Comp 424 - Project Specification
Saboteur

Course Instructor: Jackie Cheung (jcheung@cs.mcgill.ca)
Course Instructor: Adam Trischler (Adam.Trischler@gmail.com)
Project TA: Pierre Orhan (pierre.orhan@mail.mcgill.ca)

Code repository: https://github.com/PierreOrhan/SaboteurComp424

Code due: 9 April 2020
Report due: 9 April 2020

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 variant of a famous game: Saboteur. Pierre Orhan 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. Nishant Mishra is also involved in the project, and questions can also be addressed to him.

Rules

The Saboteur game is very fun, and we encourage you to play it using the GUI interface in order to get a good understanding of the rules. Some of you might have played it for real, but the version we will be working on this year is a bit different. Please carefully read the linked document that extensively describes the new rules.

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 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 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. Additional classes in the student player package are allowed. We have provided an example class called MyTools to show how this can be done.
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 Saboteur 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 (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".

Multi-threading. 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. This means built-in libraries for computation such as Math and String are allowed but libraries made specifically for machine learning or AI (e.g. Deeplearning4j) are not.

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 $\sim$ 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 (and code).

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/draws they achieve. To get a passing grade on the tournament portion, your agent must beat the random player.

Report

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 a project in group of 2. The exchange of ideas regarding the game is encouraged between groups, 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.
Any document used for this project should be strictly confined to the project and should not be shared outside the scope of the course. (For example: do not share pictures of the Saboteur GUI board).

Final tips

As we have created the game for the project, it is not clear to us what will be the best strategy. Therefore do not hesitate to try bold moves. Note that the full deck description is available in the SaboteurCard class, to consider it carefully may be a good initial start. In fact, planning ahead will be quite difficult if you do not estimate the distribution of the other players' hand (especially in late game)... Your agent will have to make compromises between keeping good tiles and control effect (obtaining the upper hand now or latter). Exploration/exploitation compromises will also be involved through the map card. These are very common problems encountered when creating a learning agent. You will learn a lot from solving them and we look forward to reading the strategies you will have designed.