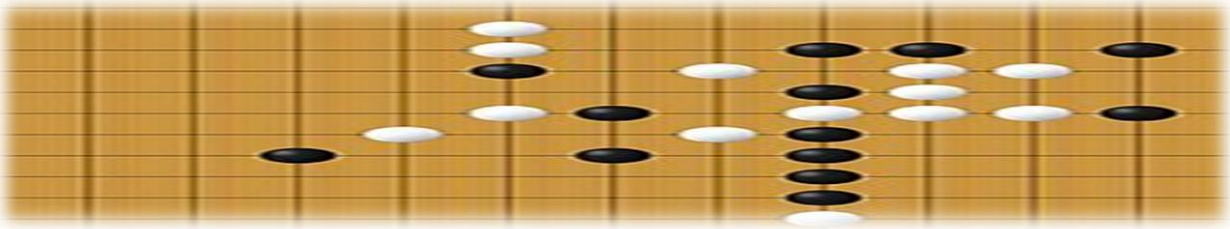
Pente



➢ Project Idea and Overview

- **Pente** is a strategy board game that involves capturing stones and forming lines of five stones in a row. The project involve developing a digital version of Pente, which can be played on desktop, web, or mobile platforms. The game would include features such as single-player mode against AI, multiplayer mode, and possibly online multiplayer with matchmaking.

😐 About the Project :

- This repository contains an AI agent implementation for the game **Pente**. The AI uses the **Minimax algorithm with alpha-beta pruning** to evaluate possible moves and make strategic decisions. The heuristic function is designed to:
  - ✓ ****Block**** the opponent's opportunities for 3-stone or 4-stone sequences.
  - ✓ ****Prioritize**** moves to create its own 3-stone or 4-stone sequences.
  - ✓ ****Optimize**** gameplay performance with efficient board evaluation.

😐 Features :

- ✓ ****Minimax Algorithm**:** Implements depth-limited search with alpha-beta pruning.
- ✓ ****Advanced Heuristic Function**:** Evaluates board states to detect potential threats and opportunities.
- ✓ ****Defensive and Offensive Strategies**:** Blocks opponent's potential winning moves and sets up AI's own winning paths.
- ✓ Configurable board size and depth for customization

➢ Applications Similar to Pente

Here are some applications similar to Pente and their functionalities:

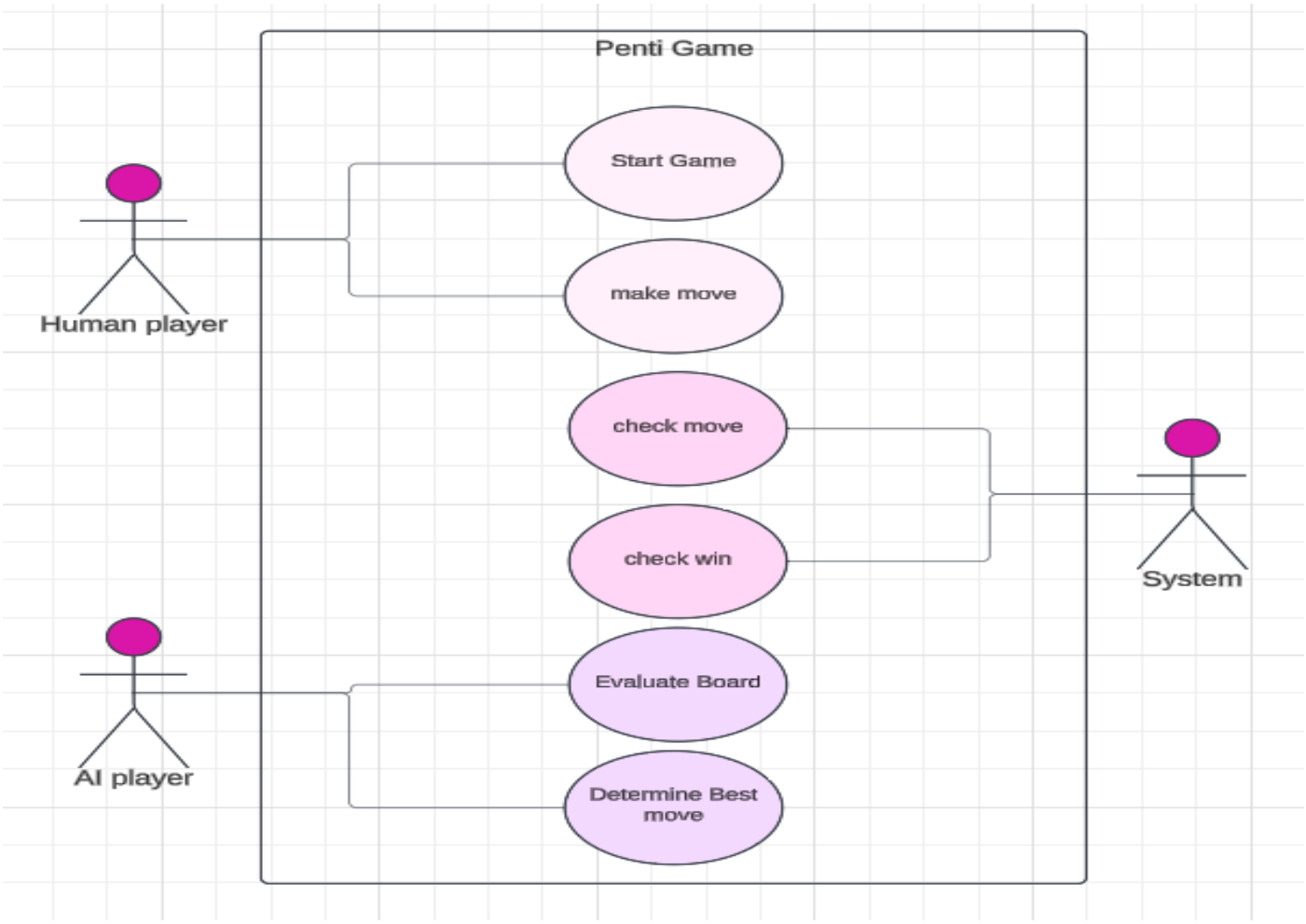
- ➔ **Gomoku:**
  - **Platform:** Desktop, Web, Mobile
  - **Features:** Single-player against AI, multiplayer, online multiplayer, different difficulty levels, and a hint system.
  - **How it works:** Players take turns placing stones on a grid, aiming to form a line of five stones.
- ➔ **Reversi (Othello):**
  - **Platform:** Desktop, Web, Mobile
  - **Features:** Single-player against AI, multiplayer, online multiplayer, various difficulty levels, and tutorials.
  - **How it works:** Players take turns placing stones, flipping the opponent's stones to their color by trapping them between two of their own stones.
- ➔ **Go:**
  - **Platform:** Desktop, Web, Mobile
  - **Features:** Single-player against AI, multiplayer, online multiplayer, different board sizes, and tutorials.
  - **How it works:** Players take turns placing stones on a grid, aiming to control the largest area of the board.

