An Intelligent Othello Player

Team Members

Member 1 Name : احمد ناصر عبد الستار احمد
Member 1 ID : 201900107

Member 2 Name : سيف الدين احمد سيف
Member 2 ID : 201900362

Member 3 Name : يارا محمد حسن علي
Member 3 ID : 201900954

Member 4 Name : سهيلة حسن حسني احمد
Member 4 ID : 201900356

Member 5 Name : اسلام حسن محمد عفيفي
Member 5 ID : 201900136

All of the team Members
Level : 3
Department : computer science

Project idea and overview:

- Reversi or Othello is a strategy board game for two players, played on an 8×8 uncheckered board. There are sixty-four identical game pieces called *disks* (often spelled "discs"), which are light on one side and dark on the other. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color.

- Othello offers multiple levels, and each level of the game is more demanding than the last one.

- The object of the game is to have the majority of disks turned to display your own color when the last playable empty square is filled.

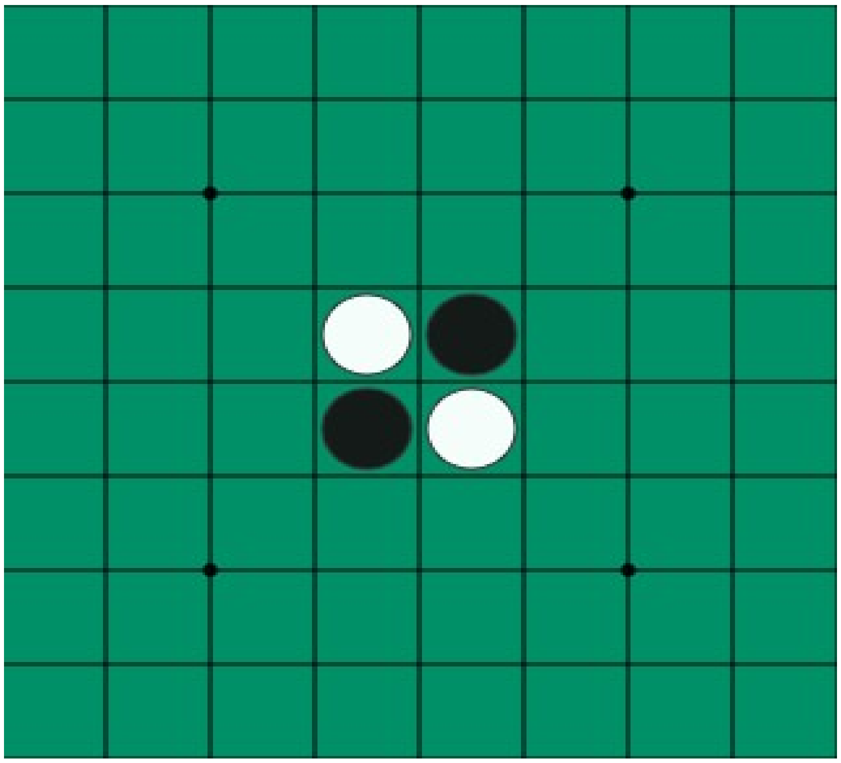
- For the specific game of *Othello*, the game begins with four disks placed in a square in the middle of the grid, two facing white-side-up, two dark-side-up, so that the same-colored disks are on a diagonal. Convention has this such that the dark-side-up disks are to the north-east and south-west (from both players' perspectives), though this is only marginally consequential: where sequential openings' memorization is preferred, such players benefit from this.

- Then the game alternates between white and black until:

  - one player can not make a valid move to outflank the opponent.
  - both players have no valid moves.

When a player has no valid moves, he pass his turn and the opponent continues.

When both players can not make a valid move the game ends.



