Project: Baroque Chess Agent

Iqra Imtiaz
iqra0908@uw.edu
Role:

- Move generator
- Alpha-beta pruning

Matthew Vaughn
mpvaughn@uw.edu
Role:

- Capturing rules
- Personality of agent

Combined efforts:

- Static evaluation
- Testing
- Dummy agent player

Purpose

The program is an agent designed to play the Baroque variant of chess.

Description

Alpha-beta pruning is used to select the best move from all the legal moves generated by a move generator. The capture results are used for each legal move to create a resulting state in successors. The first static evaluation written turned out to be expensive on time as it was using move generator. It had the following formula being used and is named expensive_static_eval in the code.

$$200 * (k - k') + 9 * (w - w') + 5 * (c - c' + f - f') + 3 * (l - l' + i - i') + 1 * (p - p') + 0.1 * (m - m') - 0.5 * (b - b')$$

Here all the pieces with (') describes the enemy variable. The description of each variable if as below;

- k = no. of king
- w = no. of withdrawer
- c = no. of coordinator
- f = no. of freezer
- l = no. of leapers
- i = no. of imitators
- p = no. of pincers
- b = no. of blocked pincers
- m = total no. of legal moves available to this agent

The cheaper designed static evaluation has the following formula;

$$200 * (k - k') + 9 * (w - w') + 5 * (c - c' + f - f') + 3 * (l - l' + i - i') + 1 * (p - p') + cp * (b - b') + 0.25 * (r - r')$$

In this formula rest of the variables have same definition as before except for the following;

- b = pincers more than 2 per row
- cp = crowd pincer penalty
- r = no. of this agent's pieces per row

this static evaluation formula allowed our agent to perform better in less time. We also implemented iterative deepening to look for more moves ahead. In one of the games with a dummy player, this static evaluation was able to reach up to 40 levels deep.

Sample session

Start of the game:

```
**** Baroque Chess Gamemaster v0.8-BETA *****
The Gamemaster says, "Players, introduce yourselves."
 (Playing WHITE:) I'm Bob, a Baroque Chess newbie and buff of Boba beverages.
              I was created by the following human boba:
              Iqra Imtiaz, iqra0908@uw.edu and Matt Vaughn, mpvaughn@uw.edu

 (Playing BLACK:) I'm Newman Barry, a newbie Baroque Chess agent.

The Gamemaster says, "Let's Play!"

The initial state is...
c l i w k i l f
p p p p p p p p
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
P P P P P P P P
F L I W K I L C
WHITE's move

Time used in makeMove: 0.2988 seconds out of  0.5
WHITE's move: the P at (6, 1) to (4, 1).
Turn 1: Move is by WHITE
Boba Boy Bob says: I feel like one of those bursting boba that's just about to pop!
c l i w k i l f
p p p p p p p p
- - - - - - - -
- - - - - - - -
- - - - - - - -
P - - - - - - -
- P P P P P P P
F L I W K I L C
BLACK's move
```

```
Time used in makeMove: 0.0000 seconds out of  0.5
BLACK's move: the p at (1, 5) to (3, 5).
Turn 2: Move is by BLACK
Newman says: I'll think harder in some future game. Here's my move
c l i w k i l f
p p p p p - p p
- - - - - - - -
- - - - - p - -
- - - - - - - -
P - - - - - - -
- P P P P P P P
F L I W K I L C
WHITE's move

Time used in makeMove: 0.2998 seconds out of  0.5
WHITE's move: the P at (6, 1) to (4, 1).
Turn 3: Move is by WHITE
Boba Boy Bob says: Stop! You're BERRYing me!
c l i w k i l f
p p p p p - p p
- - - - - - - -
- - - - - p - -
- P - - - - - -
P - - - - - - -
- - P P P P P P
F L I W K I L C
BLACK's move

Time used in makeMove: 0.0000 seconds out of  0.5
BLACK's move: the p at (1, 4) to (1, 5).
Turn 4: Move is by BLACK
Newman says: I'll think harder in some future game. Here's my move
c l i w k i l f
p p p p - p p p
- - - - - - - -
- - - - - p - -
- P - - - - - -
P - - - - - - -
- - P P P P P P
F L I W K I L C
WHITE's move
```

