ULTIMATE TICK TACK TOE AI DESIGN

MARCH 18, 2024

ARIEL SISCHY
221003350

Contents

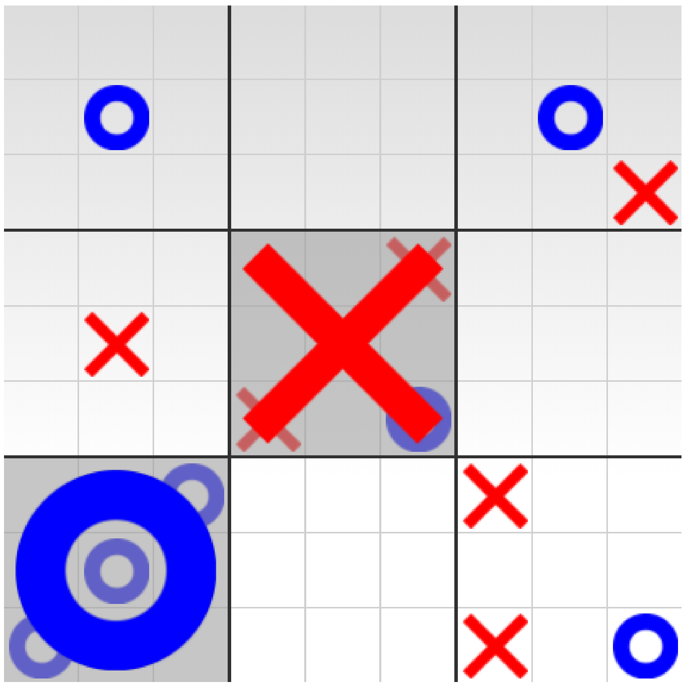
1. Background

In games, AI is often used to simulate "human" behaviour in order to increase the difficulty and enjoyment of the challenges imposed. The difficulty of these AI's is set such that the average individual can defeat it through skill and repetition. For the AI to emulate human behaviour, it must be given a sense of intelligence. This perceived intelligence can be quantified based on human rationalisation of the best choice from a set of choices. The issue comes around in how to give intelligence to an non-intelligent object. We can see that the emergence of intelligence rises from interactions between multiple components. To achieve this, it is possible to reduce an intelligence process into a set of non-intelligent computation processes.

In my solution, I will be implementing a reinforcement learning based agent to teach itself how to play the game of Ultimate Tick Tack Toe. Ultimate Tick Tack Toe is a spinoff of the classic Tick Tack Toe childhood game, where two players take turns choosing squares on an empty board, trying to get 3 in a row or diagonal. The game composes of player X and O.

In this version, there are 9, 3x3 boards, where each 3x3 board is treated like a square on one massive 3x3 board. The goal of the game is to win 3 of these small boards, in a row or diagonal. To make the game more complex and thought provoking, all 9 of these boards are linked. If player X plays on any board, such as the small middle board, and they play in the top right corner of that board, then O's turn will take place in the top right small board. If one's turn lands on a small board that has already been won or drawn, that player can play anywhere on any of the small boards instead. This complex playstyle makes thinking and planning ahead paramount.

1.1 Ultimate Tick Tack Toe Board (Next Strike Games, 2019)



2. Classification of Agent

I will be using a reinforcement-based agent. Reinforcement learning is where the AI agent learns from a sort of trial and error. During the training phase, the agent will have little to no knowledge of how to differentiate one action form another, never mind being able to play the game with any semblance of intelligence. The goal of the training is at each game end state, look back at how good or bad a previous move was, give it some reward or punishment value, store this data, update some policy governing its "thoughts", and repeat. In this manner, after a multitude of iterations, the agent will be able to differentiate in a numerical "goodness" that each possible action possesses. This will ensure the agent is more likely to take an action moving the board towards its goal: winning the game.

