Zeyad Ahmed Ibrahim Zidan 19015709

INTRODUCTION INTO AI

Programming Assignment #2 Report

TEAM DATASHEET

- Zeyad Ahmed Ibrahim Zidan 19015709
- Abdelrahman Fathy El-Lawaty 19015920
- Mohammed Mostafa Abdallah 19016506
- Youssef Saber Saeed Mohammed 19016924

ANALYSIS

We show below a table analyzing the behavior of minimax algorithm concerning nodes expansions and how long does it take the AIs to react (expand nodes) to find the best solution they would benefit from. All the data extracted are results of 2 AI agents playing against each other **with different heuristics** till the end of the game.

	With Pruning		Without Pruning	
C	Expansions	Time Taken (s)	Expansions	Time Taken (s)
01	2955	0.142609835	19607	1.038258314
02	3767	0.195471048	19607	1.106004
03	2424	0.125636101	19607	1.226750374
04	4017	0.253322124	19606	1.369306326
05	2040	0.119680166	19575	1.308536053
06	4343	0.315195322	19574	1.656530857
07	2595	0.164521456	19189	1.417242289
08	4493	0.379702568	16794	1.560829401
09	1261	0.084769249	9329	0.748996258
10	2424	0.221406698	9303	0.968409777
11	1253	0.091749191	9032	0.786859274
12	1110	0.104729652	9032	1.006011248
13	1404	0.112661362	9032	0.798867702
14	1917	0.187500238	9031	1.033239126
15	1457	0.111701012	9030	0.842744589
16	1589	0.169582605	9004	1.103047848
17	1077	0.091744423	8733	0.850693703
18	1325	0.141634703	7327	0.965125322
19	917	0.078749895	6047	0.618344545
20	764	0.079795599	2956	0.37602663

21	362	0.028944731	1363	0.137631655
22	385	0.040859461	1363	0.175531864
23	510	0.045894861	1362	0.142584801
24	524	0.058824778	1362	0.17656064
25	557	0.051892996	1361	0.141614914
26	630	0.082778454	1345	0.18550992
27	533	0.052859068	1242	0.158545017
28	425	0.053823709	908	0.144611359
29	417	0.0389328	892	0.116687775
30	408	0.055850267	799	0.138629675
31	277	0.024928331	541	0.074995995
32	129	0.013967991	190	0.025932074
33	54	0.003983498	61	0.004986286
34	54	0.005984306	61	0.00698185
35	52	0.003989458	60	0.004986048
36	49	0.004987478	54	0.004986763
37	36	0.002991438	39	0.002991915
38	17	0	19	0
39	4	0	4	0
40	3	0	3	0
41	2	0	2	0
42	1	0	1	0

PROBLEM STATEMENT

- Design a connect-four game with GUI where the AI would play against a player.
- Each player can do one move each turn.
- The turns are alternating.
- The game ends whenever the board is full, and no player can do any other move.
- The score is calculated at the end of the game showing which player has the most connected fours.
- The AI agent must use minimax algorithm with and without pruning to calculate the best move and performs it.
- Additionally, the GUI shall display the minimax tree of nodes with the maximum determined depth showing each node value.
- Whenever the minimax algorithm reaches the maximum depth allowed, the minimax algorithm must calculate node value using a heuristic.

CHALLENGES:

- Designing a user-friendly interface.

- Implementing the minimax algorithm using nodes data structure to access the root of the minimax tree and hence draw it to the user.
- Handling bits operations as each state of the game is represented in a bit-representation motive.
- Choosing the possible best heuristic and implementing it wisely.

APPROACHES:

- Designed the UI to be simple and user-friendly. The UI takes the user input from the front-end and sends the data to back-end so it can process it.
- Defined node data structure to save the state and its value to the AI whenever the game reached that state. The root is responsible for drawing the minimax tree.
- Implemented a clean-code class to handle the bits operations, which includes playing the move, changing the state, converting the state into a board, checking whether the game has ended or not, and calculating the scores of each player.

DATA STRUCTURES USED:

- Xmpz class from gmpy2, a python library.
- Node class, which is manually implemented.
- Normal Python arrays.

CODE STRUCTURE

- **agent.py** – plays the role of the AI agent and constructs minimax tree based on the value of a state. The agent picks the best value in their benefit.
- **bitoperations.py** – the core of the code
    - The file has the function that takes a state and a move and performs the action of playing. The function prevents any unsafe (invalid) move, such as playing outside the board or trying to play in a full column.
    - The file also has a method to check the end of a game at any given state.
    - Provides a function to calculate players scores at any given state.
    - Provides a function to translate a state into a 2D-matrix board.
- **heuristics.py** – this file contains 2 different heuristics (we developed a way so 2 AI agents can play against one another using these 2 heuristics).
- **node.py** – the file containing Node class.
- **runner.py** – the file which serves the front end and where the AI agent chooses their next move according to the minimax algorithm tree nodes values.

