CSCI205 Final Project
Connor Coles
Mikey Ferguson
Eric Reinhart
Jiasong Zhu

Introduction

In order to have a Pokemon battle, the creation of the Pokemon themselves must come first. An abstract class called *Creature* defines the core attributes of every Pokemon by type (fire, water, grass, normal). Each type has its own set of 2-3 Pokemon, which each have an associated name, sprite, and set of 4 possible moves to use in battle. A subclass of *Creature* is created for each type, allowing for the instantiation of a specific Pokemon by the *Player* class. Two instances of the class *Player* are created, which each generates a team of 6 non-repeating random Pokemon, one for the user and one for the enemy.

All of the available moves that can be used by the Pokemon are derived from a functional interface *Move*, which allows for the creation of two kinds of moves: attack moves and support moves. Attack moves deal damage to the opposing Pokemon, whereas support moves benefit the user in some way, such as healing itself or increasing its own speed. Each of the 4 types have 1 "weak" and 1 "strong" move associated with each of them. A "weak" move has 100% accuracy but deals less damage, whereas a "strong" move has 80% accuracy but deals more damage. Moves with less than 100% accuracy have a chance to miss, meaning the user deals no damage that turn. Depending on the type of the attack move and the opponent's type, the attack may deal double damage, half damage, or neutral damage. For example, if a fire type move is used on a grass type, the move would deal double damage. If the fire type move is used on a water type, it would deal half damage. Both of the support moves have 100% accuracy, one of which heals the user and the other increases the user's speed (making the user more likely to attack first).

Upon launching the program, the user is faced with a welcome screen, soothing music, and the choice of Normal or Hard Mode. Normal Mode is good for first-time players to learn the mechanics of the game, and Hard Mode is better for more experienced users to test their skills and try new strategies. Pressing the either button switches to the battle itself, which takes place in the *Arena*. One Pokemon from the user and one Pokemon from the enemy battle at a time, and the name, health, and sprite of both Pokemon are shown. Each turn, a player has two options: use a move or swap Pokemon. To be clear, a turn is when either the user swaps to a different Pokemon or both Pokemon select a move and ends after either both Pokemon use their moves, or one of the Pokemon faints due to an opponent's attack before it is able to use a move. If the user wishes to use a move, they can select one of the four moves on screen, which are colored based on their type and give additional information such as damage and accuracy when they are hovered over due to tooltips. If the battle is on Normal Mode, then the enemy will randomly select its moves. If the battle is on Hard Mode, then the enemy will strategically swap Pokemon and use moves that would be strong against the user's Pokemon.

After both Pokemon select a move, the speed of the user and the enemy are checked, and the faster Pokemon moves first. The animation of the moves are shown on screen and a chat log explains what occurs during each turn. The chat includes messages such as the move used by each Pokemon, the effectiveness of the attack (super effective, not very effective, or nothing if it's neutral damage), an indication if the move missed, and whether either Pokemon has fainted. If the user's Pokemon faints, they are sent to the selection screen and must select which Pokemon to be sent out next otherwise a random alive Pokemon will be sent out. If the enemy's Pokemon faints, they send out the next Pokemon in the order they were added to the team.

Instead of attacking, the user may choose to swap to a different alive Pokemon. This button brings the user to a selection screen with their 6 Pokemon and their health, and the user can only send out a Pokemon that is still alive. This is where strategy can come into play, since the user can use their knowledge about Pokemon type matchups to swap to a Pokemon that would be strong against the current opponent. If the user successfully defeats all 6 opposing Pokemon, then they are sent to a winning screen with the options to play again or quit. If the user loses, then they are sent to a losing screen with the same options.

User stories

We had three user stories that we based our user interface on. The first one was a bored child. The bored child has never played a pokemon game before. They are waiting for their dad to come back in the car from the grocery store bored out of their mind with nothing to do. The bored child wants something to pass the time, something that is exciting and entertaining. They find our game online and are interested. We designed our user interface so that anyone is able to have a great time exposing the cool creatures and moves.

