**Technical Viewpoint**

An overview of the design can be separated into four main categories, with them being the specific pokemon and their attributes, users (both bot and player), the physical game states, and lastly the JavaFX. The pokemon class uses the move class, moveInventory class, pokemonStatusEffect enumeration. The player class mainly takes from the Pokemon class to initialize the teams players use, thus the PokemonInventory belongs to the Player class. Additionally the bot class is a child class of the player class with its methods taking from the Pokemon class. The physical gameplay takes into account both the player and bot and utilizes them in the BattleMicro class. The BattleMacro class also falls under the physical gameplay. Both of these "Battle" classes are what primarily makes up the game and the JavaFX uses them to output display.

**User Stories**

Implemented Stories

Immersed Player (IP)

> This player likes to enjoy the game at any difficulty level as long as the gameplay is immersive enough. IP wants decent graphics and with a modern game expects some degree of animation, even at a minimal level. While we didn't invest a lot of time in making animations, we invested the minimum in finding animated gifs to represent a lot of our Pokemon dataset.

Challenger Player (CP)

> This player likes a challenge and wants to explore harder difficulty levels in any game they play. Thus, we made sure to include at least a harder difficulty level than our baseline, normal difficulty level. CP will come to love the harder difficulty level, as it pretty much knows everything about their current Pokemon and the player's current Pokemon at any given time and can do calculations to choose their best move against yours.

Collector Player (CP)

> The CP enjoys being able to utilize and "collect" different types of things in any game they play. In this game, we made sure to have a wide variety of Pokemon in our dataset such that this kind of player will be able to have fun using different types of Pokemon with different stats, moves, and types in their battling.

Enemy Explorer Player (EEP)

The EEP likes to explore different types of enemies, regardless of difficulty level. They are more interested in the backstory and details of their enemy. The only two enemies that this kind of player can explore in our game are the normal and the hard difficulty bots; while this is not exactly a wide array of selections to explore from, two beats one!

Non-implemented Stories

Social Player

The social player wants to interact with other players in the game. However, we realized that the Pokemon Battle Factory is a single player game at heart, and thus we did not implement a multiplayer aspect. Furthermore, we were out of time to implement and invest resources into researching how to make alterations to our current game into something with a multiplayer aspect.

