Poker AI

Devin Lehmacher - djl329

Jonathan Ho - jh964

For the final project of CS 2024, we decided to create a relatively simple AI that would play a solid strategy for heads up no limit hold'em that is theoretically balanced and unexploitable. The AI, which is implemented in C++, is paired with a Java interface that allows a user to play against it.

How To

Creating something using the Java Native Interface is fairly easy. There are a few simple steps that you need to follow in order to get it to work.

1. Write some C++ method that you want to call in Java.
2. Make a Java program that is supposed to call that method. Label the method `native`.
3. Compile the Java file (e.g. `javac File.java`)
4. Run `javah File` to create a header file that contains the prototype for the method that will be called using the JNI.
5. Implement the prototype using the methods in the `<jni.h>` header.
6. Compile your program as a dynamic library. This depends on what you are trying to do exactly but in general you will do something like: `g++ -dynamiclib -o libFoo.jnilib <c++ source files>`.
7. After obtaining your library put it in some convenient location.
8. Now you are almost done. Load the library statically using `System.loadLibrary("Foo");` or the equivalent with the name of your library but `lib` chopped off the front and `.jnilib` chopped off the back in a static initialization block (e.g. `static { <stuff> }`) for your class.
9. Run your Java program `java -Djava.library.path=<path-to-library> File`. This tells the Java VM where it can find your library and then runs your program. Whenever the `native` method is called Java will invoke the method that you wrote in your C++ code.

This was a general overview now lets see how the specific parts of this program work.

Overview

Our program is implemented roughly with a MVC paradigm. Java files are split between the packages `tui` (which implements the terminal ui) and the package `model` (which implements the game itself). We have a completely separate source tree for C++ namely `PokerBot` which is the C++ part of our code. Compiling our code can be done using `make`, however I think that this only works on Macs since neither of us had a PC or Linux machine to test on. Obviously this makes the fact that we are using Java not quite as useful since you loose the ability to use this in a cross platform way.

JNI

We use the JNI for two methods. One that calculates the next move that the AI should use and one that computes which poker hand is better than the other. These are found at

`model.State.winningMove` and `tui.PokerAI.nativeNextMove`. We made them accept parameters that are very primitive since we thought that that would make it easier to implement the JNI. The files `PokerBot/model_State.h` and `PokerBot/tui_PokerAI.h` have the method prototype that the JVM uses to to invoke the methods that we declared in our Java files. `PokerBot/PokerLib.cpp` is the implementation of these two interfaces and is the file that compiles to the library that we use when running the program.

Model

The package model implements the game. I implemented this mostly through the use of an observer paradigm. You can register listeners to the game and then one of their methods `gameChanged(Game)` is called whenever the game changes. This makes our design quite flexible and was a useful way of implementing this.

TUI

The package TUI implements the UI. This is implemented through a few controllers that give the user information that they need to make their move and takes input that they pass back to the game. `GamePrinter` prints out important points in the game such as whenever a round ends or when the game ends. It also manages the transitions between phases of the game. The main method simply sets up two controllers and starts the game. The UI looks like this when it is a human players turn.

```
(50.0): 7♠, 8♥, J♦, -, -
You (975.0): 4♣, 8♦
Opponent (975.0)
Please enter one of the following commands:
 call
 raise <amount> where amount >= 15.0
 fold
 all-in
```

AI

The AI maintains a range of possible hands it could have in each situation and makes decisions to bet, raise, call or fold based on where the actual holding is within that range. Further, the AI selects specific hands to use as bluffs in order to maintain particular ratios of bluffs and value bets. Perhaps the most important part of this feature is that regardless of situation, the AI is able to decide, with certainty, the frequency with which it is taking each action.

Running

We run the game using eclipse. We configured eclipse to use our library by setting `-Djava.library.path=lib/` as a VM arg in Run Configuration.