CS 3110 Problem Set 6: *PokeJouki*

Assigned: November 8, 2012 — Final submission due: November 30, 2012 **(no extensions)**
Design meetings: TBA

1 Introduction

In this assignment, you will develop a game called *PokeJouki*.

There are few constraints on how you implement this project. This does not mean you can abandon what you have learned about abstraction, style and modularity; rather, this is an opportunity to demonstrate all three in the creation of elegant code.

You should start by carefully designing your system, and presenting this design at a *design meeting* where you will meet with a course staff member to discuss your design. Part of your score will be based on the design you present at this meeting.

On TBA, after the problem set is due, there will be a *PokeJouki* tournament which you are encouraged to submit your bot programs to. There will be lots of free food, and the chance to watch your bot perform live. The winner gets bragging rights and will have their name posted on the 312/3110 Tournament hall of fame.

1.1 Reading this document

The types referenced in this document that are not default in OCaml are defined in `definitions.ml`. Constants follow the following naming convention: they begin with a lowercase `c`, and the rest is a descriptive name of the constant in all caps. For example, `cNUM_ETHER` specifies the initial number of ether in the inventory. The reason for this mysterious `c` in front of all the constants is that OCaml actually doesn't allow value names to begin an uppercase letter - only type constructors can do that. For all chance based constants, the value given is the percent chance that the relevant event occurs. For instance, if `cWAKE_UP_CHANCE` is 10, then there is a 10% chance to wake up. It will help to have some of the documents open in order to reference them as you read along.

1.2 Updates to Problem Set

Updates to the problem set will be marked in red.

1.3 Point Breakdown

- Design meeting – 5 pts
- Game – 40 pts
- Bot – 35 pts
- Documentation and design – 5 pts
- Written problem – 15 pts

2 Game Rules

2.1 General Rules

The game begins in the draft phase. The player chosen to have the first pick will select a steammon. Then, the next player gets 2 picks. This process continues until each player has chosen `cNUM_PICKS` steammon. Each player then selects a starter steammon simultaneously, and then the battle phase will commence. During each turn of the battle phase, each player will take an action, with the ultimate goal of making the other players team faint. The challenge is figuring out the best strategy to do that!

2.2 The Steammon

The `steammon` type specifies all of the attributes associated with a Steammon. The Steammon has up to 2 steamtypes, four attacks, stats, status changes, and modifiers.

2.2.1 Steamtypes

Steamtypes specify the specialty of a Steammon. Certain types are strong against others, taking reduced, or even zero damage from certain attacks. Steammon that use an attack that has the same steamtype as one of its own steamypes will do `cSTAB_BONUS` (SameTypeAttackBonus) times more damage.

2.2.2 Attacks

The primary instrument of battle will be attacks. Each attack has several characteristics, which change their effectiveness in battle.

- Steamtype –Each attack has an element. The element determines whether an attack will do increased damage to the opponent, or end up not affecting the opponent at all.
- PP - Each attack has a certain amount of Power Points available to it. This number indicates the number of times that a steammon can use an attack, and is usually an indicator of how powerful the attack is. More powerful attacks will have less PP so you need to watch carefully how often you use each attack.
- Power – This number is used to determine how much damage the attack will do.
- Accuracy – This number is used to determine how often an attack will hit. The accuracy represents the probability of hitting the opponent.
- Critical Hits – Each attack has a certain chance to critical hit. A critical hit will do `cCRIT_MULTIPLIER` times more damage than a regular attack and be a lucky way to change the tide of battle.
- Effects – Some attacks will have an effect associated with them. These include the ability to inflict status effects on enemies, or boost your own steammons abilities. An effect can only occur if the attack hits the opponent.

2.3 Stats

Each steammon has an attack, special attack, defense, special defense, and speed stat. The attack stat determines damage dealt, defense determines damage taken, and speed determines which steammon gets to attack first. When a pokemon uses an electric, fire, water, psychic, or ghost move, then use the special attack stat instead of attack. When a pokemon is hit by an electric, fire, water, psychic, or ghost move, then use the special defense stat instead of defense.