Our second user story is the first time player. The first time player does not play that much and is their first time playing a Pokemon spin off game. We wanted our game to be easily understandable and straightforward, but still be challenging. When you first boot up the game it is very straightforward with a start button and quit. You are then brought to the arena. Here in the arena you are given four moves each with a unique name. If you hover on the move it gives a description of the move so the player knows exactly what the move does. We wanted the player to learn on their own and experience the game however they so desired. So while we do explain what the move and what it does, how they want to proceed is completely up to the player. There is just enough information for them to grasp the concept and build on it.

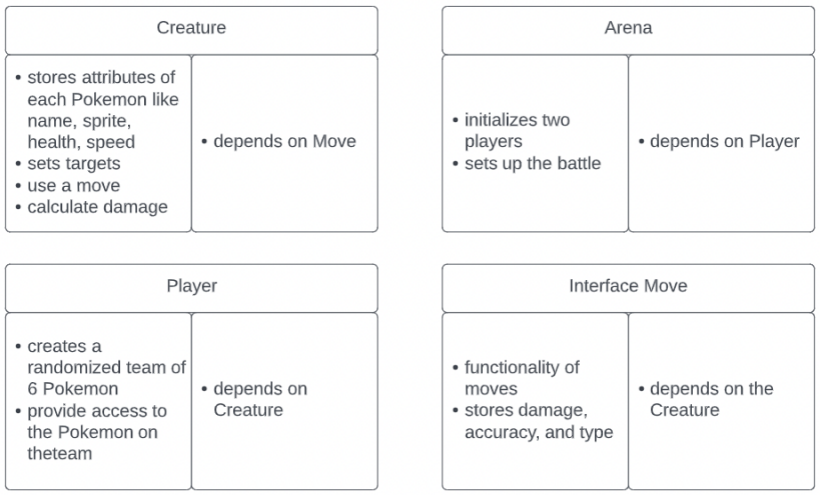
Finally our third user story is the casual gamer. The casual gamer comes home after a day of work and wants to unwind and play some games. They coincidentally find our game after looking for something new to play. They have played Pokemon in the past and have a general knowledge of how the game works and what to do. They boot up the game and are able to recognize the Pokemon and grasp how to play the game pretty quickly. After playing the game

once they find themself wanting to play again. With our user interface we have made this quite easy. Whether you win or lose you are brought to a screen that will allow you to return to the home screen to start up a new game with the pokemon randomized again so the battle is never the same. They play for an hour and are satisfied, they then go eat a nice family dinner.

Object-Oriented Design

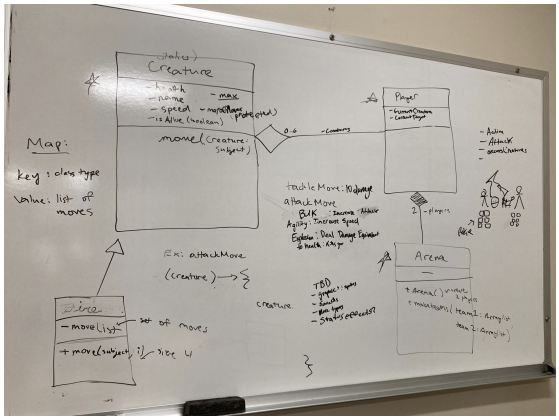
We wanted our program to be easily iterated and modified, so we decided to split it up into four main packages: Creatures, Battle,Moves and Application. These four packages include different object oriented design ideas including interfaces and abstract classes. Our Battle includes the Player and Arena class. The Player is dependent on the Arena because the arena creates both the user and the enemy team. The Creatures package contains many classes, but is mostly based around the design of one main abstract class Creature. The abstract class has tons of static maps that store all sorts of information on the moves, sprites, and names of the different types of Creatures. With an abstract class we are able to easily scale to create new types of Pokémon for our game.