2.1 Agent Task Environment Classification

- **Full Observability:** The agent is able to view the entire state of the environment. As such, they can see the entire board, with the current square Xs, Os and blanks.
- **Single Agent:** Only one agent will be deployed to play against a human player.
- **Stochastic:** Due to the agent playing against a human, a large degree of unpredictability will be present. This entails that a given action will not always bring about the predicted result given the same environment.
- **Sequential:** Previous actions the agent takes influences future actions. This can be seen where previous boards were won, captured, drawn, or where previous actions resulted in future actions taking place on different boards.
- **Semi-Dynamic:** While the passing of time does not affect the environment, the outcome of the agents' actions does.
- **Discrete:** The agent only views the environment as a singular snapshot. This update of their view will only occur once at the start of each of their turns.
- **Unknown:** The agent will have complete knowledge of the game state, but will need to learn the best actions and decisions that it can make. Only from training and reward/punishment values, will it know a 3 in a row is a desirable choice.

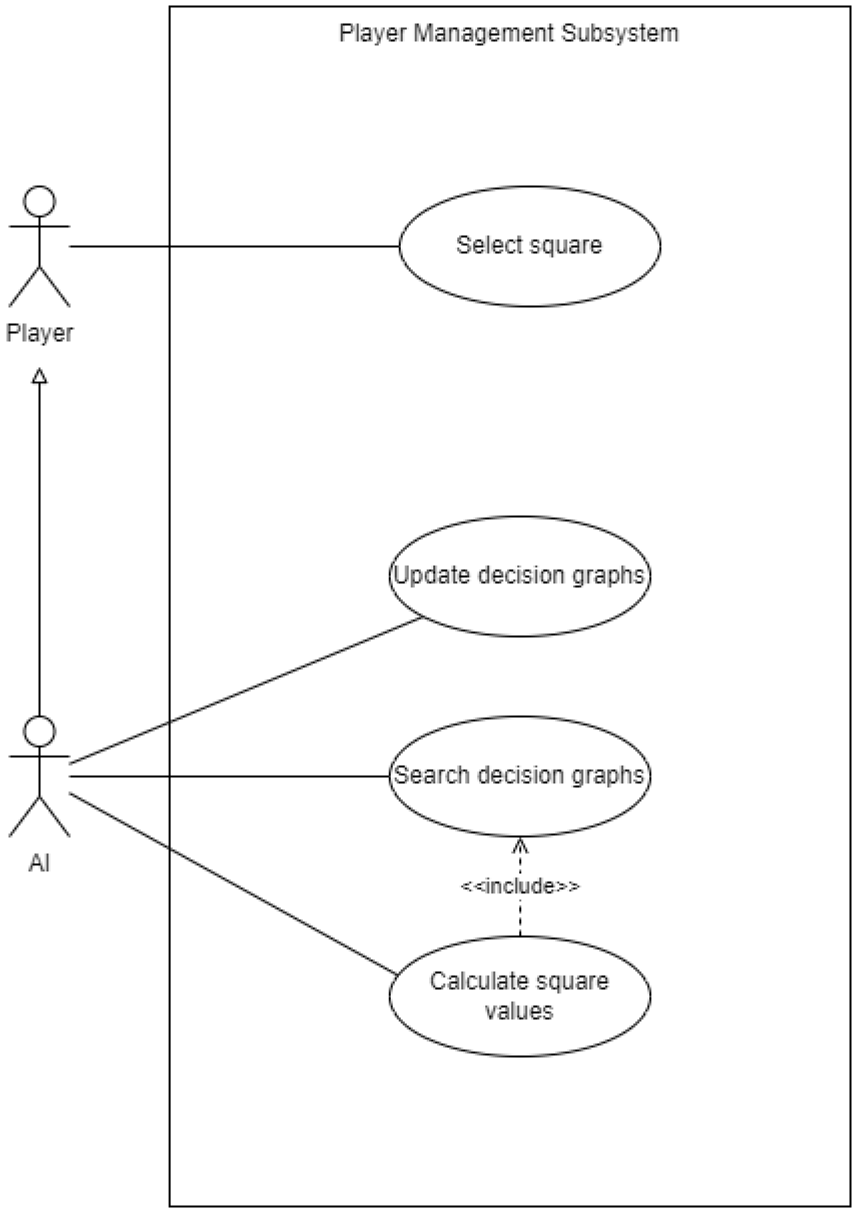
2.2 PEAS Analysis

Agent	**Performance Measures**	**Environment**	**Actuators**	**Sensors**
Ultimate Tick Tack Toe Reinforcement Learning Agent	• Wining/Losing a 3 in a row on a small board • Winning/Losing a 3 in a row on a big board • Winning/Losing a corner vs edge vs middle small board • Giving/Getting ability to choose anywhere on any of the 81 squares	• Big Board • Small Board • Xs, Os and blank squares	• Possible board squares and clicking them	• Game interface

