COMP 424 - Project Specification:
Tablut

Course Instructor: Jackie Cheung (jcheung@cs.mcgill.ca)
Project TA: Kian Kenyon-Dean (kian.kenyon-dean@mail.mcgill.ca)

CODE REPOSITORY: `https://github.com/kiankd/comp424`

Code due: April 7, 2018
Report due: April 8, 2018

Goal

The main goal of the project for this course is to give you a chance to play around with some of the AI algorithms discussed in class, in the context of a fun, large-scale problem. This year we will be working on a game called *Tablut*, a Finnish variant of the classic Viking game Hnefatafl from over a thousand years ago. Kian Kenyon-Dean is the TA in charge of the project and should be the first contact about any bugs in the provided code. General questions should be posted in the project section in mycourses.

Rules

Tablut is a two player game played on a 9 by 9 board. One player is the *Muscovites* (black), and the other is the *Swedes* (white); the invading Muscovite horde seeks to destroy the Swedish king, while the noble Swedish knights seek to assist the Swedish king to safety. The *Muscovites* start with 16 pieces and the *Swedes* start with 9 total (with one of those being the king). *Muscovites* always play first.

The objective for the *Muscovites* is to **capture** the Swedish King, while the objective for the *Swedes* is to **move their king to a corner**. The game thus ends after either of those conditions are met, or if the total number of turns exceeds the maximum limit.

All pieces move look rooks move in Chess, namely, they can move any number of tiles either vertically or horizontally from their current position, provided that their path is not impeded by another piece. There are also the following restrictions on movement:

- Only the king may move to a corner or the center.
- While any other piece cannot move *to* the center, they can still move *through* the center.

The rules for **capturing** are as follows:

1. If a piece is *captured* it is removed from the board.
2. Any piece is captured if it becomes "sandwiched" between two opponent pieces as a result of the opponent's move; i.e., a piece in position (x, y) will be captured if there is an opponent piece at both $(x-1, y)$ and $(x+1, y)$ *or* $(x, y-1)$ and $(x, y+1)$.
3. The king, however, has extra protection; the king needs to be surrounded on *all 4 sides* if it is in the center or in a position that neighbors the center (highlighted in gold colors in the game GUI). If the king is not in one of these positions, it can be captured as normally.

4. A piece may also be captured if it is sandwiched between an opponent piece and either the *center* or a *corner*. This is why it is often inadvisable for the king to move right next to a corner, since black can easily capture it just with one piece.

We will hold a competition between all the programs submitted by students in the class, with every submitted program playing one match against every other program. Each match will consist of 2 Tablut games, giving both programs the opportunity to play first.

Submission Format

We have provided a software package which implements the game logic and provides interface for running and testing your agent. Documentation for this code can be found in **code_description.pdf**; you will need to read that document carefully. Create your agent by directly modifying the code found in **src/student_player**.

The primary class you will be modifying is located in **src/student_player/StudentPlayer.java**. Comments in that source file provide instructions for implementing your agent. Your first step should be to alter the constructor for StudentPlayer so that it calls **super** with a string representing your student number instead of "xxxxxxxxx". You should then modify the **chooseMove** method to implement your agent's strategy for choosing moves.

We recommend reviewing the code written in **tablut/GreedyTablutPlayer.java**, as that provides an example of cloning the board state, testing moves on it, and how to interact with the BoardState API in a useful way for optimizing your algorithm.

When it is time to submit, create a new directory called **submission**, and copy your data directory and your modified **student_player** source code into it. The resulting directory should have the following structure:

```
submission
|
|-- src
|  |
|  |-- student_player
|     |
|     |-- StudentPlayer.java   ... the class you will edit
|     |
|     |-- MyTools.java         ... any useful methods go here
|     |
|     .
|     . ... any other useful classes can be made here
|
|-- data
   |
   .
   . Any data files required by your agent.
```

Finally, compress the **submission** directory using zip. **DO NOT USE ANY COMPRESSION OTHER THAN .ZIP**. Your submission must also meet the following additional constraints:

1. The constructor for **StudentPlayer** must do nothing else besides calling super with a string representing your student number. In particular, any setup done by your agent must be done in the **chooseMove** method.