CRC Cards

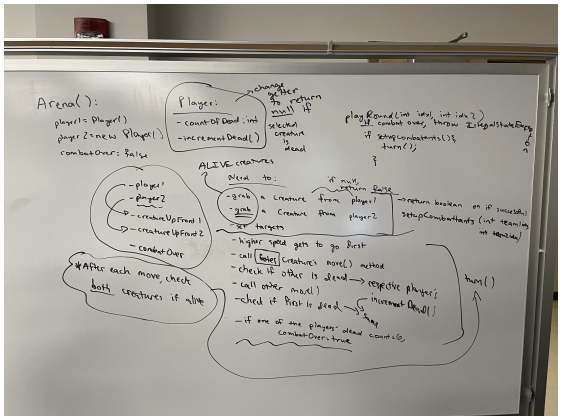


UML Diagrams

Over the course of a month we made many different iterations of our LucidChart. Although we were unable to look at all of the revisions that we made over the course of this month we were able to find some of the older mock ups for our game that we drew on whiteboards.

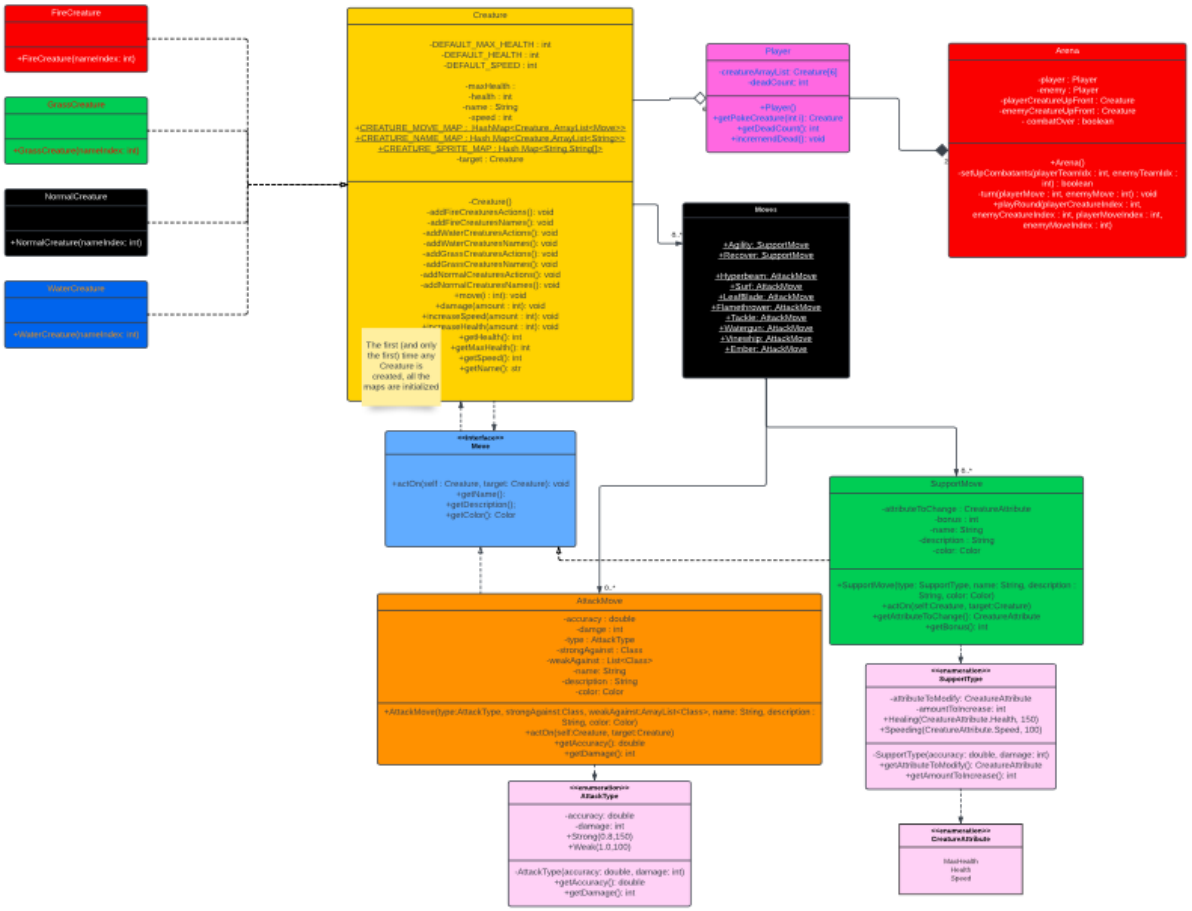


First iteration of our UML Diagram



Later iteration for the Arena class

Final UML Diagram



Below are the Intellij UML diagrams that were generated. The *Creature* class is an abstract method that holds base attributes for all types of creatures that extend *Creature*. The different creature types have unique moves, names and sprites that are set up in the abstract *Creature* class (Figure 1). *Move* is an interface that defines all of the methods for a move (Figure 2). *AttackMove* and SupportMove both implement the *Move* interface and build upon the