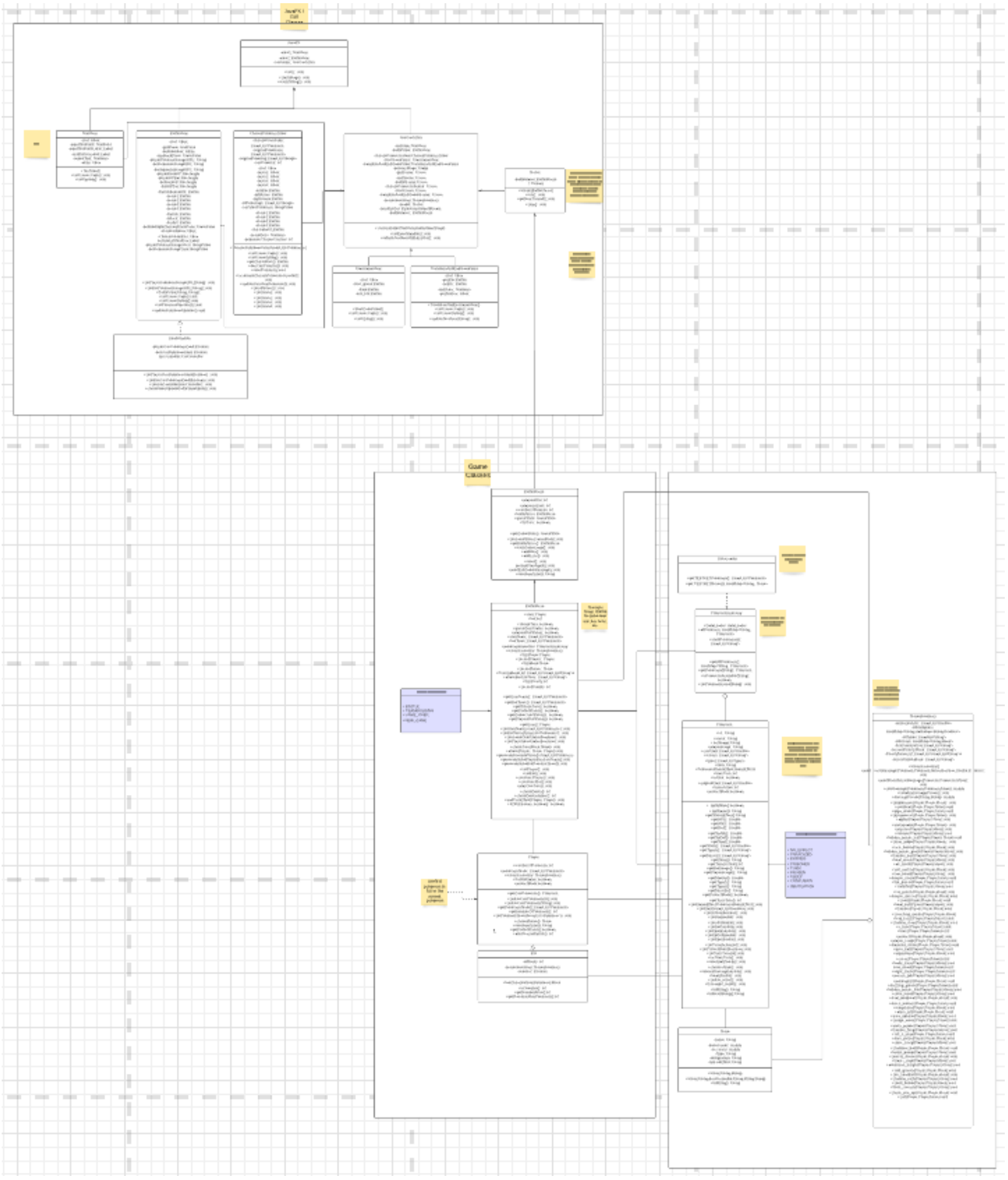
# Object Oriented Design

## CRC Cards

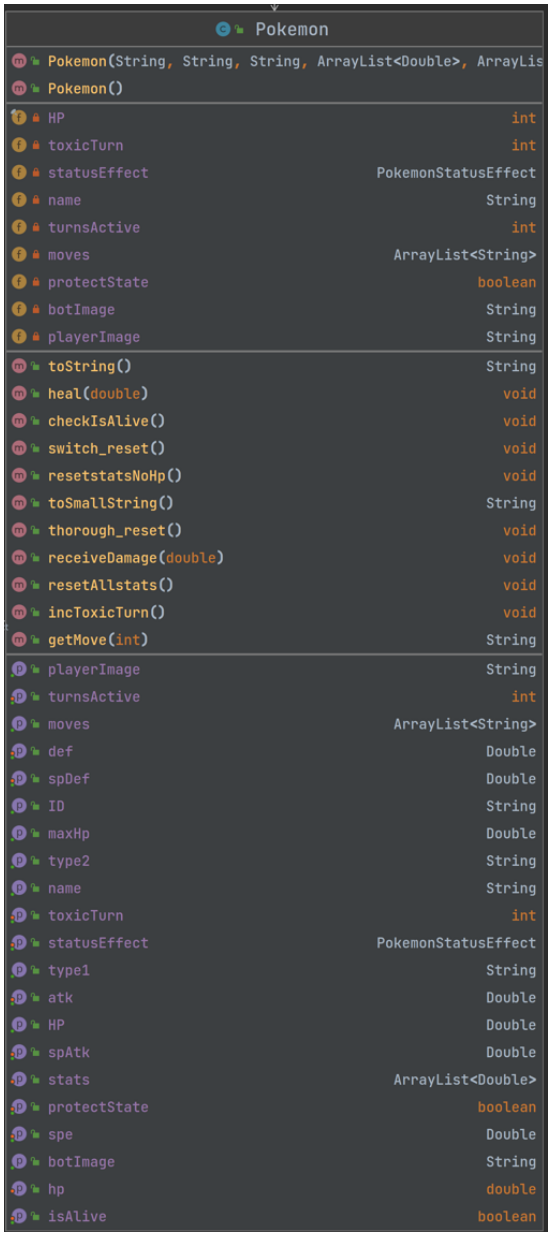

The above illustration shows all of the CRC cards used to illustrate how our JavaFX was designed and created.

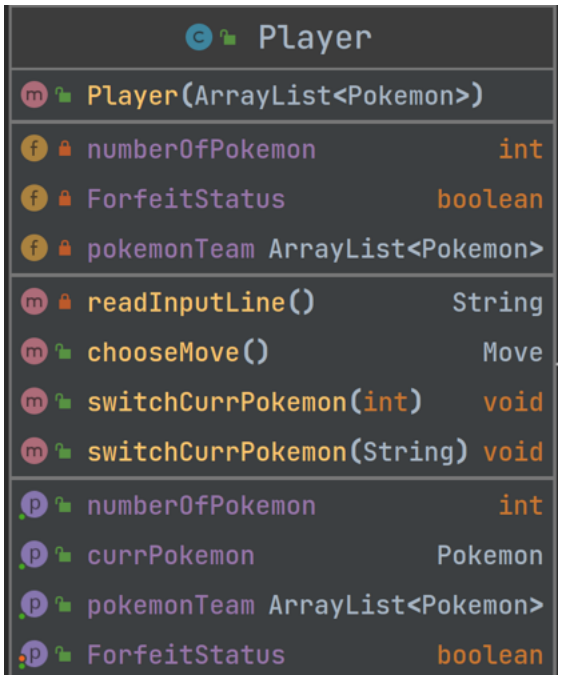

The above illustrates the CRC cards that we designed and used to construct our classes for the Main Game model.

