Design Document

Version 1.0 – 2023.11.6

Created 2023.11.6

Pokémon Adventure Game

Laurence Liu
Athena Cai
Krit Kasikpan
Inayah Dhaliwal

GitLab Repository:

https://mcsscm.utm.utoronto.ca/csc207_20239/35

SECTION 1: PROJECT IDENTIFICATION

We are motivated to create a Pokémon adventure game for our group project because it offers us valuable educational experience in game development, programming, and project management. Additionally, it provides an opportunity to enhance our teamwork and collaboration skills while enabling us to express our creativity and innovation through unique game mechanics, stories, and art assets.

Our plan to improve the existing adventure game involves visual upgrades, enhanced accessibility, additional gameplay features, extensive character abilities, dynamic audio enhancements, community-centric features, cross-platform compatibility, regular content updates, and modding support to create a more immersive and inclusive gaming experience.

SECTION 2: USER STORIES

Name	ID	Owner	Description	Implementation Details	Priority	Effort
Accessibility	1	Laurence Liu	As a player with mobility challenges, I want to be able to fully enjoy and engage with the video game, so that I can have an immersive gaming experience without limitations.	Add alt text to every Pokémon image Implement high contrast mode Allow users to zoom in on displays Implement text to speech for all in game dialogue Implement full keyboard navigation for all in game controls. Use the java.accessibility library to implement built in screen reader for all in game menus,	1	2
Exploration	2	Krit Kasikpan and Inayah Dhaliwal	As a player who loves exploring, I want to wander through various in-game locations to encounter and catch wild Pokémon, making exploration an integral part of the game.	Add Pokémon objects to each of the location in the game, and give user the ability to catch the Pokémon into their roster Allow User to see the path of the rooms they have visited via AdventureMap	1	3
Visual	3	Krit Kasikpan	As a player who loves watching Pokémon and playing visually appealing games, I want to have a visually interactive game to have a more captivating and enjoyable gaming experience	Location images inspired by Pokémon settings. Pokémon object images and enemy sprites from the show. Artistic style consistent with the Pokémon universe. Inventory displaying captured Pokémon. Pokémon details: name, moves, type.	1	3
Battle class battling logic	4	Athena Cai	As a player that likes strategy games, I want the ability to engage in Pokémon battles against other Pokémon trainers, to have exciting gameplay experience.	Create a battle class that handles the battle loop and battle logic. Enable player to interact with opponent to enter a battle against them Create turn based battle between player and opponent.	1	3
Battle class getting user input for moves	5	Athena Cai	As a player that likes hands on strategy games, I want the ability to choose strategic moves in Pokémon battles against other Pokémon trainers.	Enable user to see available Pokémon moves and relevant statistics such as energy points an and attack points Enable user to choose the Pokémon move they want to use through text input and have the battle in the game respond to their input Ability for user to pass their turn	2	3

Battle class allowing users to select Pokemon	6	Inayah Dhaliwal	As a player who has collected a roster of Pokemon, I want to choose which of my Pokemon I want to use in a battle	Enable player to see the Pokemon in their inventory Enable player to select deselect Pokemon from their inventory and begin the battle once three have been chosen	2	3
NPC Abstract Class	7	Laurence Liu	As a player who wants human interaction through the game, I want to interact with NPCs and have dialogue	Add concrete classes Villager and Opponent, place Villagers in rooms to support Players, and place Opponents in Battles. Allow NPCs to talk.	2	2

Acceptance criteria for these user stories are below.

Name	ID	Complete Acceptance Criteria List
Accessibility	1	Every game object within the game has a description attribute that accurately describe the object. The user interface includes a button to switch to toggle high contrast mode, enhancing visibility and readability. Text, images, and interactive elements should be distinguishable in high contrast mode. Users should be able to zoom in on any part of the game display without loss of clarity or functionality. The user interface features buttons for zooming in and zooming out, along with keyboard navigation controls to pan across the screen when it's zoomed in. All in game dialogue is accompanied by text to speech that reads out the displayed text in a clear and understandable manner. Users can toggle between various screen nodes by pressing the left and right arrow keys. Every button is linked to a 'make button accessible' function, which assigns a name and a description to each button for accessibility purposes. All in-game commands can be inputted using the keyboard.
Exploration	2	There are multiple locations that contain Pokémon objects. Different types of Pokémons vary by different in game location to encourage exploration There is a look command that displays to the screen every interactable object in their current room. Players can capture wild Pokémons in a room by clicking on its image or typing the “take” command. Pokémons captured are added into the players backpack and removed from the room. Players can release wild Pokémons from their backpack into the room by clicking on its image or typing the “release” command. Pokémons released are removed from the player's backpack and added into the current room the player is in.