DESIGN PATTERNS USED

- Tried as much as possible to use dependency injection and SOLID principles which helped refactoring the code into clean code by removing duplicates and having each class do a dedicated job to another.

PSEUDOCODE

MINIMAX(HEURISTIC, root, max_depth):

1. If CHECK-END(root.state)// Base case, the game ended.
    a. return score1 – score2
2. If max_depth == 0 // Maximum depth reached.
    a. return HEURISTIC(root.state)
3. If root.turn == 0 // Agent is player1 and needs to maximize their benefits.
    a. root.value = -infinity
    b. For i = 0 to 6 // Each i refers to a column number
        i. next_state ← Play in the i^{th} column.
        ii. If the next_state is valid, append it to root.state children.
        iii. next_state.value = max(root.value, MINIMAX(HEURISTIC, child, max_depth - 1)) // Crucial Line
    c. root.value = max(children.values) // Assign the maximum value of children values to the root.
    d. return root.value
4. Else // Meaning that root.turn == 1 → Maximizing Section
    a. root.value = infinity
    b. For i = 0 to 6
        i. next_state ← Play in the i^{th} column.
        ii. If the next_state is valid, append it to root.state children.
        iii. next_state.value = min(root.value, MINIMAX(HEURISTIC, child, max_depth – 1)) // Crucial Line
    c. root.value = min(children.values) // Assign the minimum value of children values to the root.
    d. return root.value

PRUNING-MINIMAX(HEURISTIC, root, max_depth, alpha, beta):

1. If CHECK-END(root.state)// Base case, the game ended.
    a. return score1 – score2
2. If max_depth == 0 // Maximum depth reached.
    a. return HEURISTIC(root.state)
3. If root.turn == 0 // Agent is player1 and needs to maximize their benefits.
    a. root.value = -infinity

b. For i = 0 to 6 // Each i refers to a column number
    i. next_state ← Play in the i^{th} column.
    ii. If the next_state is valid, append it to root.state children.
    iii. next_state.value = max(root.value, MINIMAX(HEURISTIC, child, max_depth - 1)) // Crucial Line
    iv. If beta < next_state.value → return next_state.value
    v. alpha = max(alpha, next_state.value)
c. root.value = max(children.values) // Assign the maximum value of children values to the root.
d. return root.value

4. Else // Meaning that root.turn == 1 → Maximizing Section
    a. root.value = infinity
    b. For i = 0 to 6
        i. next_state ← Play in the i^{th} column.
        ii. If the next_state is valid, append it to root.state children.
        iii. next_state.value = min(root.value, MINIMAX(HEURISTIC, child, max_depth – 1)) // Crucial Line
        iv. If alpha > next_state.value → return next_state.value
        v. beta = min(beta, next_state.value)
    c. root.value = min(children.values) // Assign the minimum value of children values to the root.
    d. return root.value

HEURISTICS :

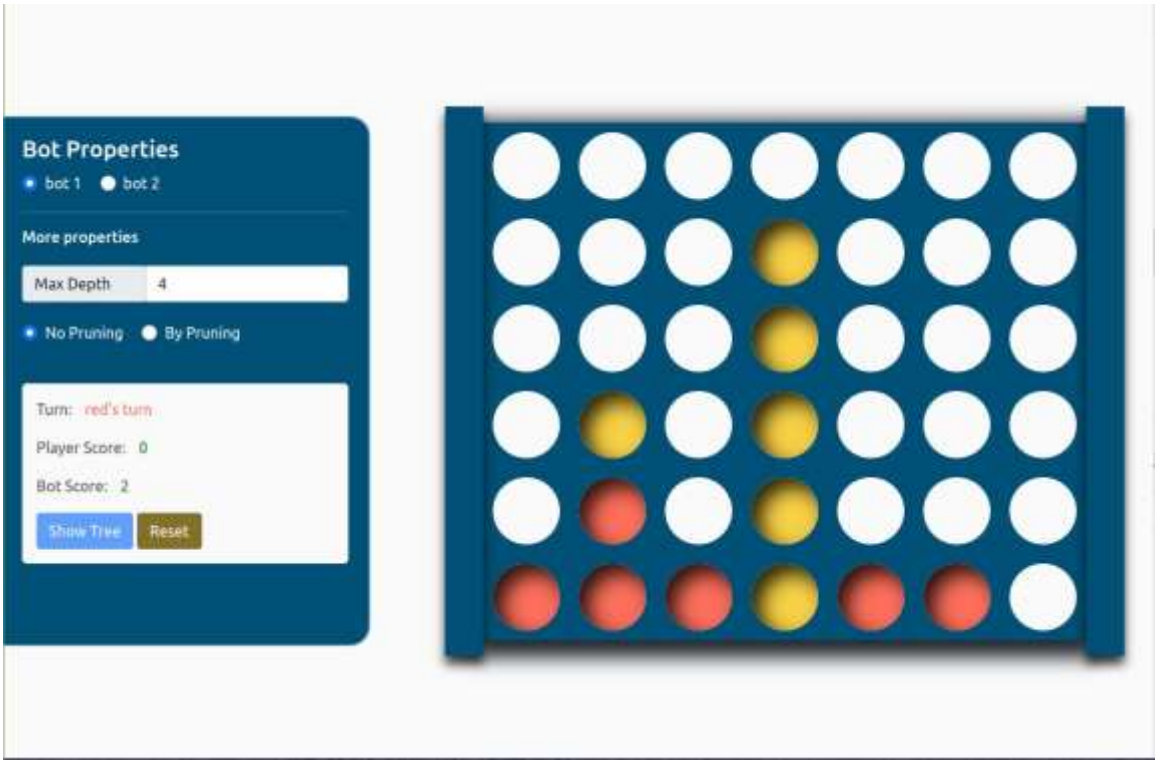
Heuristic 1 :