Main Functionalities/features, and how they work:

1)The general functions:

a - Functions important for finding places to play check_line_match, calc_legal_moves and adj_sup

are to check all the eight directions around a particular position on our board if   any of those positions supports our play then we will put it into our othello legal array .

b - Functions important for reversing flip_line and flip_token

changes color and looks in one of the 8 directions (left, right, up, down, upleft, upright, downleft, downright) to see if adjacent opponent pieces are sandwiched between our pieces. If so, it returns true. If not, it returns false. True means we should flip pieces in this direction. The -1, 0, and 1 control which direction we are looking.

c - Function make_all_not_legal

that remove all the legal moves for a new game.

d - Function return_list_of_legal_moves

that returns all the possible legal moves in a list.

e - Function game_over

if the game is finished it returns the score and the winner, else it shows only the score during the game.

2) The AI functions:

a - Function isMovesLeft

checks if there is still legal moves.

b - Function sumCoinParity

calculates the score and return it in a list .

c - Function minimaxAlphabeta

it returns the best value for the ai and also for the opponent as the ai player tries to maximize the value and the opponent tries to minimize the least possible loss.

d - Function findBestMove

it choose the best move among the available legal moves for the player and returns it.

e - Function game_heuristic

it shows all the possible heuristic functions and allow the user to choose one of them or using all them .

Similar applications:

- Tic Tac Toe :

a game in which two players alternately put Xs and Os in compartments of a figure formed by two vertical lines crossing two horizontal lines and each tries to get a row of three Xs or three Os before the opponent does

- Checkers :

checkers is a game played on a board checkered with squares of two colors. Two players compete in checkers to have the last piece on the board.

- Go-Player :

is an abstract strategy board game for two players in which the aim is to surround more territory than the opponent.

- Chess-Player :

The goal of the game is to checkmate the other king. Checkmate happens when the king is in a position to be captured (in check) and cannot escape from capture.

- Pandemic - The Board Game :

Humanity is on the brink of extinction. As members of an elite disease control team, you’re the only thing standing in the way of the four deadly diseases spreading across the world.

An initial literature review of Academic publications:

First paper:

This paper compares three strategies in using re-inforcement learning algorithms to let an artificial agent learn to play the game of Othello. The three strategies that are compared are: Learning by self-play, learning from playing
against a fixed opponent, and learning from playing against a fixed opponent while learning from the opponent's moves as well. These issues are considered for the algorithms Q-learning, Sarsa and TD-learning. These three reinforcement learning algorithms are combined with multi-layer perceptrons and trained and tested against three fixed opponents. It is found that the best
strategy of learning differs per algorithm. Q-learning and Sarsa perform best when trained against the fixed opponent they are also tested against, whereas TD-learning performs best when trained through self-play. Surprisingly, Q-learning and Sarsa outperform TD-learning against the stronger fixed opponents, when all methods use their best strategy. Learning from the
opponent's moves as well leads to worse results compared tolearning only from the learning agent's own moves.

Second paper

Game playing is a game method that require an AI (Artificial Intelligence), so that an AI can play against human in a game. Artificial intelligence involves two basic ideas[4]. First, it involves studying the thought processes of human beings. Second, it deals with representing those processes via machines (like computers, robots, etc.). AI is behavior of a machine, which, if performed by a human being, would be called intelligent. It makes machines smarter and more useful, and is less expensive than natural intelligence. Othello is one example of game playing using AI. Even though it may appear as though Othello is a fairly simple game, there still are many important aspects of the game to consider. The most important of these are the evaluation function and

searching algorithms. Why are these important? First of all, the game would be nothing
without an evaluation function. And there are many interesting aspects of the evaluation which can greatly affect both efficiency as well as game play. Second, a good searching algorithm can fulfill the ideal properties of a good
heuristic, providing a good answer in a reasonable amount of time.

Third paper:

In this article we describe reinforcement learning, a machine learning technique for solving sequential decision problems. We describe how reinforcement learning can be combined with function approximation to get approximate solutions for problems with very large state spaces. One such problem is the board game Othello, with a state space size of approximately 10 28 . We apply reinforcement learning to this problem via a computer program that learns a strategy (or policy) for Othello by playing against itself. The reinforcement learning policy is evaluated against two standard strategies taken from the literature with favorable results. We contrast reinforcement learning with standard methods for solving sequential decision problems and give some examples of applications of reinforcement learning in operations research and management science from the literature.

