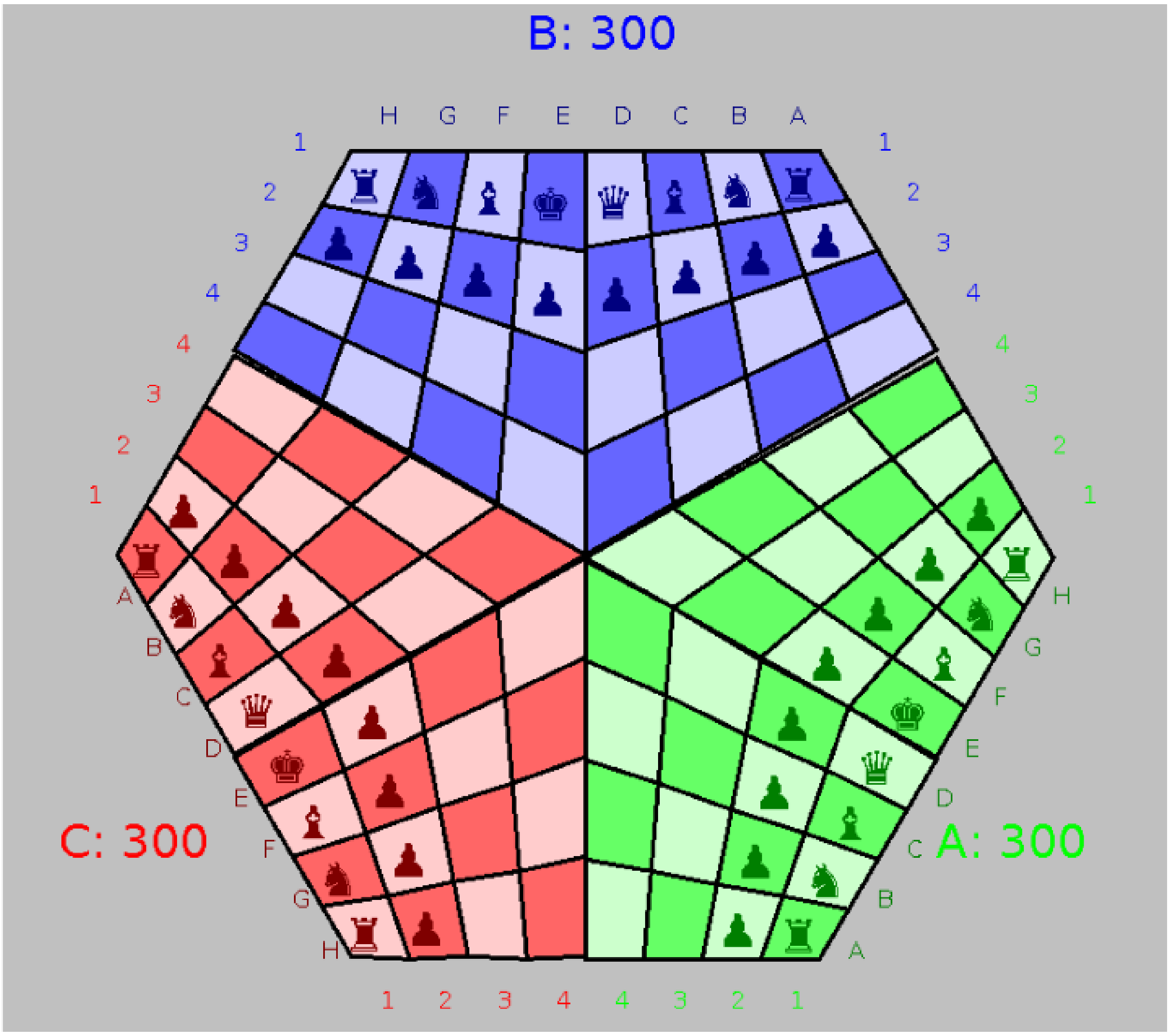
CITS3001 Project 2020

Attached Files:
3001 2020 Project assessment criteria.pdf

- Due: 5pm on Friday 23 October.
- Submit via cssubmit.
- The project contributes 30% to your final grade.
- Most people are doing the project in a pair with another student. Partners have been assigned; you can find the student ID of your partner at csmarks. Make contact soon via ABCDWXYZ@student.uwa.edu.au!

You are required to research, implement and validate artificial intelligence for the board game ThreeChess. ThreeChess is a variation of chess played on a special board between three players, with the colours Blue, Green and Red. Each player takes turns moving their pieces, where the available moves depend on the type of piece. If your piece lands on a square occupied by an opponent's piece, the opponent's piece is removed from the board (captured); the goal is to capture one of your opponents' Kings. When a King is captured the game ends; the person who took the King is the winner, the person who lost the King is the loser, and the third player neither wins nor loses.



Rules of ThreeChess

The official rules and basic hints are available here, as well as an online game you can play (it seems a bit buggy). The project works with a variation of the rules which is described below.

The initial board position is as shown in the picture. Players take turns to play, moving one piece per turn. Blue always goes first, followed by Green, and then Red.

