Faculty of Computers and Artificial Intelligence Artificial Intelligence
Fall semester 2023-2024 Artificial Intelligence project Team Members

Reop link:
https://github.com/project103/ai_bro_6.git

	Name	ID	Department
1	عبد الله محمد سعيد	20210556	CS
2	مروة عمر محمد	20210900	CS
3	رنا عصام الدين عيسى	20210335	CS
4	عمر ناصر جمال عبدالمطلب	20210628	AI
5	تقى محمد أحمد السعدي	20210242	CS
6	أسامة سميرعبدالمنعم	20210140	CS

1) Project idea in details

Tic Tac Toe 4x4x4 is an extension of the classic Tic Tac Toe game, but played on a larger 4x4x4 grid instead of the traditional 3x3 grid. The objective of the game remains the same: players take turns placing their symbols (usually "X" and "O") on the grid with the goal of getting four of their symbols in a row, either horizontally, vertically, or diagonally.

The larger grid introduces more cells and complexity, making it a bit more challenging than the standard 3x3 version. Players need to plan their moves strategically to prevent their opponent from forming a winning combination while aiming to create their own.

Project:

The project aims to create an intelligent Tic-Tac-Toe player that can play optimally against a human or another AI. The player will use advanced algorithms like Minimax with Alpha-Beta pruning and heuristic functions to make strategic decisions.

2) Main functionalities:

- function set_difficulty():
  set the difficulty level for the game choosing the algorithm

- check_and_open_gui(self):
  A method to check the difficulty level and open the corresponding GUI when the "Start" button is clicked
- **`make_move(self, i, j, k)`** :
  A method to make a move on the 4x4x4 board, updating the state accordingly.

- `is_terminal(self)`:
  A method to check if the current state of the game is a terminal state (win, loss, or draw).

- `minimax(self, depth, maximizing_player)`:
  A method implementing the minimax algorithm to find the best move for the current player. The algorithm explores the game tree to determine the optimal move.

- `evaluate_state(self, player)`:
  A method to evaluate the desirability of a terminal game state for the given player.
- `evaluate_line(self, line, player)`:
  A helper method to evaluate a line of the board for a given player.

- `alpha_beta(self, depth, alpha, beta, maximizing_player)`:
  A method implementing the alpha-beta pruning algorithm to find the best move for the current player. The algorithm explores the game tree while pruning branches that do not affect the final decision.
- `alpha_beta_wrapper(self, depth, maximizing_player)`:
  A wrapper method for the alpha-beta pruning algorithm, setting initial alpha and beta values.
- `set_AI_player(self)`:
  Determines the AI player based on the current player.
- `set_value(self, layer, row, column)`:
  Sets the value of a cell in the specified layer, row, and column if it's empty.
- `get_value(self, layer, row, column)`:
  Retrieves the value of a cell in the specified layer, row, and column.
- `turn(self)`: Switches the current player after each turn.
- `print_board(self)`: Prints the current state of the board.

- `is_win(self, player)`: Checks if the specified player has won the game.
- `is_empty(self)`: Checks if the entire board is empty.
- `is_full(self)`: Checks if the entire board is full.
- `get_invalid_spaces(self)`: Retrieves a list of invalid spaces on the board.
- `reset_board(self)`: Resets the board by clearing all cells.

- `is_symmetric(self, player, layer, row, column)`:
  Checks if there is symmetry across layers, rows, and columns for the specified player in the given layer, row, and column.

GUI:

- `set_players(self, player)`: Sets players in the TicTacToe4x4x4 instance.
- `create_widgets(self)`: Creates buttons, labels, and other widgets for the GUI.
- `back_game(self)`: Destroys the current window and opens the main menu.
- `reset_game(self)`: Resets the game state and GUI.
- `make_move(self, layer, row, col)`: Handles human moves, updates the GUI, and triggers AI moves.
- `check_game_over(self)`: Checks if the game is over (win, draw) and displays appropriate messages.
- `update_status(self)`: Updates the label displaying the current player.

3) Similar application in the market:

Tic Tac Toe Game Alternatives Tic Tac Toe Game is a classic strategy game where two players take turns marking a 4x4x4 grid with their respective symbols. If you're looking for alternatives with similar gameplay, there are several options available across different platforms. There are more than 25 games similar to Tic tac toe for a variety of platforms, including Android, Windows, Android Tablet, iPad and iPhone.

Functionalities/Features in Common:

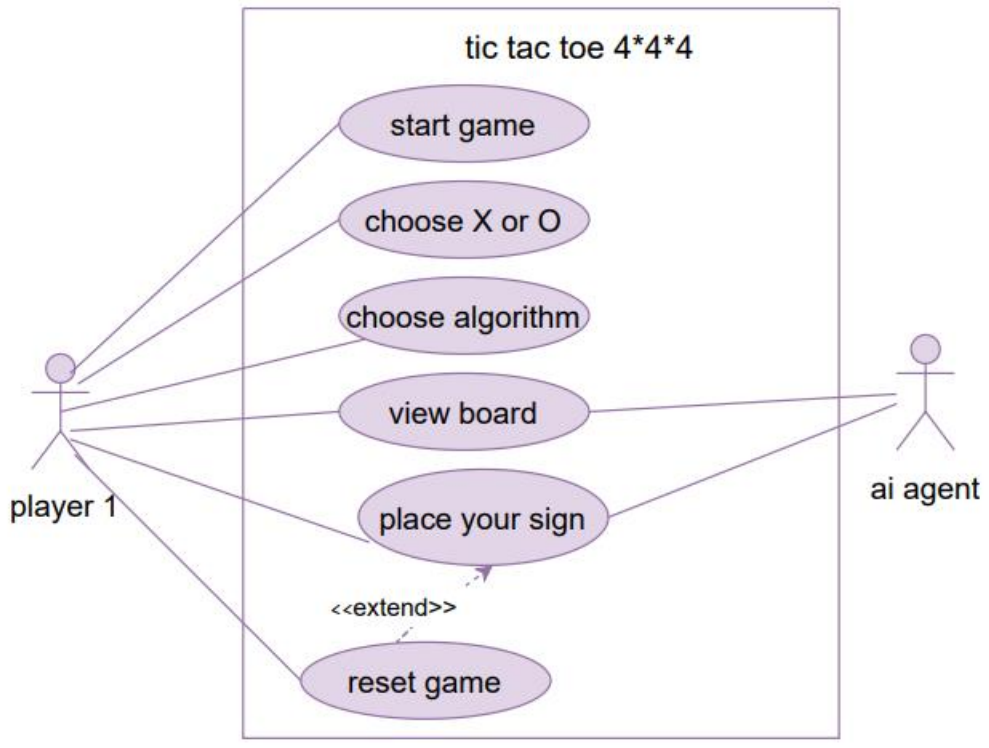
- Single-Player and Two-Player Modes
- User-Friendly Interface
- Classic 4x4x4 Grid Gameplay

