Comp 424 - Project Specification

Course Instructor: Jackie Cheung (jcheung@cs.mcgill.ca)
Project TA: Ali Emami (ali.emami@mail.mcgill.ca)

Code due: April 10, 2017
Report due: April 11, 2017

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 Das Bohnenspiel. Ali Emami 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

Das Bohnenspiel ("the bean game") is a two-player competitive game that is part of the *mancala* family of board games, which generally involve placing beans in pits and capturing them. In Bohnenspiel, the field consists of two rows of six pits each, arranged in a 2x6 grid. Each player "owns" the six pits closest to them. The game begins with every pit filled with six beans. The player whose turn it is chooses any of their pits that contains at least one bean and scoops all of the seeds in that pit. The player then *sows* the scooped beans, that is, by distributing them counter-clockwise from the scooped pit, and dropping them one-by-one in each subsequent pit until running out of seeds. Note that the seeds are sown across all twelve pits, regardless of whether or not the player owns the pit. Furthermore, if the player has just sown into his final pit and has more beans to sow, they will "spill over" into the opponent's first pit (and possibly back to the player's first pit), etc.

If a player sows their beans such that the last bean sown results in a pit of exactly two, four, or six beans (but no other number), all beans in this pit are captured and stored by the player permanently as a "score". If any capture is made, the preceding pit is checked (and its beans also possibly captured) according to the same rule, and so forth. If the player whose turn it is to move cannot move, the game ends and all beans on the board go to the other player.The goal is to capture more beans than the opponent–that is, the player with the highest score wins.

Note that the players can capture beans from any pit, whether or not they own it, but that they can only begin sowing beans from a pit on their side of the board.

As a slight modification to the classic game, our version of Bohnenspiel will allow each player to "skip" their turn (assuming they see moving in any way detrimental) up to three times per game.

Watch the following example game between two expert players here:

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.

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
|      |
|      .
|      . Any sub-packages that you write. See the ‘‘mytools’’ package for an example.
|
\--- data
   |
   .
   . Any data files required by your agent.
```

Finally, compress the **submission** directory using 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 bohnenspiel 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 1 second 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 (for example, if it gets stuck in an infinite loop) 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.

Infinite Moves. To deal with the possibility of infinite moves, we will simulate a given move for at most 200 iterations (where an iteration is defined as scooping a pit and sowing its seeds). If a move is played which has not ended by 200 iterations, the game will be cancelled and restarted. If a player is responsible for 3 such cancellations in a single game, they will suffer an automatic game loss. You can check whether a move will exceed this limit by running the move on a copy of the board state. Several methods of the BohnenspielBoardState class can be used to detect whether such an invalid move has occurred.

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. 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. 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.