The piece moves are as follows.

> ♚ King
> The King may move one square in any direction, including diagonally.
>
> ♛ Queen
> The Queen may move in a straight line in any direction (including diagonally), as many squares as are free; but she may not move through another piece.
>
> ♜ Rook
> The Rook may move in a straight line forwards, backwards, left or right, but not diagonally; and it may not move through another piece.
>
> ♝ Bishop
> The Bishop may move in a straight line in any diagonal direction; and it may not move through another piece.
>
> ♞ Knight
> The Knight moves in a L-shape (one square forwards, backwards, left, or right, and then 2 squares in any perpendicular direction). The Knight may jump over other pieces.
>
> ♟ Pawn
> The Pawn may only move 1 square forward at a time, with the following exceptions: on its first move, it may move two squares forwards; to capture a piece it *must* move diagonally forwards; and if it reaches the back rank (the edge of the board), it is automatically promoted to a Queen (unlike normal chess, where the player can choose which piece to promote to). There is no *en passant* in this version of the game.

If the King and the Rook are in their original positions, and there are no pieces between them, the King may move two squares towards the Rook, and the rook is moved to the square that was crossed by the King. This is called *castling*. Unlike normal chess you can castle across, out of, or into check, and it does not matter if the King and the Rook have moved before.

If your piece lands on an opponent's piece, that piece is captured, and it is removed from the board. You may not capture your own piece.

The game ends when a King is captured, in which case the person who took the King is the winner (+1), the person who lost the King is the loser (-1), and the third player neither wins nor loses (0).

When a piece is threatening an opponent's King, the King is said to be in *check*. Unlike normal chess, you can move into check, and you are not required to take evasive action if you are in check. The King must be captured for the game to end; it is not enough to trap a King in checkmate.

It is possible for a game to reach a situation where the same position is repeated and no player can force the end of the game. In this case the game is a draw.

In a timed version of the game, each player is given an amount of time that accumulates while they are considering their move (i.e. from the time their opponent to the right completes a move, until they move a piece). Once this time reaches a limit, the player who ran out of time is the loser. In this case, the opponent with the most remaining pieces, and who took the most pieces is the winner.

Requirements

You are required to research, implement and validate agents to play ThreeChess. You are provided with a Java interface to implement an agent, some very basic agents, and a basic class to run a game. These are available on github and will be regularly updated. Various files are provided for your agents to use, including the following.

- *Board.java* gives a representation of the current state of a game.
- *ThreeChessDisplay.java* generates a JFrame displaying the board.
- *ThreeChess* contains methods for running games and tournaments between different agents.
- *Agent.java* is an abstract class that you must subclass and provide logic for the agent to play moves given a board position. To make running tournaments easier, your agent must be in the package threeChess.agents, it must have the name *Agent########.java* (where the hashes are your student number), and it must have a zero-parameter constructor.

There are a range of other classes as well, but those are the important ones. To clone the repository, use

```
git clone https://github.com/drtnf/threeChess.git
```

To compile the files, in the root directory use

```
javac -d bin src/threeChess/\*.java src src/threeChess/agents/\*.java
```

To run a basic game, use

```
java -cp bin/ threeChess.ThreeChess
```

These files will be updated regularly, so you should regularly pull from the repository. Students are also welcome to push changes, when they find bugs or potential improvements. Documentation is available here.

Submission

You are required to submit a research report (1,500-2,000 words), and Java source code for one or two agents (pairs must submit two agents). The report should include:

- A literature review of suitable techniques: 20%
- Description and rationale of selected techniques: 20%
- Description of validation tests and metrics: 15%
- Analysis of agent performance: 15%

The source code will be assessed on:

- The quality of the code, including formatting and selection of data structures: 15%
- The performance of the agent, including in the end-of-semester tournament: 15%

The assessment criteria are given in the document above. There will be a tournament in Week 12, with the rules finalised closer to the date. The tournament will involve agents playing a series of games, with high-scoring agents awarded bonus marks.

Thus your submission should include

1. **A PDF version of your report.**
2. **A ZIP of all the source files, configuration files, and instructions to execute your code (not a RAR!).**
3. **A single source file Agent########.java (i.e. Agent[your student number]) for the agent you would like to compete in the tournament. This should not be zipped. The Agent file should be part of the threeChess.agents package.**

Getting started

There are many references for Chess AI and for perfect-information deterministic turn-based board games in general. ThreeChess adds the complication of a third player. This makes techniques such as minimax and alpha-beta pruning less practical, since it assumes the worst of your opponents, and it is unlikely both opponents will target you (and if they do, there's not much you can do about it!) The first step in designing an AI for the game is to play some games yourself and identify useful tactics. Once you have an idea of what makes a good position in the game, you can start work on an evaluation function, to give an approximate utility to a board position. You can then use that in a minimax search, a Monte Carlo tree search or a sequential decision problem. Some interesting references are included below, and we will add to this throughout the project period.

- A paper introducing Monte Carlo tree search.
- A very good project report from a student in 2016.
- If you haven't found it yet, the Wikipedia page on ThreeChess might be a good starting point for your researches.