DESIGN SPECIFICATION
for
PokéSnowdown

Version 1.0

Produced by Matthew Zang, Nicholas Shieh, Chirag Bharadwaj, Young Chan Kim

Contents

1 System Description

1.1 Core Vision

Our plan for A6 is to create a Pokémon Showdown spinoff using `OCaml` using key concepts presented in class (concurrency, mutability, functional programming) as well as integrating functions of `OCaml` not presented in class, including producing a suitable GUI and server to aid in development of the game. To set us apart from Pokémon Showdown, we are incorporating a single-player story mode in the form of a "Tournament mode", as well as allowing for full 1-Player, 2-Player, and No-Player functionality (by developing an intelligent bot that can fully play the game).

1.2 Key Features

We have the following incorporated in our following project.

1. A fully functional GUI that allows the players to see the current state of the game. The GUI shows the current Pokemon in battle, the current statuses of the Pokemon, any stat modifications, etc...

2. All the current Pokémon are ncluded in the game. We have implemented 721 Pokémon over 400 moves, around 50 abilities, and around 6 items.

3. Multiple game options. We have the following modes.
   - For 1-Player, we incorporated three different modes.
     a) **Random Battles**: This is the same game mode as commonly found on Pokémon Showdown where teams are randomized. However, the player will be playing against the AI.
     b) **Tournament mode**: This is the single player story mode. You battle Premade trainers with custom Pokémon sets. The story is that there is some Hunger Games activity, and Professor Oak is the captain who forces you to battle all the enemy trainers. As you beat trainers, you steal some of their Pokémon. Lastly, beat every trainer once and something special happens – Professor Oak goes into the cave and you are allowed to battle him once. Win and you unlock a legendary Pokémon. Lose and you have to battle all the other trainers once more.
     c) **Preset Battles**: The player gets to create his own Pokémon team and play against AI that also use his unlocked Pokémon.
   - For 2-Player, we hope to have a two game modes.
     a) **Random Battles**: Similar to One Player random battles, but against another player instead of an AI.
     b) **Preset Battles**: Both players get to choose 6 Pokémon from the unlocked Pokémon before battling.
   - Lastly, for No-Player, we hope to have a single game mode.
     a) **SkyNet Battles**: Player watches the AI face each other with randomized Pokémon.

4. We provided a Pokémon team editor that allows a player to choose Pokémon and movesets. This team is saved and can be loaded up for future use. These teams will be used in the game modes described above.

5. We wish to include a unique menu system and good GUI Animations.

6. We included a simple interactive story mode/dialogue that will describe the underlying storyline in our Tournament mode.

7. We have included a very nice auto-saving feature that saves the game for you.

8. We have also included key-board input. During battle, the user must press SPACE BAR to proceed to the next text option. In addition, keyboard input is used to move the character during Tournament mode.