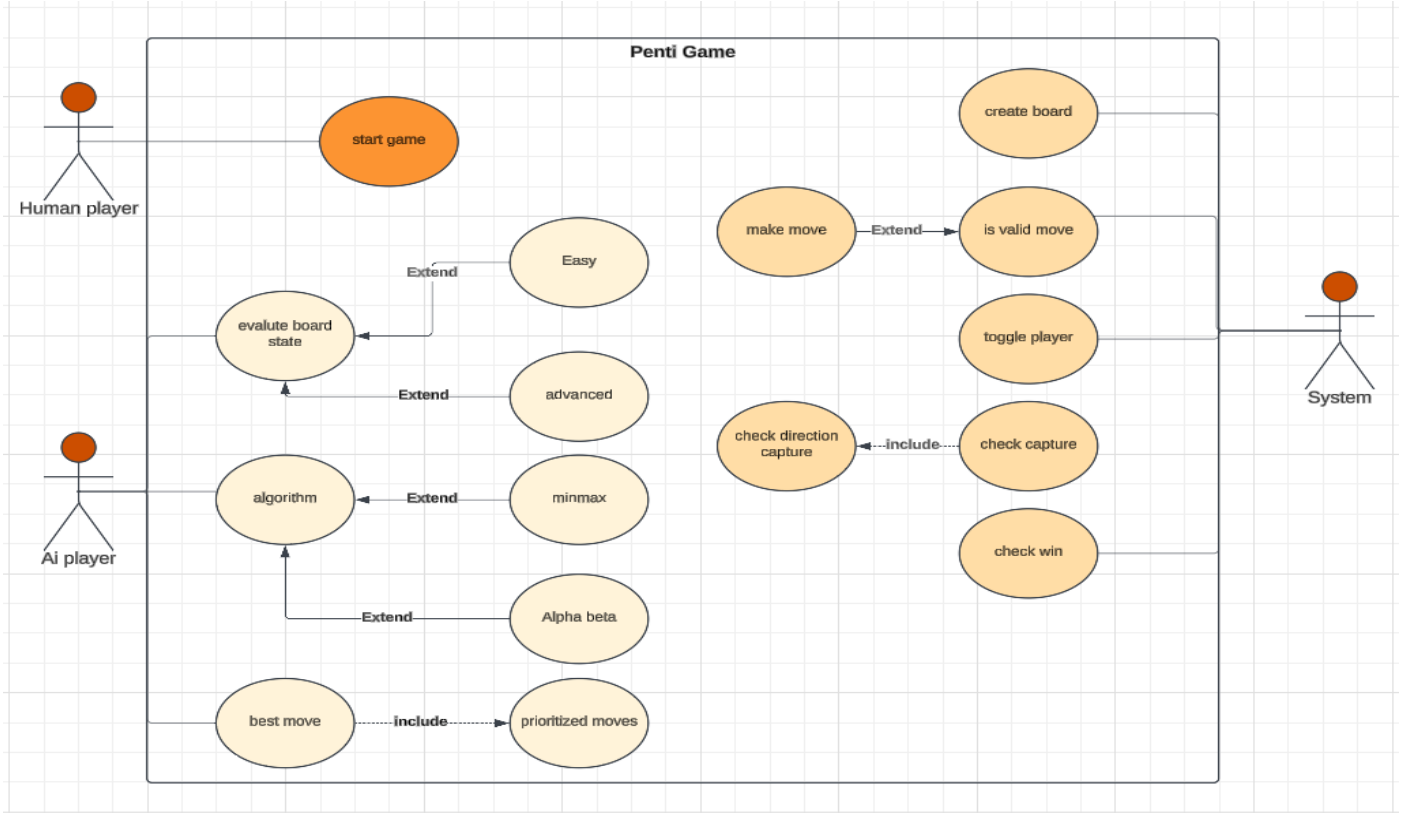
➢ Literature Review

Here are some academic publications relevant to the development of a digital Pente game:

- ➔ **"Artificial Intelligence in Board Games":**
  - **Summary:** This paper explores the use of AI in board games, discussing various algorithms and techniques used to create challenging AI opponents.
  - **Relevance:** Useful for developing the AI component of the Pente game.
- ➔ **"User Interface Design for Strategy Games":**
  - **Summary:** This article examines best practices in designing user interfaces for strategy games, focusing on usability and player engagement.
  - **Relevance:** Important for creating an intuitive and engaging interface for the Pente game.
- ➔ **"Multiplayer Game Design":**
  - **Summary:** This book covers the principles of designing multiplayer games, including matchmaking, player interaction, and network considerations.
  - **Relevance:** Essential for implementing the multiplayer features of the Pente game.
- ➔ **"Game Theory and Strategy":**
  - **Summary:** This publication delves into game theory and its applications in strategy games, providing insights into strategic decision-making.
  - **Relevance:** Can help in understanding the strategic elements of Pente and improving AI strategies.
- ➔ **"Mobile Game Development":**
  - **Summary:** This paper discusses the challenges and techniques in developing games for mobile platforms, including performance optimization and user experience.
  - **Relevance:** Crucial for ensuring the Pente game runs smoothly on mobile devices.

Use case:


Penti Game
Start Game
make move
check move
check win
Evaluate Board
Determine Best move
Human player
AI player
System


Penti Game
start game
Easy
Extend
evalute board state
Extend
advanced
algorithm
Extend
minmax
Extend
Alpha beta
best move
include
prioritized moves
create board
make move
Extend
is valid move
toggle player
check direction capture
include
check capture
check win
Human player
Ai player
System

Main functionalities/features (from the users' perspective) in your proposed software/solution

Pente game with an AI opponent. The primary functionalities from a user's perspective are:

1. Human vs. AI Gameplay:

- Users can play against an AI opponent with varying levels of difficulty.
- The AI opponent uses the minimax algorithm (potentially with alpha-beta pruning) to make moves.

2. Game Board:

- A visual representation of the Pente board is displayed.
- Users can interact with the board by clicking on empty cells to place their stones.

3. Game Rules:

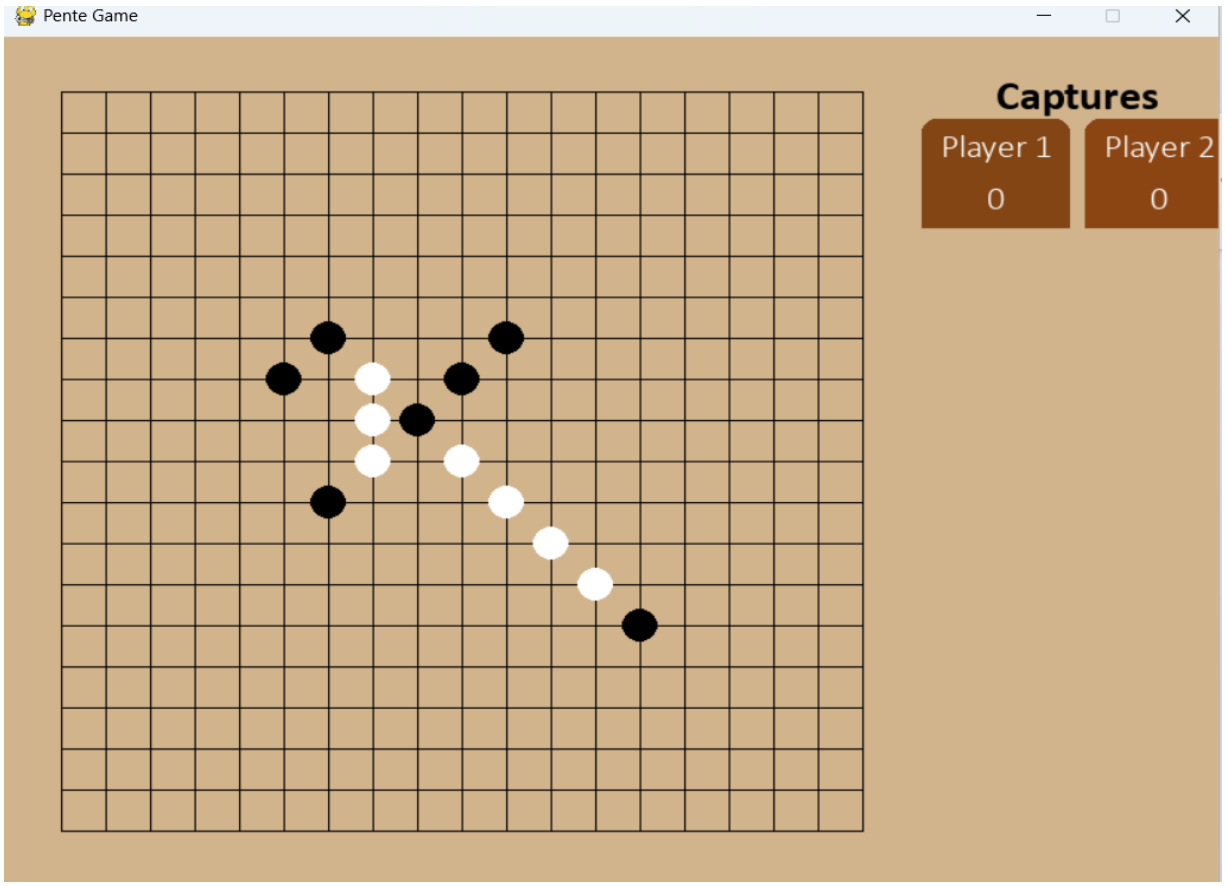
- The software enforces the rules of Pente, including:
- Placing stones on the board.
- Capturing opponent's stones.
- Winning conditions (five-in-a-row or five captures).

4. AI Difficulty:

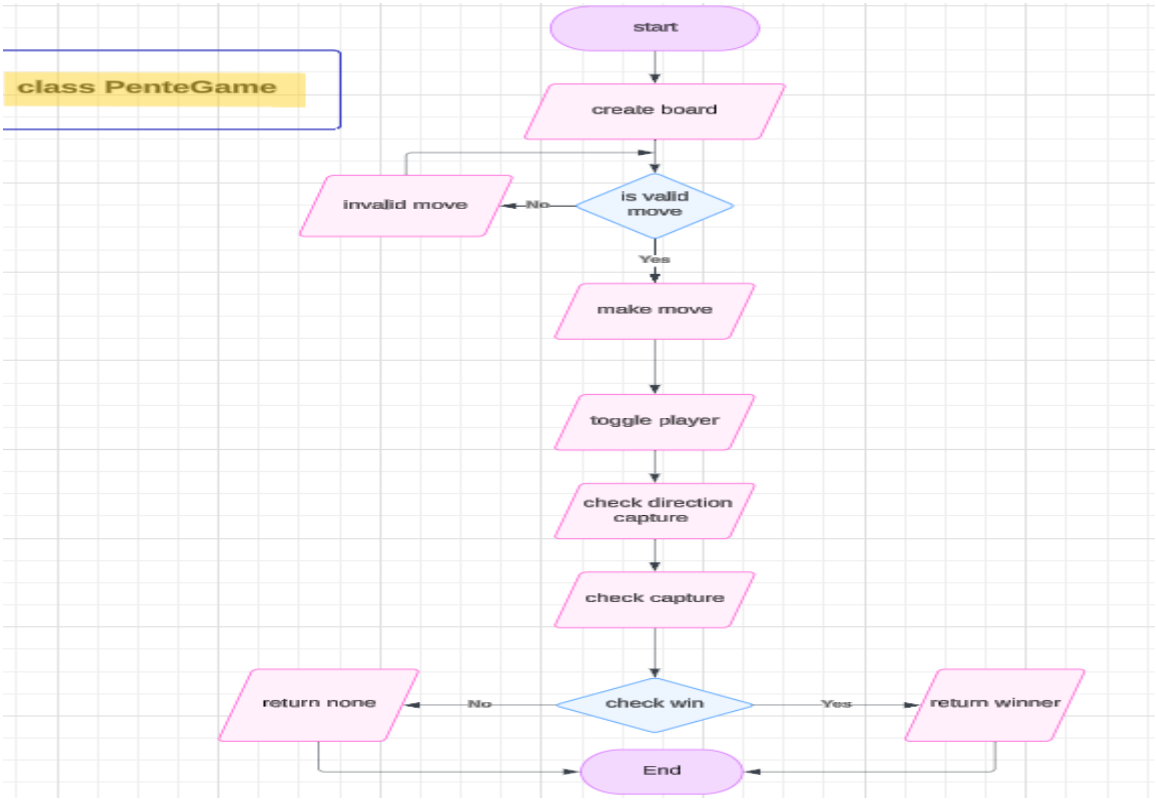
- Users can choose between different difficulty levels for the AI opponent, affecting the AI's decision-making process and search depth.