2.4 Status

- Paralyzed – When a steammon is paralyzed, there is a chance that the steammon will not attack that turn, `cPARALYSIS_CHANCE`. The speed of a paralyzed steammon is lowered by `cPARALYSIS_SLOW`. A steammons new speed is determined by the formula: $\frac{steammon.speed}{cPARALYSIS\_SLOW}$
- Poisoned – When a steammon is poisoned, it will take extra damage at the end of each turn. It takes `cPOISON_DAMAGE`*(steammon.max_hp)
- Asleep – A sleeping steammon is unable to attack. However, prior to each attack, the steammon has `cWAKE_UP_CHANCE` to wake up and execute its attack.
- Confused – When a steammon is confused there is `cSELF_ATTACK_CHANCE` that the steammon will attack itself instead of the opponent. If a steammon attacks itself, it will attack itself with an attack that has power `cSELF_ATTACK_POWER`. There is `cSNAP_OUT_OF_CONFUSION` chance that the steammon will no longer be confused prior to the attack. Confusion is removed when the steammon is withdrawn.
- Frozen – When a steammon is frozen, it is unable to attack. There is `cDEFROST_CHANCE` that the steammon will defrost at the end of its turn.

Steammon can have up to two statuses. However, if the steammon has two statuses, then one of them must be Confused. In other words, a steammon can be confused and have one more status.

Note: Fainted steammon lose all status effects when they are revived.

2.5 Modifiers

Each steammon has the ability to have its stats raised from items and attack effects. These will allow your steammon to become more effective in battle. However, be careful, if a steammon is switched out, it will lose all of the modifiers, and return back to its base stats. Each modifier can be raised or lowered 3 levels. When a stat has a modifier on it, the steammons base stat is treated as c⟨attribute⟩UP⟨level⟩ or c⟨attribute⟩DOWN⟨level⟩ depending on which stat is modified and what level that modifier currently is.

2.6 Items

In addition to a set of steammon, each team is given an inventory. Each player can choose to use an item instead of attack for a turn.

- `Ether` – Restores 5 PP to each attack of the target Steammon
- `MaxPotion` – Restores a Steammon back to full HP. This item cannot revive a fainted steammon.
- `Revive` – Restores a fainted Steammon back to half of its maximum HP. Revive can **only** be used on fainted steammon.
- `FullHeal` – Removes all status effects from the target Steammon.
- `XAttack` – Increases a Steammons Attack modifier by 1. This does not apply to Special Attack.
- `XDefense` – Increases a Steammons Defense modifier by 1. This does not apply to Special Defense.

- `XSpeed` – Increases a Steammons Speed modifier by 1.
- `XAccuracy` – Increases a Steammons Accuracy modifier by 1.

A note on targets: The `X` items (`XAttack`, `XDefense`, `XSpeed`, and `XAccuracy`) can only be used on the current actively picked steammon. If the target is a steammon that isn't the current actively picked steammon, it should still be considered a valid command and the game server should still apply the item on the current actively picked steammon, despite the target being wrong.

2.7 Game Flow: Initialization Step

Both players connect to the server, at which point the server calls `init_game`, initializing the game. This includes: reading the files `attack.txt` and `steammon.txt` to obtain the information about the steammon that will be in play for this game, and selecting the player that will get first pick.

2.8 Game Flow: The Draft Step

The player who is given first pick selects a steammon out of the pool of steammon. Then, the second player picks two steammon. This process proceeds in rounds with the first pick of the round alternating until `cNUM_PICKS` steammon have been chosen. This creates a picking sequence of 1 for the first team, 2 for the second team, 2 for the first team, and continuing on until `cNUM_PICKS` is picked for the first team, and then 1 for the second team. The server will send out PickRequest to the team that is going to pick, and `DoNothing` to the team that is to wait. However, the AI will never know that it was sent a `DoNothing`, and will only recieve Request commands from the server.