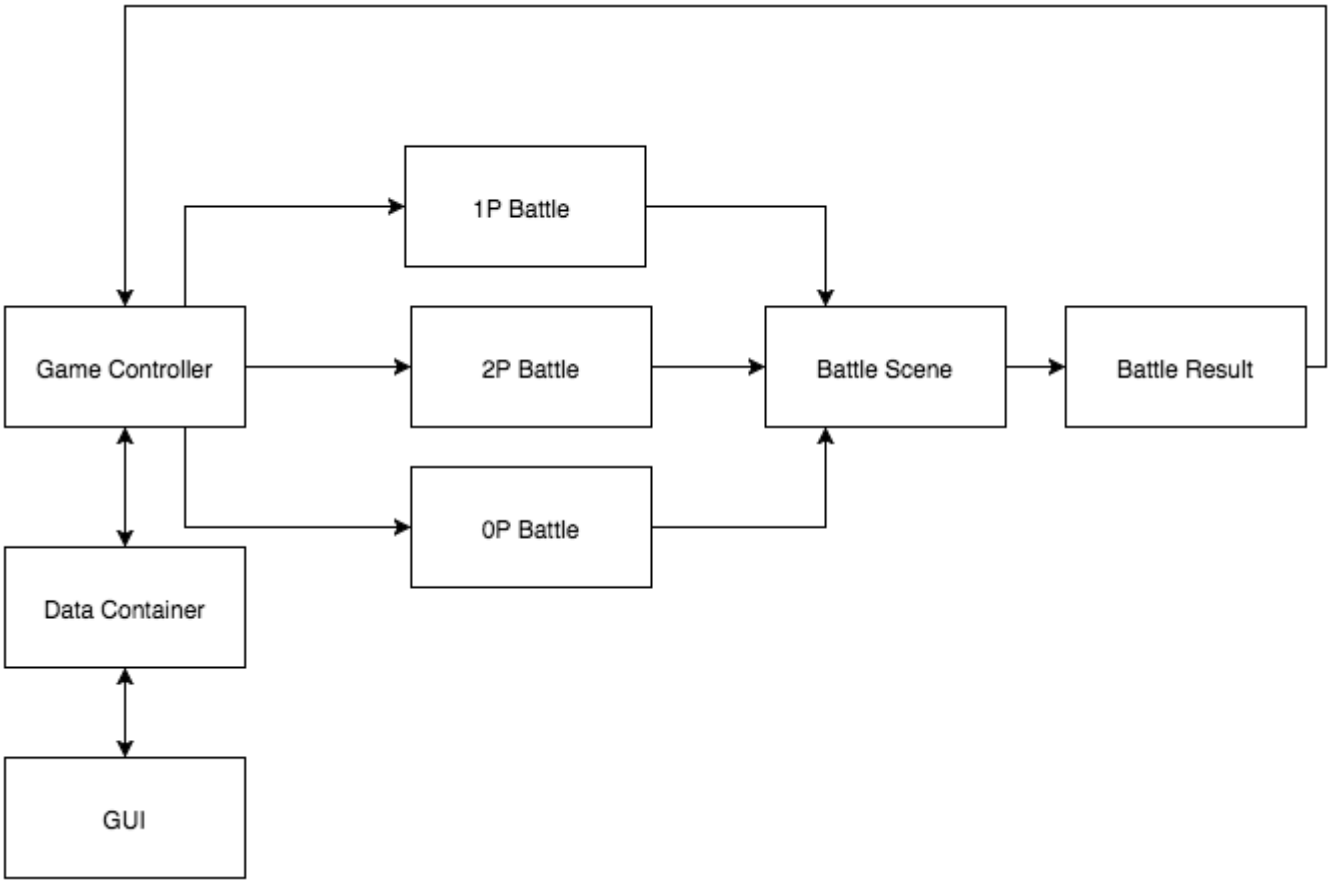
1.3 Narrative Description

This game will be an upgraded version of Pokémon Showdown, suitable and to the scale for a small console video game. It will have the main aspects of Pokémon Showdown including random battles between two players, as well as premade team battles between two players and a fully functional Pokémon team editor. However, it will also include single player capabilities, such as random battling or premade battling against an AI. The AI must be able to battle effectively and use advanced Pokémon techniques in the battle. In addition to this, there will be a Tournament mode which is essentially a story mode. The player has to play against several bots before proceeding to the final boss. This means that bots should have different levels of intelligence in order for this game mode to be challenging.

2 Architecture

We produced a pipe and filter architecture. We were initially going to produce a client-server architecture with a game server keeping track of the current game state and updating the GUI will allowing clients (players) to send messages to the server to retrieve information and request actions; however, we realized that creating a server would make it hard to implement all the other features we wanted to implemented (since none of us knew what creating a server entailed). Instead of having a client-server architecture that allowed for two players to do concurrent battling, we realized that a pipelined architecture where each player would take turns choosing a move would work just as well. The input from the user will be passed along a pipeline and transforms the data into output. This is sufficient enough for the turn based Pokémon game.

However, in addition to the pipe and filter architecture (in relation to the game flow), we have some aspects of Shared Data Architecture. The Game and the GUI use the shared data architecture to communicate by reading and writing to the shared data. The C&C diagram is provided below. The pipeline flow is shown from left to right, while the shared data architecture is shown extending downwards from Game Controller.



3 System Design

3.1 Important Modules

1. **Game**: This is the game controller. It is mainly involved in handling the GUI and passing control to the battle controller when a battle occurs/is set up.

2. **Gui**: The Gui is used to diplay the current state of the game. There are two main phases in the Gui, the menu screens and the battle scenes. The Gui will either communicate with the game controller when the menu screens are being displayed or communicate with the battle controller when a battle is occuring.

3. **Info**: The Info module contains all the data types that are needed/shared between the module. It essentially contains everything the other modules need to use to communicate with one another and helps in avoiding circular dependencies. All data types needed for the game will be declared here.

