Programming Assignment 2: Stochastic Pokemon

Due: Friday 03/28/2025 @ 11:59pm EST

The purpose of programming assignments is to use the concepts that we learn in class to solve an actual real-world task. To that end you will be writing java code that does not use Sepia, I have developed the game from scratch. In this assignment you will be writing tree-search agents to play stochasic pokemon (i.e. pokemon that has randomness. This is "normal" pokemon).

1. Copy Files

Please copy the files from the downloaded lab directory to your cs440 directory. You can just drag and drop them in your file explorer.

- Copy `Downloads/pa2/lib/pokePA-<VERSION>.jar` to `cs440/lib/pokePA-<VERSION>.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/pa2/lib/argparse4j-<VERSION>.jar` to `cs440/lib/argparse-4j-<VERSION>.jar`
  This is a java implementation of python's `argparse` module. It makes it easier for programs to define and parse command line arguments.
- Copy `Downloads/pa2/junit-<VERSION>.jar` to `cs440/lib/junit-<VERSION>.jar`
  This library provided unit testing functionality in Java.
- Copy `Downloads/pa2/hamcrst-<VERSION.jar` to `cs440/lib/hamcrest-<VERSION>.jar`
  This library is needed for the `junit` library to function correctly.
- Copy `Downloads/pa2/src/pas` to `cs440/src/pas`.
  This directory contains our source code `.java` files.
- Copy `Downloads/pa2/pokePA.srcs` to `cs440/pokePA.srcs`.
  This file contains the paths to the `.java` files we are working with in this lab. Just like last lab, files like these are used to speed up the compilation process by preventing you from listing all source files you want to compile manually.
- Copy `Downloads/pa2/doc/pas` to `cs440/doc/pas`. This is the documentation generated from `pokeLab-<VERSION>.jar` and will be useful in this assignment. After copying, if you double click on `cs440/doc/pas/pokemon/index.html`, the documentation should open in your browser.

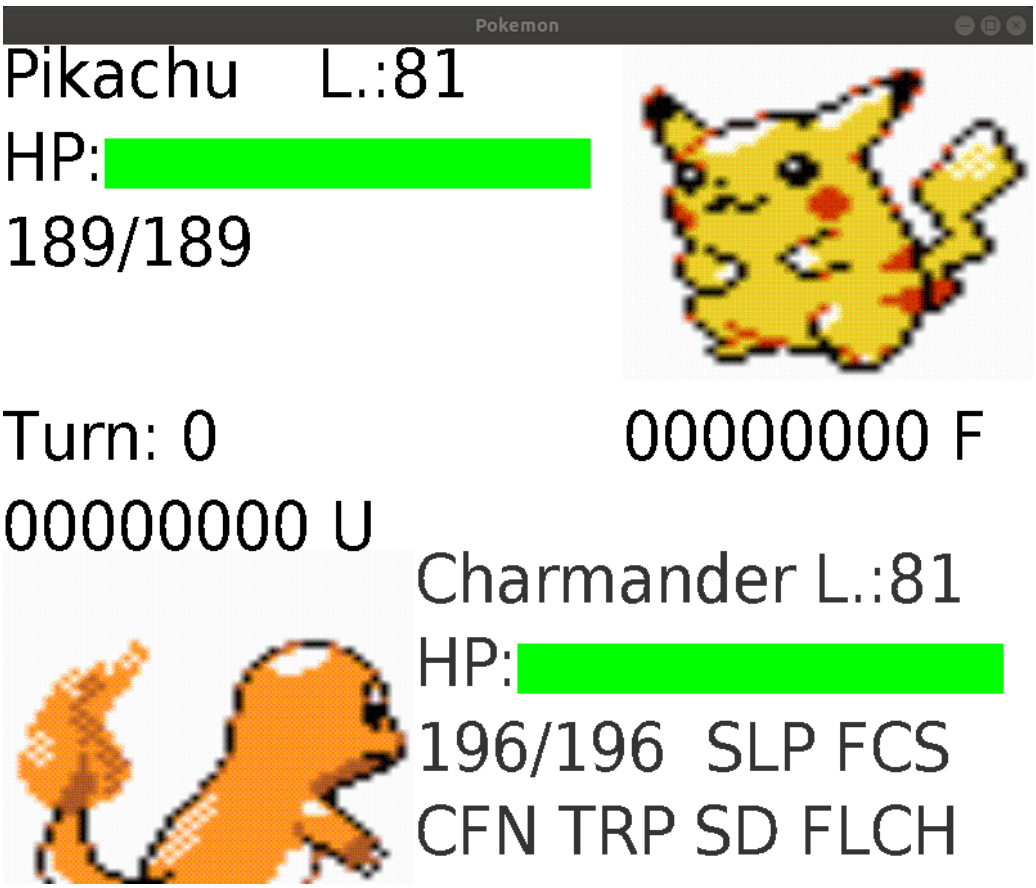
2. Test run

If your setup is correct, you should be able to compile and execute the given template code. You should see a window appear showing a Bulbasaur battling aginst a Geodude.

```
# Mac, Linux. Run from the cs440 directory.
javac -cp "./lib/*:." @pokePA.srcs
java -cp "./lib/*:." edu.bu.labs.pokemon.Easy

# Windows. Run from the cs440 directory.
javac -cp ./lib/*;. @pokePA.srcs
java -cp ./lib/*;. edu.bu.labs.pokemon.Easy
```