Each steammon can be on at most one team. The attacks for each Steammon are the same. For instance, if Squirtle and Horsea both know Water Gun, both of their Water Guns will have the same characteristics, though they will not share PP, or any other characteristics that would be specific to a particular steammon. The pool will be balanced so that the player with first pick does not get an unfair advantage. When the Draft Step is over, the server sends PickInventoryRequests to both teams, and then the Inventory Step begins.

2.9 Game Flow: The Inventory Step

Players can choose their own inventory. The server sends out PickInventoryRequests to each player. Each player begins with a given amount of money specified by `cINITIAL_CASH` and the AI must send back how many of each item they want in their inventory. If the total cost of the inventory chosen by the player exceeds `cINITIAL_CASH`, then the game assigns a default inventory using the counts specified in `constants.ml`. When the Inventory Step is over, the server sends StarterRequests to both teams, and then the Battle Phase begins!

2.10 Game Flow: The Battle Phase

During the Battle Phase, the server will only send out StarterRequests and ActionRequests. StarterRequests mean that the current active steammon has fainted, and the team must select a new battling Steammon. Otherwise, only ActionRequests are used until the game is over.

The teams are allowed to use items, switch steammon, and use attacks during this phase. The priority of these actions is the order in which they will be discussed.

2.10.1 Switch Steammon

The player specifies one of his non-fainted steammon to switch with his current active steammon. The opposite player will attack the new active steammon after the switch, protecting the old active steammon. A steammon that is switched out loses all of its previous modifiers, and only loses confusion.

2.10.2 UseItem

If a player wants to use an item, they must have at least one of that item in their inventory, and after use, they lose one copy of the item. Item descriptions are specified above.

2.10.3 UseAttack

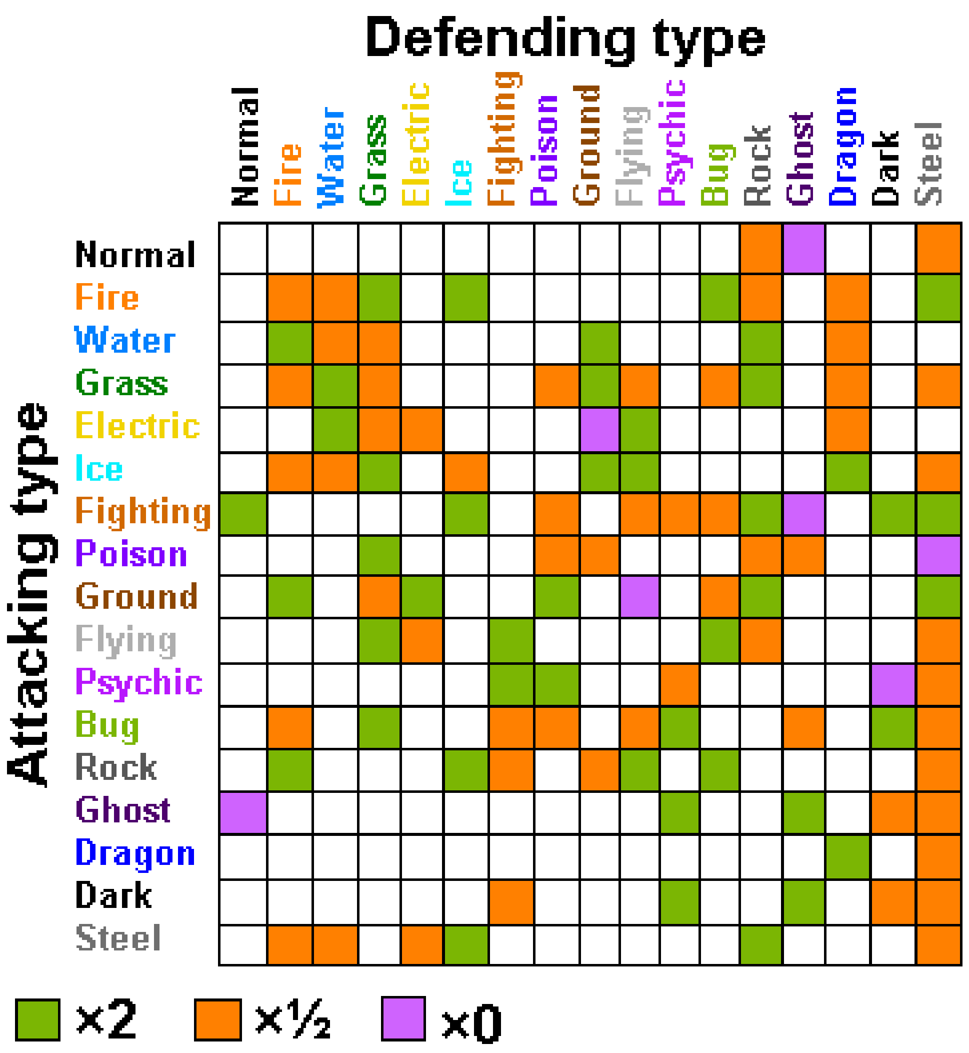
The player specifies the attack that it wants to use. If both players choose to attack, the steammon with the higher speed will attack first. If both steammon have the same speed, the game will pick an arbitrary team to attack first. The attack will hit the other steammon with probability that is represented by its accuracy. An attack may still not go through for reasons other than a miss. If a steammon is Frozen or Asleep, it is unable to attack. If a steammon is confused or Paralyzed, there is a chance that the attack may fail. If the attack hits, the attacks effect may also trigger, with the chance associate with the effect.

