Matthew Oey

Mike Wang

CS 3110 Problem Set 6: POKEJOUKI

Design Outline

Summary

We found implementing the bot to be one of the hardest parts of the assignment. We treated the game as a function that required optimization and the bot as an information search. This required a lot of linear algebra, probability theory, and exhaustive bookkeeping as we tried to formulate an algorithm that found the optimal attack for every possible state.

One of the major design decisions we decided to do was to implement the game rules in the state.ml file rather than in the game.ml file. We did this because many of the game rules involved handling commands and updating the state of the game. It also made dividing the assignment easier, especially given the long break in the middle which made it difficult to meet up, because the game.ml file handled the game implementation while state handled the game rules and they did not require that much code of each other's code.

Specification

In our module state.ml, we had to implement many of the game rules. The most important functions in state.ml were add\_steammon, select\_inventory, switch\_steammon, attack, and use\_item. These five methods represented the core of the game.

Design and Implementation

Our modules are separated into three folders: game, shared, and team. The modules in the shared folder handles the shared files including utility functions, definitions, and constants. These were all left unchanged throughout the assignment. The modules in the game folder handle the implementation of the game. It contains our game, state, netgraphics, and server modules. The module netgraphics.ml handles the GUI and server.ml handles the concurrency between the game, the bots, and the GUI. The module state.ml handles many of the game rules and keeps track of the current game state. We talked about if we should create mutable variables however we decided that we were better off creating a new game state every turn. The module game.ml handles implementation of the game as well as its various requests and commands. In the teams folder we have the bots which handle all the AI operations.

The architecture of our program is fairly simple. game.ml is dependent on state.ml and all three main classes we had to implement are all dependent on constants.ml, definitions.ml, and util.ml.

The main data structure we used was game\_state. Game state contained all the information about every steammon on each player's teams and kept track of both player's inventories. To organize this amount of data it used tuples to keep track of the various aspects and records to keep track of things that required a lot of data such as steammon and attack.

Testing

Known Problems

Comments

I thought the problem set was interesting although a slightly exhaustive. It didn't require the implementation of any challenging algorithms like we saw in earlier problem sets but it did provide good practice in software engineering. For future implementations of the pokemon game, I would suggest more difficult problems but less volume of coding. For example, maybe give some of the game rules such as calculating damage but require the student to code the server.