Fourth paper:

In this article we present an overview on the state of the art in games solved in the domain of two- person zero-sum games with perfect information. The results are summarized and some predictions for the near future are given. The aim of the article is to determine which game characteristics are predominant when the solution of a game is the main target. First, it is concluded that decision complexity is more important than state-space complexity as a determining factor. Second, we conclude that there is a trade-off between knowledge-based methods and brute-force methods. It is shown that knowledge-based methods are more appropriate for solving games with a low decision complexity, while brute-force methods are more appropriate for solving games with a low state-space complexity. Third, we found that there is a clear

correlation between the first-player's initiative and the necessary effort to solve a game. In particular, threat-space-based search methods are sometimes
able to exploit the initiative to prove a win. Finally, the most important results of the research involved, the development of new intelligent search methods, are described.

Fifth paper:

This paper surveys the evaluation and search techniques utilized by the strongest Othello programs of their time during the past twenty years. In this time span computer Othello has experienced considerable progress which culminated in the convincing 6-0 victory of LOGISTELLO against the then World-champion Takeshi Murakami in 1997. The focus of this article is the evolution of Othello evaluation functions and heuristic search techniques which quite nicely reflect the general A.I. trend of replacing slow and error-prone manual tuning by automated machine learning approaches.

Dataset used:

No dataset used.

Algorithms used:

Alpha-Beta Search Strategy with depth first , it's a backtracking algorithm that used in decision-making and often used in 2-players games to find the optimal move for a player assuming that the opponent also plays optimally.
The player is named maximizer that wants to maximize the win chances and his enemy named minimizer that want to minimize the win chances of the maximizer. Where it's maximizer's turn the state score will tend to be increased. Where it's minimizer's turn the state score will tend to be decreased. The score of the board states calculated by some heuristic functions.
Heuristics used in the algorithm:

- **Coin Parity**

  gets the difference between the number of points of the max player and the min player,
  The return value is determined as follows :
  100 (Max Player Coins - Min Player Coins) / (Max Player Coins + Min Player Coins)

- **Mobility**

  gets the relative difference between the number of possible moves for the max and the min players, with the intent of restricting the opponent's mobility and increasing one's own mobility.
  Calculated by:

  ```
  if ( Max Player Moves + Min Player Moves != 0)
      Mobility Heuristic Value =
              100 * (Max Player Moves - Min Player Moves) / (Max Player Moves + Min Player Moves)
  else
      Mobility Heuristic Value = 0
  ```

- **Corner occupancy**

  If a player can capture the corners, he cannot be flanked by the opponent. It also allows a player to build coins around the opponent and provide stability to the player's coins. This value is Calculated by:

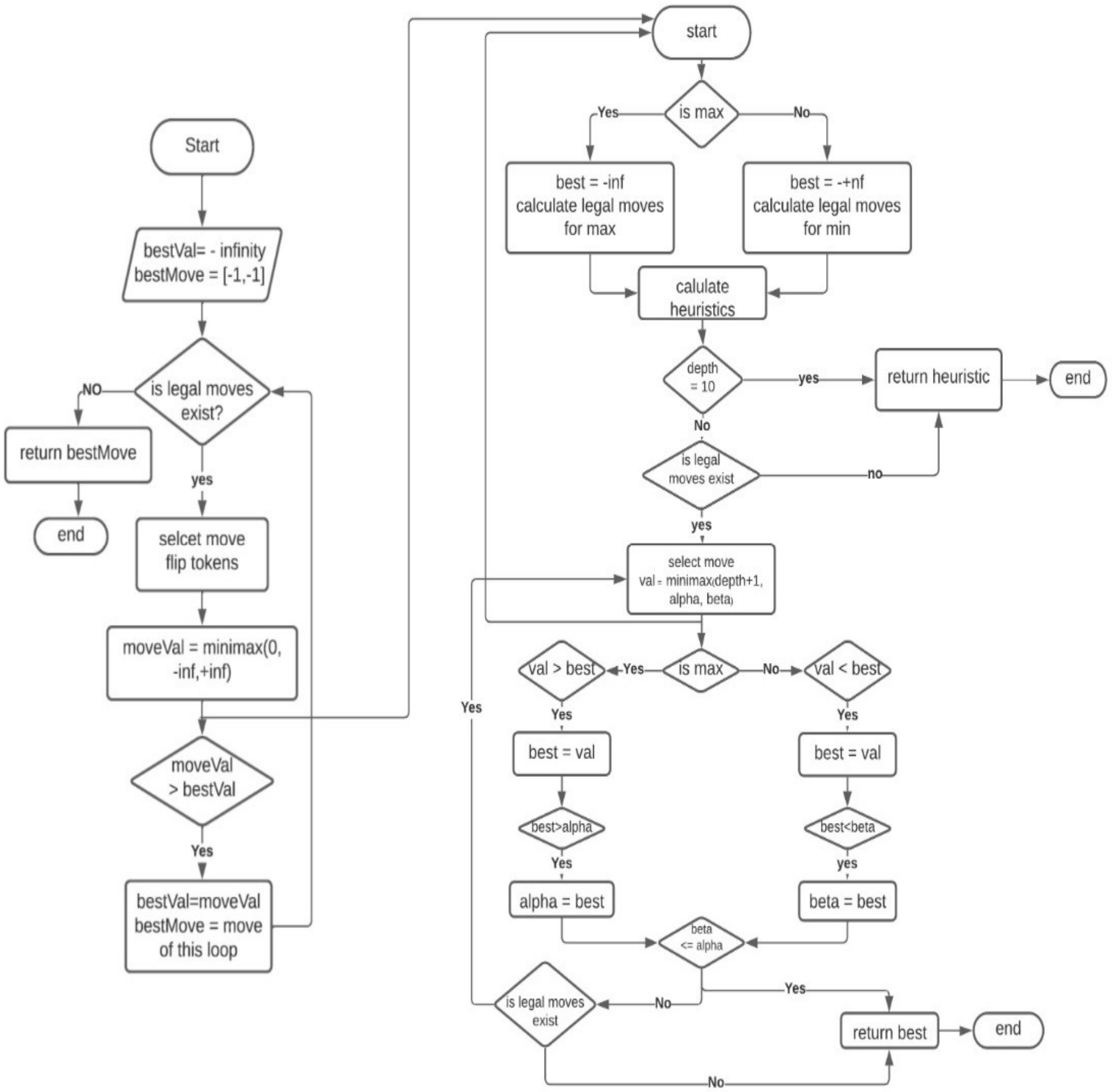
  ```
  if ( Max Player Corners + Min Player Corners != 0)
      Corner Heuristic Value =
          100 * (Max Player Corners - Min Player Corners) / (Max Player Corners + Min Player Corners)
  else
      Corner Heuristic Value = 0
  ```

In Alpha-Beta Search, states are expressed as a tree , it prunes branches that will not give a better score to be much faster.
It has 2 parameters alpha and beta.
Alpha is the best value the maximizer can get; beta is the best value the minimizer can get.