The Formula for attack damage is:

$$\frac{AttackPower * AttackersAttack * CritEffect * STABBonus * SteamtypeMultiplier}{OpponentsDef}$$

- AttackPower – the power rating for the attack
- AttackersAttack – the attack rating for the steammon
- CritEffect – If a critical hit occurs, this is `cCRIT_MULTIPLIER`, otherwise, this is 1
- If a steammon uses an attack that has the same element as one of its steamtypes, STABBonus is `cSTAB_BONUS` otherwise, it is 1.
- OpponentsDef – is the Defense stat of the defending steammon
- SteamtypeMultiplier – This is determined by the attacking steammon's attack type and the defending steammons types. For steammon with 2 types, use the product of the results on each type. Use this chart to determine it:

Note: If the attack is an electric, fire, water, psychich, or ghost attack, then replace AttackersAttack by the attackers Special Attack stat, and replace OpponentsDef by the opponent's Special Defense stat.



Defending type

Attacking type

	Normal	Fire	Water	Grass	Electric	Ice	Fighting	Poison	Ground	Flying	Psychic	Bug	Rock	Ghost	Dragon	Dark	Steel
Normal													🟧	🟪			🟧
Fire		🟧	🟧	🟩		🟩						🟩	🟧		🟧		🟩
Water		🟩	🟧	🟧					🟩				🟩		🟧		
Grass		🟧	🟩	🟧				🟧	🟩	🟧		🟧	🟩		🟧		🟧
Electric			🟩	🟧	🟧				🟪	🟩					🟧		
Ice		🟧	🟧	🟩		🟧			🟩	🟩					🟩		🟧
Fighting	🟩					🟩		🟧		🟧	🟧	🟧	🟩	🟪		🟩	🟩
Poison				🟩				🟧	🟧				🟧	🟧			🟪
Ground		🟩		🟧	🟩			🟩		🟪		🟧	🟩				🟩
Flying				🟩	🟧		🟩					🟩	🟧				🟧
Psychic							🟩	🟩			🟧					🟪	🟧
Bug		🟧		🟩			🟧	🟧		🟧	🟩			🟧		🟩	🟧
Rock		🟩				🟩	🟧		🟧	🟩		🟩					🟧
Ghost	🟪										🟩			🟩		🟧	🟧
Dragon															🟩		🟧
Dark							🟧				🟩			🟩		🟧	🟧
Steel		🟧	🟧		🟧	🟩							🟩				🟧

🟩 ×2 🟧 ×½ 🟪 ×0

Remember, a Steammon may have up to two types, so both types need to be considered for this calculation. Attacks may also have effects. These effects are determined by their name. Nada is an attack with no effect.

