COMP 424: Project Details
Pentago-Twist

Course instructor: Jackie Cheung (jcheung@cs.mcgill.ca)
Project TAs: Erfan Seyedsalehi and Samin Yeasar

Due date: April 13th, 2021, 9:00pm Eastern

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 **Pentago-twist** (follow the link: https://github.com/SaminYeasar/pentago_twist), a variation on a more popular game called Pentago. Erfan Seyedsalehi and Samin Yeasar are the TAs in charge of the project and should be the contact about any bugs in the provided code. General questions should be posted in the Microsoft Teams. **This is an individual project**.

Rules

Pentago-Twist falls into the Moku family of games. Other popular games in this category include Tic-Tac-Toe and Connect-4, although Pentago-Twist has significantly more complexity. The biggest difference in Pentago-Twist is that the board is divided into quadrants, which can be rotated 90 degree clockwise or flipped horizontally during the game.

Setup Pentago-twist is a two-player game played on a 6x6 board, which consists of four 3x3 quadrants. To begin, the board is empty. The first player plays as white and the other plays as black.

Objective. In order to win, each player tries to achieve 5 pieces in a row before their opponent does. A winning row can be achieved horizontally, vertically or diagonally. If all spaces on the board are occupied without a winner then a draw is declared. If rotating/flipping single quadrant results in a five-in-a-row for both players, the game also ends in a draw.

Playing. Moves consist of two phases: placing and rotating/flipping. On a given player's turn, a piece is first placed in an empty slot on the board. The player then selects a quadrant, and can choose either to flip horizontally (see figure 2) or rotate 90 degress clockwise (see figure 1). A complete move therefore consists of placing a piece, then rotating or flipping.

Strategy. Allowing quadrants to be rotated/flipped introduces significant complexity and your AI agent will need to contend with this high branching complexity. Since quadrants can be rotated/flipped, blocking an opponent's row is not as easy as simply placing an adjacent piece. A good AI agent might consider balancing seeking to win while preventing their opponent from achieving the same.

Evaluation. In the first round of the evaluation, we will hold a competition where every submitted program will be playing one match against a random subset of all submissions. Each match will consist of 2 Pentago-twist games, giving both programs the opportunity to play first. For the second round, the 10 percent highest scoring agents from the first game will play against each other in the playoff round where each agent is paired against all other agents

to find the highest scoring agents. The evaluation phase will require a lot of matches so please be mindful of your program's runtime. In order to reduce the compute load, we might do another round before these two where agents will be paired against simple agents (random or Alpha-Beta pruning with simple evaluation functions).

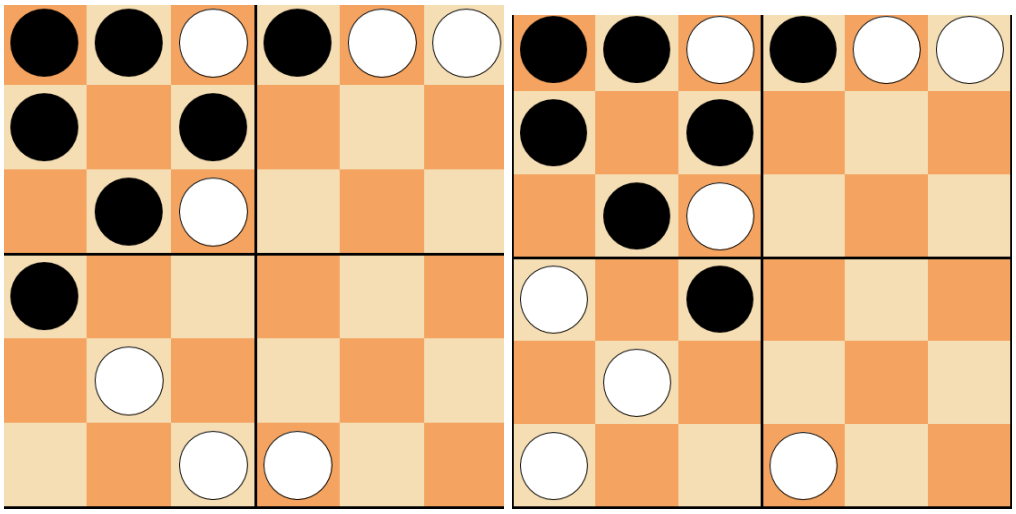

Figure 1: White player places its move at (0,5) and choose the third Quadrant to rotate 90 degrees clockwise.

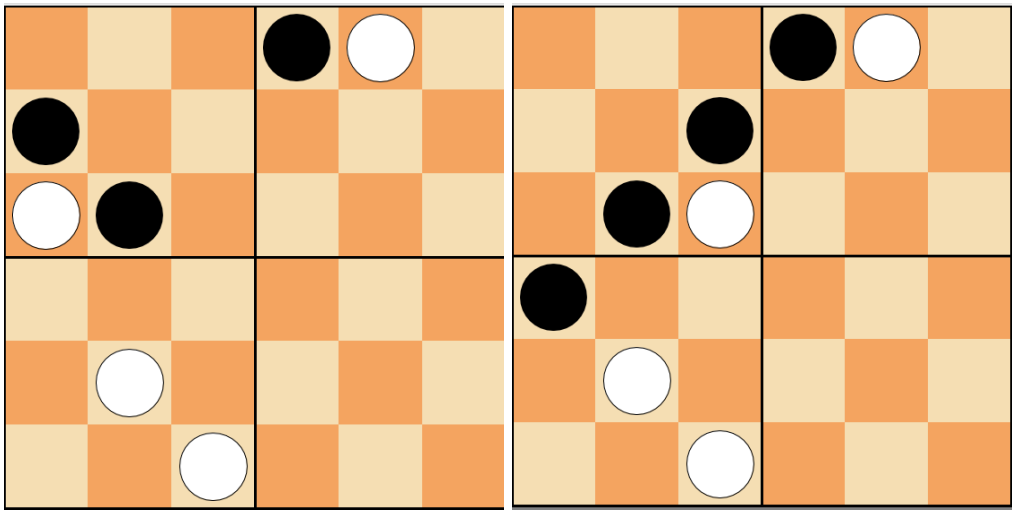

Figure 2: Black player places its move at (0,3) and choose the first Quadrant to flip horizontally.

Submission Format

The code and the report needs to be submitted through MyCourses. Submit your project as **"Submission.zip"**, which should contain your **"report.pdf"** and your code following the code structure described below.

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 (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
    | — report.pdf
```

Finally, compress the submission directory using zip. **DO NOT USE ANY COMPRESSION OTHER THAN .ZIP**. Your submission must also meet the following additional constraints, **OTHERWISE YOUR SUBMISSION WILL NOT BE EVALUATED**.

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 (with the unmodiffied pentago-twist 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 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 300 words per page), but the most important constraint is that the report be clear and concise. Although it is not necessary, it is highly suggested you use LATEXto write up the report. 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 Marks distribution

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.