By default, the agent controlling the Bulbasaur will do nothing, and will lose to the Geodude (being controlled by a uniform-move-choosing agent by default). The game rendering will look something like this:



There is a lot of information about the game being conveyed in the rendering. Besides describing the turn number, let us focus on the Charmander's rendering as it shows more settings than the Pikachu:

- `Turn: XYZ` shows the current turn number of the game. This number will increment as the game progresses.
- `0000000 U`. This string either appears above or below the sprite of a pokemon. The integer numbers show the stage multipliers for the seven stats a pokemon keeps track of. During battle, a pokemon's stats can either be strengthened or weakened, and that is reflected in these numbers. If the number is positive, that pokemon's stat has been strenghtened. If the number is negative, that pokemon's stat has been weakened. Please click here to learn about how these values are

used to change a pokemon's stat. We are using GenI-II rules when they are combined, and Gen II rules when they are separate. The stats these numbers modify are in the following order:

1. ATK: The strength of a pokemon when dealing physical attacks.
2. DEF: The defense of a pokemon against physical attacks.
3. SPD: The speed of a pokemon.
4. SPATK: The strength of a pokemon when dealing special attacks.
5. SPDEF: The defense of a pokemon against special attacks.
6. ACC: The accuracy of a pokemon. This affects how likely a move this pokemon uses is to hit its target.
7. EVASIVE: The evasiveness of a pokemon. This affects how likely a move is to hit this pokemon.

Each stage multiplier is bounded in the interval [-6, +6].

The `U` shows the height of a pokemon. If a pokemon is underground (i.e. using the move `Dig`), you will see `U`. If a pokemon is flying in the air (i.e. using the move `Fly`), you will see `F` (as in the case of the Pikachu). If a pokemon is neither flying nor underground, you will not see a character in this location.

- `Charmander L.:81`. This string shows the name of the pokemon as well as its level. Generally, the higher the level, the more powerful a pokemon is.
- `HP: <BAR>`. This shows how much health a pokemon has as a percentage of its initial health. Over 50% health and the bar is green, less than 50% but over 20% and the bar is yellow. Below 20% the bar will be red.
- `196/196`. This is a numerical format of the HP shown in the HP bar.
- `SLP`. This is where the persistent status (called a `NonVolatileStatus` in the code) of a pokemon will be shown. Currently the Charmander is asleep, but there are other persistent statuses that a pokemon can have. Note that a pokemon can only have a single persistent status, so if they are afflicted with one already, they cannot be afflicted with another until the existing one is removed. Here are the persistent statuses that a pokemon can be afflicted with:
  - PARALYSIS: If paralyzed, a pokemon has a 25% of losing their turn due to being fully paralyzed. Additionally, a pokemon's SPEED is rounded down to 75% of its value when determining the move order. The string `PAR` will appear when a pokemon is paralyzed.
  - POISON: When poisoned, a pokemon will take 1/8 of its maximum HP as damage at the end of each turn. The string `PSN` will appear if a pokemon is poisoned.
  - TOXIC: This is like poison but on steroids. When affliced by toxic, a pokemon will take N/16 of its maximum HP as damage at the end of every turn, where N counts the number of turns since being afflicted with toxic. The string `TOX` will appear if a pokemon is affliced with toxic.
  - BURN: Like poison, a pokemon will take 1/8 of its maximum HP as damage at the end of every turn. However, if a pokemon is frozen and a move applies a burn, that pokemon will thaw and will no longer be frozen (and it will also not be burned). The string `BRN` will appear if a pokemon is burned.
  - FREEZE: When a pokemon is frozen, it will stay frozen (and unable to make a move) for at maximum 7 turns. At the beginning of that pokemon's move, it has a 9.8% chance of thawing on its own. As mentioned, a frozen pokemon can be thawed early by being burned when frozen. The string `FRZ` will appear if a pokemon is frozen.