2. Do not change the name of the **student_player** package or the **StudentPlayer** class.

3. Sub-packages are allowed. We have provided an example sub-package called **mytools** to show how sub-packages can be used.

4. Data required by your program should be stored in the **data** directory.

5. You are free to reuse any of the provided code in your submission, as long as you document that you have done so.

We plan on running several thousand games and cannot afford to change any of your submissions. Any deviations from these requirements will have you disqualified, resulting in part marks.

You are expected to submit working code to receive a passing grade. If your code does not compile or throws an exception, it will not be considered working code, so be sure to test it thoroughly before submitting. If your code runs on the Trottier machines (with the unmodified hus and boardgame packages), then your code will run without issues in the competition. We will run your agent from inside your submitted **submission** directory.

Competition Constraints

During the competition, we will use the following additional rules:

Turn Timeouts. During each game, your agent will be given no more than 30 seconds to choose its first move, and no more than 2 seconds to choose each subsequent move. The initial 30 second period should be used to perform any setup required by your agent (e.g. loading data from files). If your player does not choose a move within the allotted time, a random move will be chosen instead. If your agent exceeds the time limit drastically then you will suffer an automatic game loss.

Memory Usage. Your agent will run in its own process and will not be allowed to exceed 500 mb of RAM. The code submission should not be more than 10 mb in size. Exceeding the RAM limits will result in a game loss, and exceeding the submission size will result in disqualification. To enforce the limit on RAM, we will run your agent using the following JVM arguments: "-Xms520m -Xmx520m".

Multithreading. Your agent will be allowed to use multiple threads. However, your agent will be confined to a single processor, so the threads will not run in parallel. Also, you are required to halt your threads at the end of your turn (so you cannot be computing while your opponent is choosing their move).

File IO. Your player will only be allowed to *read* files, and then only during the initialization turn. All other file IO is prohibited. In particular, you are not allowed to *write* files, so your agent will not be able to do any learning from game to game.

You are free to implement any method of choosing moves as long as your program runs within these constraints and is well documented in both the write-up and the code. Documentation is an important part of software development, so we expect well-commented code. All implementation must be your own. In particular, you are not allowed to use external libraries.

Write-up

You are required to write a report with a detailed explanation of your approach and reasoning. The report must be a **typed PDF file**, and should be free of spelling and grammar errors. A professional report in any scientific setting *should be written in* LaTeX, which is unquestionably objectively superior to Microsoft Word and Google Docs; while this is advisable and it would be better for you in the long run to learn LaTeX, this is not required. The suggested length is between 4 and 5 pages (at ∼300 words per page), but the most important constraint is that the report be clear and concise (which would be assisted by writing separate sections as is done in all serious scientific works – fortunately LaTeXmakes it quite easy to make separate numbered sections). The report must include the following required components:

1. An explanation of how your program works, and a motivation for your approach.

2. A brief description of the theoretical basis of the approach (about a half-page in most cases); references to the text of other documents, such as the textbook, are appropriate but not absolutely necessary. If you use algorithms from other sources, briefly describe the algorithm and be sure to cite your source.
3. A summary of the advantages and disadvantages of your approach, expected failure modes, or weaknesses of your program.
4. If you tried other approaches during the course of the project, summarize them briefly and discuss how they compared to your final approach.
5. A brief description (max. half page) of how you would go about improving your player (e.g. by introducing other AI techniques, changing internal representation etc.).

Marking Scheme

50% of the project mark will be allotted for performance in the tournament, and the other 50% will be based on your write-up.

Tournament

The top scoring agent will receive full marks for the tournament. The remaining agents will receive marks according to a linear interpolation scheme based on the number of wins/losses they achieve. To get a passing grade on the tournament portion, your agent must beat the random player.

Write-up

The marks for the write-up will be awarded as follows:

```
Technical Approach:                    20/50
Motivation for Technical Approach:     10/50
Pros/cons of Chosen Approach:           5/50
Future Improvements:                    5/50
Language and Writing:                   5/50
Organization:                           5/50
```

Academic Integrity

This is an individual project. The exchange of ideas regarding the game is encouraged, but sharing of code and reports is forbidden and will be treated as cheating. We will be using document and code comparison tools to verify that the submitted materials are the work of the author only. Please see the syllabus and www.mcgill.ca/integrity for more information.