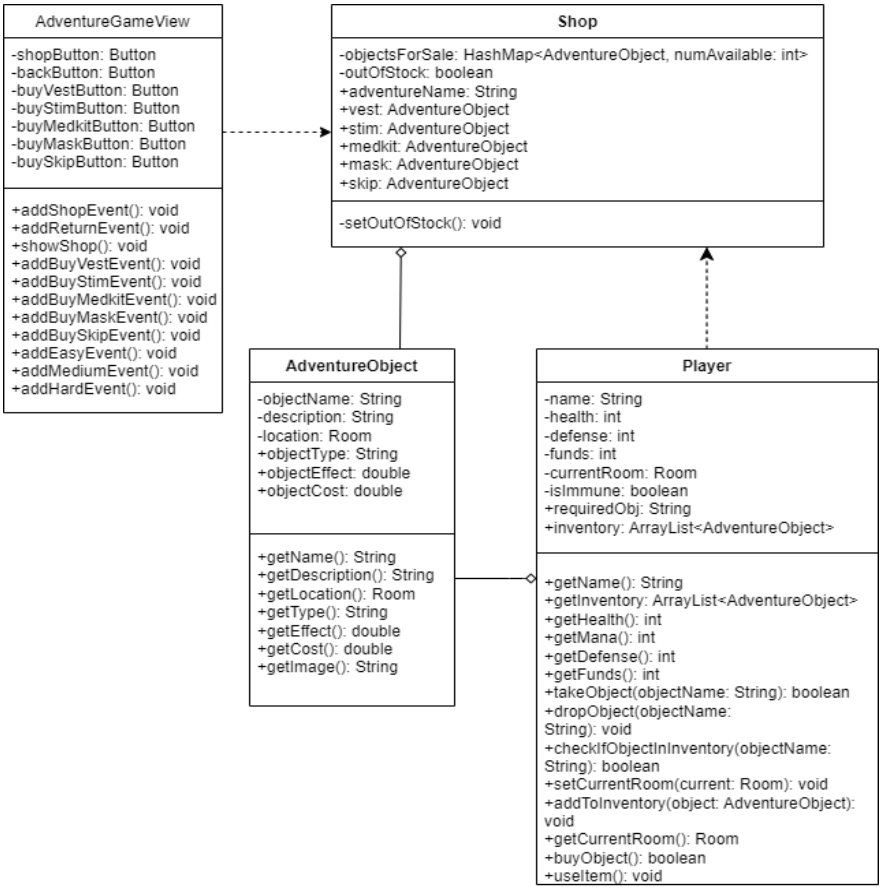
Implementation Details: The UML diagram outlines these main components:

- The *Map* class, *Map* will generate a fresh map with only the visited rooms being visible. *Map* will observe *AdventureGame* whenever the player moves, and use the *rooms* variable from *AdventureGame* to update itself.
- The *AdventureGame* class will update itself through actions made in *AdventureGameView* which *Map* will observe.

- The *AdventureGameView* class will create an instance of *Map* which will generate the given state of the game by accessing *AdventureGame* through *AdventureGameView* and return the updated map to *AdventureGameView* to display through the *showMap* function.

Design Pattern #2: Visitor Pattern

Overview: This pattern will be used to implement a shop structure, where the player (visitor in this case) is able to buy and operate on AdventureObjects (elements of the shop structure) that share similar properties.



Implementation Details: The UML diagram outlines these main components:

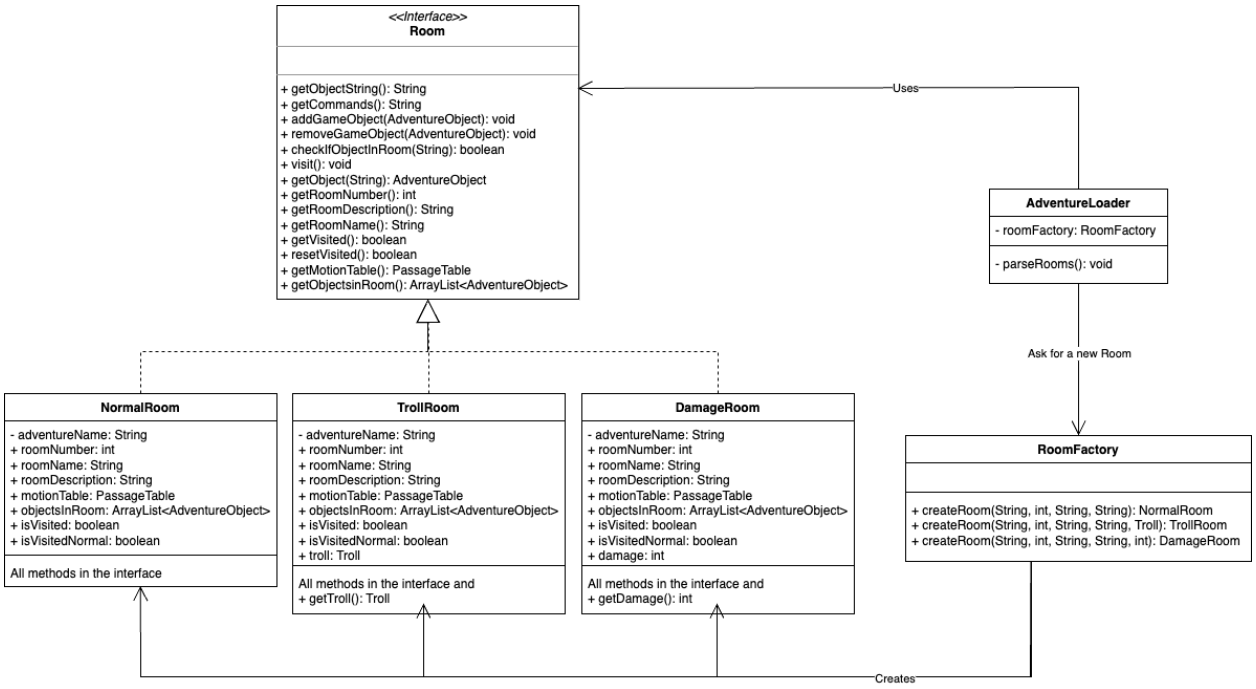
- *Shop* class: The shop class essentially revolves around a Hashmap that maps *AdventureObjects* to their respective quantity available. Attribute outOfStock notifies players if all objects for sale are depleted.
- When there is a successful purchase of an object, the quantity of that object

in the shop will decrease and enter the player's inventory. The behaviour of a *Shop* object will behave differently than that of an object picked up in a room; in this case, the object will be consumed and have its effect applied to the player instead of dropped into the room.

- Both the *Player* class and the *Shop* class share the "has a" relationship with the *AdventureObject* class.
- The interface for the *Shop* class is implemented in *AdventureGameView*, where buttons are created for each object sold in the shop.
- Because every object sold in the *Shop* class is similar in property, we will operate on an object to transfer them onto the player (the visitor). In our case, a transaction. The shop structure holds these objects and keeps track of them. Similarly, the shop structure will include different elements (objects) for each difficulty of the game selected. The methods *addEasyEvent()*, *addMediumEvent()*, and *addHardEvent(),* in *AdventureGameView* determine the difficulty that the shop is allocated to.

Design Pattern #3: Factory Pattern

Overview: This pattern will be used to implement how different kinds of rooms are constructed in different difficulties