2.11 Bad Teams

The game server must also handle bots that do not send valid responses or any responses at all. This section describes the rules the game server should use when dealing with bad teams.

2.11.1 Invalid Commands

If the game server receives an invalid command, the game should treat it as if the bot had sent no command for that step. See the next section for information on how to handle a nonresponse.

An invalid command includes, but is not limited to:

- Commands that are not of the form `Action of action`.
- Attempting to use an item that is not in the team's inventory.
- Sending anything other than `PickSteammon` in response to a `PickRequest`.
- Sending anything other than `SelectStarter` in response to a `StarterRequest`.

- Using a `UseItem` action that has a target that is not a Steammon in the team.
- Using a `UseAttack` action that is not a valid attack.

2.11.2 No Response

If a bot sends no valid action within `cUPDATE_TIME`, the action for the team that is passed into `Game.handle_step` will be `DoNothing`. The game logic should continue running the game on behalf of the team. Here is the logic the game should use for each game phase:

- For a `PickRequest`, the game should pick an arbitrary available steammon.
- For a `PickInventoryRequest`, the game should pick a default inventory based on the default item counts in `constants.ml`
- For a `StarterRequest`, the game should pick an arbitrary non-fainted steammon for the team.
- for an `ActionRequest`, the game should not perform any action for the team, but the team can still be affected by the opposing team's actions.

3 Communication

3.1 Client-Server Framework

PokeJouki makes use of a client-server framework. Under this framework, the game server is responsible for keeping track of the game state, applying the game rules, and so on. Clients (i.e., players) are run as an entirely separate process and can keep track of whatever information they want, but need to send messages to the server to perform game actions or receive information.

Players communicate with the game server by sending information over channels. The protocol for messages is defined by the type `Action` in `definitions.ml`.

There are five types of messages defined in the protocol:

- Control messages, which deal with starting and ending the game
- the `DoNothing` message, which is used by the server to correct mistakes that occur in the game
- Action messages, which the team uses to specify what it wants to do
- Request messages, which allow the server to tell the players what its next action choice is.

See the sections below for details on the three types of messages.

3.2 Communication as a client

3.3 Control Messages

3.3.1 Control Quick Reference

We have primarily taken out the need for the player to connect to the client. The player just needs to be able to respond when given a Request type.

3.3.2 Action Quick Reference

- SelectStarter- This is used in response to a StarterRequest. A player chooses which steammon they would like to send into battle. This occurs at the beginning of the battle phase or when the previous steammon fainted.
- PickSteammon- This is used in response to a PickRequest. The team specifies which steammon from the pool it wants to select for its team.
- PickInventory- This is used in response to a PickInventoryRequest. A player specifies an inventory, as in how many of each item the player wishes to buy with the money specified by `c_INITIAL_CASH`.
- SwitchSteammon- This is used in response to an ActionRequest. The player recalls their current battle steammon and selects another one to send out.
- UseItem - This is used in response to an ActionRequest. The player uses an item, on the target steammon.
- UseAttack - This is used in response to an ActionRequest. The player commands his battle steammon to use the specified attack.

4 GUI

4.1 GUI Client

In order to view the game, you will have to set up a GUI client program. The client has been coded for you, and is located in the `gui` directory. The client renders the game using Java and Swing. This module will be sufficient for simple rendering of the game, though you are welcome enhance it for karma if you wish. To use the GUI, you must have Java.