  - SLEEP: When a pokemon is asleep, it will sleep for a maximum of 7 turns. Like being frozen, it will have a 9.8% chance of waking up at the beginning of its move. This deviates from the traditional sleep rules, but I found it was easier to implement. The string `SLP` will appear is a pokemon is asleep.
  - NONE: A pokemon that does is not afflicted with anything gets this value. No string will appear.

- `FCS`. This string appears if a pokemon has used a move called `Focus Energy`. This move increases the chances that a pokemon will inflict a critical hit the next turn. Unlike Gen-1 pokemon, where this is bugged (in Gen-1, focus energy actually **halves** your critical hit chance instead of **doubling** it), I have made sure the chances of a critical hit have been **doubled**. This is considered a "volatile" stat, so if you use `Focus Energy` and then swap the pokemon out, this stat will be lost.

- `CFN`. This will appear when a pokemon is confused. A pokemon will remain confused for anywhere between 2 to 5 turns maximum. At the beginning of using a move, a pokemon has a 50% chance of doing damage to itself instead. This is cured when the confused pokemon is swapped out or the confusion duration ends.

- `TRP`. This will appear when a pokemon is trapped. Trapped pokemon cannot be swapped out until the trap ends. The duration of a trap are set by moves designed to trap pokemon (e.g. `Fire Spin`, etc.).

- `SD`. This will appear if a pokemon is seeded. The move `Leech Seed` plants seeds on the opponent pokemon, which at the end of every turn, sap health from the seeded pokemon and deliver that health to the non-seeded one (who recovers a little bit of health). This is cured when the seeded pokemon is swapped out.

- `FLCH`. This will appear if a pokemon has flinched. Some moves may cause the target of a move to flinch when struck by that move. When a pokemon flinches, it will not execute the move it had inteded to during that turn (if the flinched pokemon goes after it has flinched). This is reset at the end of every turn.

If you wish to change the agent (the Charmander will now win the battle), you can do so by specifying the `--t1Agent` command line argument when running the program (here is the Linux/Mac version of how you would do this):

```
java -cp "./lib/*:." edu.bu.labs.pokemon.Main --t1Agent src.labs.pokemon.agents.MinimaxAgent
```

There are a few other command line arguments you can provide to this program. To see the list of what arguments you can provide, please trigger the help message by doing the following (here is the Linux/Mac version):

```
java -cp "./lib/*:." edu.bu.labs.pokemon.Main -h
```

Task 3: Tree Traversal

I have provided one kind of tree traversal functionality, and that is the method `getPotentialEffects` in the `Move` and `MoveView` class. This method will return a `List` of all potential outcomes of a move. However, I have not provided any tree traversal functionality beyond this. One of the important parts of this programming assignment is that you develop tree traversal infrastructure to help you tree-traversal agent examine all of the potential outcomes of using a `MoveView`. Remember, your tree needs to have `CHANCE` nodes inside it along with `MAX/MIN` nodes. There are several layers of randomness in stochastic pokemon (listed in order) that happen every turn of the game:

1. The ordering of which pokemon's moves will occur happens first. Each move has a priority stat to it, and pokemon have speed stats. If one pokemon submits a move with a higher priority (i.e. larger priority value) than the other, the move with the higher priority will occur first. Moves where a user decides to switch out their pokemon (called `SwitchMoves`) have high priority. If both pokemon submit moves with the same priority, then the speed stat of the two pokemon deicide the order (i.e. the faster pokemon goes first). Remember that being paralyzed reduces you speed to 75% of its original value here! Finally, if both pokemon are equally fast, then its a pure 50/50 chance you go first. You will have to program this randomness to generate nodes for your tree.

2. Once the move order is decided, the moves submitted are attempted to be executed in that order. However, if a pokemon is asleep/frozen/paralyzed, then there is a chance that the submitted move will not occur. Provided that the submitted move makes it through this check, if the pokemon is confused, there is a 50% chance that the submitted move will not occur and the pokemon will hurt itself instead. You will have to program this randomness to generate nodes for your tree. When dealing with the hurting yourself due to confusion, I would recommend generating a synthetic move and use the `getPotentialEffects` api. You will have to do something similar to this to generate the synthetic move:

```
hurtYourselfMove = new Move(
    "SelfDamage",           // move name
    Type.NORMAL,            // damage type (should be typeless but we'll tell the
                            // to ignore STAB and type terms in damage calculation
    Category.PHYSICAL,      // move category
    40,                     // base power for hurting yourself from confusion is 40
    null,                   // infinite accuracy
    Integer.MAX_VALUE,      // number of uses
    1,                      // critical hit ratio
    0                       // priority
).addCallback(
    new MultiCallbackCallback(
        new ResetLastDamageDealtCallback(), // new damage so reset old value
        new DoDamageCallback(
            Target.CASTER,                  // hurt yourself
            false,                          // dont include STAB term in damage calc
            false,                          // ignore type terms in damage calculation
            true                            // damage ignores substitutes
        )
    )
);

List<Pair<Double, BattleView> > confusionDamageOutcomes = hurtYourselfMove
    .getView().getPotentialEffects(...); // get all the potential outcomes
```

3. If the move you submit has made it this far, it can resolve in a bunch of different ways. For instance, moves can miss, or the damage they cause can be random, or they might cause additional side effects, etc. I have handled getting the potential ways the move can resolve for you with the `getPotentialEffects(...)` api.