Visual	3	Every Pokémon object has an attached image_address attribute that stores the string of its corresponding image base on its index. Every room object has an attached image_address attribute that stores the string of its corresponding image base on its index. Every villager object has an attached image_address attribute that stores the string of its corresponding image base on its index. Every opponent object has an attached image_address attribute that stores the string of its corresponding image base on its index. Visuals are clear and sharp and are easily viewable by the user.
Battle class battling logic	4	The opponent's active Pokémon selects a move to execute at random from its available move set during each turn. When the player's active Pokémon executes a move, the correct amount of damage, based on the move's specified damage value, is dealt to the opponent's active Pokémon. When the opponent's Pokémon executes a move, the player's active Pokémon receives damage according to the move's damage value. The opponent's active Pokémon will be knocked out when its health point falls below 0. The player's active Pokémon will be knocked out when its health point falls below 0. When an active Pokémon is knocked out, the player is able to selec
Battle class getting user input for moves	5	The active Pokémon in battle will display a string of available moves that the user can enter. The player can enter available Pokémon moves which calls for the Pokémon to execute selected move. All Pokémon's can execute PASS to pass their turn and gain energy.
Battle class allowing users to select Pokémon	6	The player can enter a battle if the player has at least 3 Pokémon's in their backpack. When a player enters a battle, the player can select 3 Pokémon's from their backpack to battle with.
NPC Abstract Class	7	There are interactable opponents added in between rooms that trigger a battle. There are interactable villagers added to various rooms in the game. There is a talk command that allows the user to interact with the villager if there is a villager present in the room.

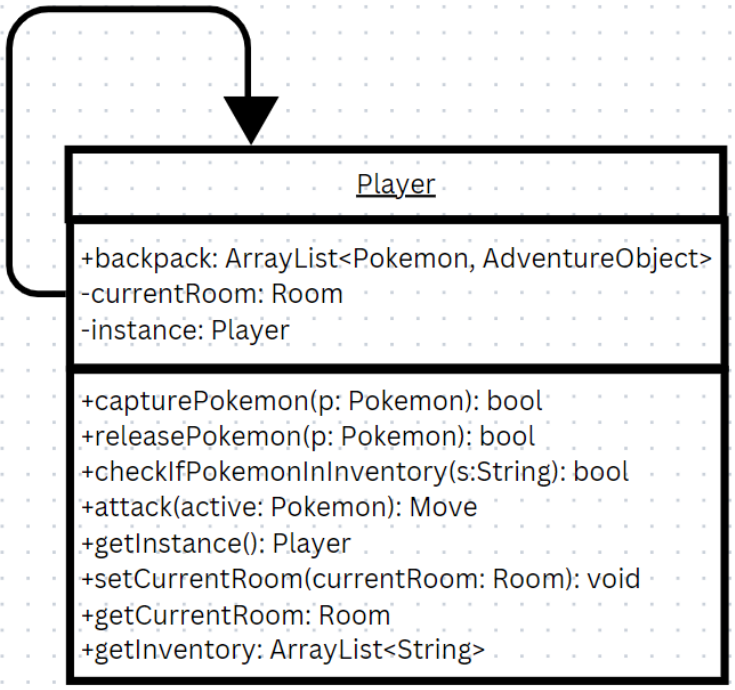
The talk command will display the next phrase in the villager's dialogue.

Villagers has the ability to drop Pokémon's into the room.

SECTION 3: SOFTWARE DESIGN

Design Pattern #1: Singleton

UML Diagram:



Overview: This pattern will be used to implement a class Player that ensures only one instance of itself exists throughout the application's lifecycle

Implementation Detail: The UML diagram outlines the main component:

- Class Player which includes the *getInstance()* method

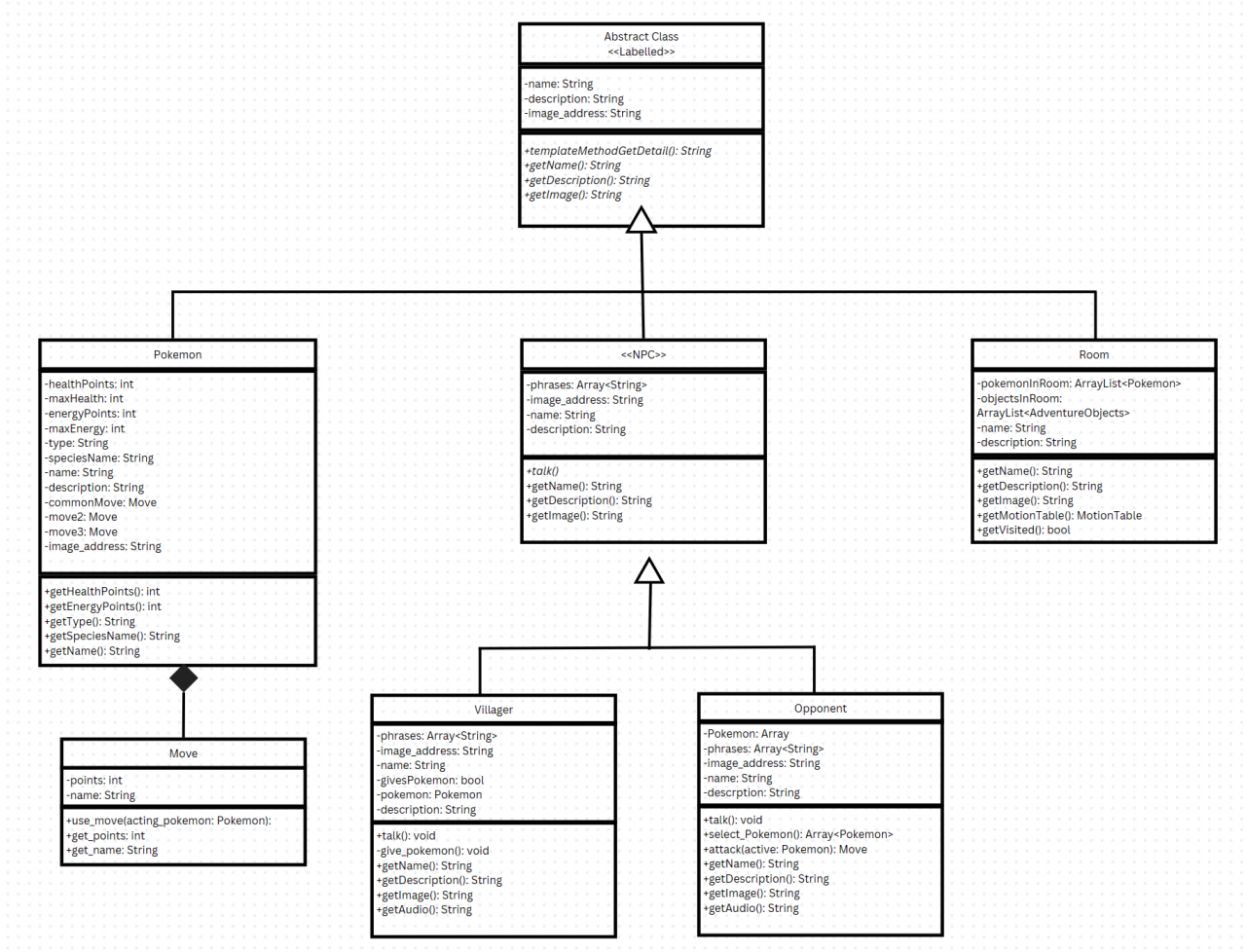
The method *getInstance()* works as follows:

- Initialization: When the application starts or when the first call to *getInstance()* is made, the Singleton class creates an instance of itself and stores it.

- Subsequent Calls: For all subsequent calls to *getInstance()*, the method checks whether an instance of the class already exists. If an instance exists, it returns that instance; otherwise, it creates a new instance.
- Returning the Instance: The *getInstance()* method returns a reference to the single instance of the class, allowing other parts of the program to access and use it.