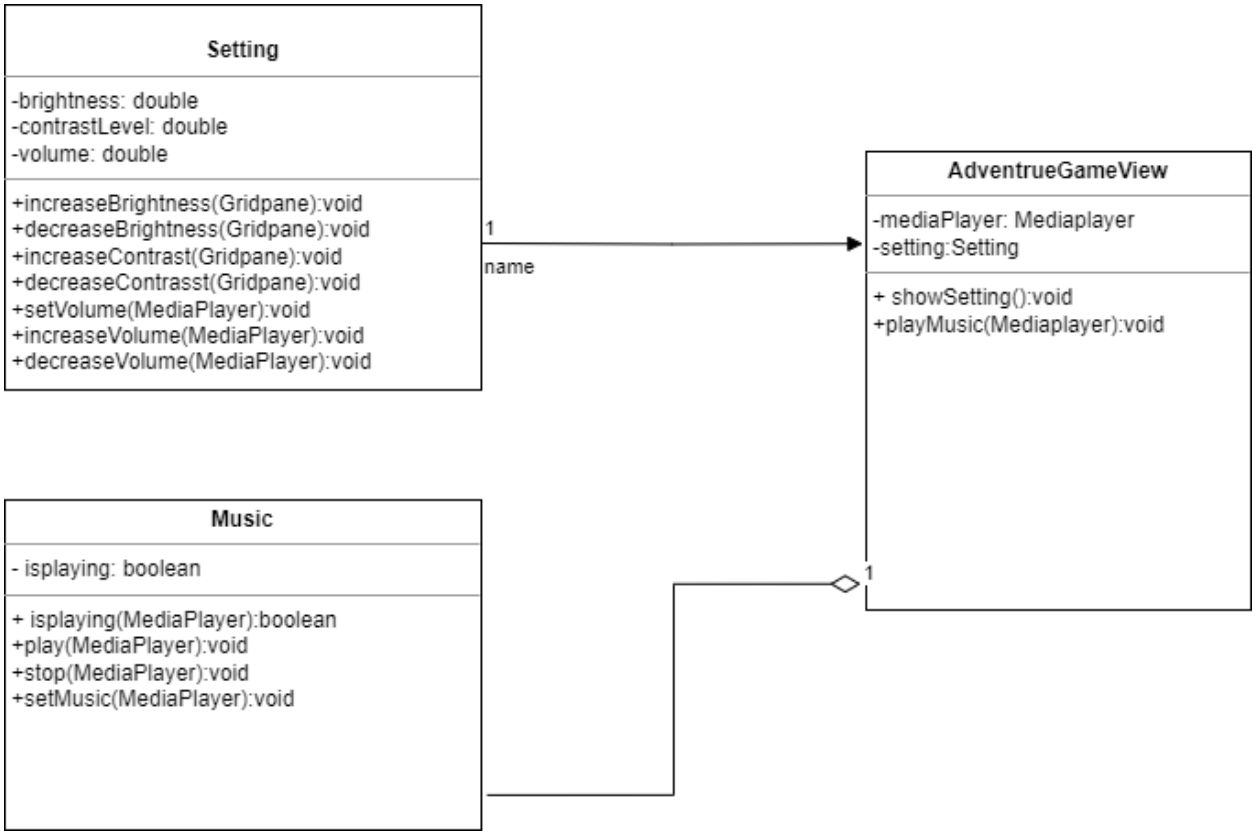
Implementation Details: The UML diagram outlines four main components:

- The AdventureLoader will load the game as usual, but it will call a roomFactory attribute with type RoomFactory to create a specific kind of rooms that AdventureLoader needs. And AdventureLoader will use either NormalRoom/TrollRoom/DamageRoom when creating the game.
- The RoomFactory will be called by the AdventureLoader to create a specific room by calling either NormalRoom/TrollRoom/DamageRoom. The type of room will be created based on the parameters given in the AdventureLoader.
- The Room Interface generates all the methods that is needed for an arbitrary room,
- NormalRoom/TrollRoom/DamageRoom implements the Room interface and creates extra attributes or methods based on the type of the room. E.g. DamageRoom will generate damage on the player, therefore, it has a damage

attribute and a getDamage method. This tells the amount of damage it will do on the player.

Design Pattern #4: Chain of Responsibility

OverView:This pattern will be used to set Audio/Display/Accessibility



Implementation Details: The UML diagram outlines three main components:

- Setting Class: Allows users to change volume, brightness, and set accessibility features. Uses Optional colorContract, text display, and Audio class. If the Game does not provide these classes, related features would be unable to be used.
- Music Class: Responsible for the background music of the game. Can set new music to play or check the status of the current MediaPlayer.
- AdventrueGameView Class: Main class displaying the Game. Have functions that require music and setting class and their methods. Its functions are distributed into setting and music classes.