4. **BattleController**: This battle controller module is called by the game controller when a battle is taking place. During this time, the battle controller handles all the game mechanics and updates the Gui. The game controller waits for the battle to end before resuming control.

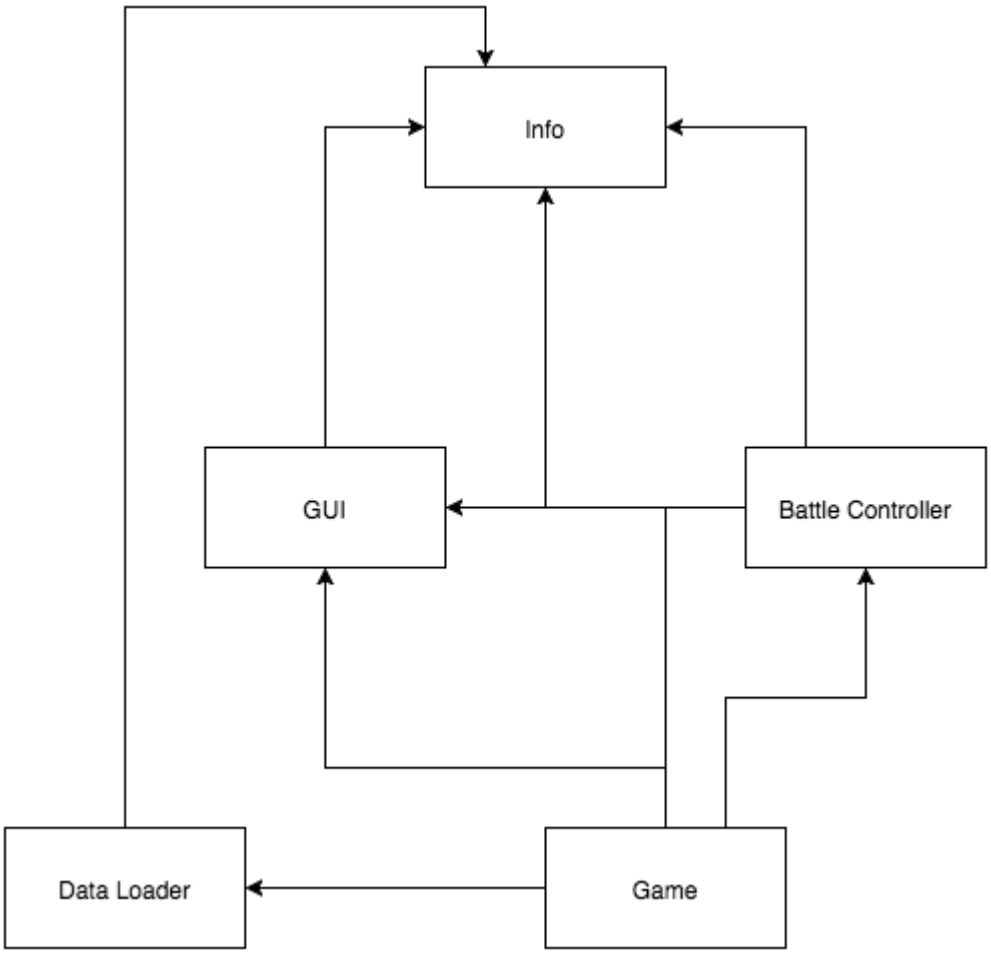
5. **Pokémon**: This module is involved in loading the data from the json files and getting Pokémon data. It is responsible for turning the Json data into a Pokémon. For example, functions in this method will be called to convert a string containing the name of the Pokémon to some value of type Pokémon (defined in the Info module).

6. **AI**: This module will be involved in the artificial intelligence aspect of our game, where we will handle the bots playing the game automatically for the SkyNet Battles. For instance, this module should contain functions that will allow a bot to intelligently select moves to maximize damage to the opposing bot while minimizing its own damage and moves used.

7. **Tournament**: This contains all the tournament data. It has the opponent's in tournament mode, the map of the tournament mode, the different trainer classes, and it makes calls to Save after a battle has been won.

8. **Save**: This module makes all the save calls. While Pokémon turns json to Pokémon, Save turns json to a Pokémon.