methods (Figure 3). They define the attributes for the unique Attack and Support moves. *AttackMoves* and *SupportMoves* both have Enumeration classes *AttackType* and *SupportType* that allow users to define if a move is strong, weak , heals or increases speed. The class *Moves* instantiates *AttackMove* and *SupportMove* to create all of the unique moves (Figure 4).

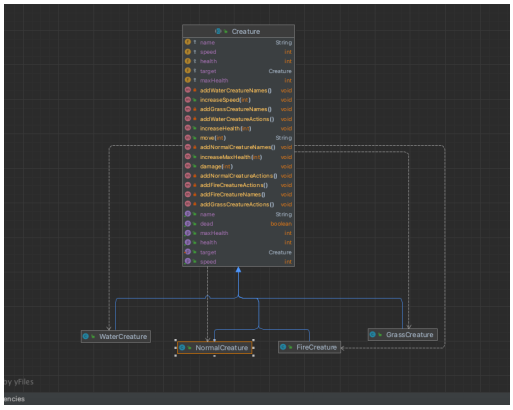


Figure 1

Figure 2

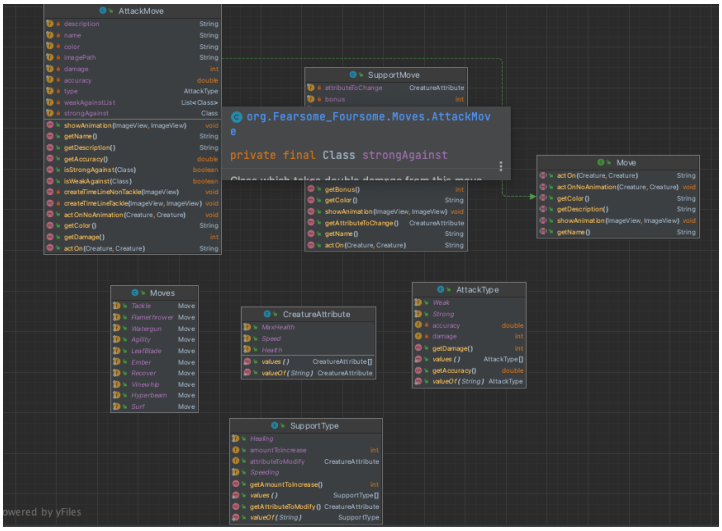


Figure 3

Figure 4

The *Player* class relies heavily on *Creature* since it holds a team of 6 randomized, non-repeating Pokemon, and allows for easy access to the Pokemon on each team (Figure 5). The *Arena* class initializes two players: one for the user, and one for the enemy (Figure 6). The *Arena* is responsible for setting up the 2 Pokemon on the field at that time and handles all of the battle mechanics.

