DESIGN SPECIFICATION
for
PokéSnowdown

Version 1.0

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

Contents

1 Update Log

This log will hold all the updates/git pushes to our game, showing our current progress/what needs to be completed, etc...

2 System Description

2.1 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).

2.2 Key Features

We hope to have the following implemented in our completed project:

1. A fully functional GUI that allows the players to see the current state of the game. The GUI will most likely show the Pokémon but the attack animations might be too complex to complete.
2. All the current Pokémon will be included in the game, as well as items that are commonly used in competitive battling will also be included.
3. Multiple game options. We are hoping to have the following modes.
    - For 1-Player, we hope to incorporate 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 that we plan on incorporating.
        c) **Preset Battles**: The player gets to create his own Pokémon team and play against AI that use pre-made Smogon team compositions.
    - For 2-Player, we hope to have a single game mode.
        a) **Friend Battles**: Players have the options of choosing a Random team or a Preset team to head into battle with their friend.
    - Lastly, for No-Player, we hope to have a single game mode.
        a) **SkyNet Battles**: Player has option of choosing the Pokémon team for two bots, and watch them as they face off.
4. We wish to provide 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 integrate music into our game based upon the game mode as well as the current battle status.
6. We want to include an interactive story mode/dialogue that will describe the underlying storyline in our Tournament mode.
7. We will have to include concurrency for two player battling, meaning that there will have to be a server to respond to player requests as well as the GUI.

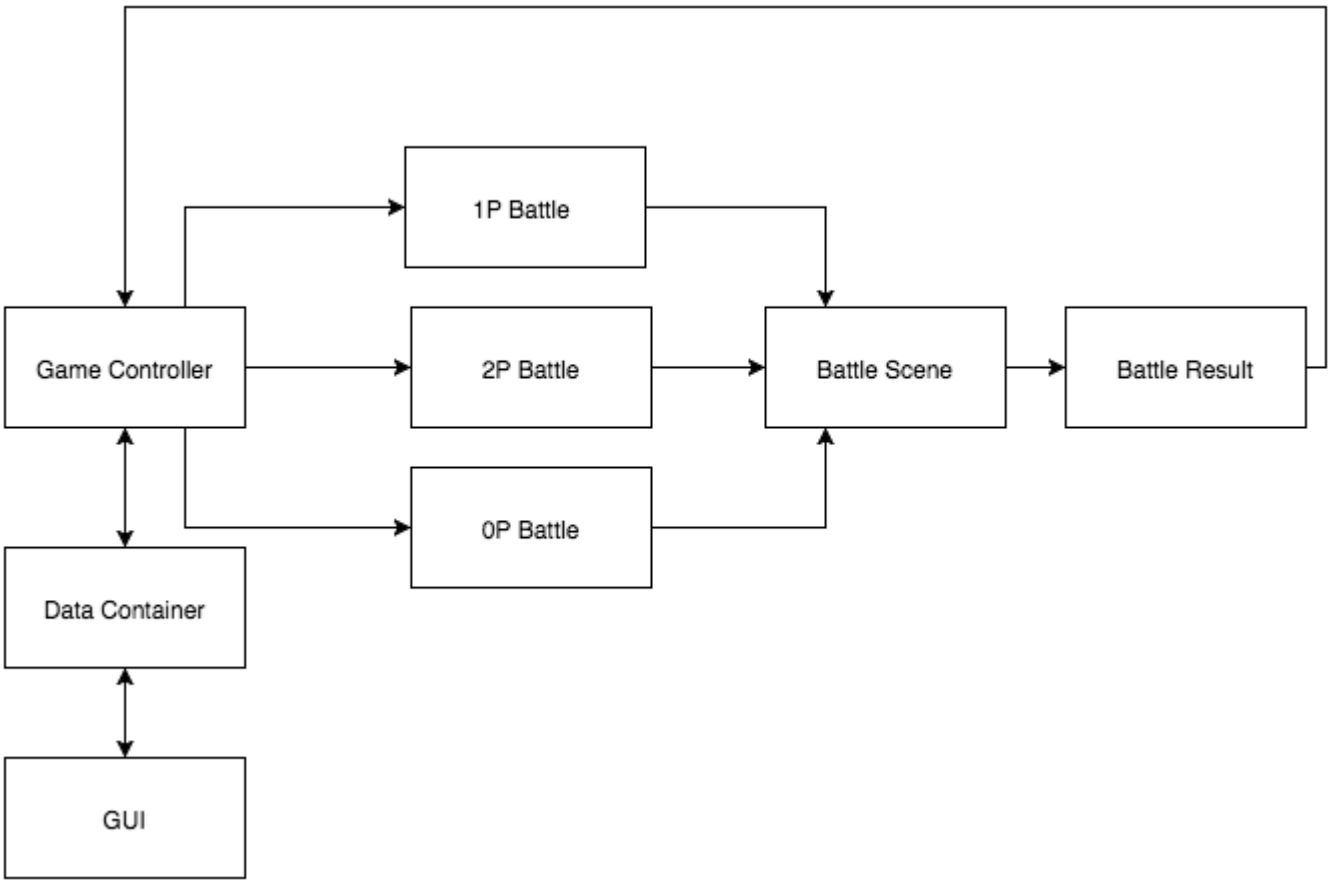
2.3 Narrative Description

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.