## High-level UML Diagrams

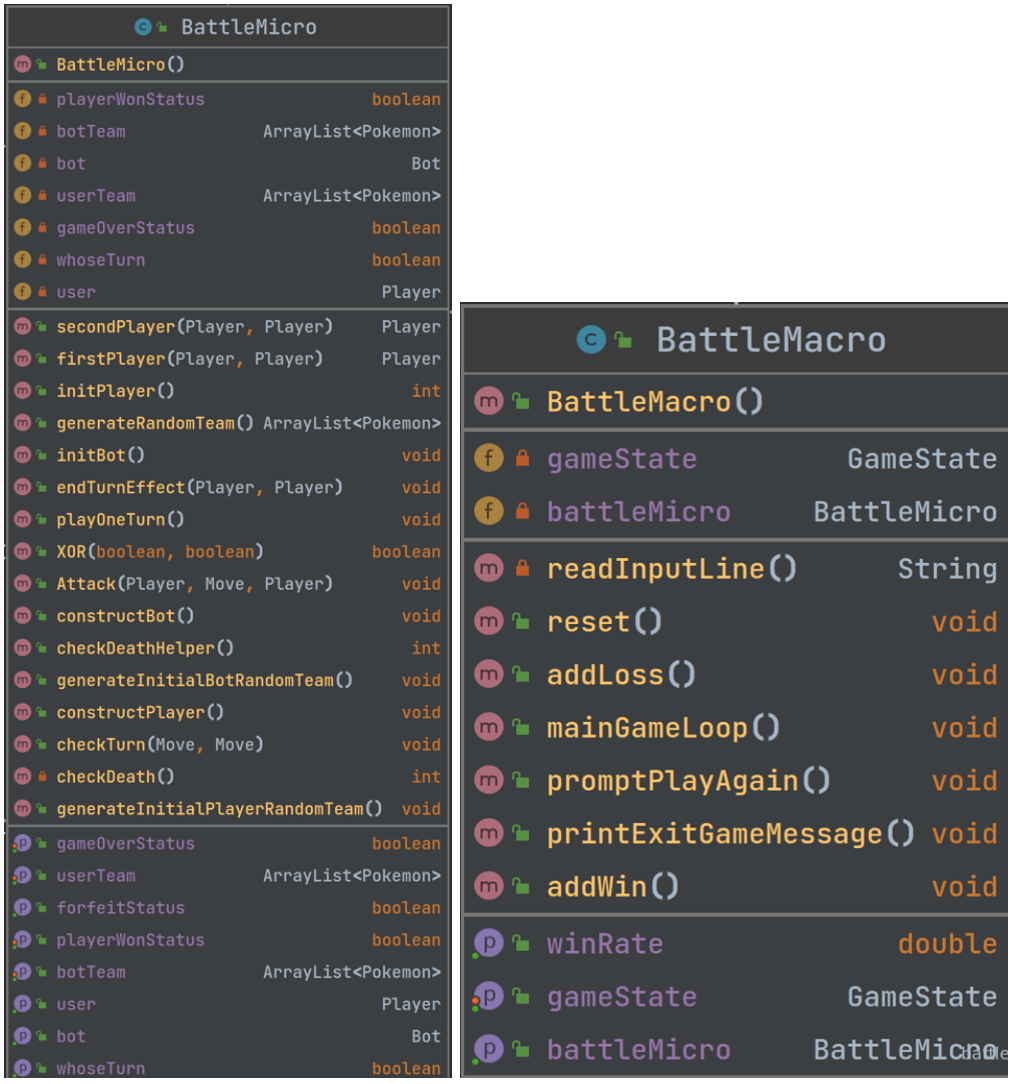


## Key Screen Shots of the IntelliJ Generated UML Diagrams.

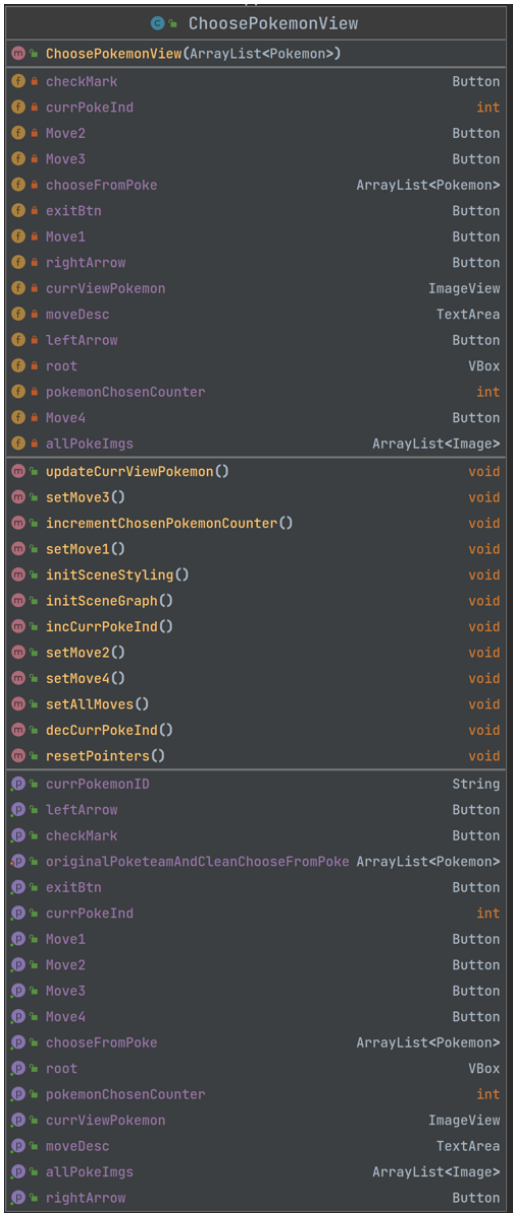



The above two screenshots show the UML class diagrams generated from Intellij for our Pokemon and Player classes. We want to emphasize the shear number of functionals and attributes that each class has in order to properly represent our game entities.




The above two screenshots show the classes that handle all of the higher level game functions. They ultimately serve the main game loop and how different classes come together to make the game functioning.


ChoosePokemonView
ChoosePokemonView(ArrayList<Pokemon>)
checkMark Button
currPokeInd int
Move2 Button
Move3 Button
chooseFromPoke ArrayList<Pokemon>
exitBtn Button
Move1 Button
rightArrow Button
currViewPokemon ImageView
moveDesc TextArea
leftArrow Button
root VBox
pokemonChosenCounter int
Move4 Button
allPokeImgs ArrayList<Image>
updateCurrViewPokemon() void
setMove3() void
incrementChosenPokemonCounter() void
setMove1() void
initSceneStyling() void
initSceneGraph() void
incCurrPokeInd() void
setMove2() void
setMove4() void
setAllMoves() void
decCurrPokeInd() void
resetPointers() void
currPokemonID String
leftArrow Button
checkMark Button
originalPoketeamAndCleanChooseFromPoke ArrayList<Pokemon>
exitBtn Button
currPokeInd int
