Program Architecture

This is a Tic Tac Toe game that is written in Java 1.7.0_10. It implements two different ai's. One is a thoughtful ai that makes intelligent moves and the other is a naïve ai that will occasionally make bad moves. The program is divided up into 7 classes:

1. TicTacToe: This is the main class that handles command line arguments and starts the game.
2. Engine: This class runs the game by calling the getMove() method from the Player subclasses, keeps track of the board, and checks for end of game conditions.
3. Node: This class represents a single square on the game board and is mainly used for display purposes.
4. Player: This class is an interface for the different types of players so that getting the move from the players can be a generic call.
5. Human: This class is used for a human playing the game and handles command line input and illegal move choices.
6. Thoughtful: This class runs the thoughtful ai (discussed below).
7. Naive: This class runs the naïve ai (discussed below).

Thoughtful AI

The thoughtful ai uses two rules in the tic tac toe game. The first rule it checks is if it can get three spaces in a row so that is can win the game. This is done by representing each row, column, and diagonal as a string of 0's for free spaces, 1's for player 1 and 2's for player 2, and matching it to a winning string of the players number. Below is a trace from a game where the agent is player 1 and finds a winning move in the third column:

Looking for winning move.
Checking 221
Checking 010
Checking 201
Checking 202
Checking 210
Checking 101
Winning move found in third column.

Since the ai is player 1 it is looking for one of three winning strings: 110, 101, or 011, and it finds one in the third column. The second rule thoughtful ai checks is if the opponent can get three spaces in a row so that it will win the game. Since it is illegal to pick a square that has already been chosen this will stop the opponent from winning. The ai does this in the same way it checks for its own winning move but this time it looks for a pattern matching the opponents player number, 2, which can be seen in the trace below:

Looking for opponent winning move.
Checking 201
Checking 010
Checking 200
Checking 202
Opponent winning move found in first column.

Since the opponent is player 2 it finds the pattern in the first column and picks the opponents winning move so that the opponent cannot and therefore does not win. If the thoughtful ai finds that it cannot make a winning move and the opponent cannot make a winning move it will make a strategic move.

Once it has decided what kind of move it is going to make it will search its knowledge base to find the correct move to use. The knowledge base is sorted into winning, blocking, and strategic moves. The knowledge base is represented as strings in a text file. Each state is saved on three lines:

1. they type of knowledge represented
2. the game board
3. the optimal move

When the ai is sorting through the knowledge base it will only look at the saved board if it is the same type of knowledge as it is looking for (ex: if it is looking for a winning move it will skip the knowledge about blocking or strategy). The game board is represented as a 9-character string where free spaces are 0's, opponent moves are !'s and the players moves are ~'s. The optimal move is a number related to the index on the board of the optimal move where the board is represented with numbers 0-8. Below is an example board and how it would be saved in memory if the player was X's:

```
  | X |O
--------
  | X |O
---------
X|   |
```

```
w
0~!0~!~00
7
```

The w is for a winning move because if the ai selects position 7 it will win the game.

Naive AI

The naïve ai uses the same exact technique as thoughtful ai. It has the same methods for finding its own winning moves and for finding the opponent's winning moves. It also uses the same knowledge base as thoughtful ai. The difference between naïve and thoughtful is that naïve has a chance to not look for good moves. Even naïve players of tic tac toe know the rules, but sometimes they either don't see a good move or forget to check for an opponent's good move. Naive ai has a random chance to not check for a winning move, an opponent's winning move, or a strategic move. If it can make a winning move, but does not check for a winning move, then it will not find the current board in the knowledge base because it would be under the winning move type of knowledge. It will then choose randomly. This goes the same for not checking for opponent winning moves or strategic moves. Below is an example of not checking for an opponent's (X's) winning move:

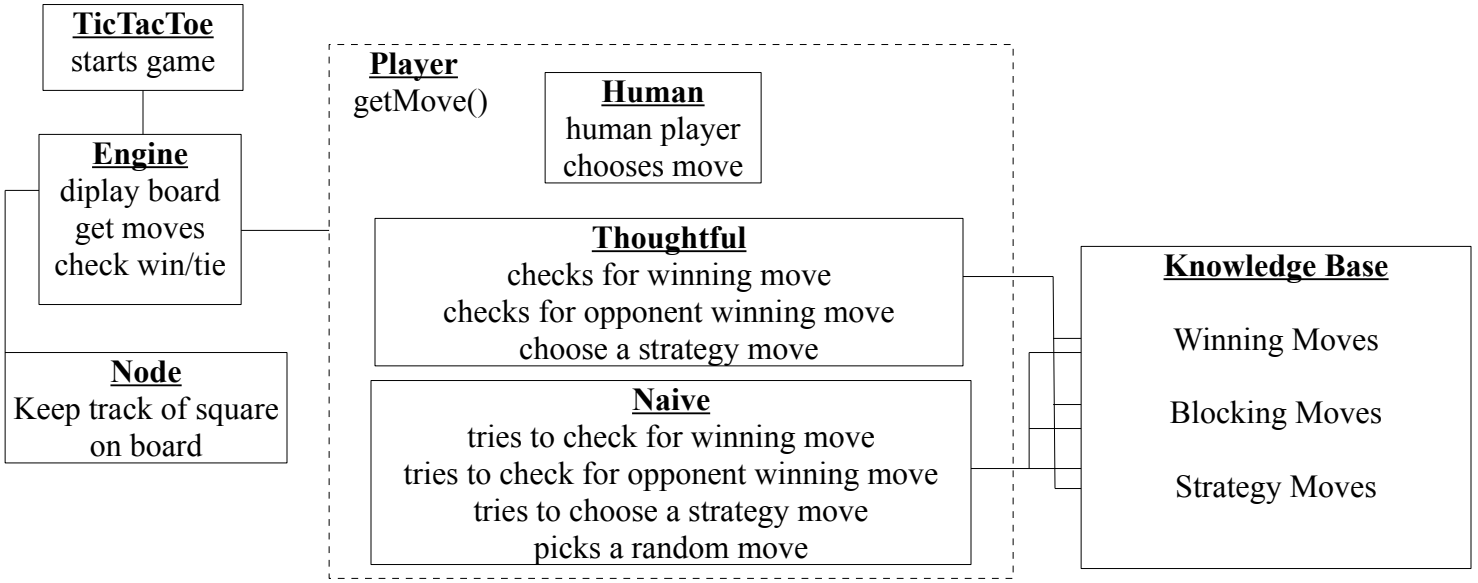
```
  | O | X
-----------
  | X |
-----------
 O |   | X
```

```
Looking for winning move.
Checking 021
Checking 010
Checking 201
Checking 002
Checking 210
```

Checking 101
Checking 011
Checking 112
Picking a strategy move.

```
 O | O | X
-----------
   | X |
-----------
 O |   | X
```

Normally the ai would look for an opponent's winning move after looking for its own winning move, but in this case it “forgot” to check and picked a random move instead.



Thoughtful vs. Naive Experiment

This experiment was run with thoughtful ai as player 1 and naïve ai as player 2. In this experiment thoughtful ai won 75% of the time and tied the other 25%. Whenever the thoughtful ai won it was because the naïve ai failed to check for a blocking move and lost or failed to pick a strategy move and was caught in a no win situation like the one below:

```
O |   |
-----------
  | X | O
-------------
X |   | X
```

In this situation O's cannot block both of X's winning moves so X will win. In the games that ended up as a tie the naïve ai either did not fail to check for a type of move, failed to pick a good strategy move but recovered later, or picked a random move that happened to be the best move. This was what I was expecting the experiment to look like. The naïve ai misses looking for an opponent's winning move almost every game and the thoughtful ai shouldn't make any mistakes so the thoughtful ai should be winning most of the games it plays. Below is an example of the experiment being run, my comments are inside of the parenthesis.