How They Work:

- User Interface: Players interact with the game through a graphical interface, making moves by selecting cells on the grid.
- AI Strategies: Some games utilize AI algorithms like minimax or heuristics to provide challenging single-player experiences.
- Multiplayer (Online Game): Online multiplayer games involve server-based communication for real-time matches between players.

Proposed Solution & Dataset:

Use case diagram:



no dataset used

Applied Algorithms:

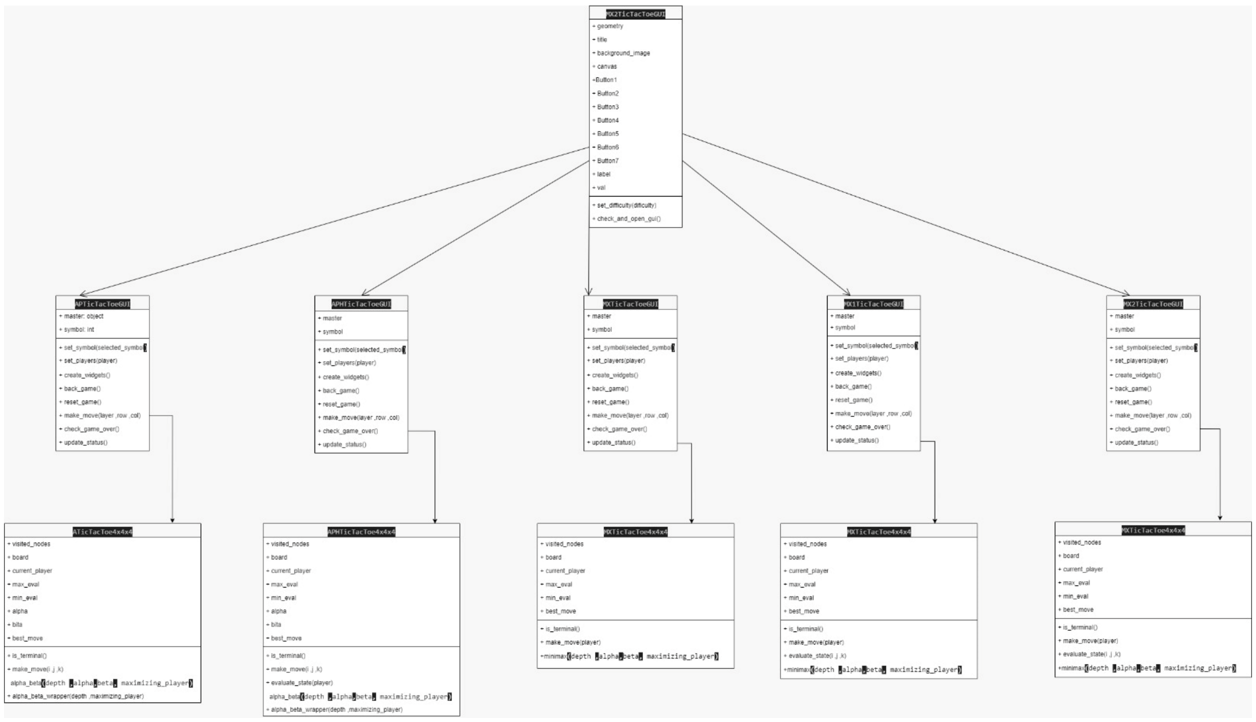
The Minimax algorithm is a decision-making algorithm commonly used in two-player turn-based games, such as Tic Tac Toe, Chess, or Checkers. Its primary objective is to find the optimal move for a player, assuming that both players play optimally and with the goal of maximizing their chances of winning or minimizing their chances of losing.

Alpha-Beta Pruning is an optimization technique applied to the Minimax algorithm to reduce the number of nodes evaluated in the search tree. It helps improve the efficiency of the Minimax algorithm by pruning branches of the tree that are guaranteed not to affect the final decision. This can significantly reduce the number of nodes explored, making the algorithm faster.