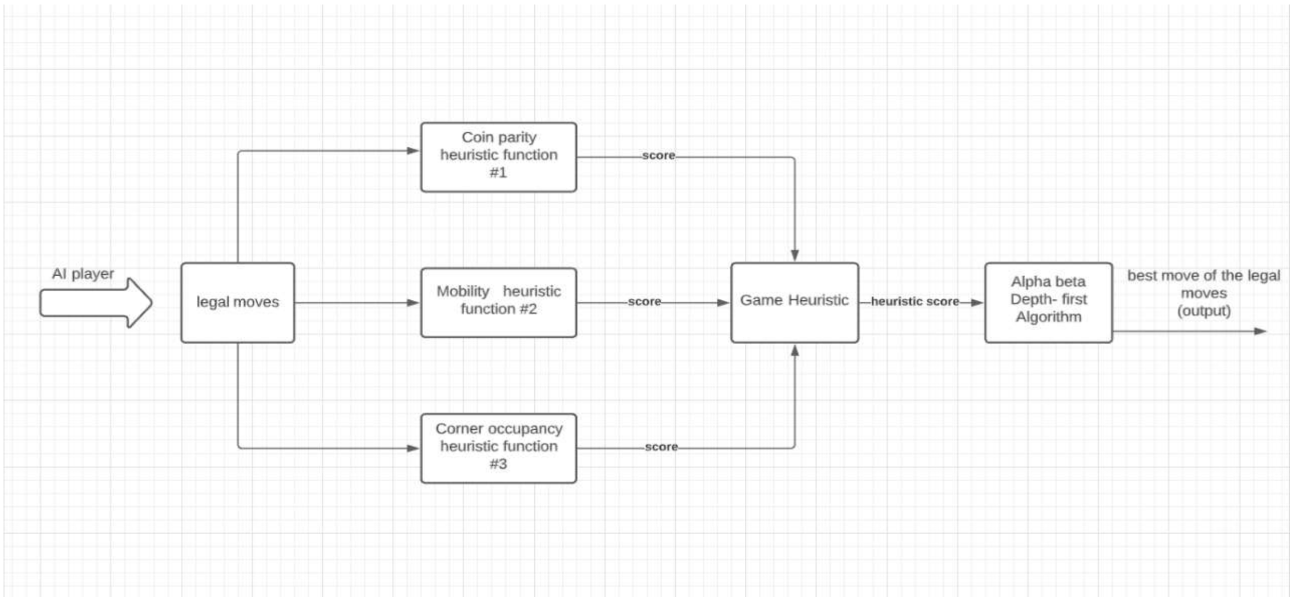
Development platform:

- Python language
- Numpy and copy libraries
- Tkinter (Python GUI)