3 Architecture

We are aiming to 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.



4 System Design

4.1 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.

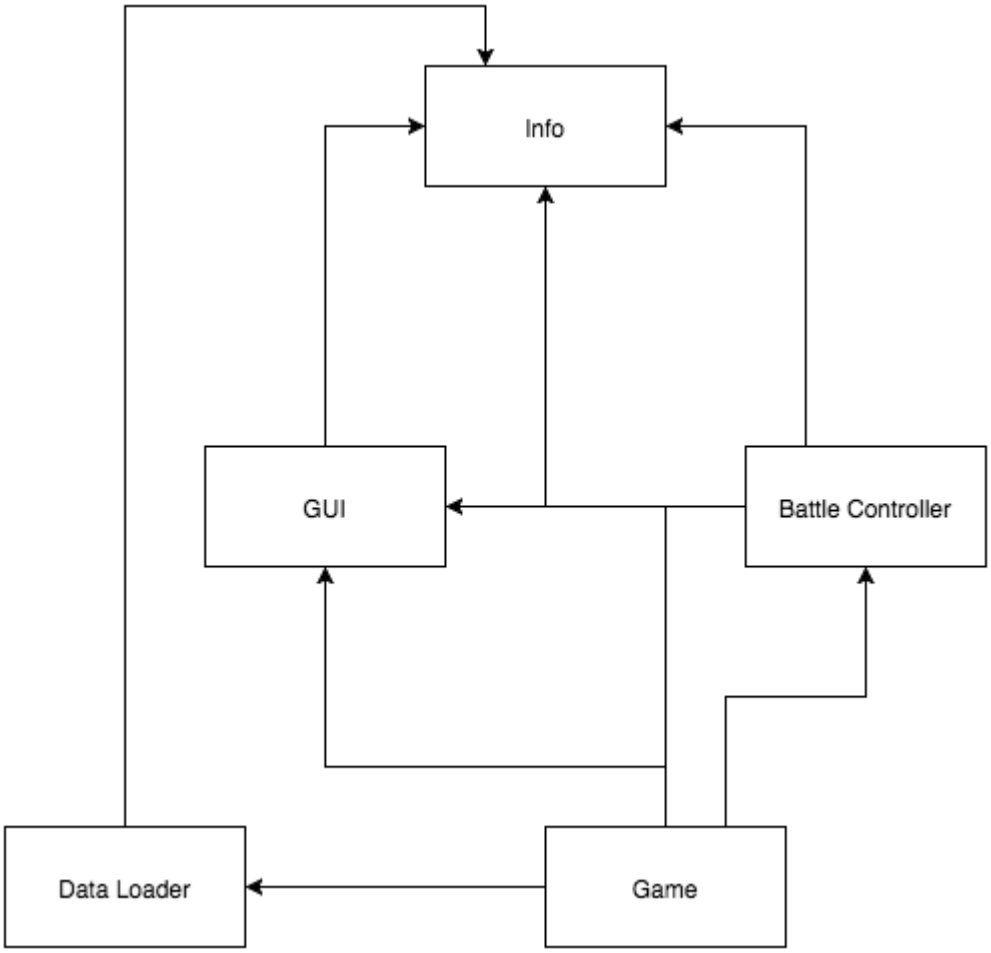
2. **Gui**: The Gui is used to diplay the current state of the game. There are two main phases in the Gui, the menu screens and the battle scenes. The Gui will either communicate with the game controller when the menu screens are being displayed or communicate with the battle controller when a battle 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. **DataLoader**: This module is involved in loading the data from the json files. For example, functions in this method will be called to convert a string containing the name of the Pokémon to something of type Pokémon (defined in the Info module).

4.2 Module Dependency Diagram



5 Data

5.1 Internal Data

We plan on having json files to represent all the Pokémon, moves, abilities, etc... Since json files aren't readily available online with all the Pokémon data, we plan to write Python scripts to fetch the data from databases and convert them to a json format. To load the data, we are most likely going to use the Yojson module, similar to how we used it in A2.

5.2 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 communicate by emptying and filling the Ivar. When Game is ready to handle another input, Game resets the Ivar by creating a new one. GUI then fills in the Ivar when it gets some input. Thus, Game and GUI can communicate without creating any circular dependency within modules. The Ivar is filled with some variable that is of type `game_state` defined in the Info module.

5.3 Data Structures

To hold all the Pokémon data we will most likely be using some implementation of a dictionary. We are thinking of using a HashMap or the infamous Two-Three trees.

6 External Dependencies

We will be using `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`.

7 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.