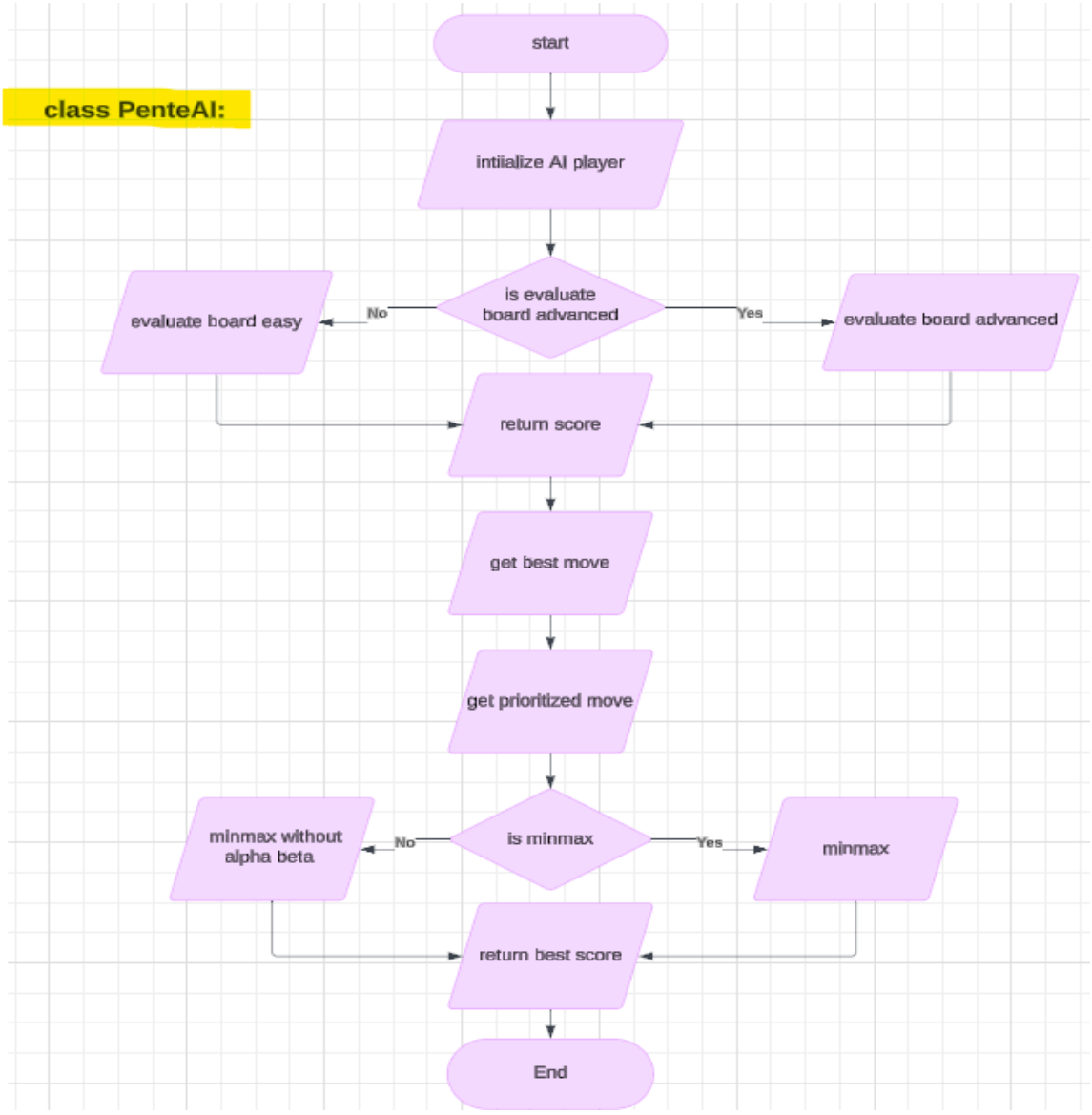
5. Win/Loss/Draw:

- The software determines the winner or if the game ends in a draw.
- It displays the outcome to the user.

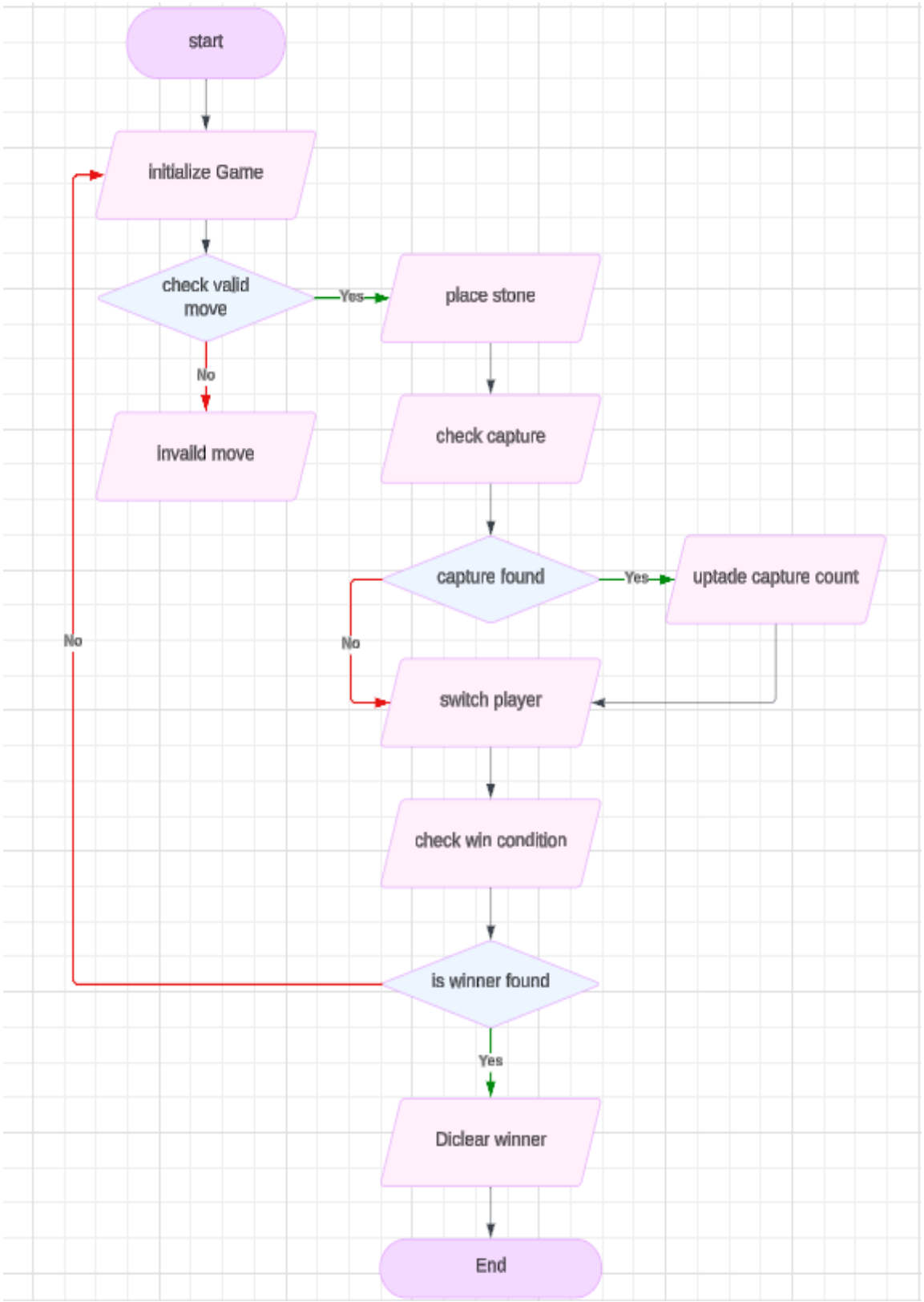


Flow chart :

