Artificial Intelligence
LAB B Report
Sirajus Salekin, Sowmma Roy and Ali Shahram Musavi

For this lab we had to design a breakthrough game environment where two AI agents would play against each other. The representation scheme for the game is a 2D array, having the experience of working on the previous lab that involved maze solving, we came to the conclusion that a 2D array was the best option.

The two array is represented in the class Board, where it keeps track of the turn, players and the game state. For every move, we make sure to check if it is valid, meaning it is within the game environment. The position of the coordinates are represented by a tuple as (row, column). For the mimmax, we are using a tree representation. Based on the discussion we had with Dave and our teammates we concluded that given the time and space limitation branching out to a depth of three for the minmax tree was the best option. For each move, the AI agent is going to look up to a depth of three for the move it makes, before making any decision.

The game environment is displayed such that the two players are “X” and “O” and the empty positions are shown as “.” . The player X plays top down, and the player O plays bottom up. It is important to mention that for the game, the agent O is always going to make the first move.

Additionally, we have considered two cases for the terminal state. One is where one player run out of pieces and the other is where one players piece reach to the other end of the board.

Part A
Evasive vs Evasive

Evasive: e(s) = number_of_own_pieces_remaining + random()

The utility function evasive is very simple, as it values a given board state by the number of pieces remaining for the player. The evasive is more inclined to keep its pieces safe, and is not very keen to attack the other player. Thus when two agents play against each other using evasive function both will try to keep their pieces on the board. We believe evasive is a good name for the strategy.

We begin by having to AI play against each other on a 5X5 board, both using evasive as their utility function.

Initial Board state:

```
Welcome to Breakthrough!
This game has 4 different intelligent agents you can play with.
Please choose the opponent agents
1 : Evasive
2 : Conquerer
3 : House Lannister
4 : House Stark
Enter two numbers between 1 and 4, separated by a space:
1 1
You have chosen Evasive and Evasive to play

Please decide how your board should look like.
Enter # of rows, # of columns and # of rows with players
separated by spaces
5 5 1
This is how the board looks like:

#########################################################

X X X X X

. . . . .

. . . . .

. . . . .

O O O O O

#########################################################
```

First run:

```
#########################################################
. . . . X
. . O . .
. O . . .
. . X . X
. . . . .
#########################################################

#########################################################
. . . . X
. . O . .
. O . . .
. . X . X
. . . . .
#########################################################
X's turn now.
#########################################################
. . . . X
. . O . .
. O . . .
. . . . X
. X . . .
#########################################################
This game Ended. To play again, run `game.py`
Total number of moves made: 22
```

Player O captures two pieces
Player X captures two pieces

Second run:
This time we run it on a 8x8x2 environment

```
##########################################################

. . . . . O . .

. . . . . X . .

. . . X X . . .

. X . . . . . .

X X X . . . O X

. . . . . .

. X X X . . . .

. . . . . .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 103
```

Player O captures 5 pieces
Player X captures 14 pieces

As mentioned above it seems like both AI are defensive and trying to save their pieces. We believe that is the reason it takes reasonably more moves for a terminal state. This is because since on a 8x8 environment both players are hesitant to move their pieces forward. In general, we could conclude that evasive is a simple, save strategy.

Part B
Conqueror VS Evasive

Conqueror: (0 - number_of_opponent_pieces_remaining) + random()

Conqueror is a more offensive function. It is more focused at capturing the opponents pieces in every turn. The value of a state is determined based on the number of pieces a player has captured of its opponent. Conqueror seems to be working against evasive, since it is more outgoing and inclined to capture more pieces. Thus we expect the games to end quicker, either by one player running out of pieces or reaching the end of the board.

First run:
Player O is using evasive
Player X is using conqueror
Board state: 5x5x1

```
Welcome to Breakthrough!
This game has 4 different intelligent agents you can play with.
Please choose the opponent agents
1 : Evasive
2 : Conquerer
3 : House Lannister
4 : House Stark
Enter two numbers between 1 and 4, separated by a space:
1 2
You have chosen Evasive and Conquerer to play

Please decide how your board should look like.
Enter # of rows, # of columns and # of rows with players
separated by spaces
5 5 1
This is how the board looks like:

#########################################################

X X X X X

. . . . .

. . . . .

. . . . .

0 0 0 0 0
```

Ending state:

```
#########################################################

. . X . .

X . . . .

. X . . O

. X . . O

. . . X .

#########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 20
```

Player O captures no pieces
Player X captures three pieces

Second run:
Player O is using evasive
Player X is using conqueror
Board state: 6x6x2

```
###########################################################

. . . . . .

X . . X X .

. . . X X .

. . . . . X

. . . O O .

X . . . . O

###########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 44
```

Player O captures 5 pieces
Player X captures 9 pieces

Third run:
Player O is using evasive
Player X is using conqueror
Board state: 8x8x2

```
##########################################################

. . X X . . . .

. . . . X . . X

X X . . . X X X

X . . . . . O .

. X X . O . O O

. . . . O . . .

. O . O . O . .

O . O . . . . X

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 62
```

Player O captures 3 pieces
Player X captures 6 pieces

Observation:
As mentioned above conqueror works more offensively while evasive plays more defensively. The different instances of the game that we run, the AI agent using conqueror always ends up capturing more pieces of the opponents this was true for all the different sizes of the board. In 90% of the times, conqueror was the winner, but in the remaining 10% evasive won the games.