Middle of the game:

```
Time used in makeMove: 0.0000 seconds out of  0.5
BLACK's move: the p at (1, 2) to (3, 2).
Turn 18: Move is by BLACK
Newman says: I'll think harder in some future game. Here's my move
c l i - k i - f
- p - p p p P -
p - - - - - - P
- - p - - - - -
- P - - - - - -
P - - - - - - -
- - - P P P P -
F L I W K I L C
WHITE's move

Time used in makeMove: 0.2988 seconds out of  0.5
WHITE's move: the P at (2, 7) to (2, 2).
Turn 19: Move is by WHITE
Boba Boy Bob says: THAI(M) for you to give up! I'm too far ahead!
c l i - k i - f
- p - p p p P -
p - P - - - - -
- - - - - - - -
- P - - - - - -
P - - - - - - -
- - - P P P P -
F L I W K I L C
BLACK's move

Time used in makeMove: 0.0000 seconds out of  0.5
BLACK's move: the p at (1, 1) to (1, 2).
Turn 20: Move is by BLACK
Newman says: I'll think harder in some future game. Here's my move
c l i - k i - f
- - p p p p P -
p - P - - - - -
- - - - - - - -
- P - - - - - -
P - - - - - - -
- - - P P P P -
F L I W K I L C
WHITE's move
```

End of the game:

```
Boba Boy Bob says: This position is the best! Just like my favorite boba, Mango Green Tea!
- l i - k i - f
- P - - P - P -
- - P - - - - -
- - - - - - - -
- - - - - - - -
P - - - - - - -
- - - P - P P -
F L I W K I L C
BLACK's move

Time used in makeMove: 0.0000 seconds out of  0.5
BLACK's move: the k at (0, 4) to (1, 3).
Turn 28: Move is by BLACK
Newman says: I'll think harder in some future game. Here's my move
- l i - - i - f
- P - k P - P -
- - P - - - - -
- - - - - - - -
- - - - - - - -
P - - - - - - -
- - - P - P P -
F L I W K I L C
WHITE's move

Time used in makeMove: 0.3058 seconds out of  0.5
WHITE's move: the P at (1, 1) to (1, 2).
Turn 29: Move is by WHITE
Boba Boy Bob says: Move GINGERly! I have the advantage here!
- l i - - i - f
P - - - P - P -
- - P - - - - -
- - - - - - - -
- - - - - - - -
P - - - - - - -
- - - P - P P -
F L I W K I L C
BLACK's move

Win for WHITE
Game over.
```

```
Congratulations to the winner: WHITE
```

Demo instructions

We would suggest playing our agent against a dumb player Newman we have provided. Newman just randomly choose a move from all the legal moves possible. The following line of code should run a game between our agent and the dumb agent.

```
py BaroqueGamemaster.py Bobby_Boba_BC_PLAYER Newman_BC_PLAYER
```

Code excerpts

The following excerpt is from the move generator. It checks if the piece we are looking at is our imitator and look for all the adjacent enemy pieces. Then, it runs a loop on all the enemy pieces to generate legal moves from the imitator's location, the way any other friendly piece would have generated. This code is within the main loop of the board and the method getMoves generate all the legal moves possible for some piece x from location (i,j) on the current board.

```
1.  if enemyPieces[freezer] not in adjacentPieces: #freeze if enemy's freezer adjacent
2.      if mark == friendlyPieces[imitator]: #Imitator
3.          for x in adjacentPieces:
4.              moves += getMoves(currentState, i,j,x+1,adjacentLocations, friendlyPieces,
    enemyPieces)
5.      else:
6.          moves = getMoves(currentState, i,j,mark,adjacentLocations, friendlyPieces, enem
    yPieces)
```

The following excerpt is from alpha beta pruning. The algorithm is similar to toro tile agent we created in homework 4 but here we are returning the state itself along with the best move found and the provisional value. The reason for this is that capture results are already being found in the successors function where we create new states and to take a turn we need both the move made, and the resulting captures of that move. So directly returning the state when best move was found through alpha-beta seemed more efficient.

```
1.  if whoseMove == myPlayer:
2.      provisional = -100000
3.  else:
4.      provisional = 100000
5.  best_move = False
6.  newState = state
7.  for s in successors(state, whoseMove):
8.      (lastMove, lastState, newVal) = alpha_beta_pruning(s[0], alpha, beta, other(whoseMo
    ve), plyLeft-1)
```