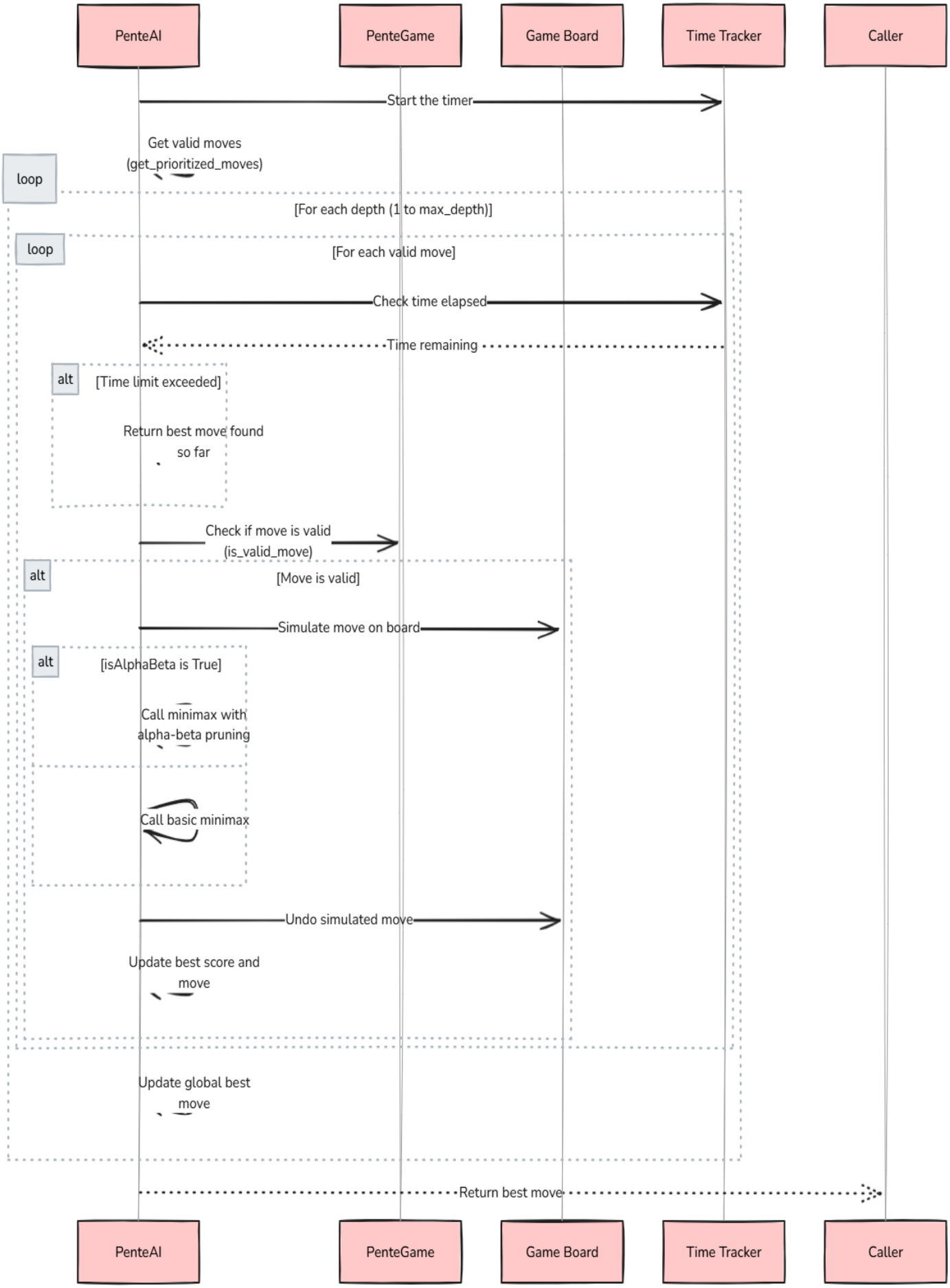

class PenteGame
start
create board
is valid move
No
invalid move
Yes
make move
toggle player
check direction capture
check capture
check win
No
return none
Yes
return winner
End


class PenteAI:
start
intiialize AI player
is evaluate board advanced
No
evaluate board easy
Yes
evaluate board advanced
return score
get best move
get prioritized move
is minmax
No
minmax without alpha beta
Yes
minmax
return best score
End

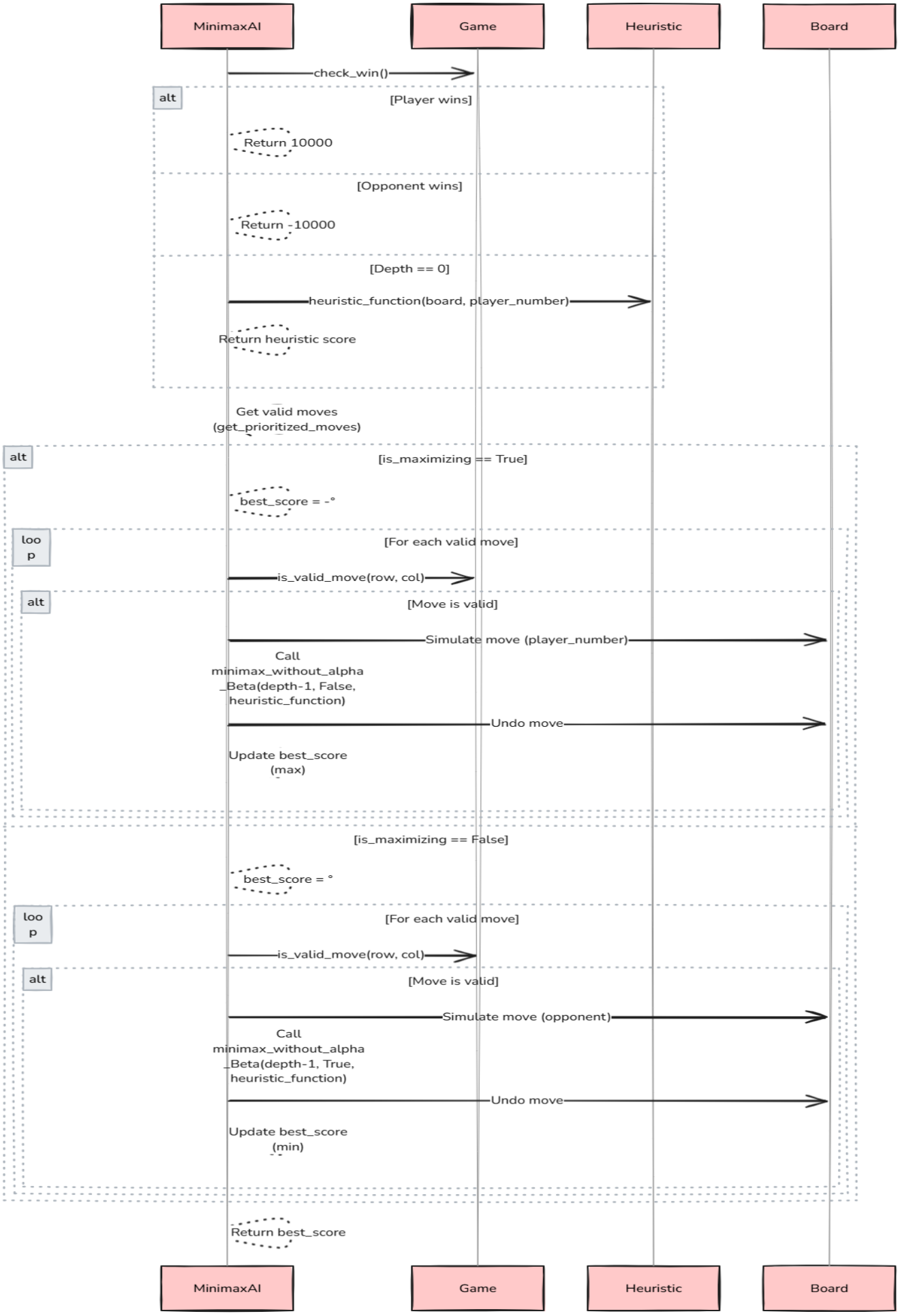
Flow chart :


start
initialize Game
check valid move
Yes
place stone
No
invalid move
check capture
capture found
Yes
uptade capture count
No
No
switch player
check win condition
is winner found
Yes
Diclear winner
End

