Report

1. Title: Baroque Chess AI agent.
2. A) Zoheb Siddiqui: Created the static eval, minimax(without αβ pruning), IDDFS and Zobrist hash functions.

B) Edmund Lai:

1. The program is designed to play the game of baroque chess smartly, using various AI techniques learnt in class.
2. Minimax: This function takes in a state and a specified depth. It figures out the best possible move that can be made in the current state by analyzing the static eval values for successor states and maximizes/ minimizes them depending on whose turn it is. This process is repeated till all states for the given depth have been analyzed or if the method runs out of time. In both cases the best possible state analyzed(yet) is returned.

Zobrist Hashing: This technique is used to map a state to a unique hash value. The unique hash is the key to the static eval value of the state. Storing static eval values in a hash table saves time while running the minimax as it avoids recomputing values for states that have been analyzed before.

αβ pruning: This is another technique to optimize minimax to save time and avoid computing unnecessary states.

1. Screenshot:
2. Demo instructions:
3. Code excerpt:

def staticEval(state):

weights = [0,0,1,1,5,5,3,3,6,6,4,4,10,10,4,4]

board = state.board

white\_list = []

black\_list = []

piece\_present\_sum = 0

can\_move\_sum = 0

kill\_sum = 0

for i in range(0,8):

for j in range(0,8):

piece = board[i][j]

if piece != 0 and piece != 1:

if piece % 2 == 0:

piece\_present\_sum -= weights[piece]

black\_list.append([piece,(i,j)])

else:

piece\_present\_sum += weights[piece]

white\_list.append([piece,(i,j)])

to\_return = 0.3\*(piece\_present\_sum)

w\_moves = generateNewMoves(state, BC.WHITE)

b\_moves = generateNewMoves(state, BC.BLACK)

can\_move\_sum = len(w\_moves) - len(b\_moves)

for white\_move in w\_moves:

kill\_list = white\_move[2]

for elem in kill\_list:

kill\_sum += weights[elem]

for black\_move in b\_moves:

kill\_list = black\_move[2]

for elem in kill\_list:

kill\_sum -= weights[elem]

to\_return += 0.6\*(kill\_sum)

to\_return += 0.1\*(can\_move\_sum)

return to\_return

This is the static eval function. It returns a value for the state depending on if it’s good for white or black. If the state is good for white a high value is returned and if it is good for black a low value is returned.

Each piece is given a value depending on it’s importance. E.g the king is 10 and pincer is 1. The value returned consists of 3 parts. The first is the weighted difference between pieces remaining for white and black. This makes up 30% of the value. The 2nd is the difference between the total number of moves possible for white and black. This makes up 10% of the value. 60% of the value consists of the difference in kill sum for white and black. The kill sum is a weighted sum of all the pieces that can be killed for each opponent in the current state.

1. A) Zoheb Siddiqui: I learnt how to implement Zobrist hashing and integrate it into minimax. I also learnt how to correctly implement a time limit for the minimax function.

B) Edmund Lai:

1. If we had more time we would have implemented Q learning to teach the agent about the consequences to moving to a new state to make it smarter. We would also try to optimize our current code to work more efficiently to process more states in a limited time.
2. We used Wikipedia to learn about the different ways the individual pieces act and how Baroque chess is played.