NOTE: THIS IS A MARKDOWN FILE, BEST VIEWED IN THE INCLUDED PDF, OR A MARKDOWN VIEWER.

Project 4: Konane Game Implementation

By: Kevin Bohinski & Hannah Richman & Brittany Plummer

12/12/2016

CSC 435 Programming Languages

Readme File

Files

```
p4.zip
+-- .gitignore
+-- build.sbt
+-- index.html
+-- readme.md
+-- readme.pdf
\-- project
    +-- plugins.sbt
\-- src
    \-- main
        \-- scala
            \-- com
                \-- plteamscala
                    \-- konane
                        +-- Board.scala
                        +-- Driver.scala
                        +-- HumanPlayer.scala
                        +-- MinimaxPlayer.scala
                        +-- Player.scala
```

`*.scala, index.html` contains our source, and `readme.md` or `readme.pdf` contains this readme.
`*.sbt` contains information for the `sbt` build tool.

Dependencies

- The version of Scala required to run this project is 2.12.0.
- The user also requires sbt version 0.13.13.
- We use PubNub for networking communication. This should be downloaded with sbt.

Running the Project

- Unzip the folder.
- Type: `sbt clean update run clean`
- Done! Output should be printed.

Input & Output

Our file requires you to pick between the web version and the standard version, input the size of the board, if the user wants the board displayed between every move, how many games the user wants to play, who will go first, and if the user would like the game to run automatically against a "random" player. After the completion of the games, the percentage of wins as well as the actual number of wins for both the human and minimax players is printed.

When selecting the web version, the user must also enter the room number provided to them on the page. Having different rooms allows for multiple users to play multiple different games at the same time.

Positives

- Project has all required rules implemented
- Can run automatically without user input which makes for easier debugging
- Prints out statistics after running the specified number of games
- We made a realtime web version which enhances a player's sense of visualization for the game
- Minimax wins 99% of the time!

Limitations and Bugs

- None that we know of...

Conventions

- Attempted to follow conventions found in the Scala documentation.

Abstractions

- We took advantage of inheritance in Scala to simplify the Player classes.

Data Structures

- A two-dimensional array is used for the board, and an ArrayBuffer (similar to an ArrayList) is used in conjunction with Tuples for the list of possible moves that can be made. Maps and JSON are used for web communications.