Move1 Button
Move2 Button
Move3 Button
Move4 Button
chooseFromPoke ArrayList<Pokemon>
root VBox
pokemonChosenCounter int
currViewPokemon ImageView
moveDesc TextArea
allPokeImgs ArrayList<Image>
rightArrow Button


BattleView
BattleView(String, String)
Poke1Btn Button
Move4 Button
Poke0Btn Button
currPokemon Pokemon
botCurrPokemon Pokemon
botPokemonImageURL String
Attack Button
Switch Button
Move2 Button
playerHpBar Rectangle
root VBox
Move1 Button
Move3 Button
BotHpBar Rectangle
Forfeit Button
playerPokemonImageURL String
initSceneStyling() void
updatePokemonSprites() void
bottomRightBoxToggleChoices(int) void
updateMovesBox() void
initSceneGraph() void
updateBottomLeftTextBox(String) void
initPokemonSprites() void
updateSwitchPoke(String, boolean, String, boolean) void
BotHpBar Rectangle
Switch Button
Poke0Btn Button
NAMETEXTBARWIDTH int
botCurrPokemon Pokemon
Move1 Button
playerHpBar Rectangle
Move2 Button
playerPokemonImageURL String
Move3 Button
Move4 Button
botPokemonImageURL String
root VBox
currPokemon Pokemon
Attack Button
Poke1Btn Button
Forfeit Button

The above two screenshots illustrate what two of our main views look like under the hood. Our game heavily depends on these two views being able to update via control by our GuiController class that handles the communication between the player with the GUI and our game model underneath.