3.2 Module Dependency Diagram



4 Data

4.1 Internal Data

We have all the json files to represent all the Pokémon, moves, abilities, etc... Since json files aren't readily available online with all the Pokémon data, we wrote `Python` scripts to fetch the data from databases and convert them to a json format. To load the data, we used the `Yojson` module, similar to how we used it in A2. In addition, we have many jpgs and gifs that represent the Pokémon. We used many of the resources on Smogon and Bulbapedia to obtain information.

4.2 Communication

In order to let the GUI communicate with the Game without the use of a server, we came up with a method of communication through Ivar references. When Game initializes the GUI, we will make Game pass in as an argument an Ivar reference. Game and GUI will communicate by emptying and filling the Ivar. When Game is ready to handle another input, Game will reset the Ivar by creating a new one. GUI will then fill in the Ivar upon receiving some input. Thus, Game and GUI will be able to communicate without creating any circular dependency among modules. The Ivar will be filled with some value that is of type `game_state` defined in the Info module.

4.3 Data Structures

We held all the Pokémon data in Json files. We did have many Variants to hold the secondary effects of the game. While Battle Controller did the damage calculation and secondary effect dependencies, it used variants to hold all the different secondary effects that happened during the move processing. It then sends this variant to the GUI which unpacks it nicely as a message in the battle to the user. In addition, these variants are also decomposed to get the right attack animation. The Pokémon themselves are stored as mutable records. The trainers are stored as records with a field for the current Pokémon in battle, a field for a list of alive Pokémon and a field for a list of dead Pokémon.

5 External Dependencies

We used `Lablgtk2` to create the GUI and `Yojson` to parse json files. The json data will primarily be collected from https://github.com/veekun/pokedex/tree/master/pokedex/data/csv using `Python`.

6 Testing Plan

Previous Testing Plan:

Our testing plan will mainly consist of debugging and verifying individual functions through a combination of black-box and white-box testing for each module, as well as regression tests to ensure any changes to some aspect of our program does not break an existing functionality that was previously verified. These test cases will be mostly randomized and automated, and testing will take place during and after the implementation process. Of course, once we are finished with the implementation and modular testing, we will also test the overall gameplay by playing the game many times and letting some of our friends outside of the course to play the game to assure that the game can be played without issues and also receive useful feedback. If we have time, we may even formally verify the correctness of some of our functions.

Our Testing Plan:

Our testing plan was primarily done during the hard coding phase. We divided our project into three main phases – data preparation, gui creation, and hard coding. Data Preparation was necessary to get all the data of the Pokémon. This was obtained by downloading csv files and parsing them into JSON format using Python. No testing had to be done in this phase.

The GUI Creation phase was done next. It was simply the creation of the GUI and creating all the menus/simple game animations. This was done with peer review testing. We made sure all code was functional before it was pushed. There are no big bugs in the GUI. However, there are times when the Game is stuck in the Loading Screen. This is easily fixed by closing and rerunning the game.

After GUI Creation, we had to do the hard coding. We hard coded all the different moves/abilities. There are no known bugs in the moves/abilities. This was because we thoroughly tested every move/secondary effect before allowing it to be in the game.

7 Division of Labor

Matthew Zang:

He primarily did the GUI, managing the team, getting the json data, and setting up the project. He also assigned tasks to everyone. He made the menu for the GUI, the loading screens, battle animations, and designed the overall Game Logic. He also wrote the Poke Editor and the main menu screens.

Nick Shieh:

He was primarily in charge of hard coding the moves. He helped code the majority of the moves and making sure the interactions were clear. Nick was also in charge of coding the AI for the game. He and Young made two different difficulty levels for the AI, a simple one and a more complex one. However, only the complex one was included as an opponent in the game, as the simpler one served to be way too naive to even people that have never played Pokémon before.

Chirag Bhardawaj:

He was in charge of testing, code verification, and getting the game animations and images. He also made the Tournament mode of our game, the tile-maps, and retrieved the sprites for the opponents. He also coded the Save File handler, which was vital for the saved state of our game. Additionally, he was in charge of the graphics design of our game.

Young Chan Kim:

He was also in charge of hard coding the moves. He helped code the majority of the moves and handling the GUI interactions for the Pokemon moves. He also made a major contribution to the AI of the game. In addition, he also was primarily in charge of coding the abilties and items that were used throughout the game. He also designed the formats for the Data structures, such as the records for the Pokémon, records for the moves, variants for elements and secondary effects, etc.