1. **title of project**

OPTION 2: Game Playing Techniques

2. **names and roles of each teammate**

Jingqiang Wang: Designed Zobrist Hash and implemented it in the original code.
Junhao Zeng: Created the static evaluation function and the movement generator.

3. **what the program is supposed to do**

This program is supposed to be able to play a game of Baroque Chess smartly by using some AI techniques.

4. **technique used and brief description (half a page) of how that technique works. If you use multiple AI techniques then describe each one but with somewhat less detail for each one**

Minmax Search:
It is function that is used to figure out the best state which could be reached by the current player. One of the players will try to make the value of state as high as possible and the other one will try to decrease it. Based on this assumption, we could find the final state that it will lead to. And we actually used an IDDFS way to find the values of states, which means it will try to find the states values by different depth iteratively.

Alpha-beta pruning:
During the minimax search, many states are explored unnecessarily. Alpha-beta pruning is a technology to optimize the minimax search. Due to the decisions made by the maximizer of the game is always trying to choose the state with higher value in the end and minimizer will choose decision that lead to lower state value, states between them are unworthy to explore. Therefore, process to explore those states could be cut off, which saves time to explore more states.

Zobrist hashing:
Zobrist hashing is used to store the values of states that we have calculated in a more efficient way with less collisions. The idea is creating an 8×8 table, in which each cell is a dictionary, and stored 2 random number. And for the first time we calculate a state value, it will be stored in the dictionary according to the hash value of this state. The next time we meet this state, we will get the state value from the hash dictionary instead of calculating it again. We also try implement another way to hash the state. The difference is for each cell in the pre-generated table, we store a few random numbers according to the piece placed here.

5. either a screen shot or a transcript of an interesting sample session

```
WHITE's move

The Four value:
Number of states_EXPANDED 207
Number of states EVALS and FOUND 4520
Number of states get from the hash table  1122
Number of CUTOFFS  165
ply:
3
Back up state value 52
Time used in makeMove: 1.0462 seconds out of  1.5
WHITE's move: the L at (7, 6) to (3, 6).
Calling the move validation service.
valid move
valid move
Turn 13: Move is by WHITE
Thanos says: You are weak.
c l i w k - l f
p p p p p i p -
- P - - - F - -
- - - - - - L -
- - P - - P - -
- - - - - - - -
P - - P - P - P
- L I W K I - C
BLACK's move
```

6. brief demo instructions

In order to play Baroque chess, you need to store BaroqueGameMaster.py and Thanos_BC_Player in a same folder, and then run BaroqueGameMaster.py to begin.

7. code excerpt showing some interesting part(s) of your Python code and some explanation of it

```
def staticEval(state):
    ally_points = 0
    enemy_points = 0
    enemy_king_down = True
    for yditection, row in enumerate(state.board):
        for xdirection, piece in enumerate(row):

            # do not need to do any calculations for empty square
            if piece == 0:
                continue

            # store nearby pieces
            neighbors = surrounding(state, yditection, xdirection, moves)

            # find piece score: you get a better score if you have more pieces
            if piece % 2 == 0:
                enemy_points -= importance['living_reward'] *
advanced_static_value[piece // 2]
            else:
                ally_points += importance['living_reward'] *
advanced_static_value[piece // 2]

            # the more enemies it can freeze, the higher the points
            if piece == 14 or (piece == 8 and 15 in neighbors):
                enemy_points -= importance['freezing_reward'] \
                                * sum([advanced_static_value[x // 2] for x in
neighbors if x % 2 == 1])

            elif piece == 15 or (piece == 9 and 14 in neighbors):
                ally_points += importance['freezing_reward'] \
                               * sum([advanced_static_value[x // 2] for x in
neighbors if x % 2 == 0])

            # the more a pincer can kill in one move, the higher the points
            elif piece == 2:
                enemy_points -= importance['killing_reward'] * pincer_kill(state,
yditection, xdirection)

            elif piece == 3:
                ally_points += importance['killing_reward'] * pincer_kill(state,
yditection, xdirection)

            # check how many allies there are near king
            elif piece == 12:
```

```
                enemy_king_down = False
                enemy_points -= importance['king_guard'] * sum([1 for x in neighbors
if x % 2 == 0])
                enemy_points += importance['king_guard'] * sum([1 for x in neighbors
if x % 2 == 1])

            elif piece == 13:
                ally_points += importance['king_guard'] * sum([1 for x in neighbors
if x % 2 == 1])
                ally_points -= importance['king_guard'] * sum([1 for x in neighbors
if x % 2 == 0])

    total_points = ally_points + enemy_points
    if enemy_king_down:
        return 100000
    return total_points
```