Thoughtful(X's) vs. Naive(O's):

```
   |  |
-----------
   |  |
-----------
   |  |
```

Looking for winning move.
Checking 000 (this is the first row)
Checking 000 (second row)
Checking 000 (third row)
Checking 000 (first column)
Checking 000 (second column)
Checking 000 (third column)
Checking 000 (diagonal left to right)
Checking 000 (diagonal right to left)
Looking for opponent winning move.
Checking 000
Checking 000
Checking 000
Checking 000
Checking 000
Checking 000
Checking 000
Checking 000
Checking 000
Picking a strategy move.
(The thoughtful agent cannot pick a winning move and the opponent cannot win on the next move so it chooses a strategy move.)

```
   |   |
-----------
   | X |
-----------
   |   |
```

Looking for winning move.
Checking 000
Checking 010
Checking 000
Checking 000
Checking 010
Checking 000
Checking 010
Checking 010
Looking for opponent winning move.
Checking 000
Checking 010
Checking 000
Checking 000
Checking 010
Checking 000
Checking 010
Checking 010
Picking a strategy move.
(Naive does not see a winning move or an opponent winning move so it chooses a strategy move.)

```
   |   |
-----------
   | X |
-----------
 O |   |
```

Looking for winning move.
Checking 000
Checking 010
Checking 200
Checking 002
Checking 010
Checking 000
Checking 010
Checking 012
Looking for opponent winning move.
Checking 000
Checking 010
Checking 200

Checking 002
Checking 010
Checking 000
Checking 010
Checking 012
Picking a strategy move.
(There is no winning move for the player or the opponent so a strategy move is chosen.)

```
  |   | X
-----------
  | X |
-----------
 O |   |
```

Looking for winning move.
Checking 001
Checking 010
Checking 200
Checking 002
Checking 010
Checking 100
Checking 010
Checking 112
Looking for opponent winning move.
Checking 001
Checking 010
Checking 200
Checking 002
Checking 010
Checking 100
Checking 010
Checking 112
Picking a random move.
(There is no winning move for the player or the opponent and it does not look for a strategy move so it chooses randomly.)

```
  | O | X
-----------
  | X |
-----------
 O |   |
```

Looking for winning move.
Checking 021
Checking 010
Checking 200
Checking 002
Checking 210
Checking 100

Checking 010
Checking 112
Looking for opponent winning move.
Checking 021
Checking 010
Checking 200
Checking 002
Checking 210
Checking 100
Checking 010
Checking 112
Picking a strategy move.
(There is no winning move for the player or the opponent so a strategy move is chosen.)

```
  | O | X
-----------
  | X |
-----------
 O |   | X
```

Looking for winning move.
Checking 021
Checking 010
Checking 201
Checking 002
Checking 210
Checking 101
Checking 011
Checking 112
Picking a strategy move.
(It does not see a winning move and it does not look for an opponent winning move so it misses a blocking move. It attempts to pick a strategy move but this situation would be in the blocking category in the knowledge base, so it chooses randomly.)

```
 O | O | X
-----------
  | X |
-----------
 O |   | X
```

Looking for winning move.
Checking 221
Checking 010
Checking 201
Checking 202
Checking 210
Checking 101
Winning move found in third column.
Picking a winning move.

(It looks for a winning move and finds one in the third column so it picks the move and wins.)

////////////////////////////////
Player 1 wins!

```
 O | O | X
-----------
   | X | X
-----------
 O |   | X
```

In project one the difference between thoughtful and naïve was that naïve had incorrect and incomplete data in its knowledge base. In this experiment naïve had access to the exact same knowledge base as the thoughtful ai. The difference between the two in this project was how they followed the rules of the game. Thoughtful ai followed the two rules it was given every single turn while naïve ai would sometimes “forget” to follow the rules and choose randomly. The way the domain knowledge, or rules in this case, were factored out caused the ai agents to “think” about the game differently from project one as well. In project one they would only try to find the current board in the entire knowledge base without thinking about rules because the rules were hidden in the suggested moves. In this project the knowledge base was sorted by the rule it followed so the ai agents looked for boards that were related to the current rule it was trying to follow.