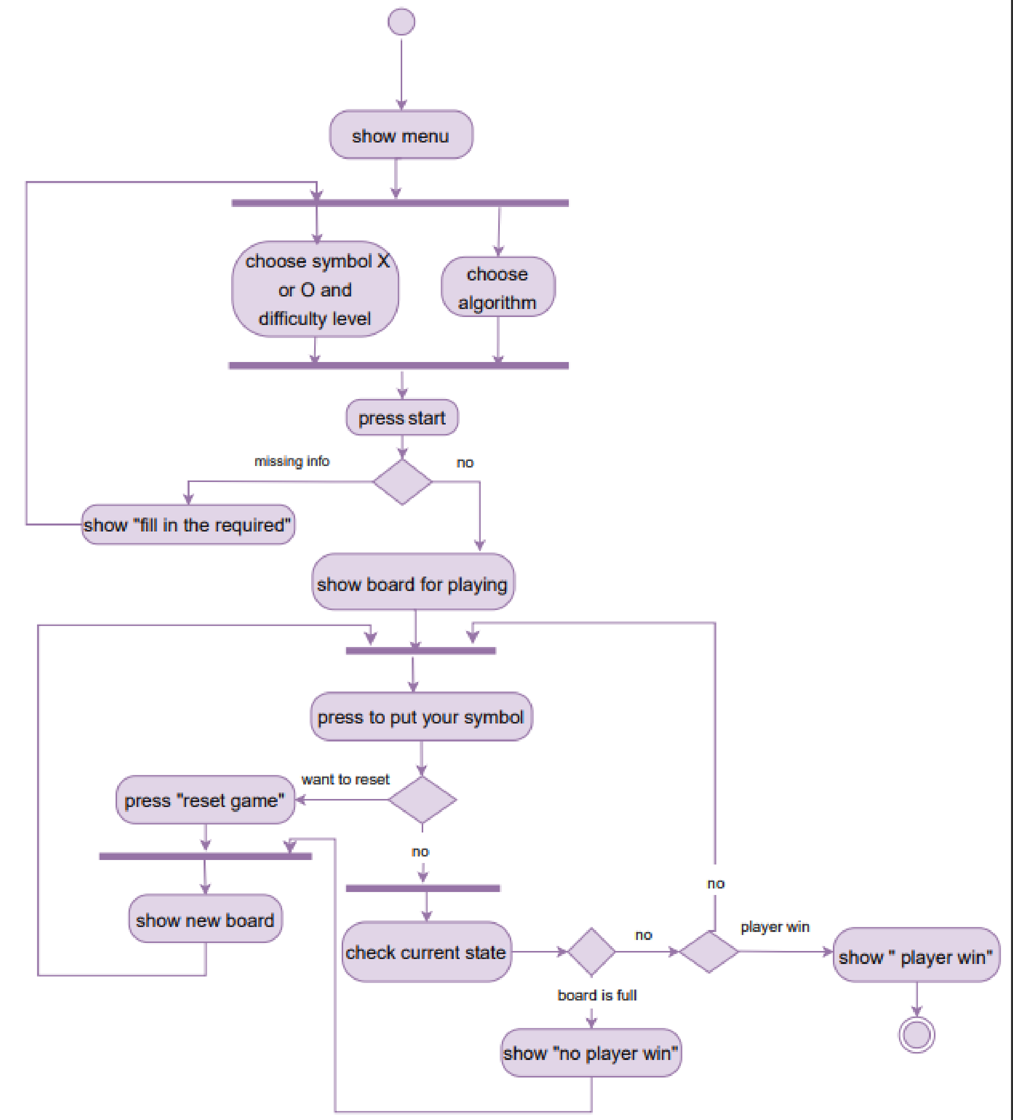
Minimax with Heuristic is an extension of the classic Minimax algorithm, incorporating a heuristic evaluation function to estimate the desirability of a game state when the maximum depth of the search tree is reached. This modification allows the algorithm to make informed decisions even when it cannot explore the entire game tree.

Alpha-Beta Pruning with Heuristic is an enhancement to the Alpha-Beta Pruning algorithm, incorporating a heuristic evaluation function to improve the efficiency of the search for optimal moves in game trees. This modification aims to reduce the number of nodes evaluated by the algorithm while still making informed decisions

Class diagram:

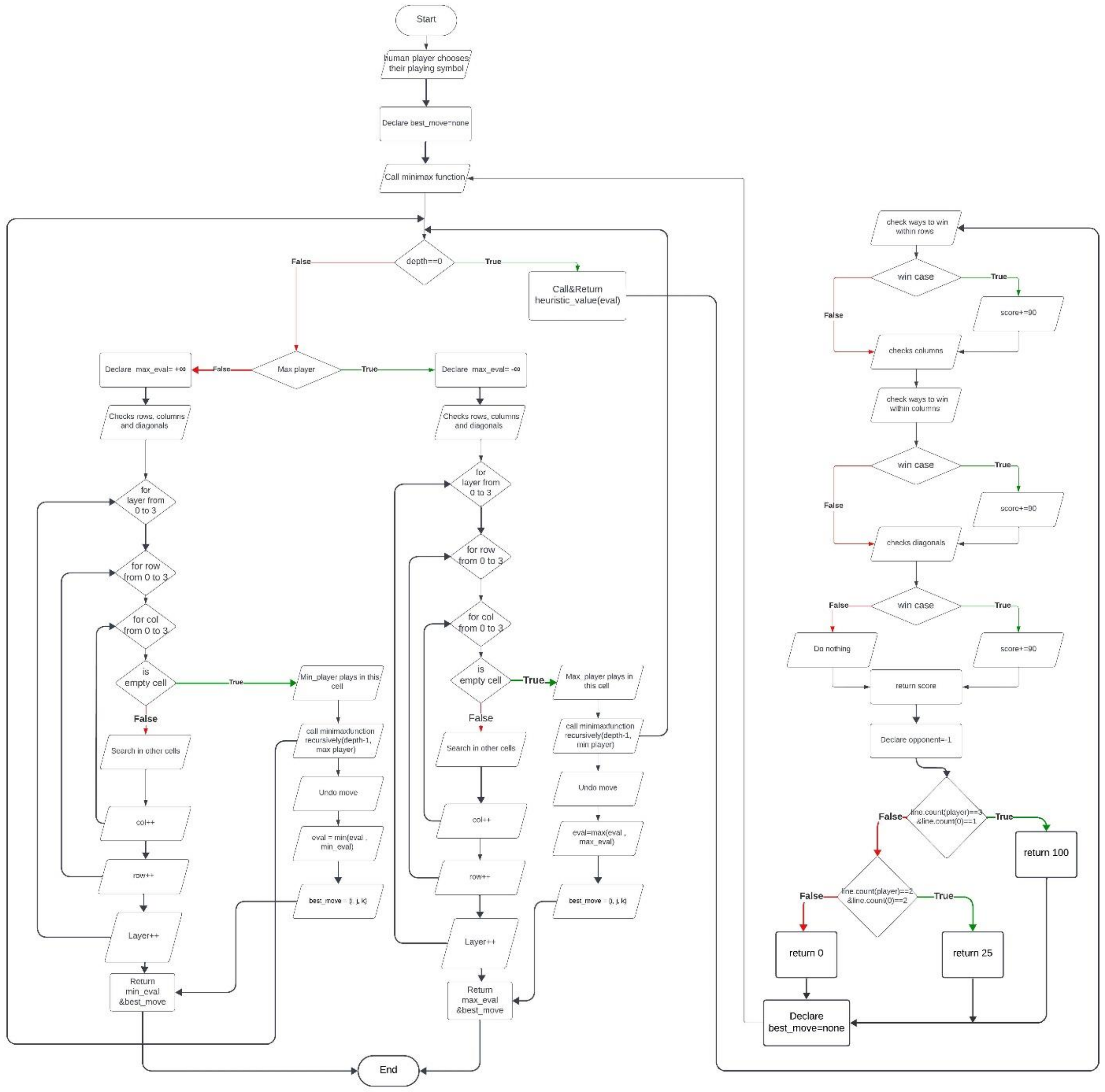


Activity diagram:

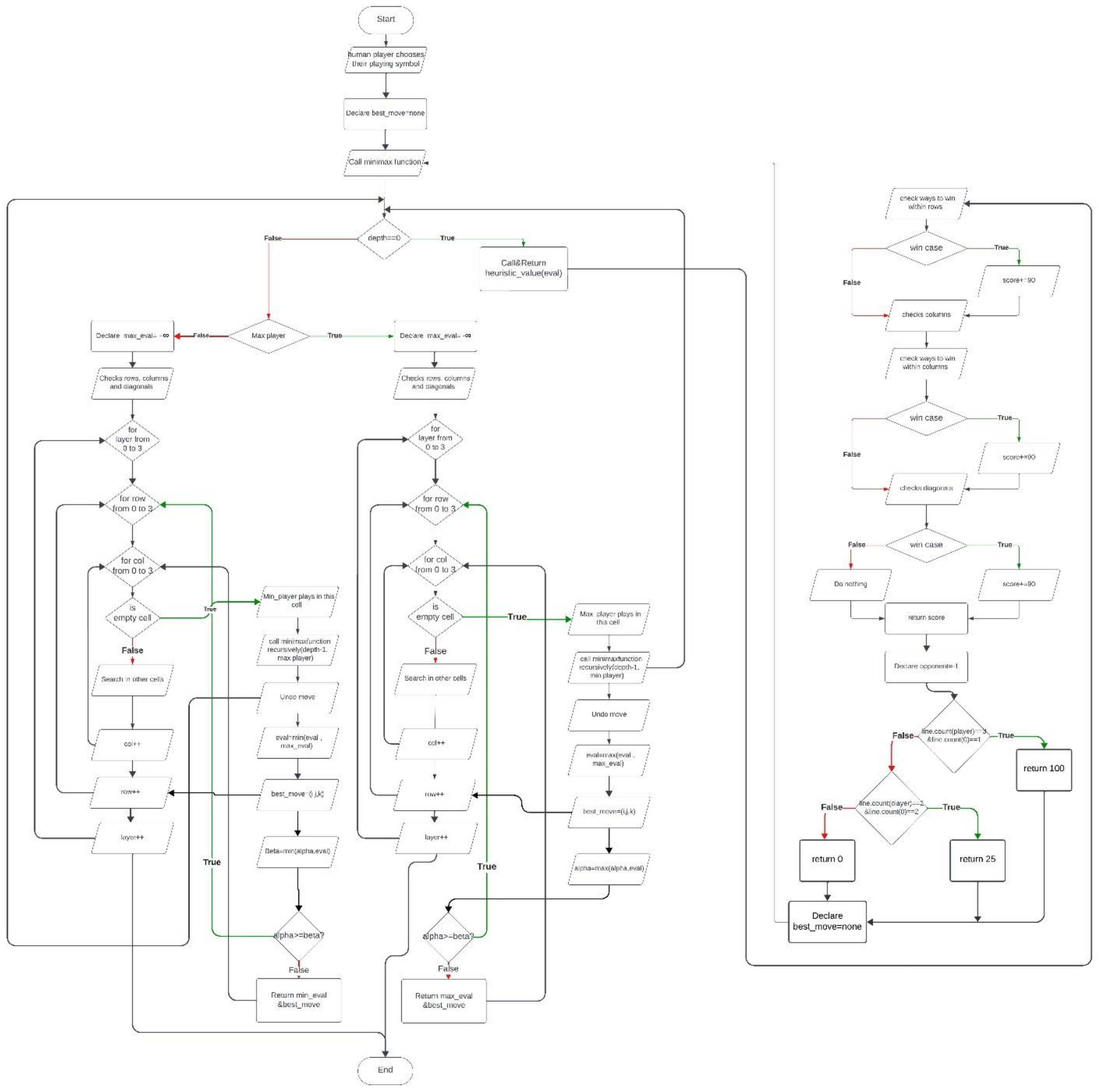


Flowchart:

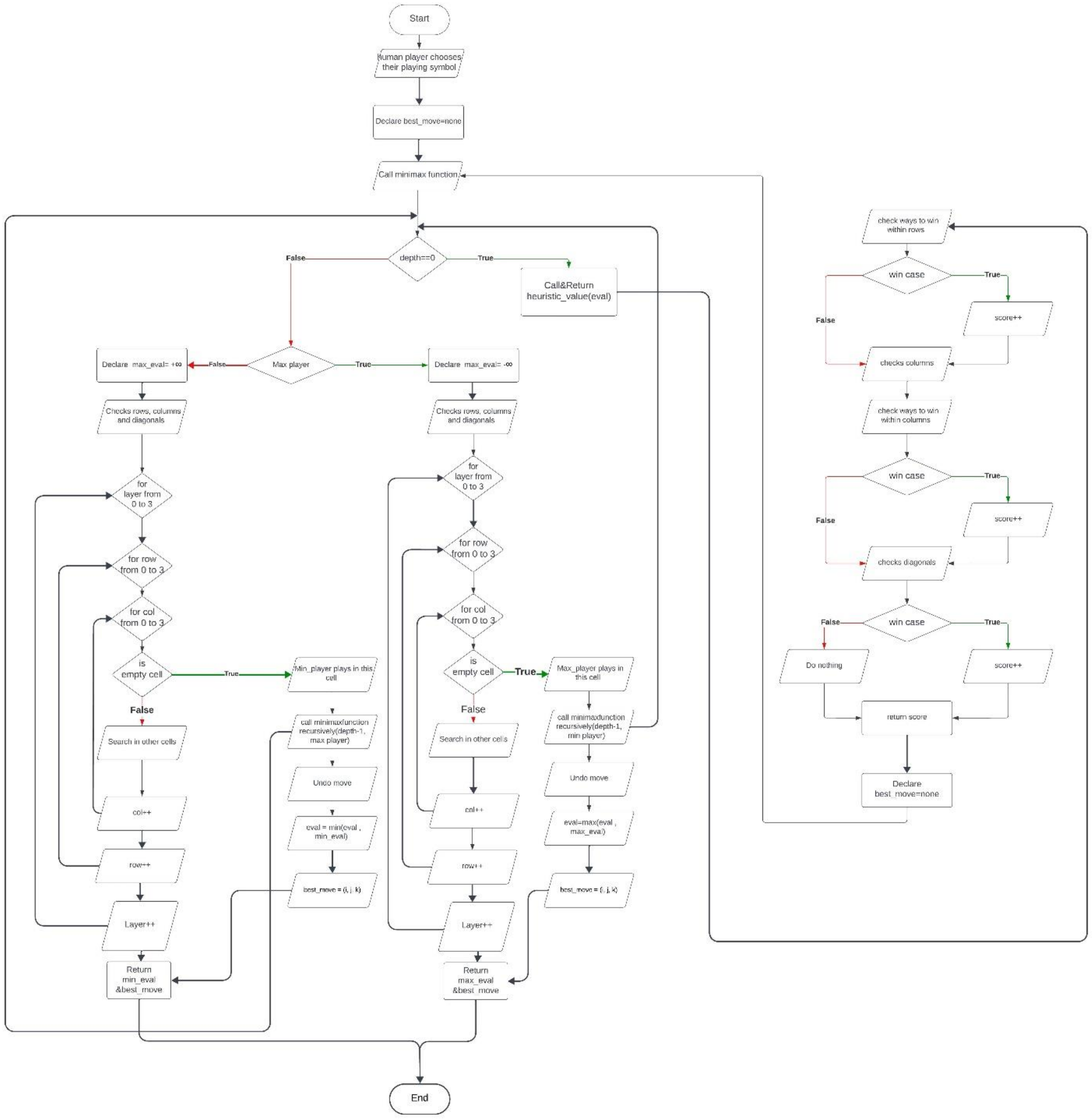
1)minimax with heuristic function1



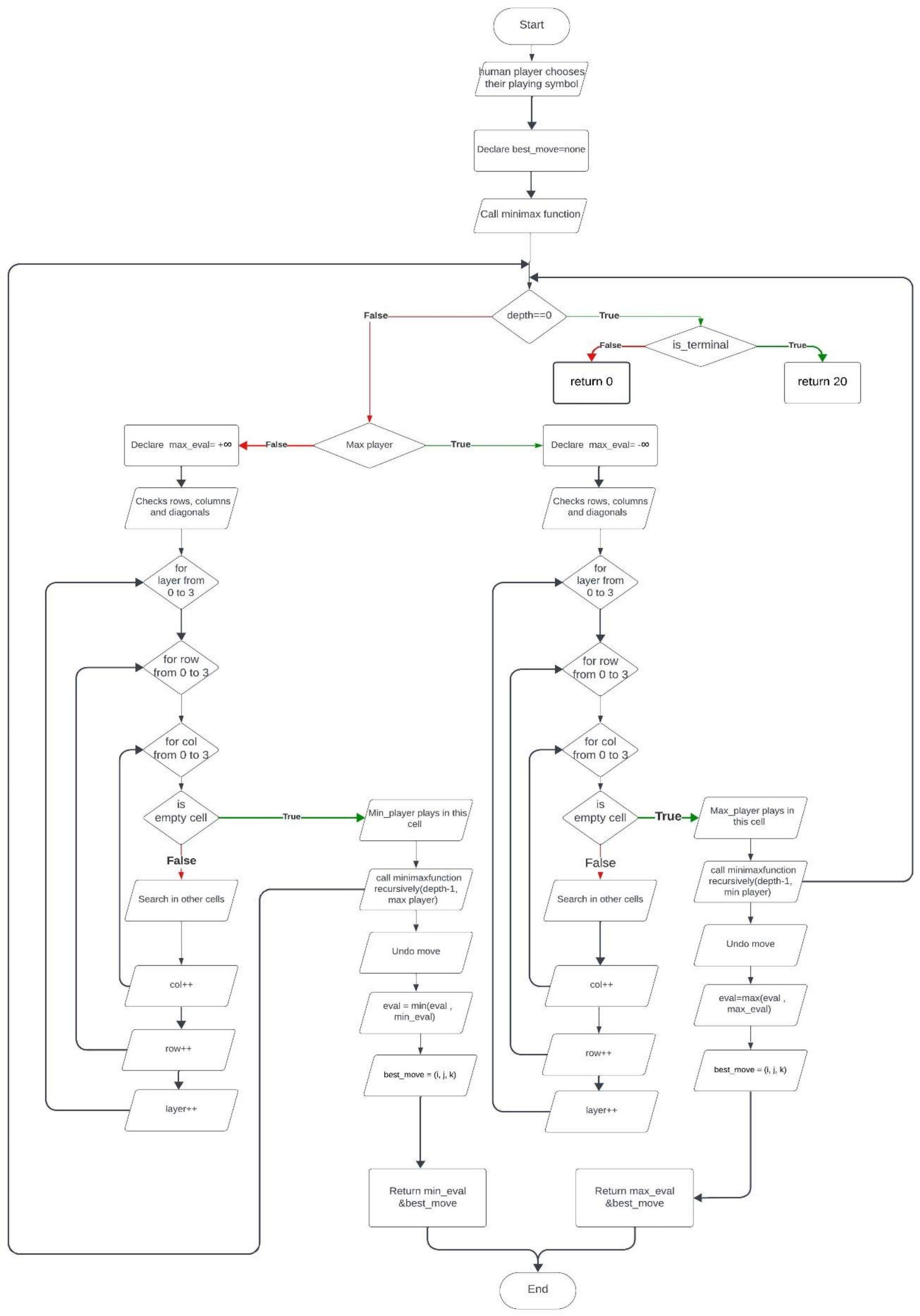
2)Alpha-Beta with heuristic function1


Start
human player chooses their playing symbol
Declare best_move=none
Call minimax function
depth==0
False
True
Call&Return heuristic_value(eval)
Max player
Declare max_eval=-∞
Checks rows, columns and diagonals
for layer from 0 to 3
for row from 0 to 3
for col from 0 to 3
is empty cell
Min_player plays in this cell
call minimaxfunction recursively(depth-1, max player)
Undo move
eval=min(eval , max_eval)
best_move=(i,j,k)
Beta=min(alpha,eval)
alpha>=beta?
Return min_eval &best_move
Search in other cells
col++
row++
layer++
Max_player plays in this cell
call minimaxfunction recursively(depth-1, min player)
eval=max(eval , max_eval)
alpha=max(alpha,eval)
Return max_eval &best_move
End
check ways to win within rows
win case
score+=90
checks columns
check ways to win within columns
checks diagonals
Do nothing
return score
Declare opponent=1
line.count(player)==3 &line.count(0)==1