Design Pattern #2: Template Method

UML Diagram:



Overview: This pattern will be used to implement a behavioural pattern that provides an abstract class Labelled for defining the skeleton of an algorithm in an operation; in this case it is getting details about objects and deferring some steps to subclasses for class-specific operations. This design pattern lets our subclasses redefine certain steps of an algorithm without letting them to change the algorithm's structure.
Implementation Detail: The UML diagram outlines these main components:

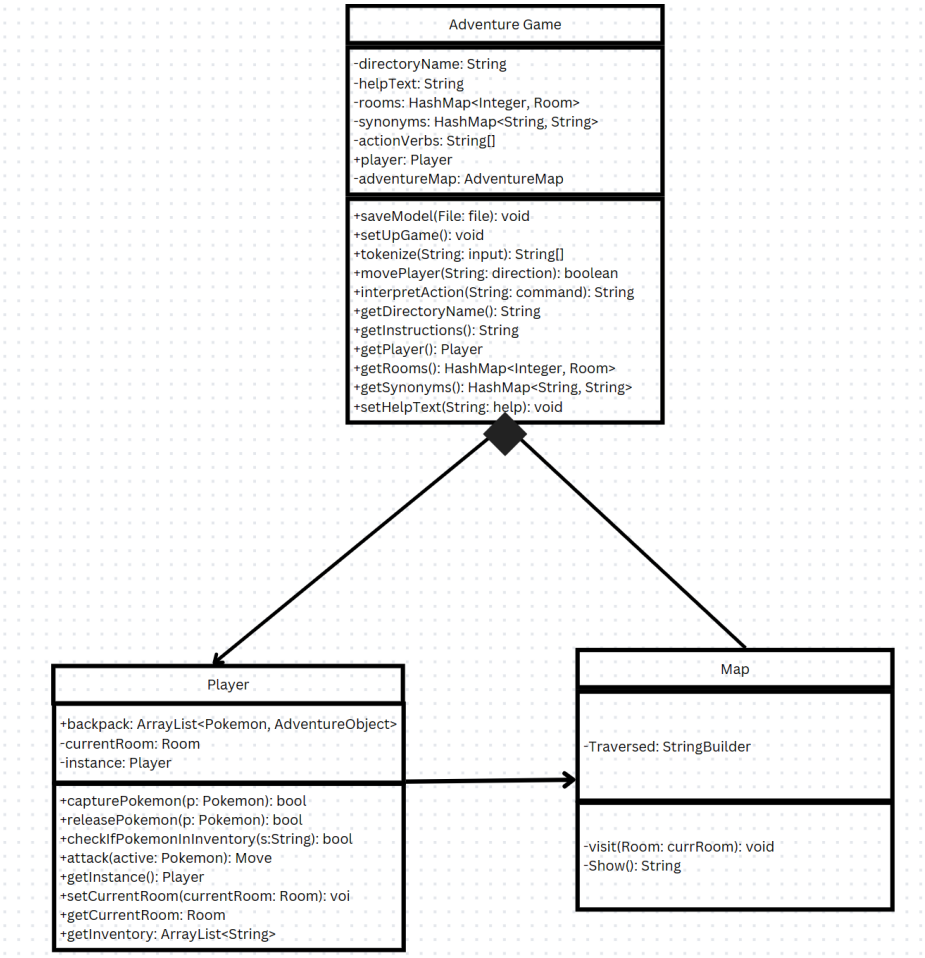
- Template Method: Abstract Class Labelled contains three common methods among its concrete classes: *getName(), getDescription(), getImage().*
- getName() will return the name of the object

- getDescription() will return the description of the object
- getImage() will return the directory address the of the image
- Concrete Factories: these are the classes that may change methods in the Template Method abstract class. These are the Pokémon, NPC, and Room class.

This pattern promotes loose coupling between the client code and the specific classes, allowing player to switch between different families of objects without modifying the client code.

Design Pattern #3: Observer

UML Diagram:



Overview: This design pattern defines a one-to-many dependency between objects so that when the Player object changes state all its dependents are notified and updated automatically. The single instance of the Player class is the Subject. The Adventure Game and AdventureMap classes are the Observers.

Implementation Detail: The UML diagram outlines these main components:

- Adventure Game will use Player and Map to show Observer pattern in Action
- Traversed stores a String of all rooms visited by the Player which we get from the

