Pokémon Showdown: Design Commentary (Diego Urena)

For our final project we decided to create a small-scale replica of the Pokémon battle simulator, Pokémon Showdown. We wanted users to be able to practice their battling skills against the AI to hone their knowledge of type matchups, so we decided to make a version of the battle simulation that included only attacking moves and the fully-evolved Pokémon from the first generation of Pokémon games. Although a lot of finer details were sacrificed (such as abilities, items, and status moves) since we couldn’t find data files online, we still managed to create what we believe is a faithful replica of the Pokémon battling system.

Our original plan was to make a class for Pokémon moves, Pokémon types, Pokémon themselves, a Pokémon battle class, and one for the UI. We decided to split up the development of the classes amongst the group members, with Fletcher being responsible for the UI and importing the data from the .json files available online while Ethan and I would be responsible for the other classes, as well as adding a more details to the .json files available online (such completing the movesets, adding the second types of all of the Pokémon, adding HP and speed as Pokémon stats, and adding PP for moves). I took on the coding of the Move and Pokémon classes as well as the Battle class, specifically the designing of the AI and attacking the opponent, although I also worked with Fletcher on the actual process of the user battling the AI and the design of the UI. I also completed the movesets of all of the Pokémon in the .json file, going through their data on Serebii.com to create the best possible moveset for each Pokémon.

The first decision I made was to cut the design of a Pokémon Type class. I decided that there was no need to have an entire Object for a Pokémon Type as it could be represented by a String since it has no other data other than the name and effectiveness, which varies so much depending on the type matchup that it didn’t really make sense to store that as a variable for an object. So with that decision made I got to work on the Move and Pokémon classes. I decided to work on these classes simultaneously since both classes worked off of each other; some Move methods were going to need a Pokémon object to be passed to it, and Pokémon objects would have 4 Move objects associated with them. I was initially going to make the stats for a Pokémon object an array of integers, but I decided it would be better to make them individual variables to make it more clear which stat we were accessing in things like the damage calculation for battling. I was also originally going to leave the Pokémon with their base stats and not worry about giving them a level, but I decided instead to randomize the level of each Pokémon between 79 and 89 and then do a calculation for their effective stats, which is the Pokémon’s stats after it reaches a certain level. I also needed to include a variable that indicated whether the Pokémon was healthy or not that we could use in the Battle class, so I added a boolean to represent that value. I also decided to use an array instead of an array list to represent the moveset of the Pokémon because every Pokémon has 4 moves (it does not vary, so it makes sense to have it at a set size). I also decided to make only one getType() method that returns an array list of the Pokémon’s type(s) instead of two methods for each possible type of the Pokémon because I thought it would be easier to call only one method whenever we needed the types of a Pokémon and to deal with an ArrayList that might have an empty String as the second value than calling two methods for each Pokémon only for one of them to return an empty String if the Pokémon only had one type. For the Move class, a change I decided to make from the original plan was adding the critical hit modifier and boolean value into the Move object itself instead of leaving that calculation for the damage calculation in the battling. I made this decision after discussing with Fletcher about adding a message for whenever the hit was a critical hit and realizing that if we wanted to do that, we would need to store that value in the Move object since it was not enough to have it generated in a private method since that wouldn’t be stored in a way for us to be able to print it and associate it with a move. I also decided to differentiate between base PP and current PP like how in the Pokémon class I differentiated between base HP and current HP so that that information could be displayed on the UI.

The Battle class was made with significant collaboration from everyone. I provided the code for hitting the opponent and designed the AI. Originally I was going to make the AI randomly choose a move and it would never swap out, like the AI in the actual games. But then I decided to make an “advantage system” that would allow the AI to make smarter decisions. If the AI had any advantage (a number greater than 0), it would attack, if it was in a “neutral state” (advantage of 0), there is a high change it would attack, but it might swap, and if it’s in varying disadvantage states, there are varying high chances that it will swap. To create the advantage calculation, I made 4 different subsets of the advantage calculation: a move advantage, a type advantage, an HP advantage, and a speed advantage that returned various ints depending on comparisons between the AI’s Pokémon and the user’s Pokémon to check the AI’s chances of succeeding that turn in damaging or KO’ing the user while not being KO’d itself. The AI’s turn would then be decided using this advantage as well as a random calculation. If the method for the AITurn returned true, the AI would hit the user; if not, it would swap out into the Pokémon with the best advantage against the user’s Pokémon. Originally I had this method perform the actual actions of the AI instead of returning a boolean, but after our battling code had a lot of bugs and Fletcher decided to create the Decision class to clean up the code, I made this change to correspond with his Decision class and make creating Decision objects for the AI easier. I then worked with Fletcher to make the code for handling a turn to cover all the possible cases in the battle, like a user and/or the AI swapping, the user attacking first and knocking the AI out before it can retaliate, etc. I also worked with him on some of the design decisions for the UI, such as the code for displaying whether a move was superffective or a critical hit and similar things. I also then worked on some bugs we found with Fletcher as well as on my own, such as fixing the effective stat calculation (it was only ever returning 10 or 5) and fixing the message for the AI’s move’s effectiveness when the user switched out (for example it would say Earthquake was super effective when I swapped out into a Dragonite from an Electrode even though it had no effect, so I needed to fix that). After those bug fixes, our code worked as expected and I am very proud of the final product.