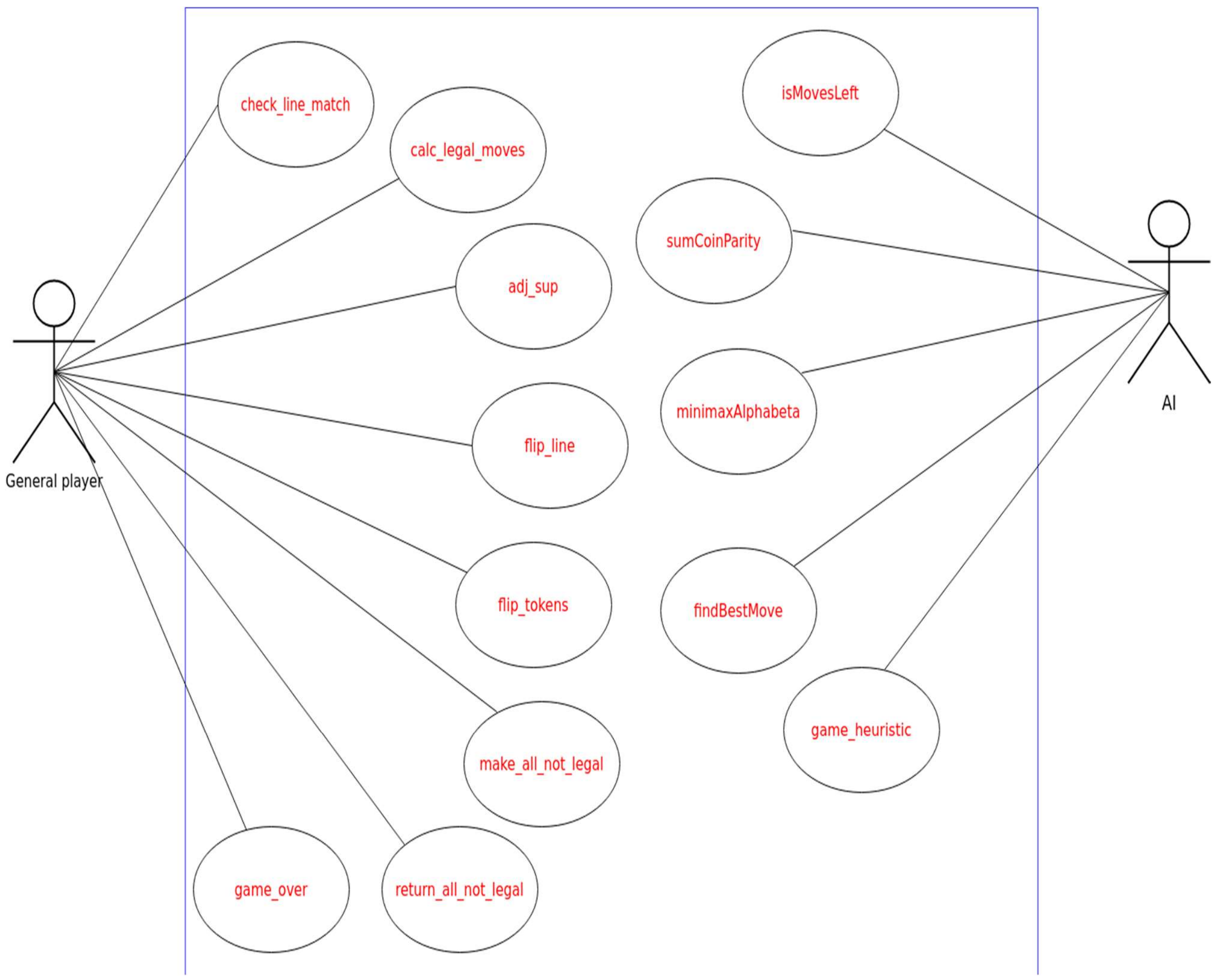
Flowchart Diagram:



Block Diagram:


AI player
legal moves
Coin parity heuristic function #1
Mobility heuristic function #2
Corner occupancy heuristic function #3
score
score
score
Game Heuristic
heuristic score
Alpha beta Depth- first Algorithm
best move of the legal moves (output)

Use-Case diagram:



Analysis of the results, what are the insights?

- The results suggest that mobility is a very powerful heuristic when used in conjunction with the corner and coin parity heuristics.

- The mobility heuristic ensures that the opponent does not have too many moves to choose from, hence restricting the opponents control over the board. Though mobility has a low weight, it has such a great impact on the game play, as demonstrated by the results. This suggests that increasing the weight of the mobility heuristic might enhance play quality, but that was not true during our experimentation, because high mobility downplayed other heuristics leading to bad moves.

Team members role:

In the general functions:

احمد الستار عبد ناصر احمد :

Made the calc_legal_moves, check_line_match and adj_sup functions.

سيف الدين احمد سيف:

Made the Gui frame and functions.

يارا محمد حسن علي:

Made the flip_line and flip_tokens .

سهيلة حسن حسني احمد:

Made the make_all_not_legal , return_list_of_legal_moves and game_over functions.

اسلام حسن محمد عفيفي:

Made the one_on_one_main_fun and ai_on_one_main_fun functions.

In the AI functions:

We all worked together in the minimaxAlphabeta, findBestMove and game_heuristic functions.

Source code URL:

http://rb.gy/bnfl1l

References:

- Project idea :

Reversi – Wikipedia

- First paper :

Reinforcement Learning in the Game of Othello

- Second paper :

Evolutionary Neural Network for Othello Game - ResearchGate

- Third paper :

Reinforcement Learning and its Application to Othello

- Fourth paper :

Games solved: Now and in the future - Science Direct

- Fifth paper :

The evolution of strong othello programs