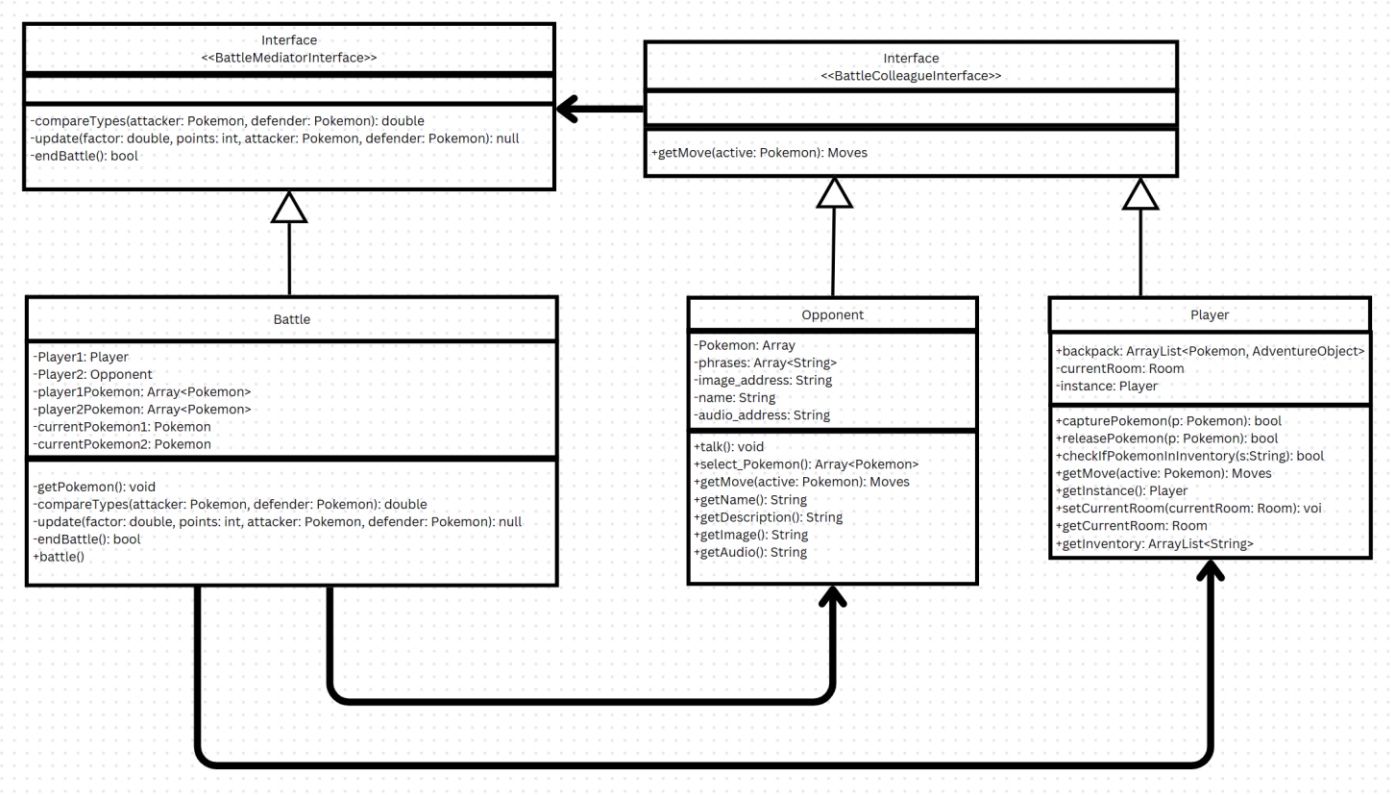
getCurrentRoom() method and is saved in visit()

- The Show() method shows the traversed rooms in the RoomDescLabel

In conclusion, when the Player moves through a room, it is stored in AdventureMap and hence displayed on the screen with a list of all the rooms visited by the Player in order of which they were visited

Design Pattern #4: Mediator

UML Diagram:



Overview: Battle is a Mediator class that encapsulates how Player and Opponent objects interact. The Battle Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets us vary their interaction independently.
Implementation Detail: The UML diagram outlines these main components:

- Battle requests Pokémon moves from Player and Opponent, with proper timing, I.e. ensuring the players take turns.
- Battle updates points of Player and Opponent Pokémon based on attacks that were made.

The compareTypes method performs type logic based on the types of Pokémon used by Opponent and Player to decide how effective an attack move is.

The battle method loops between Player and Opponent turns. For each turn, the following steps are taken:

1. Request a move
2. Compare types to determine effectiveness factor and multiply the factor by the attack

points of the move

3. Update the health and energy points of the Pokémon according to the move
4. Check if either of the active Pokémon are empty on health points. If they are, then switch out with the other Pokémon in the Pokémon arrays. If all Pokémon of either the Player or Opponent have 0 health, then battle is over. If the Player’s Pokémon all have 0 health when the battle ends, then the Player has lost the battle. Reset the health and energy points of all the Pokémon that were involved in the battle and return true if the Player won and false otherwise.