Figure 5

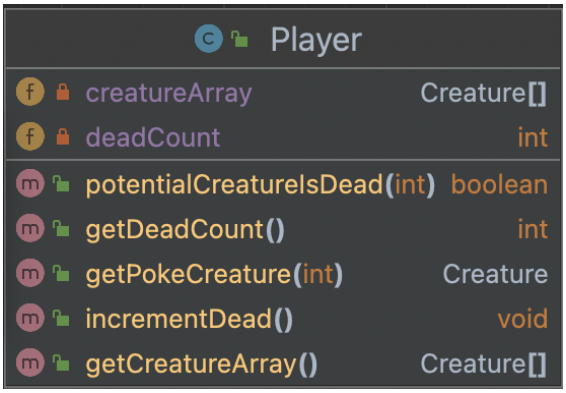
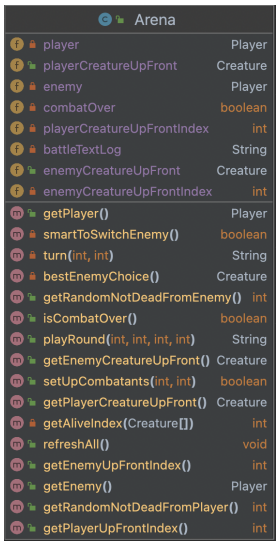


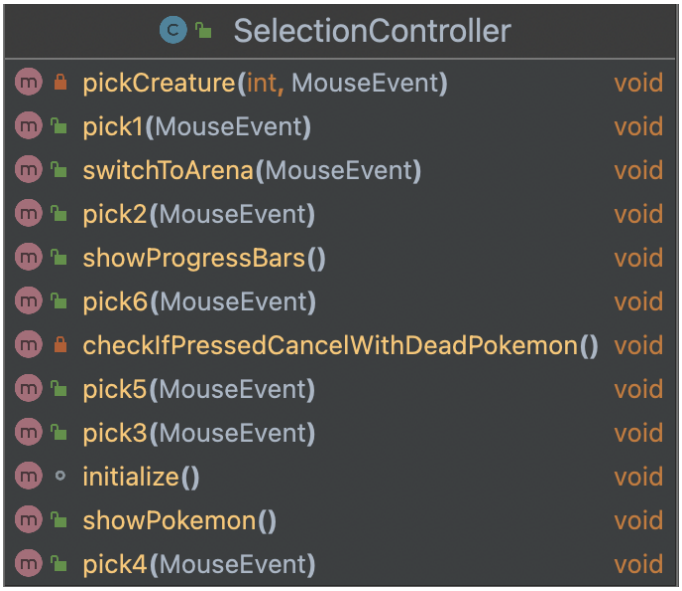
Figure 6



Now, onto the Controllers. *ArenaController* is easily the most vital class for the functionality of the game, since it controls the entire flow of the battle by relying on *Arena*. This class is responsible for taking in user input, running calculations, and handling nearly everything that happens on screen during the battle (Figure 7). When the user's Pokemon on the Arena dies or the user chooses to swap, they are sent to the *Selection* screen, which is controlled by *SelectionController*. This Controller displays the name, health, and sprite of all of the user's Pokemon and allows them to switch to any of the six pokemon of their choosing. The Arena class is able to display our unique moves that we created for different types using the method *setUpMoves*. Another pivotal method in the *ArenaController* is the *playARound* method. It sets up the targets for each Pokemon, keeps track of dead and alive pokemon, and makes sure the game ends properly.

Figure 7

Figure 8



WinnerController and *LoserController* are very simple, as they change the scene to winning and losing, respectively, then play corresponding music (Figure 9). *MenuController* is also very simple, and it allows the user to select Normal or Hard difficulty or Quit the game.

Figure 9

Figure 10