explanation:
Another enhanced techniques used in our project is using the more complicated static evaluation method to evaluate the board.
First, for a specific status of the board, we count how many enemies all pincer can kill in one move, and adding five points for each kill. Vice versa, we count how many allies can be killed by enemy pincer in one move, and deducting five points for each kill.
Besides, because the more allies a king has in his surrounding, the safer the king is, so we check the surroundings of our king, the surrounding means the eight cells which connect to the king cell. For each surrounding cell, if it is an ally, we add five points, if it is an enemy, we deduct five points. Vice versa, for each surrounding cell of enemy king, if it is an ally, we deduct five points, if it is an enemy, we add five points.
What's more, we count how many allies are frozen by enemy's freezer or imitator, for each frozen ally, we deduct one point. Vice versa, we count how many enemy are frozen by our freezer or imitator, for each frozen enemy, we add one point.
In the end, we add or de a specific points for each piece depending on what it is.

8. brief description of what each team member learned in this project

Junhao Zeng:

After this project, I learned how to use python to design complex code. Before this project, I mainly use java, and life becomes so easier after I use python to write code.
Besides, after finished this project, I know how to implement minimax search, alpha-beta pruning and Zobrist hash, and I also know how to write a good static eval function.

Jingqiang Wang:
I learned how to implement Zobrist hash. Zobrist hash is mentioned in course, but the way to implement it is slightly different in python. Besides, implement a method should be careful. We have tried several ways to implement it, but some of them are not really efficient, which will directly affect the number of states it can explore. What is more, sometimes we can choose

simpler evaluate function for it is efficient and could save time to explore deeper layer in this search.

9. what you would like to add to your program if you had more time

If we had more time, we would implement feature-based learning approach to improve our static eval, the weighted feature vector could end up acting as decent static eval with enough training.

10. citations for any references you used in the project. This should include the names and URLs of any websites that you used and whose ideas or other resources were incorporated into your project. In each case, describe in a sentence what role that website played in your project and what you incorporated from it.

We used Wikipedia to learn about the basic movement of the Braoque chess.
https://en.wikipedia.org/wiki/Baroque_chess

We used Wikipedia to learn how to implement Alpha-beta pruning.
https://en.wikipedia.org/wiki/Alpha–beta_pruning

we used Stackoverflow to learn how implement Zobrist Hashing
https://stackoverflow.com/questions/10067514/correctly-implementing-zobrist-hashing

11. For 5 points of extra credit, include a section with a heading "Partners' Reflections" with two subsections, one for each partner. Each subsection should give the partner's name, main role(s) in the project, a description of the challenges and benefits of the partnership from that partner's perspective.

Junhao Zeng:

Main role: Created the static evaluation function and the movement generator.
Challenges: Since each of us only wrote a part of codes for this project, one of the challenges were how to make codes more readable for parter. In order to solve this problem, we wrote some comments when some part of codes were tricky. In this way, this saves us a lot of time.

Benefits: The benefits are very obvious. By breaking up this project into two pieces, both of us can focus on the part for which we are responsible, so we can finish this project better. Besides, after we finished individual part, we can share idea with each other. Therefore, we can actually learn more than doing this project alone.

Jingqiang Wang:
Main role: implement the alpha-beta pruning and the Zobrist hash.
Challenges: The Zobrist hash need to be implemented in an appropriate way to avoid mistake. Another challenge is that we want to find another way to store the state will less collisions in hash table.

Benefits：We leaned a lot to debug our codes and cooperate with each other. The active communication between us improved the result of our project, sometimes your partner has a different view of yours and can help you figure out your problem easily.