return 100
line.count(player)==2 &line.count(0)==2
return 0
return 25
Declare best_move=none

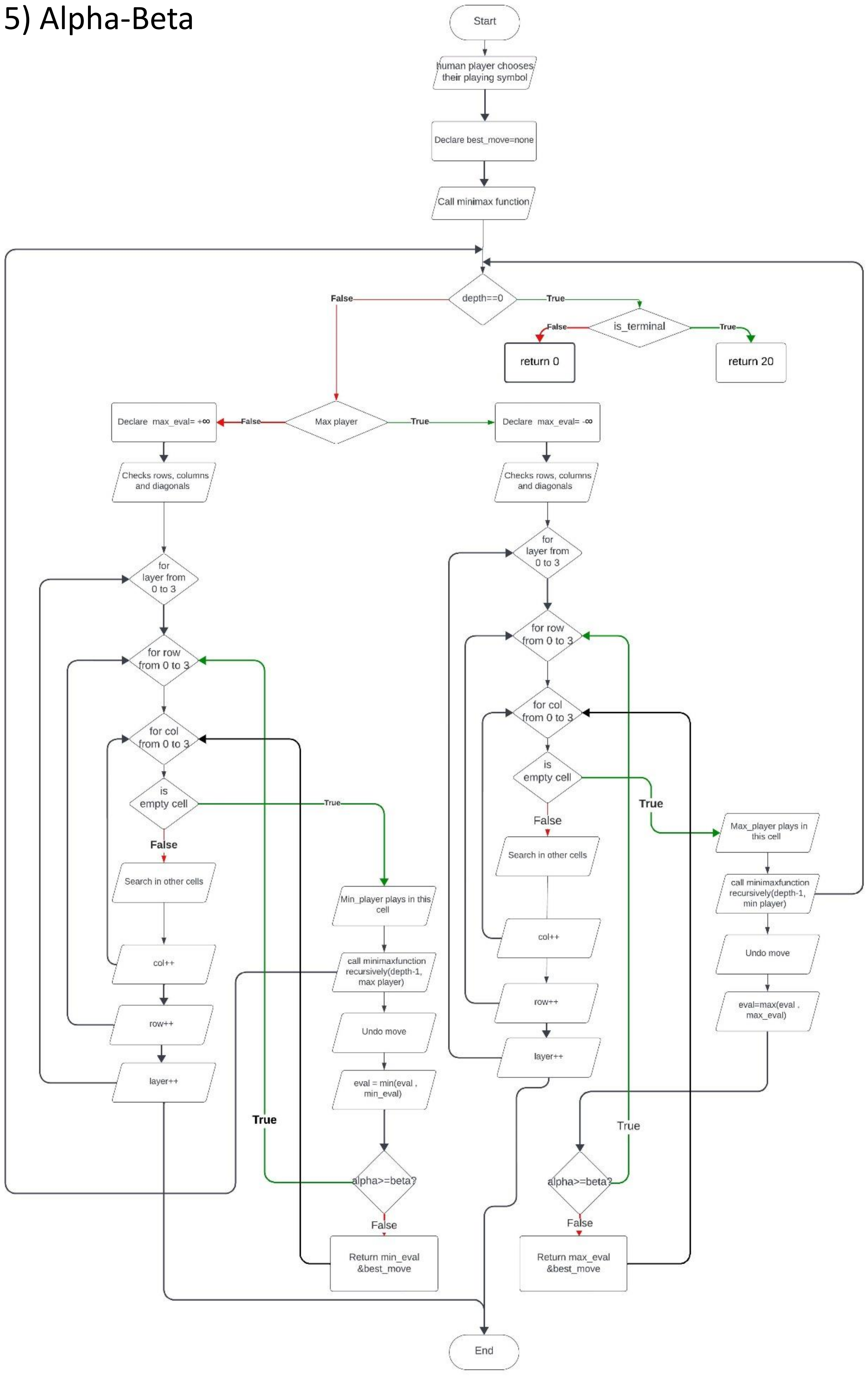
3) Minimax with heuristic function 2


Start
Human player chooses their playing symbol
Declare best_move=none
Call minimax function
depth==0
False
True
Call&Return heuristic_value(eval)
Max player
False
True
Declare max_eval= +∞
Declare max_eval= -∞
Checks rows, columns and diagonals
for layer from 0 to 3
for row from 0 to 3
for col from 0 to 3
is empty cell
True
False
Min_player plays in this cell
call minimaxfunction recursively(depth-1, max player)
Undo move
eval = min(eval , min_eval)
best_move = (i, j, k)
Search in other cells
col++
row++
Layer++
Return min_eval &best_move
Max_player plays in this cell
call minimaxfunction recursively(depth-1, min player)
eval=max(eval , max_eval)
Return max_eval &best_move
check ways to win within rows
win case
score++
checks columns
check ways to win within columns
checks diagonals
Do nothing
return score
Declare best_move=none
End