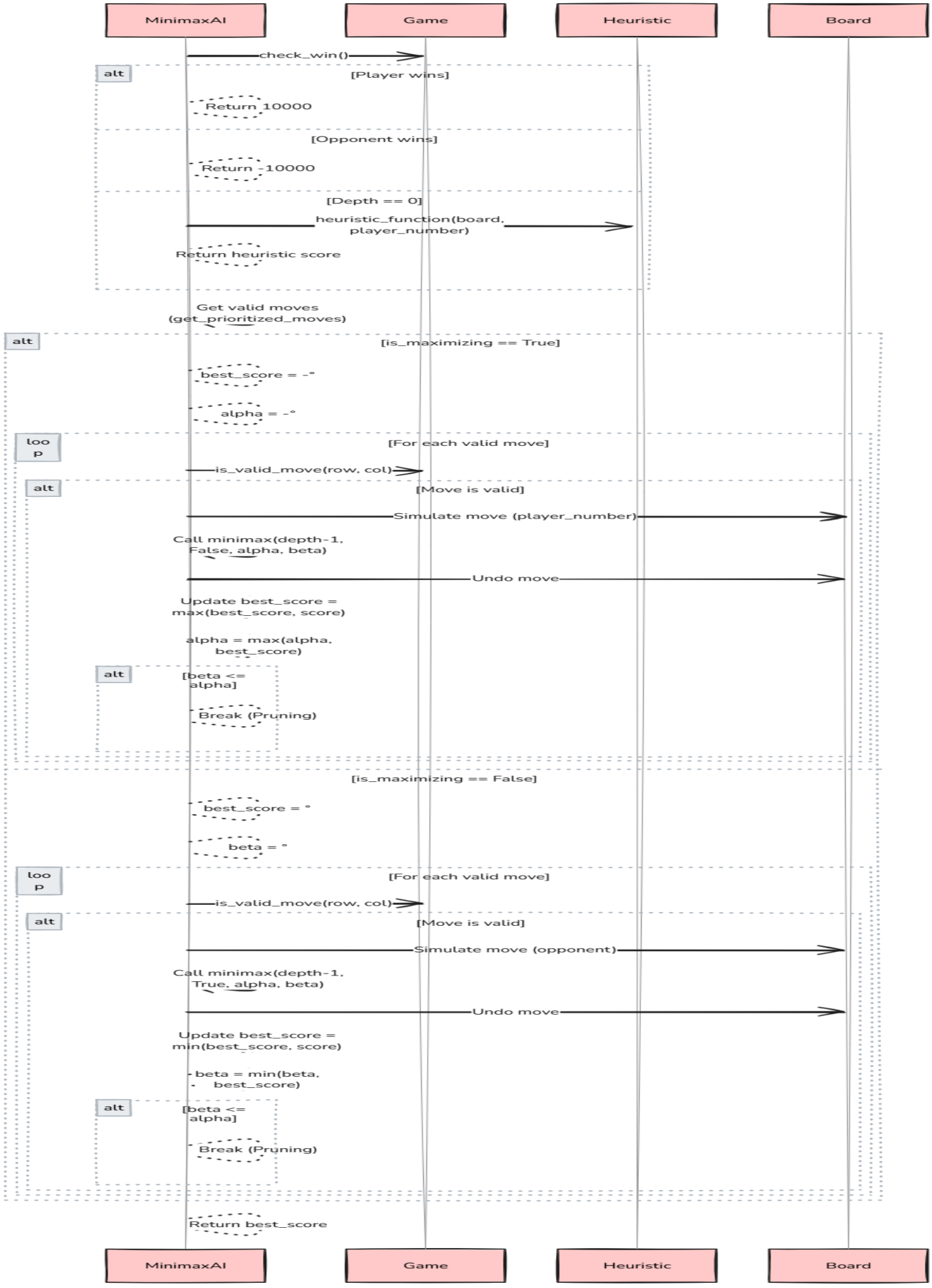
sequence diagram (Get best move)



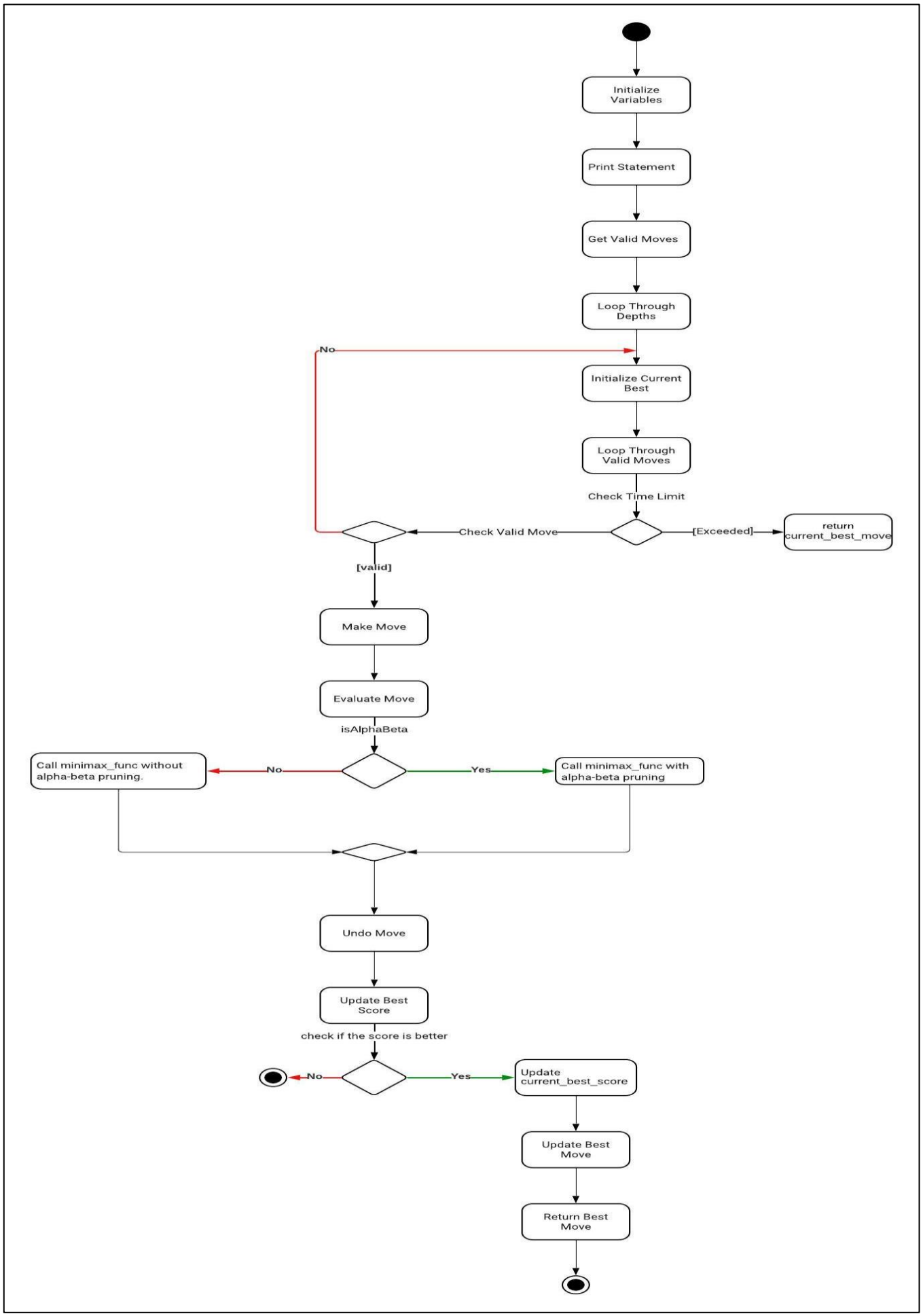
sequence diagram (minimax_without_alpha_Beta)



sequence diagram (minimax)



Activity diagram (Get best move)



Functions' descriptions:

Init(PenteGame) description	
Goal	Set up a clean board with default settings for a new game.
How	• Initializes the game board as a 2D list with all cells set to 0. • Sets the starting player (current_player) to 1. • Initializes capture counts (captures_p1 and captures_p2) to 0.
Outcome	Players begin with an empty board, and Player 1 starts.
Initiator	**System** (when a new game is started).
Preconditions	• board_size is an integer greater than 0.
Postconditions	• self.board_size is set to board_size. • self.board is a 2D list of size board_size x board_size with all elements set to 0. • self.current_player is set to 1. • self.captures_p1 and self.captures_p2 are set to 0.

is_valid_move description	
Goal	Ensure a player's move is within bounds and on an empty cell.
How	• Checks if the specified row and col are within the board size. • Verifies that the target cell (board[row][col]) is 0.
Outcome	Returns True if the move is valid, False otherwise.
Initiator	**Player** (via a move action).
Preconditions	• row and col are integers within the range of the board size.
Postconditions	• Returns True if the move is within bounds and the cell is empty. • Returns False otherwise.

make_move description	
Goal	Place a stone on the board, check for captures, and switch turns.
How	• Validates the move using is_valid_move. If invalid, returns False. • Updates the board by setting board[row][col] to the current_player value. • Calls check_captures to evaluate and remove captured stones, updating capture counts. • Invokes toggle_player to switch to the next player.
Outcome	Updates the board, modifies capture counts, and switches turns if the move is valid.
Initiator	**Player** (during their turn).
Preconditions	• row and col are valid integers within the board size. • The move is valid (checked by is_valid_move).
Postconditions	• If the move is valid: ➔ The stone is placed on the board. ➔ Captures are checked and updated. ➔ The current player is toggled. ➔ Returns True. • If the move is invalid: ➔ Returns False.