We observed that the behavior of the two agents is very different as one is focused on keeping its pieces, and the other is more focused on attacking and capturing as many pieces as possible.

Part C

As seen above the two utility functions Evasive and Conqueror doesn't seem to be particularly strong players. They don't have strong preference for the board states that are win, and not particularly averse to those that lead to a loss situation. In fact their goal is not even to win, one only tries to save its pieces while the other tries to capture as many pieces as possible.

At this part of the lab we create our own to functions. Inspired by Dave's naming for evasive and conqueror we named our two functions house_lannister and house_stark, both taken from the series: Game of Thrones.

```
def house_lannister(player, input_state):
    result = 2 * (myscore(player, input_state) - 1) * enemyscore(player, input_state)
    percentange = random.uniform(0,1)   #Generates random number between 0-1
    Num = round(percentange , 2)
    result = result + Num
    return result


def house_stark(player, input_state):
    result = 1 * (myscore(player, input_state) - 2) * enemyscore(player, input_state)
    percentange = random.uniform(0,1)   #Generates random number between 0-1
    Num = round(percentange , 2)
    result = result + Num
    return result
```

House lannister is an offensive player, it keeps track of its own score through keeping the count of how many pieces it has in the board, as well as how far its pieces are in the enemy's ground. Additionally, to make the player smarter it also considers and keeps in mind how many players are left from the opponent and how far their pieces are in its ground, and based on that compares the utility of each state. It has a strong preference for keeping the maximum number of its players, while also moving forward and capturing the enemy's pieces.

House Stark keeps track of the same variable, however, its more focused leads its opponent to less players or losing the game. It keeps track of its own score and the enemy's score. House Stark is however more conservative and defensive compared to House Lannister.

First run
House Lannister against Evasive
5x5x1
Player O is evasive
Player X is House Lannister

```
#########################################################

X . . . .

X . X . .

O . X O .

. . . . O

. X . O .

#########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 16
```

Player O captures nothing
Player X captures 1 piece

Second run
Player O is evasive
Player X is House Lannister 8x8x2

```
########################################################

. . . . . . . X

. X . . O . . .

. X X O . . . O

X . . . . . . .

. X . . . X . .

. . . . . . . .

X . . . . . O .

. . . . X . . .

########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 96
```

Player O captures 7 pieces
Player X captures 12 pieces

House Stark vs Elusive
Player O is evasive
Player X is House Stark
8x8x2

```
##########################################################

. . . . X . X .

X X X X X X . .

. . X . X . X .

. . X . . X . .

O . . . . . . O

. . . . . O O .

O . O O . O . .

O . . X . . O O

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 48_
```

Player O captures 4 pieces
Player X captures 5 pieces

House Lannister vs Conqueror
Player O is conqueror
Player X is House Lannister
8x8x2

```
##########################################################

. . . . . . . .

. . . . . . O O

. . . . . . O .

X X . . X . O .

. X . X . . . .

. . . . . O . O

. . X . O . . .

X . . . . . . .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 102
```

Player O captures 9 pieces
Player X captures 9 pieces

House Stark vs Conqueror
In this instance we have a **8 8 2** board.
O = conqueror | X = house stark

```
##########################################################

. . . . . . . .

. . . . . . . X

. X . . . . X .

. X . X . . . .

. X . . . . . .

. . X . . . . .

. . . . . . X .

. . . . . . . .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 90
```

The game ends with 8X and 0 O
O captures 8 X
X captures 16 O

```
##########################################################

. . . . . .

. . . . . .

X . . . O . . X

X . . . . . .

. . . . O . O .

. . . . . .

O . . . . . .

. . X . . . . .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 94_
```

The game ends with 4X and 4 O
O captures 14 X
X captures 14 O

House Lannister vs House Stark
In this instance we have a **8 8 2** board.
O = lannister | X = house stark

```
##########################################################

. . . . . . . .

. . . O . O O .

. . . . . . . .

. . . O . . . .

O . . . . . . .

. . . . . . . .

. . . O . . O .

O . O O . O O .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 73
```

The game ends with 0X and 12 O
O captures 16 X
X captures 4 O

```
##########################################################

X . . . . . . .

X . . . . . . X

. . . . . . . X

. . . X . . X .

. . . . . . . .

. . . X . X . .

. . . . . . . .

. . . . . . . .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 88
```

The game ends with 8X and 0 O
O captures 8 X
X captures 16 O

```
##########################################################

X X . . . . . .

. . . . X . . X

X . X . . . . .

. . . . . . . .

. . . X . . . .

. . . . X . X .

. . . . . . . .

. . . . . . . .

##########################################################

This game Ended. To play again, run `game.py`
Total number of moves made: 80
```

The game ends with 9X and 16 O
O captures 7 X
X captures 16 O

Observations:
Based on running the players against each other multiple times, we believe it is safe to conclude that House Lannister is the champion. It has won games against evasive, conqueror and House Stark. We believe this is because it tends to move forward with more pieces and at each state it keep count of how many pieces it has and how forward they are, as well as it's enemies. There are definitely shortcoming with it, as it can become more smarter and have stronger preferences to the states to its victory.