4) Minimax


Start
human player chooses their playing symbol
Declare best_move=none
Call minimax function
depth==0
False
True
is_terminal
False
True
return 0
return 20
Max player
False
True
Declare max_eval= +∞
Declare max_eval= -∞
Checks rows, columns and diagonals
for layer from 0 to 3
for row from 0 to 3
for col from 0 to 3
is empty cell
True
False
Search in other cells
col++
row++
layer++
Min_player plays in this cell
call minimaxfunction recursively(depth-1, max player)
Undo move
eval = min(eval , min_eval)
best_move = (i, j, k)
Return min_eval &best_move
Checks rows, columns and diagonals
for layer from 0 to 3
for row from 0 to 3
for col from 0 to 3
is empty cell
True
False
Search in other cells
col++
row++
layer++
Max_player plays in this cell
call minimaxfunction recursively(depth-1, min player)
Undo move
eval=max(eval , max_eval)
best_move = (i, j, k)
Return max_eval &best_move
End

5) Alpha-Beta


Start
human player chooses their playing symbol
Declare best_move=none
Call minimax function
depth==0
False
True
is_terminal
False
True
return 0
return 20
Max player
False
True
Declare max_eval= +∞
Declare max_eval= -∞
Checks rows, columns and diagonals
Checks rows, columns and diagonals
for layer from 0 to 3
for layer from 0 to 3
for row from 0 to 3
for row from 0 to 3
for col from 0 to 3
for col from 0 to 3
is empty cell
is empty cell
True
True
False
False
Search in other cells
Search in other cells
Min_player plays in this cell
Max_player plays in this cell
call minimaxfunction recursively(depth-1, max player)
call minimaxfunction recursively(depth-1, min player)
col++
col++
row++
row++
layer++
layer++
Undo move
Undo move
eval = min(eval , min_eval)
eval=max(eval , max_eval)
True
True
alpha>=beta?
alpha>=beta?
False
False
Return min_eval &best_move
Return max_eval &best_move
End

literature review:

Paper one: minimax algorithm

The full game tree has a root representing the initial layout of the game and vertices and edges representing all possible moves to the end of the game. All possible moves for the current player are children nodes of the root and then all moves available to the next player are children nodes of these nodes, and so forth. Each branch of the tree represents a possible move that player could make at a given point in the game. Evaluating the game at a leaf of the game tree yields the papered status of the game after that sequence of moves is made by the players. Game tree search is aimed at finding optimal strategy for the game. The algorithm assumes that the second player tries to minimize the gain of the first player, while the first player tries to maximize his gain, hence the name of the algorithm. The game tic-tac-toe is a simple game in which two players, represented as O and X, alternate in marking spaces on a 3x3 grid . The game tree of tic-tac-toe with the possible combinations of the first two moves with symmetrical positions omitted for simplicity. The min-max algorithm traverses the entire tree in a depthfirst fashion, and depending on whether a node is maximizing or minimizing, the algorithm keeps track of the largest or the smallest score, respectively. When a leaf node is reached, its score is determined by an evaluation function. depicts the min-max algorithm. Since Min-max explores every node in the game tree, the algorithm is not practical for a game tree with many branches or depths .

Paper two: Alpha-Beta Algorithm

The min-max algorithm can has been improved by proposing an efficient algorithm, alpha-beta, for sequential game tree search. The idea to cut-off unnecessary branches is to keep two scores in the search. The first one is alpha (lower bound), which keeps track of the highest score obtained at a maximizing node higher up in the tree and is used to perform pruning at minimizing nodes. Any move made from the maximizing node with score less than or equal to alpha is of no improvement and can be pruned, because there is a strategy that is known to result in a score of alpha. The second score is beta (upper bound, which keeps track of the lowest score obtained at a minimizing node higher up in the tree and is used to perform pruning at maximizing nodes. Beta can be viewed as the worst-case scenario for the opponent, because there is a way for the opponent to force a situation no worse than beta. If the search finds a move that returns a score of beta or greater, the rest of the legal moves do not have to be searched, because there is some choice the opponent will make to prevent that move from happening. The resulting algorithm, called alpha-beta algorithm

Paper three: The performance of search algorithms as AI gameplay agents for two-player quantum games, especially Quantum Chess, is explored in this thesis. Before applying the search algorithms in the quantum domain, their advantages and disadvantages is studied in the context of a simple classical game. Tic-tac-toe is a ubiquitous game that can be played on a piece of paper or in electronic formats. Two search algorithms have been adapted for the game of tic-tac-toe: Minimax and Monte Carlo Tree Search (MCTS). According to study, Minimax is a more effective algorithm than MCTS for a game with small search depth like tic-tac-toe. The MCTS algorithm has many knobs to turn to tune its effectiveness. To determine the optimized parameter range for the MCTS such that it compares well with the Minimax algorithm, an extensive comparative analysis between Minimax and MCTS algorithm is carried out. The investigations from this thesis and further applications are also applicable to quantum games. The performance of Minimax as applied to the game of quantum chess is analyzed through a few simple puzzles.

Paper four: The literature presents several connections between games and one or more learning theories, such as cognitivism or social constructivism. The cognitivist paradigm of Piaget's theory states that learning and knowledge occur through interaction with one's environment and culture (Butts and Brown 1989). The other significant influence of social constructivism is associated with the developmental theories of Vygotsky. The primacy of social constructivism emphasized social interaction as the driving force and prerequisite to individuals' cognitive development. In collaborative group work, it is crucial to the development of thought that members arrive at a mutual understanding of the topic. Vygotsky assumes that the language of the learners' development occurs from the result of working together in activities involving problem-solving (Schinke-Llano 1993). Through social interaction or discussion, the experience can further encourage higher mental functions, such as critical thinking skills. Tic-Tac-Toe As explained by Crowley and Siegler (1993), tic-tac-toe is a simple game where two players take turns to draw symbols (Xs or Os) on a three-by-three grid. Furthermore, the player has to place three symbols in a row, column, or diagonal to win the game. It appears to be simple, but drawing from other studies, the use of tic-tac-toe for game-based learning could result in enhancement and an active learning environment in classrooms. For instance, Honarmand, Rostampour, and Abdorahimzadeh (2015) carried out research on game-based learning that involved the use of tictac-toe and flashcards. Although the focus of this study was on students' performance in vocabulary instead of mathematics, their findings showed that game-based learning had increased the students' performances through challenges and motivation. The use of tic-tac-toe in game-based learning in mathematics is rather scant. One study by Ke (2008) explored the impact of digital game-based learning toward students' performances in mathematics, where tic-tac-toe had been utilized as one of the many games in the lesson intervention. The results of the study indicated that computer math drill games significantly improved students’ positive attitudes toward mathematics learning (Ke 2008). Furthermore, based on Ke’s (2008) findings, the students were reported to have performed all the given tasks enthusiastically despite playing certain games where math drills were the main focus of the gameplay or were appropriately challenging.