The game server is responsible for sending graphical update messages to the GUI, as described below. The game server will listen for clients to send the updates to throughout the game. However, if a GUI connects halfway through the game, it will have missed the message that initialize the board. So be sure the GUI connects before the game starts if you want to be able to see everything. In other words, start the process for the GUI before you start the processes for the teams.

4.2 Sending messages to the GUI

We have provided a simple module called `Netgraphics` with functions to send graphical updates. The functions are specified in `netgraphics.ml`. Note that the `init` function is called by the Server module, and `sendUpdates` is called reguarly by the `Server` module. So in order to send a graphics update to the clients, the game module calls `Netgraphics.addUpdate` with the appropriate `update` type. The one exception to this is that the `InitGraphics` update must be sent by itself, so the game module should send this update directly by calling `Netgraphics.sendUpdate`.

4.3 Graphics Commands

	Arguments	Meaning
`InitGraphics`		Tell the GUI client to initialize the graphics in preparation for a new game starting.
`UpdateSteammon`	`species, hp, maxhp, team`	Update (or add) steammon by species to given team
`SetChosenSteammon`	`species`	Set the currently active Steammon (team looked up by species name)
`NegativeEffect`	`msg, team, hp`	Show a negative effect for `team` with message `msg` and the amount of damage `hp` (0 hp indicates no damage)
`PositiveEffect`	`msg, team, hp`	Show a positive effect for `team` with message `msg` and the amount of bonus `hp` (0 hp indicates no bonus)
`Message`	`msg`	Display a status message `msg` to the GUI console.
`SetFirstAttacker`	`team`	Sets which team attacks first in the round. Must be called before the `NegativeEffect` and `PositiveEffect` messages are queued for the round.
`SetStatusEffects`	`species, statuses`	Sets the currently active statuses for the steammon

A note on negative and positive effects: Effects should be shown for: hits, misses, heals, and special attacks (e.g. freezes). Misses can be considered a negative effect on the targeted team, with 0 hp damage. So if Red team attacked Blue team and missed, the update would be `NegativeEffect(‘‘Miss’’, Blue, 0)`.

5 Provided source code

Many files are provided for this assignment. Most of them you will not need to edit at all. In fact, you should only edit and/or create new files in the `game` and `bot` directories (plus any edits you need to make to the compilation scripts). Here is a list of all the files included in the release and their contents.

`build*.bat`	Build scripts for the game server, teams, and GUI client (see Section 6)
`game/game.mli`	Signature file for handling actions, time, rules, and the game state
`game/game.ml`	Stub file for actions, rules, and time
`game/netgraphics.mli`	Signature file for sending updates to the GUI client
`game/netgraphics.ml`	Implementation of sending updates to the GUI
`game/server.ml`	Starts the game server and deals with communication
`shared/connection.mli`	Signature file for connection helper module
`shared/connection.ml`	Implementation of connection helper module
`shared/constants.ml`	Definitions of game constants
`shared/definitions.ml`	Definitions of game datatypes
`shared/util.ml`	General use helper functions
`shared/thread_pool.mli`	Signature file for thread pool helper module
`shared/thread_pool.ml`	Implementation of thread pool helper module
`team/team.ml`	Basic framework for a client to interact with a game server
`team/babybot.ml`	Stub for a team AI
`game/attack.txt`	Attacks for the steammon
`game/steamon.txt`	Steammon for the game

5.1 Code Structure

To understand how to implement the game, you must first understand how the code we have provided you operates. The crucial aspect to understand is the relation between the Server module and the Game module. The server module deals primarily with receiving connections from the teams, and calls the Game module for all issues related to the game rules. *You do not need to modify the Server module, graphics commands, or GUI client.* Your modifications and additions will take place in the Game module, State module, and any other modules you choose to add.