toggle_player description	
Goal	Alternate between Player 1 and Player 2 after a valid move.
How	Sets current_player to 3 - current_player (switches between 1 and 2).
Outcome	The current player is switched, and the next player is ready for their turn.
Initiator	**System** (internally called by make_move).
Preconditions	• self.current_player is either 1 or 2.
Postconditions	• self.current_player is either 1 or 2.

check_captures description	
Goal	Detect and remove pairs of opponent stones when they are surrounded by the active player's stones.
How	• Iterates over eight possible directions (horizontal, vertical, diagonal). • Calls check_direction_captures for each direction to identify and remove captured stones. • Updates the capturing player's capture count based on results.
Outcome	Captured stones are removed, and the capturing player's count increases.
Initiator	**System** (internally called by make_move).
Preconditions	• row and col are valid integers within the board size. • A stone has been placed at self.board[row][col].
Postconditions	• Captures in all directions are checked. • Captured stones are removed from the board. • Returns the total number of captures.

check_direction_captures description	
Goal	Examine a specific direction for two opponent stones surrounded by the active player's stones.
How	• Checks the board for a pattern: two consecutive opponent stones followed by the current player's stone. • If the pattern is detected, sets the opponent stones to 0 (removing them from the board). • Returns the number of captures detected (typically 1 for each pattern).
Outcome	Captures stones in that direction, if any, and updates the capture count.
Initiator	**System** (internally called by check_captures).
Preconditions	• row, col, dx, and dy are valid integers. • current_stone and opponent_stone are valid player numbers (1 or 2).
Postconditions	• Checks for a specific capture pattern in the given direction. • Removes captured stones if the pattern is found. • Returns the number of captures in this direction.

check_win description	
Goal	Determine if a player has won the game.
How	• Iterates through the board to check for: ➔ Five consecutive stones in a row, column, or diagonal for a player. ➔ A player achieving at least five captures. • If any condition is satisfied, returns the winning player's number (1 or 2).
Outcome	Returns the winning player or None if no winner yet.
Initiator	**System** (automatically after each move) or **Player** (to manually check the game's status).
Preconditions	• The game board is initialized and has valid player stones placed.
Postconditions	• Checks for five consecutive stones in a row, column, or diagonal for each player. • Checks if any player has achieved at least five captures. • Returns the winning player's number (1 or 2) if a win condition is met. • Returns None if no winner yet.

Init(PenteAI)description	
Goal	Initialize the AI player.
How	• Takes in the game instance and the AI's player number (1 or 2). • Sets the opponent's player number.
Outcome	Sets up the AI player with the game instance and player numbers.
Initiator	System (when creating an AI player instance).
Preconditions	• game is an instance of PenteGame. • player_number is an integer (1 or 2).
Postconditions	• self.game is set to the provided game instance. • self.player_number is set to the provided player number. • self.opponent is set to the opponent's player number (3 - player_number).

basic_evaluations description	
Goal	Evaluate the board state for basic patterns and captures.
How	• Iterates through the board to check for specific patterns and captures. • Uses predefined patterns to assign scores based on the presence of these patterns. • Calculates captures by checking sequences of opponent stones surrounded by the player's stones.
Outcome	Returns a score representing the board's evaluation for the given player.
Initiator	System (as part of the AI's decision-making process).
Preconditions	• board is a 2D list representing the game board. • player is an integer (1 or 2).
Postconditions	• Iterates through the board to check for specific patterns and captures. • Returns a score representing the board's evaluation for the given player.

evaluate_board_state_advanced description	
Goal	Provide an advanced evaluation of the board state considering multiple strategic aspects.
How	• Uses a combination of basic evaluations, capture evaluations, and positional evaluations. • Considers both offensive and defensive patterns. • Evaluates the control of the board's center and potential moves.
Outcome	Returns a comprehensive evaluation score of the board state for the given player.
Initiator	System (as part of the AI's decision-making process).
Preconditions	• board is a 2D list representing the game board. • player is an integer (1 or 2).
Postconditions	• Uses a combination of basic evaluations, capture evaluations, and positional evaluations. • Considers both offensive and defensive patterns. • Returns a comprehensive evaluation score of the board state for the given player.

evaluate_captures description	
Goal	Evaluate the board for potential captures by the given player.
How	• Iterates through the board to identify sequences where the player's stones capture opponent's stones. • Checks for specific patterns where two opponent stones are surrounded by the player's stones. • Adds a bonus score for strategic capture locations and multiple captures.
Outcome	Returns a score representing the capture potential for the given player.
Initiator	System (as part of the advanced board evaluation process).
Preconditions	• board is a 2D list representing the game board. • player is an integer (1 or 2).
Postconditions	• Iterates through the board to identify sequences where the player's stones capture opponent's stones. • Adds a bonus score for strategic capture locations and multiple captures. • Returns a score representing the capture potential for the given player.

check_pattern description	
Goal	Check if a specific pattern exists on the board.
How	• Verifies if the pattern fits within the board boundaries. • Iterates through the pattern to check if it matches the board's current state.
Outcome	Returns True if the pattern is found, otherwise False.
Initiator	System (as part of the pattern recognition process).
Preconditions	• board is a 2D list representing the game board. • row and col are integers within the board size. • dx and dy are integers representing the direction. • pattern is a tuple representing the pattern to match.
Postconditions	• Verifies if the pattern fits within the board boundaries. • Iterates through the pattern to check if it matches the board's current state. • Returns True if the pattern is found, otherwise False.

evaluate_board_state_easy description	
Goal	Provide a simple evaluation of the board state based on specific patterns and positions.
How	• Uses basic evaluations to score the board. • Adds positional evaluation, particularly focusing on the center of the board.
Outcome	Returns a score representing the board's evaluation for the given player.
Initiator	System (as part of the AI's decision-making process).
Preconditions	• board is a 2D list representing the game board. • ai_agent is an integer (1 or 2).
Postconditions	• Uses basic evaluations to score the board. • Adds positional evaluation, particularly focusing on the center of the board. • Returns a score representing the board's evaluation for the given player.

get_best_move description	
Goal	Determine the best move for the AI player using the minimax algorithm.
How	• Iterates through valid moves and evaluates them using the minimax algorithm. • Considers a time limit and depth limit for the search.
Outcome	Returns the best move found within the given constraints.
Initiator	System (when the AI needs to make a move).
Preconditions	• minimax_func is a function implementing the minimax algorithm. • isAlphaBeta is a boolean indicating whether to use alpha-beta pruning. • heuristic_fun is a function for evaluating the board state. • max_depth is an integer representing the maximum search depth. • time_limit is a float representing the time limit for the search in seconds.
Postconditions	• Iterates through valid moves and evaluates them using the minimax algorithm. • Considers a time limit and depth limit for the search. • Returns the best move found within the given constraints.

get_prioritized_moves description	
Goal	Generate a list of valid moves, prioritizing strategic positions.
How	• Prioritizes moves near the center and around existing stones. • Adds remaining positions to the list.
Outcome	Returns a list of valid moves, ordered by priority.
Initiator	System (as part of the move generation process).
Preconditions	• The game board is initialized and has valid player stones placed.
Postconditions	• Prioritizes moves near the center and around existing stones. • Adds remaining positions to the list. • Returns a list of valid moves, ordered by priority.

minimax_without_alpha_Beta description	
Goal	Implement the minimax algorithm without alpha-beta pruning.
How	• Recursively evaluates moves to a given depth. • Maximizes or minimizes the score based on the current player.
Outcome	Returns the best score for the current board state.
Initiator	System (as part of the AI's decision-making process).
Preconditions	• depth is an integer representing the search depth. • is_maximizing is a boolean indicating whether the current move is maximizing or minimizing. • heuristic_funtion is a function for evaluating the board state.
Postconditions	• Recursively evaluates moves to a given depth. • Maximizes or minimizes the score based on the current player. • Returns the best score for the current board state.

Minimax description	
Goal	Implement the minimax algorithm with alpha-beta pruning.
How	• Recursively evaluates moves to a given depth. • Uses alpha-beta pruning to eliminate branches that won't affect the final decision.
Outcome	Returns the best score for the current board state.
Initiator	System (as part of the AI's decision-making process).
Preconditions	• depth is an integer representing the search depth. • is_maximizing is a boolean indicating whether the current move is maximizing or minimizing. • alpha and beta are floats representing the alpha and beta values for pruning. • heuristic_funtion is a function for evaluating the board state (optional).
Postconditions	• Recursively evaluates moves to a given depth. • Uses alpha-beta pruning to eliminate branches that won't affect the final decision. • Returns the best score for the current board state.

Init(PenteGameGUI)description	
Goal	Initialize the Pente game GUI.
How	• Sets up the game configuration, including board size, cell size, margin, and screen size. • Initializes Pygame and sets up the display. • Initializes fonts and game logic variables. • Calls the method to show the algorithm selection screen.
Outcome	Sets up the game GUI and prepares it for user interaction.
Initiator	System (when creating a PenteGameGUI instance).
Preconditions	• board_size is an integer greater than 0. • cell_size is an integer greater than 0. • margin is an integer greater than 0.
	• self.BOARD_SIZE is set to board_size. • self.CELL_SIZE is set to cell_size.

