

Baroque Chess Agents

Baroque GO

1. Title: Baroque Chess Agents

2. Names and Roles:

Erik Huang (huangti):

Implemented Zobrist Hashing, how to account for the best move for each state.

Tuning minimax algorithm and alpha-beta pruning to fit into this project.

Minghao Liu (lmh98):

Implemented generation of move list for each piece, rules for capturing and moving, making move functions and some useful helper functions.

We work together in debugging and constantly optimizing our code to make it shorter, more readable and manageable.

3. The program, Baroque Go, supposed to play the Baroque Chess game as either White Chess or Black Chess with its best move picked using results from static evaluation of all possible states. The program has a feature of using Zobrist Hashing. Basically it puts each state with its corresponding hash key generated by a zhash function into a hash table to optimize the Alpha-Beta pruning in terms of efficiency and the depth of IDDFS.

4. Our chess agent Baroque Go will generate a hash table to store Zobrist keys each game. For each move, the chess agent generates all possible moves from the current state as a dictionary that has a move as a key and the list of captured pieces as values. Then, from this dictionary, we construct new states from the

current state and pass them into the static evaluation function. While we are getting the evaluated value for each state, the program also uses either minimax or alpha-beta pruning to compare and find the best state. While the program is running its alpha-beta pruning inside the make move function, it also follows a time constraint, that is, if the program realizes that it has gone past the time-limit given, it will instantly quit searching and return the current best move. This is achieved by IDDFS.

Our agent's custom static evaluation would generate the value of each state by adding the potentially vulnerable opponent chess pieces with its corresponding value and minus the ally piece that can potentially be captured by opponent. To do this we simply look around the current piece to see if there's any threats nearby. We highly evaluate kings and freezer in our game because we found them important to the result of the game. As a result, our agent is very aggressive when using its custom static evaluation function.

5.

Below is a screenshot of my agent playing with "Pro", as soon as he moved his freezer forward against my coordinator, Baroque GO decides to bring back the pawn behind him and capture the freezer.

```

Time used in makeMove: 1.9999 seconds out of 2
WHITE's move: the F at (2, 1) to (1, 0).
Calling the move validation service.
valid move
valid move
Turn 9: Move is by WHITE
Pro says: i move freezer, andoh, you look very strong
c l - - k i l f
F p p - p p p p
- - - - -
- - - w - - -
p - - p - - i -
P - - - - -
- P P P P P P P
- L I W K I L C
BLACK's move

playing as: black
=====
Report of Zobrist Hashing Status for BaroqueG0:
PUT COUNT: 14941
Success GET COUNT: 570
Failed GET COUNT: 14573
Game Collision COUNT: 368
Saved Static Eval COUNT: 14941
Max play reached: 3
=====
Number of States Expanded: 2649
Number of Cut Offs: 2272
=====
Time used in makeMove: 1.9918 seconds out of 2
BLACK's move: the p at (4, 0) to (2, 0).
Calling the move validation service.
valid move
valid move
Turn 10: Move is by BLACK
BaroqueGo says: Here's my move, I think this is an obvious move
c l - - k i l f
- p p - p p p p
p - - - - -
- - - w - - -
- - - p - - i -
P - - - - -
- P P P P P P P
- L I W K I L C
WHITE's move

```

6. Instruction for demoing:

I prefer to have the game master setting changed as this:

TIME_PER_MOVE = 4

TURN_LIMIT = 30


I also feel like that the simple static evaluation function is more conservative,

which is good against agents like Pro, so that I set “using custom evaluation

function” to False.

I also had a printing section that displays the Zobrist hashing status, which can be disabled if we just want to watch the game.

7. Below is a screenshot of a code segment that we are proud of:

A screenshot of a code editor showing two Python functions for generating chess moves. The first function, `getNonDiagonalMoves`, iterates over four non-diagonal directions. The second function, `getDiagonalMoves`, iterates over four diagonal directions. Both functions use a helper `getIndicesInDirection` to find valid squares and `getCaptured` to check for pieces. The code is color-coded with syntax highlighting.

```
# Check non diagonal directions
def getNonDiagonalMoves(board, row, col, myKing):
    moveList = {}
    for direction in moveDirections[:4]:
        for rc in getIndicesInDirection(row,col,BOARD_SIZE,direction):
            if board[rc[0]][rc[1]] == 0:
                captured = getCaptured(board, row, col, rc[0], rc[1], direction, myKing)
                moveList[((row, col), (rc[0], rc[1]))] = captured
            else:
                break
    return moveList

# Check diagonal directions
def getDiagonalMoves(board, row, col, myKing):
    moveList = {}
    for direction in moveDirections[4:]:
        for rc in getIndicesInDirection(row,col,BOARD_SIZE,direction):
            if board[rc[0]][rc[1]] == 0:
                captured = getCaptured(board, row, col, rc[0], rc[1], direction, myKing)
                moveList[((row, col), (rc[0], rc[1]))] = captured
            else:
                break
    return moveList
```

These 2 helper functions can help generate the diagonal and non-diagonal move list for a given piece, I think it looks neat and concise thanks to our helper function that outputs arrays of indices in 8 directions.