5.2 Server

At a high level, `server.ml` does the following things:

1. Calls `Netgraphics.init`, which accepts connects from GUI clients
2. Waits until for enough teams to join the game (see Section 6.1 for more details)
3. Repeatedly calls `Game.handleStep` with the responses of the clients, outputting the state and request to the client.
4. Uses the Actions of team to call `Game.handle_step`.
5. Repeatedly sends the graphics updates to the GUI.

You will need to think carefully about how you design and implement the game to meet the Server module's expectations.

5.3 Game

All the functions in the `Game` module referred to above are specified in `game.mli`.

Your design may require you to add or modify the type declarations or functions we have provided.

5.4 State

We strongly suggest that you have a `State` module in your final design, as the distinction between game rules and game state is significant. In other words, you should NOT put all of your code in the `Game` module.

6 Running the game

You will need four command prompts to run the game.

6.1 Game Server

From `ps6/game`, run `build_game.bat` to build the game server. In this command prompt, run `game.exe`.

6.2 GUI

The GUI is already written for you in Java. It is distributed in a jar file called `gui_client.jar`. The GUI needs the `data` folder for game assets. To run the GUI, just run the jar file (from the command line, `java -jar gui_client.jar`). Enter your Game Server connection information (by default it is localhost at port 10501), and press the `Connect` button.

6.3 Team

From `ps6/team` run `build_team.bat teamname` (or `build_team.sh teamname` for Mac/Linux systems) to build the bot in the file `bot/teamname.ml`. Next, run `teamname.exe localhost 10500`. Run this command in two separate command prompts, and watch the magic happen!

6.4 The Whole Game

7 Your tasks

There are several parts to the implementation of this project. Make sure you spend time thinking about each part before starting. Start on this project *early*. There are many things you will have to take into consideration when designing the code for each section.

7.1 Design meeting

Your first task is to create a design for your *PokeJouki* implementation and meet with the course staff to review it. Each group will use CMS to sign up for a meeting, which will take place in days to be determined. If you are unable to sign up for any of the available time slots on CMS, contact the course staff, and we will try to accommodate you.

At the meeting, you will be expected to explain the design of your system, explain what data structures you will use to implement the design, and hand in a printed copy of the signatures for each of the modules in your design. You are also expected to explain your initial thoughts about strategies for your player bot. In designing module interfaces, think about what functionality needs to go into each module, how the interfaces can be made as simple and narrow as possible, and what information needs to be kept track of by each module. Everyone in the group should be prepared to discuss the design and explain why the module signatures are the way they are. We will give you feedback on your design.

7.2 Implementing the game

Your second task is to implement the *PokeJouki* game in the file `game/game.ml`, and any files you choose to add. Note that you should add files only to the `game` and `team` directories. You must implement the rules as described in Section 2 and handling of actions as described in Section 3.3. You must also make sure that the actions units take are rendered in the graphic display using the interface detailed in Section 4. You can use the sample team program we provide to test your game, but for full testing coverage you will need to write your own tests.

7.3 Designing a bot

Your third task is to implement a bot to play the game. A very weak bot that you can use as a basis for your bot code is provided.

There are many different strategies for building a good bot. This is your chance to be creative and have fun creating a good AI. There will be a tournament where you will get a chance to see your bot compete.

7.4 Documentation

Your final task is to submit a design overview document for this project. Since this project is both large and quite open-ended regarding the way one may choose to implement it, documentation becomes even more important.

Your design overview document should cover *both* your implementation of the game itself and the bot you created. In discussing your bot, you should make note of what strategies you experimented with, and what you found to be most effective.

7.5 Things to keep in mind

Here are some issues to keep in mind when designing and implementing the game:

- **You need to make a good design**. This project is both large and complicated; without spending time on making a design that is both solid and complete, you *will* very quickly get bogged down when you go to implement things. The importance of design cannot be overemphasized. Trying to write code before your have your design is a recipe for disaster on a project of this magnitude.