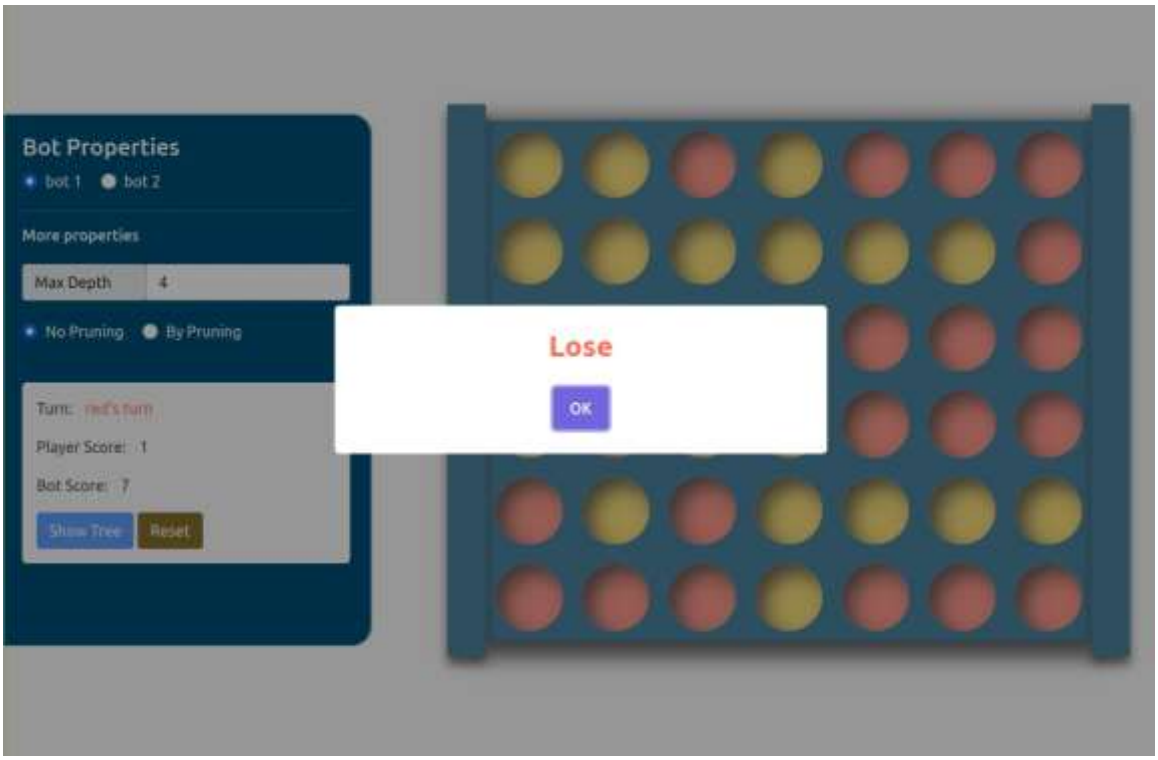
For each elements calc all sequences nodes which have same color in all directions and calculate the score as a function of this numbers and for each sequence see the next node , if it have no thing that increases the score else the score is decreased

Heuristic 2 :

For each elements calc all sequences nodes which have same color in all directions and for each sequence calculate number of next and before zeros and max next sequence , and calculate the score as a function of this parameters

TEST CASE :


Bot Properties
bot 1
bot 2
More properties
Max Depth
4
No Pruning
By Pruning
Turn: red's turn
Player Score: 0
Bot Score: 2
Show Tree
Reset


Bot Properties
bot 1
bot 2
More properties
Max Depth
4
No Pruning
By Pruning
Turn: red's turn
Player Score: 1
Bot Score: 7
Show Tree
Reset
Lose
OK