Postconditions	• self.MARGIN is set to margin. • self.STONE_RADIUS is set to 12. • self.SCREEN_SIZE is calculated based on the board size, cell size, and margin. • Pygame is initialized, and the display is set up. • Fonts are initialized. • Game logic variables (self.pente_game, self.ai, self.winner, self.current_algorithm self.current_difficulty) are set to None. • The algorithm selection screen is displayed.

create_button description	
Goal	Create a button with specified text and position.
How	• Draws a button rectangle with specified dimensions and position. • Renders the button text and centers it within the button.
Outcome	Returns the button rectangle, text rectangle, and rendered text.
Initiator	System (when creating buttons for the GUI).
Preconditions	• text is a string. • x, y, width, and height are integers.
Postconditions	• A button rectangle is created and drawn on the screen. • The button text is rendered and centered within the button. • Returns the button rectangle, text rectangle, and rendered text.

show_algorithm_selection_screen description	
Goal	Display the screen for selecting the game algorithm.
How	• Fills the screen with a background color. • Displays welcome text and buttons for algorithm selection. • Waits for user input to select an algorithm.
Outcome	Sets the selected algorithm and proceeds to the difficulty selection screen.
Initiator	System (when initializing the game GUI).
Preconditions	• Pygame is initialized, and the display is set up.
Postconditions	• The screen is filled with a background color. • Welcome text and buttons for algorithm selection are displayed. • Waits for user input to select an algorithm. • Sets self.current_algorithm based on user selection. • Proceeds to the difficulty selection screen.

show_difficulty_selection_screen description	
Goal	Display the screen for selecting the game difficulty.
How	• Fills the screen with a background color. • Displays welcome text and buttons for difficulty selection. • Waits for user input to select a difficulty level.
Outcome	Sets the selected difficulty and starts the game.
Initiator	System (after selecting the algorithm).
Preconditions	• self.current_algorithm is set to a valid algorithm ("Alpha-Beta" or "Min-Max").
Postconditions	• The screen is filled with a background color. • Welcome text and buttons for difficulty selection are displayed. • Waits for user input to select a difficulty level. • Sets self.current_difficulty based on user selection. • Starts the game.

start_game description	
Goal	Start the Pente game with the selected algorithm and difficulty.
How	• Initializes the game logic and AI with the selected settings. • Sets the heuristic function and alpha-beta pruning flag based on the selected difficulty and algorithm. • Calls the method to run the game.
Outcome	Starts the game and prepares the AI for making moves.
Initiator	System (after selecting the difficulty).
Preconditions	• self.current_algorithm is set to a valid algorithm ("Alpha-Beta" or "Min-Max"). • self.current_difficulty is set to a valid difficulty ("Easy" or "Hard").
Postconditions	• Initializes the game logic by creating a PenteGame instance. • Sets the heuristic function based on the selected difficulty. • Sets the alpha-beta pruning flag based on the selected algorithm. • Initializes the AI player with the game instance and player number. • Sets self.winner to None. • Calls self.run_game() to start the game loop.

run_game description	
Goal	Run the main game loop.
How	• Continuously updates the game display, including the board, stones, and captures. • Handles user input for making moves and checks for a winner. • If the AI is playing, it makes moves based on the selected algorithm and heuristic function.
Outcome	Runs the game until the user quits or a winner is determined.
Initiator	System (after starting the game).

Preconditions	• The game logic and AI are initialized. • The game board is displayed.
Postconditions	• Continuously updates the game display, including the board, stones, and captures. • Handles user input for making moves and checks for a winner. • If the AI is playing, it makes moves based on the selected algorithm and heuristic function. • Ends the game loop when the user quits or a winner is determined.

draw_board description	
Goal	Draw the game board grid.
How	• Fills the screen with a background color. • Draws horizontal and vertical grid lines based on the board size and cell size.
Outcome	Displays the game board grid on the screen.
Initiator	System (as part of the game display update).
Preconditions	• The game board is initialized.
Postconditions	• Fills the screen with a background color. • Draws horizontal and vertical grid lines based on the board size and cell size. • Displays the game board grid on the screen.

draw_stones description	
Goal	Draw the stones on the board.
How	• Iterates through the board and draws stones at the appropriate positions based on the board state.
Outcome	Displays the stones on the board.
Initiator	System (as part of the game display update).
Preconditions	• The game board is initialized and has valid player stones placed.
Postconditions	• Iterates through the board and draws stones at the appropriate positions based on the board state. • Displays the stones on the board.

get_board_coordinates description	
Goal	Convert mouse position to board coordinates.
How	• Calculates the row and column based on the mouse position and cell size.
Outcome	Returns the row and column corresponding to the mouse position.
Initiator	System (when handling user input).
Preconditions	• mouse_pos is a tuple representing the mouse position (x, y).
Postconditions	• Calculates the row and column based on the mouse position and cell size. • Returns the row and column corresponding to the mouse position.

draw_captures description	
Goal	Draw capture information for both players.
How	• Creates a surface for displaying capture information. • Draws boxes and text for each player's captures.
Outcome	Displays the capture information on the screen.
Initiator	System (as part of the game display update).
Preconditions	• The game board is initialized and has valid player stones placed. • Capture counts for both players are updated.
Postconditions	• Creates a surface for displaying capture information. • Draws boxes and text for each player's captures. • Displays the capture information on the screen.

display_winner description	
Goal	Display the winner and show reset and exit buttons.
How	• Creates an overlay with semi-transparent black. • Displays the winner text and buttons for resetting or exiting the game. • Waits for user input to reset or exit.
Outcome	Displays the winner and handles user input for resetting or exiting the game.
Initiator	System (when a winner is determined).
Preconditions	• The game has ended, and a winner is determined.
Postconditions	• Creates an overlay with semi-transparent black. • Displays the winner text and buttons for resetting or exiting the game. • Waits for user input to reset or exit. • Handles user input for resetting or exiting the game.

Understanding the Pente AI Algorithm

The provided code implements a Pente AI player that primarily relies on a **Minimax algorithm** with **Alpha-Beta Pruning** to make decisions. This algorithm, coupled with a **heuristic evaluation function**, allows the AI to explore possible moves and select the optimal one.

Here's a breakdown of the key components:

1. Minimax Algorithm with Alpha-Beta Pruning:

- ✓ Recursive Search : The algorithm explores the game tree, alternating between maximizing (AI's turn) and minimizing (opponent's turn) player perspectives.
- ✓ Evaluation Function: At each leaf node (terminal state), the heuristic function is used to evaluate the board state.
- ✓ Pruning: Alpha-beta pruning is employed to significantly reduce the search space. It prunes branches that cannot possibly yield better results than those already explored.

2. Heuristic Evaluation Function:

- The `evaluate_board_state` function assesses the board state and assigns a score.

Key factors considered include:

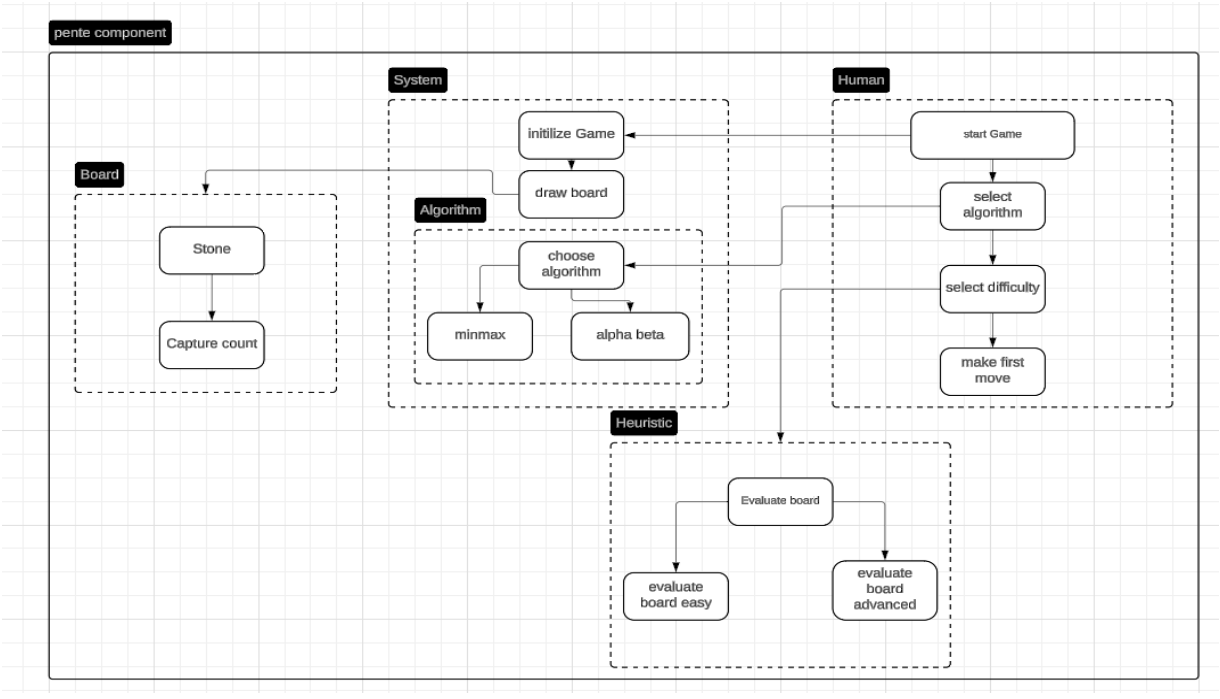
- ✓ **Winning Sequences:** The presence of potential winning sequences (e.g., four connected stones).
- ✓ **Threat Detection**: Identifying potential threats (e.g., open three-in-a-row).
- ✓ **Center Control:** Prioritizing moves that control the center of the board.
- ✓ **Capture Opportunities:** Evaluating the potential for capturing opponent's stones.