8.

This is the first time we implement Zobrist Hashing, and we both have a deeper understanding of why we need it and how much it can improve our program.

We also learned a lot about this new type of chess game, Baroque chess. Its

rules are really interesting and inspiring.

9.

If we have more time, we would definitely make our custom static evaluation function stronger so that our agent has better strategies. I also would like to make the make move function more efficient so that we can explore more states for each move.

10.

Reference:

1). *Wikipedia – Baroque Chess*

https://en.wikipedia.org/wiki/Baroque_chess

2). *Comparative study of performance of parallel Alpha Beta Pruning for different architectures*

<https://arxiv.org/pdf/1908.11660.pdf>

From the Wikipedia source, we found the basic rules of the game and was able to make all the move list functions and capturing functions based on that.

From the paper, I read it to gain a better understanding of how alpha-beta pruning can be applied to multiple types of board games and its limitations.

11. Partners' Reflections

Minghao Liu:

My partner is Erik Huang who is mainly in charge of the project. Our partnership has worked well through the entire project with collaboration and job distribution. With team working, he helped me to optimize multiple lines of codes including the helper method to generate the moving list. He also helped me with few bugs that are hard to fix. The major challenge is that without meeting in person, it is hard for us to combines our codes together and work efficiently.

Erik Huang:

Mark and I had worked together for a really long time so that we know each other very well. I often do pair programming with him so that I can have someone who points out my minor mistakes and inspire me with great ideas. Without him being around, I wouldn't be able to come up with the helper function for the move list. We certainly faced some challenges as well, during the time when we couldn't meet with each other, I wrote many lines of code by myself without testing, which results in a number of minor issues that took forever to debug.

OPTION 1 Report and Thoughts About Zobrist Hashing

Zobrist Hashing is implemented within our minimax and alpha-beta pruning algorithm to increase efficiency. Each state has its own hash key generated by a zhash function. We use the put function every time when we discover a new state never seen before. We store each state within a tuple that contains hash value, play used, evaluation value and number of static evaluations in our hash table. The main improvement of Zobrist hashing is that if we see a state that already exist in our table, we can just retrieve the evaluation value for that state and skip the evaluation process. Every time this process happens, we can save lots of time in searching for new states, especially if we have a very time-consuming static evaluation function and a long time limit to run our IDDFS.

From our experiments, with a time limit of 40 seconds per move, our algorithm with Zobrist Hashing is able to successfully call get function 24598 times and explore 5179 states for its first move. While the Zobrist Hashing is turned off, the program only expanded 4547 states in its first move. I think that the improvement would be more significant if our static evaluation function is more complicated.

Note that since this is the first move, no collisions will present.

=====

Report of Zobrist Hashing Status for BaroqueGO:

PUT COUNT: 54253

Success GET COUNT: 24598

Failed GET COUNT: 54253

Game Collision COUNT: 0

Saved Static Eval COUNT: 54253

Max play reached: 4

=====

Number of States Expanded: 5179

Number of Cut Offs: 4207

Without using Zobrist Hashing to improve efficiency:

=====

Report of Zobrist Hashing Status for BaroqueGO:

PUT COUNT: 68711

Success GET COUNT: 0

Failed GET COUNT: 68711

Game Collision COUNT: 0

Saved Static Eval COUNT: 68711

Max play reached: 4

=====

Number of States Expanded: 4547

Number of Cut Offs: 3712

=====

During our collaboration, the first obstacle between us is to communicate and persuade each other about our own ideas while implementing the move list. We are both not good at explaining why our own implementation is better so that we spent lots of time struggling over the structure. The second challenge for us is to merge each other's work after working separately. We found many bugs and had a hard time debugging them because we were not familiar with the code written by the other partner.

Individual Statement:

Erik Huang:

It's a pleasure to work with Mark. He is very creative and strategic while we are brainstorming ideas and constructing the program. He also motivated me a lot in this project.

Minghao Liu:

Erik has a focused and clear mind while coding, he solved many issues in a very short amount of time and makes remarkable progress every time we work together. I enjoyed working with him.