  Before writing any code, you should have a very clear idea of *all* of the following:

  - What concurrency issues exist and how to deal with them
  - What information needs to be kept track of to fully represent the game
  - How that information will be stored and accessed efficiently
  - What the interface between your modules will be
  - What invariants will hold between your modules
  - Which modules will enforce those invariants

- **Think carefully about how to break up your program into loosely coupled modules.** The program will be complex and difficult to debug unless you can develop modules that encapsulate important aspects of the game. Design the interfaces to these modules carefully so that you can work effectively with your partner and can do unit testing of the modules as you implement.
- **Make sure that what is going on in the game matches what is going on in the graphics.** Updating one does not automatically update the other. If you are watching the game and something seems to go wrong, remember, it could just be the code controlling the output to the screen. Moreover, just because the graphics look correct doesn't mean the game is acting properly. It would behoove you to maintain some sort of invariant between the status of the game and the status of the graphics.
- **Problems in the game might actually be problems with the bots.** If you are using your own bots to test the actions and something seems wrong, the bots could just as easily be at fault.
- **Implement and test the actions one at a time.** Don't try to implement all of the actions and test them with one single bot. Start with easier actions and work up to the harder ones.
- **Remember to finish the game in time to write a good bot.** The bot part of the project is worth almost as much as the game part, so don't put off the bot part until the end. And unlike in past semesters, we WILL be grading it based on how well it performs. It will be VERY hard to get full credit on this part.

7.6 Final submission

You will submit:

1. A zip file of all files in your `ps6` directory, including those you did not edit. We should be able to unzip this and run the `build_game.bat` script to compile your game code, and the `build_team.bat` script to compile your bot code (i.e., you should modify the scripts to include all necessary files). This should include:
   - your game implementation
   - your bot, named `bot.ml` in the `team` directory along with any files it needs to build and run

   It is very important that you organize your files in this manner, as it greatly simplifies grading.
2. Your documentation file, in `.pdf` format.

Although you will submit the entire `ps6` directory, you should only add new files to the `game` and `team` folders; the other folders should remain unchanged. If you add new `.ml` or `.mli` files, you should add them to the compilation scripts. Note again that we expect to be able to unzip your submission and run the `build_game.bat` script in the newly created directory to compile your code without errors or warnings. **Submissions that do not meet this criterion will lose points.**

8 Tournament

On TBA, after the problem set is due, there will be a *PokeJouki* tournament which you are encouraged to submit your bot programs to. There will be lots of free food, and the chance to watch your bot perform live. The winner gets bragging rights and will have their name posted on the 312/3110 Tournament hall of fame.

9 Written Problem

In addition to the game and bot implementation tasks described above, this project also includes a written problem on amortized complexity. This written question should be submitted to CMS, in `.pdf` format.

Recall the binary search algorithm. The time required for finding an element is $O(\log n)$, but to add a new element, we require $O(n)$ time in the worst case.

Über-hacker Zen "Kode" T-Rex has come up with a new data structure: the Zardoz array. This data structure supports two operations, SEARCH and INSERT. We wish to support these operations on a collection of n total elements.

Define $k = \lceil \log_2(n+1) \rceil$. We denote the binary representation of n as $n_{k-1}n_{k-2}...n_0$. A Zardoz array has k sorted arrays, $A_0, A_1, ..., A_{k-1}$. Each A_i has space for 2^i elements. If $n_i = 0$, then A_i contains 0 elements, and if $n_i = 1$, then A_i contains 2^i elements. Note that an array can't be partially filled. We also note that the total number of elements in a Zardoz array is still n, since $\sum_{i=0}^{k-1} n_i 2^i = n$. Finally, while each A_i is sorted, we do not enforce any relationship between the elements of different arrays.

1. Describe how to perform SEARCH on a Zardoz array. Analyze its worst-case running time.
2. Describe how to perform INSERT on a Zardoz array. Analyze its worst-case running time, and, using the potential method, analyze its amortized running time. Your amortized running time must be better than $O(n)$, that is $o(n)$.