3. Move Prioritization:

- The `get_prioritized_moves` function identifies strategic moves and explores them first.

- **Prioritized moves include:**

- ✓ Moves that create potential winning sequences.
- ✓ Moves that block opponent's threats.
- ✓ Moves that capture opponent's stones.
- ✓ Moves that control the center of the board.



Experiments & Results

1. Description of Experiments

"We tested the PenteGame, PenteAI, and PenteGameGUI classes to ensure they function correctly both individually and together. The primary goals were to validate the correctness of game logic, the effectiveness of the AI, and the performance of the algorithms."

2. Testing Methodology & Results

➢ Unit Testing

Tested individual functions (is_valid_move, make_move, and check_win) in PenteGame.

```
----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK
PS C:\Users\Omnia\Desktop\projects\AI project>
```

➢ Integration Tests

Test how different components interact with each other.

```
Current player after toggle: 2
Current player after move: 2
Evaluating best move...
0
0
0
0
0
0
0
0
0
0
0
0
10
10
10
10
10
10
```

```
10
10
10
10
10
10
10
10
Current player after toggle: 1
Current player after move: 1
.
----------------------------------------------------------------------
Ran 1 test in 2.007s

OK
```

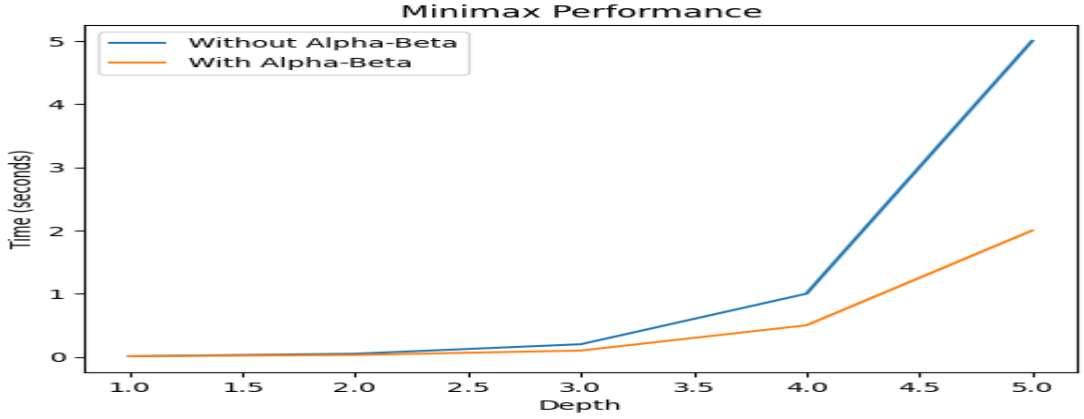
➢ Performance Tests

Measure the performance of algorithms.

```
Depth: 3, Without Alpha-Beta: 2.0007s, With Alpha-Beta: 2.0108s
Evaluating best move...
```

```
10
Depth: 5, Without Alpha-Beta: 2.0067s, With Alpha-Beta: 2.0047s
PS C:\Users\Omnia\Desktop\projects\AI project>
```

Plot



Analysis, Discussion, and Future Work:

Strengths:

- ✓ **Clear Structure:** The code is well-organized and easy to follow, with clear function and class definitions.
- ✓ **Robust Game Logic:** The PenteGame class implements the core game logic, including move validation, capture detection, and win condition checking, accurately.
- ✓ **Effective AI Implementation:** The PenteAI class utilizes the minimax algorithm with alpha-beta pruning, a strong foundation for AI decision-making in games.
- ✓ **Heuristic Evaluation:** The heuristic functions provide a reasonable evaluation of board states, guiding the AI towards strategic moves.
- ✓ **Prioritized Move Selection:** The get_prioritized_moves function improves the efficiency of the AI by focusing on promising moves**.**
- ✓ **User-Friendly GUI:** The GUI provides a visually appealing and intuitive interface for human players to interact with the game.

Weaknesses and Potential Improvements:

- **Heuristic Function Limitations**: While the heuristic functions provide a good foundation, they may not always capture subtle strategic nuances. More sophisticated heuristics or machine learning techniques could potentially improve the AI's performance.
- **Minimax Search Depth:** The depth of the minimax search can impact the AI's decision-making time and accuracy. Deeper searches can lead to better decisions but may be computationally expensive. Experimenting with different search depths and time limits can help find a balance.
- **Alpha-Beta Pruning Optimization:** The alpha-beta pruning implementation could be further optimized by carefully selecting the order of moves to explore and using more aggressive pruning techniques.
- **GUI Optimization:** The GUI could be optimized for performance, especially when dealing with large board sizes or complex AI calculationsusing techniques like double buffering or asynchronous updates to improve responsiveness.
- **Error Handling:** The code could benefit from more robust error handling. For example, handling invalid moves or unexpected input can prevent the game from crashing.

Potential Insights and Future Directions:

- ➢ **Experiment with Different Heuristic Functions:** Evaluate the impact of different heuristic functions on the AI's performance. (combining multiple heuristics or using machine learning techniques to learn optimal strategies).
- ➢ **Optimize Minimax Search:** Implement more advanced search algorithms like Monte Carlo Tree Search (MCTS) to handle complex game states more efficiently.
- ➢ **Improve GUI Performance:** Use techniques like asynchronous updates and optimization of rendering routines to enhance the game's responsiveness.
- ➢ **Add Multiplayer Mode:** Implement a multiplayer mode where human players can compete against each other or the AI.
- ➢ **Incorporate Machine Learning:** Explore the use of machine learning techniques, such as reinforcement learning or neural networks, to train the AI on a large dataset of game play.

What are the advantages / disadvantages?

Feature	Advantages	Disadvantages
Code Structure	Clear and modular, easy to understand	-
Minimax Algorithm	Strong foundation for AI decision-making	Can be computationally expensive for deep searches
Heuristic Functions	Guide the AI towards strategic moves	May not capture all nuances of complex board states
Prioritized Moves	Improves efficiency by focusing on promising moves	Might miss some optimal moves in certain scenarios
User Interface	User-friendly and visually appealing	Could benefit from additional features and optimizations
Alpha-Beta Pruning	Improves the efficiency of the minimax algorithm	Implementation might not be fully optimized
Error Handling	Could be improved for robustness	-
User Experience	Could be enhanced with more customization options and features	-

Why did the algorithm behave in such a way? What might be the future modifications you'd like to try when solving this problem?

😑 Why the Algorithm Behaved in Such a Way

The behavior of the algorithm, particularly its decision-making process, is primarily influenced by the following factors:

1. **Heuristic Function** : The heuristic function is crucial for evaluating board states and guiding the search. A well-designed heuristic function can significantly improve the AI's decision-making. However, if the heuristic function is not accurate or efficient, it can lead to suboptimal moves.

2. **Minimax Algorithm Depth :** The depth of the minimax search determines how far ahead the AI can look. A deeper search allows for better evaluation of long-term consequences, but it also increases computational cost. A shallow search might lead to short-sighted decisions.

3. **Prioritized Moves**: The `get_prioritized_moves` function influences the order in which moves are explored. Prioritizing strategic moves can improve efficiency, but it's important not to overlook potentially good moves that aren't immediately obvious.

4.**Alpha-Beta Pruning**: While the code doesn't explicitly implement alpha-beta pruning, it's a powerful technique to reduce the search space and improve efficiency.

😐 Future Modifications:

1. Enhanced Heuristic Function:

- Consider incorporating more sophisticated evaluation criteria, such as capturing patterns, threats, and positional advantages.
- Experiment with different weighting schemes for different factors to fine-tune the evaluation.

2. Improved Minimax Algorithm:

- Implement alpha-beta pruning to significantly reduce the search space and improve performance.
- Explore iterative deepening to gradually increase the search depth as time permits.
- Consider using transposition tables to store and reuse previously calculated evaluations.

3. Strategic Move Prioritization:

- Refine the `get_prioritized_moves` function to prioritize moves that are more likely to lead to winning positions or advantageous board states.
- Experiment with different prioritization strategies, such as focusing on center control, creating threats, and blocking opponent's threats.

4. Machine Learning Integration:

- Train a machine learning model on a large dataset of games to learn optimal strategies.
- Use the model to guide the AI's decision-making process, either by directly predicting the best move or by improving the heuristic function.

5. User Interface and Experience:

- Enhance the user interface with features like game history, move suggestions, and difficulty levels.
- Provide feedback to the user about the AI's thought process or the reasoning behind its moves.