Paper five: In domains with multiple competing goals, people face a basic challenge: How to make their strategy use flexible enough to deal with shifting circumstances without losing track of their overall objectives. This article examines how young children meet this challenge in one such domain, tic-tac-toe. Experiment 1 provides an overview of development in the area; it indicates that children's tic-tac-toe strategies are rule based and that new rules are added one at a time. Experiment 2 demonstrates that even young children flexibly tailor their strategy use to meet shifting circumstances. Experiment 3 indicates that these adaptations are accomplished through a process of goal-based resource allocation, whereby children focus their cognitive resources on applying rules most consistent with their current primary goal. A computer simulation specifies how this process works and demonstrates its sufficiency for producing behavior much like that of the children. The findings are discussed as part of a broader framework of mechanisms for generating problem-solving approaches.

Paper six: China and the United States (US) have established respective cultural institutions in each other, namely the Confucius Institute and the American Cultural Centre. While they differ in the establishment background and management model, in their counterpart countries’ view, both Confucius Institutes and American Cultural Centres serve the function of public diplomacy as a ‘quasi-diplomatic agency’, attracting extensive political attention. When comparing the history of their developments in the US and China, a similar trend of ‘rise-to-fall’ and ‘positivity-to-negativity’ can be observed. The stagnation of American Cultural Centres in China and the crisis of Confucius Institutes in the US both originate from the same source — political pressure. The development paths of both institutions are closely related to the development and transformation of Sino–US political relations. In this process, China’s attitude is relatively moderate in an attempt to maintain the continuous development of a friendly bilateral relation, while the US is anxious about China's rising status as its economic and strategic competitor, thereby adjusting its China policy and imposing rigid political controls over Confucius Institutes. Affected by the Sino–US trade dispute and the COVID-19 pandemic, Sino–US relations deteriorate further; accordingly, the future outlook of Confucius Institutes in the US is worrying. However, with abundant uncertainties, the future of the Confucius Institute remains to be further studied.

Analysis of the results, what are the insights?

From the best to the worst:

1- Alpha-Beta with Heuristics

2- Minimax with Heuristics

3- Alpha-Beta Pruning:

4-Minimax

Minimax algorithm

Advantages:

1. Guarantees optimal play: Minimax ensures that the AI player makes the best move possible given the current state of the game, assuming the opponent also plays optimally.
2. Works well for games with manageable state space: Tic Tac Toe has a relatively small state space, making it feasible to calculate the entire game tree and find the optimal move.

Disadvantages:

1- Computationally expensive for larger games: In games with larger state spaces, such as chess or Go, the minimax algorithm becomes computationally expensive due to the exponentially increasing number of possible moves and states.
2- Limited to perfect information games: Minimax requires complete knowledge of the game state and all possible moves, which makes it unsuitable for games with hidden information or uncertainty.
3- May not scale well for complex games: While suitable for Tic Tac Toe, minimax can struggle to scale to more complex games due to the exponential growth in the game tree, leading to increased computational demands.

Alpha-beta:

Advantages:

1- Improved efficiency over minimax: Alpha-Beta pruning reduces the number of nodes evaluated in the game tree, leading to faster computation compared to the basic minimax algorithm.
2- Effective for games with limited depth: In games like Tic Tac Toe with a relatively shallow game tree, Alpha-Beta pruning can significantly reduce the number of nodes evaluated.

Disadvantages:
quality of solution not the best

Minimax with heuristic:

Advantages:

1- Reduced search space: By using a heuristic evaluation function, the Minimax algorithm can focus on more promising moves, reducing the depth of the search tree and making it more efficient.
2- Better scalability: With a well-designed heuristic, the algorithm can scale better to larger games or deeper search depths by guiding the search towards more relevant parts of the game tree.
3- Allows for imperfect information: Heuristic evaluation functions can handle imperfect information or incomplete game states by approximating the value of a position based on observable features.

Disadvantages:

1- Risk of suboptimal moves: Heuristic evaluation functions are not perfect and can lead to suboptimal decisions if the heuristics do not accurately reflect the true value of a game state.

alpha-beta with heuristic:
Advantages:

2- Improved efficiency: Alpha-Beta pruning reduces the number of nodes evaluated in the game tree, and when combined with a heuristic evaluation function, it can further reduce the search space, leading to faster computation.
3- Deeper search: With a more efficient search, Alpha-Beta pruning with a heuristic evaluation function can potentially explore deeper into the game tree, allowing for better decision-making in complex game states.
4- Better scalability: The combination can be more scalable than pure Minimax with heuristics, as the pruning helps manage the increased complexity of larger game trees.

Disadvantages:
Difficulty in balancing depth and accuracy: In some cases, the use of heuristics with Alpha-Beta pruning can make it challenging to balance the depth of the search (i.e., how far into the game tree the algorithm explores) with the accuracy of the evaluations. Deeper searches might sacrifice accuracy if the heuristic is not sophisticated enough to provide meaningful evaluations at deeper levels.

Heuristic accuracy: The effectiveness of the combined approach heavily relies on the accuracy of the heuristic evaluation function. If the heuristic does not accurately estimate the value of game states, the algorithm might make suboptimal or outright incorrect decisions.