The excerpt above is from static evaluation function. The first two lines show how the no. of leapers and imitators of friendly pieces are calculated by comparing it against the list of friendly pieces we have. The if condition next shows how pincers are counted as total and per each row if they are more than 2 in number. That's being done to apply penalty on the states where pincers get crowded.

```
1.  L += board[i][j]==myPieces[leaper]
2.  I += board[i][j]==myPieces[imitator]
3.
4.  if board[i][j] == myPieces[pincer]:
5.      pincer_count += 1
6.      P += 1
7.      if pincer_count >= 2:
8.          bad_pincer_count += 1
```

Learning

Matt:

I learned how to play Baroque chess. I also learned about the difficulty of enumerating over a state space. While it appears simple in lecture (e.g. start with a given state, use the transition functions to generate new states), modeling the game itself becomes an interesting part of the problem. The model of the problem impacts how the user approaches solving the problem.

Iqra:

I learnt how to think about complex static evaluations and build a smarter agent through that. Generally, I learnt how to create game engines from scratch. Writing move generator was tiring and time consuming but it taught me how to think about calculations of moves made in the games.

Extra features given more time

Zobrist hashing. Given that we used IDDFS, Zobrist hashing would significantly reduce the amount of static evaluations run because IDDFS runs over some of the same states multiple times. By using Zobrist hashing we would likely reduce the number of static evaluation function calls needed by a fairly large factor.

The Jamboree algorithm. I learned about this from some friends taking CSE 332. It's a twist on minimax/alpha beta pruning where the user decides that it's unlikely that any remaining states which are going to be generated will be pruned by alpha-beta pruning, so they handle each of the remaining states on this level with a parallel algorithm which can use as many cores as the computer has available for the remainder of the states to be enumerated from the current state.

Citations

[1] Baroque Chess, by Chad W Smith;
URL: https://www.scribd.com/document/312467727/Baroque-Chess (accessed March. 01, 2019)

[2] Ultima - The Chess Variant Pages,
available online at: https://www.chessvariants.com/other.dir/ultima.html

From the first citation above, we learnt the moves each piece can make. That helped in creating move generator for our chess agent. The second citation was used to learn about the capturing rules of the game. The animations on that website were specifically helpful to learn everything fast specially for the leaper.

Partner reflections:

Matthew Vaughn

Main role:

My largest contribution in terms of number of lines of code written was modeling how the pieces capture enemy pieces. I also wrote some tests for move generation, contributed some ideas to the static evaluation function and helped with write the successors function to generate new moves. Except when

I was sick (and worked exclusively on capture() stuff), we worked on the project together in person, so we both had a hand in all of the ideas contributed to essentially the entire project.

Benefits of the partnership:

It helped a lot to talk through how the Baroque Chess moves work with Iqra. One of the biggest difficulties of the project was just learning how the game works, so it was hugely useful to be able to ask if a piece moves/captures the way you think it moves/captures. Additionally, discussing our plans for the agent was useful as well. It helped to talk through our plans for some aspect of the agent.

Challenges of the partnership:

Coordination is tough. It was always hard to know what the other person was working on, even if we were in the same room. This was mostly a problem while Iqra worked on move generation and I didn't have a ton to do, but I made the most of my time and wrote some tests modeling how the move generation should be expected to work. Overall being on a team probably did save me a certain amount of time coding anyways.

Iqra Imtiaz

Main role:

My largest contribution in the project was to write a move generator. Besides that, we each wrote a static evaluation and choose the better function timewise. We mostly worked on the project together so almost all the sections were completed together in person.

Benefits of the partnership:

This was my first experience in pair programming, and it was pretty useful as we overcame most of the issues we encountered, by brainstorming together and discussing about it. This was specifically useful as I have experience from my previous projects when I stay stuck on something for a day and suddenly something clicks. But working together on this project made that process really fast, as both of us though of different issues that could be at fault and that gave me different ideas to tackle to issues really fast.

Challenges of partnership:

The only challenge I faced was pulling new code for GitHub specially when we were working together on something. But the issues were solved quickly as Matt helped me figure out how to merge code from local machine to the github.