4. Finally, once the turn is over, there are some post-conditions that can occur. For instance, any damage caused by POISON/TOXIC/BURN/SEEDING needs to be calculated and applied. Flags need to be reset (like FLINCHING), counters for FREEZE/SLEEP need to be decremented, etc. If a pokemon has fainted (i.e. its HP is now zero), then that pokemon must be swapped out for a new, unfainted one. Since you don't know which pokemon you want to choose if your pokemon faints, or if its your opponents, you don't know which one they'll choose, you have to consider all available options. I have not implemented this part of the tree traversal for you, so you will need to program this randomness to generate nodes for your tree. **Note** that this only occurs at the end of every **turn**, which is every **two** layers of the tree (and should not be applied every layer).

Note: I have **not** implemented any functionality for getting the set of available moves. You will have to implement this! The only thing I have done is check that the move you submit is legal. If you submit an illegal move, your pokemon will not move that turn!

Task 4: Tree-Traversal Agent (100 points)

Please take a look at the file `src/pas/pokemon/agents/TreeTraversalAgent.java`. I want you to complete this file with an implementation of your chosen stochastic tree-traversal algorithm. Inside of this file are several areas for implementation:

- You will probably want to create some data type(s) to help create/traverse the game tree. I recommend making a `Node` class where you can put all logic for getting children, utilty values, etc. If you choose to do so, you are welcome to make this another file, **however** any files you create should go inside the same directory as `TreeTraversalAgent` and should belong to the same `src.pas.pokemon.agents` package!

- The game tree in stochastic pokemon is not only massively wide, it is also infinite (because you and your opponent can technically constantly switch our your pokemon forever). You **will** need to artificially constrain the size of your tree, meaning that you need utility heuristics. Like earlier, you are welcome to create a new file to contain your heuristics, as long as it is the same directory as `TreeTraversalAgent` and belongs to the same package.

- Method `stochasticTreeSearch` inside the `StochasticTreeSearcher` class. The `stochasticTreeSearch` method is where your stochastic tree-traversal should go. Your traversal algorithm should use any infrastructure code (like your utility heuristics, etc.) in it. Since the game tree is massively wide, you will likely want to implement an algorithm that prunes subtrees for speed.

- Since your tree algorithm is pruning subtrees, you will need to decide the order that the algorithm examines nodes. This ordering scheme, like the others, is welcome to be placed in a separate file provided that it follows the rules (same directory and package as `TreeTraversalAgent`).

There are three difficulty settings in this assignment. Please click here for more information:

- EASY: You will face Brock's Gen-1 team (a Geodude and an Onix). You are given a Bulbasaur, who has two moves super-effective against Brock's pokemon. This battle should be easy to win.

- MEDIUM: You will face Sabrina's Gen-1 team (Kadabra, Mr. Mime, Venemoth, and Alakazam). You are given a Snorlax, a Charizard, a Parasect, and a Beedrill. This will be difficult match to win: even though you have pokemon that know moves super effective against Sabrina's, your pokemon are also partially weak to psychic damage.

- HARD: You will face Lance's Gen-1 team (Gyarados, Dragonair, Dragonair, Aerodactyl, and Dragonite). You are given a Snorlax, a Venusaur, a Dragonite, a Lapras, and a Machamp. Even though you have pokemon who know moves that are super effective against Lance's, his pokemon do tons of damage, and your pokemon are also partially weak against his. This should be a hard fight to win.

Task 5: Extra Credit (50 points)

We are currently devising a team and a custom agent who **will** be extremely difficult to beat. We will release more information on this in the future. If you can beat this agent at all, we will award you full extra credit.

Task 6: Submitting your lab

Please submit the file: `TreeSearchAgent.java` on gradescope (no need to zip up a directory or anything, just drag and drop the file) as well as any additional java files you created for this assignment.

Task 7: Tournament Eligibility

In order for your submission to be eligible in the tournament, your submission must satisfy all of the following requirements:

- Your submission must be on time.
- You do not get an extension for this assignment.
- Your agent compiles on the autograder.
- Your agent can beat the `EASY` difficulty more than 80% of the time, the `MEDIUM` difficulty more than 50% of the time, and the `HARD` difficulty more than 25% of the time.