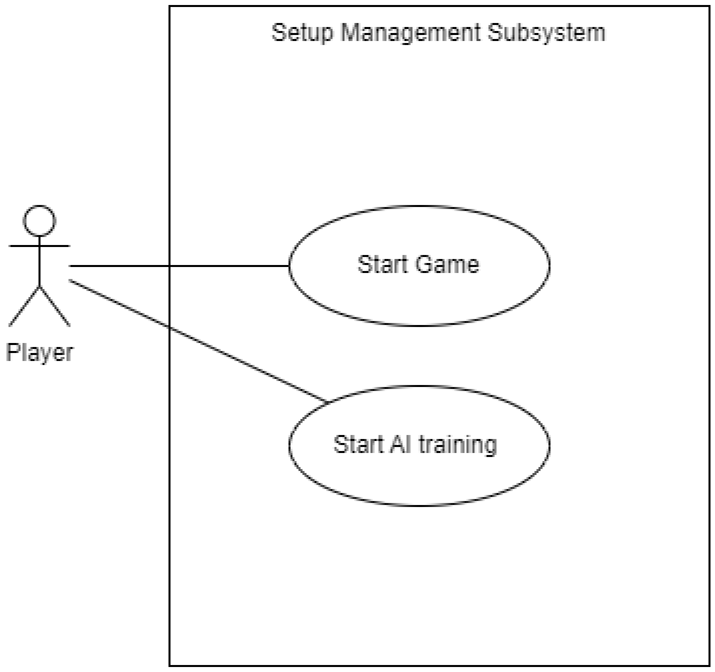
3. Use Cases and Use Case Diagrams

There are two main use case diagrams. The player management subsystem which focuses on the playing of the ultimate tick tack toe game, and the setup management subsystem which focuses on starting the game and entering training mode.

3.1 Use Case Diagram: Player Management



3.2 Use Case Diagram: Setup Management



4. Use Case Descriptions

4.1 Player Management Subsystem

4.1.1 Use Case: Select Square

- **Input:** All possible square selections and their values
- **Output:** Chosen square
- **Process:**
    1. The AI agent takes in all possible square actions and compares their reward values. In the players case, they just decide and pick their square.
    2. The AI agent chooses the square with the highest reward value.

4.1.2 Use Case: Update Decision Graphs

- **Input:** Sequence of plays leading up to end state
- **Output:** Updated reward values in decision graphs
- **Process:**
  1. The AI agent starting at the end state, assigns reward and punishment values to each square selection. A reward and punishment value also effect previous moves, at a decreasing percentage based on distance.
  2. The AI stores the updated values in the decision graphs.

4.1.3 Use Case: Search Decision Graphs

- **Input:** All sequences from decision graphs
- **Output:** Corresponding sequence from decision graphs with reward values
- **Process:**
  1. The AI takes in all sequences from the decision graphs and searches for a corresponding sequence based on the current board state as a whole, or a specific 3x3 board state if none are found.
  2. The AI selects the found sequence for processing.
- **Extension:**
  a. If no sequence is found, a no sequence found will be outputted.

4.1.4 Use Case: Calculate Square Values

- **Input:** Sequence from decision graphs with values
- **Output:** Playable square values
- **Process:**
  1. The AI takes either a matching past sequence to assign reward values, or if none is found, uses past 3x3 board positions to predict what each square value is worth.
  2. If a no sequence found occurred, no values will be assigned, leading to a more random selection.

4.2 Setup Management Subsystem

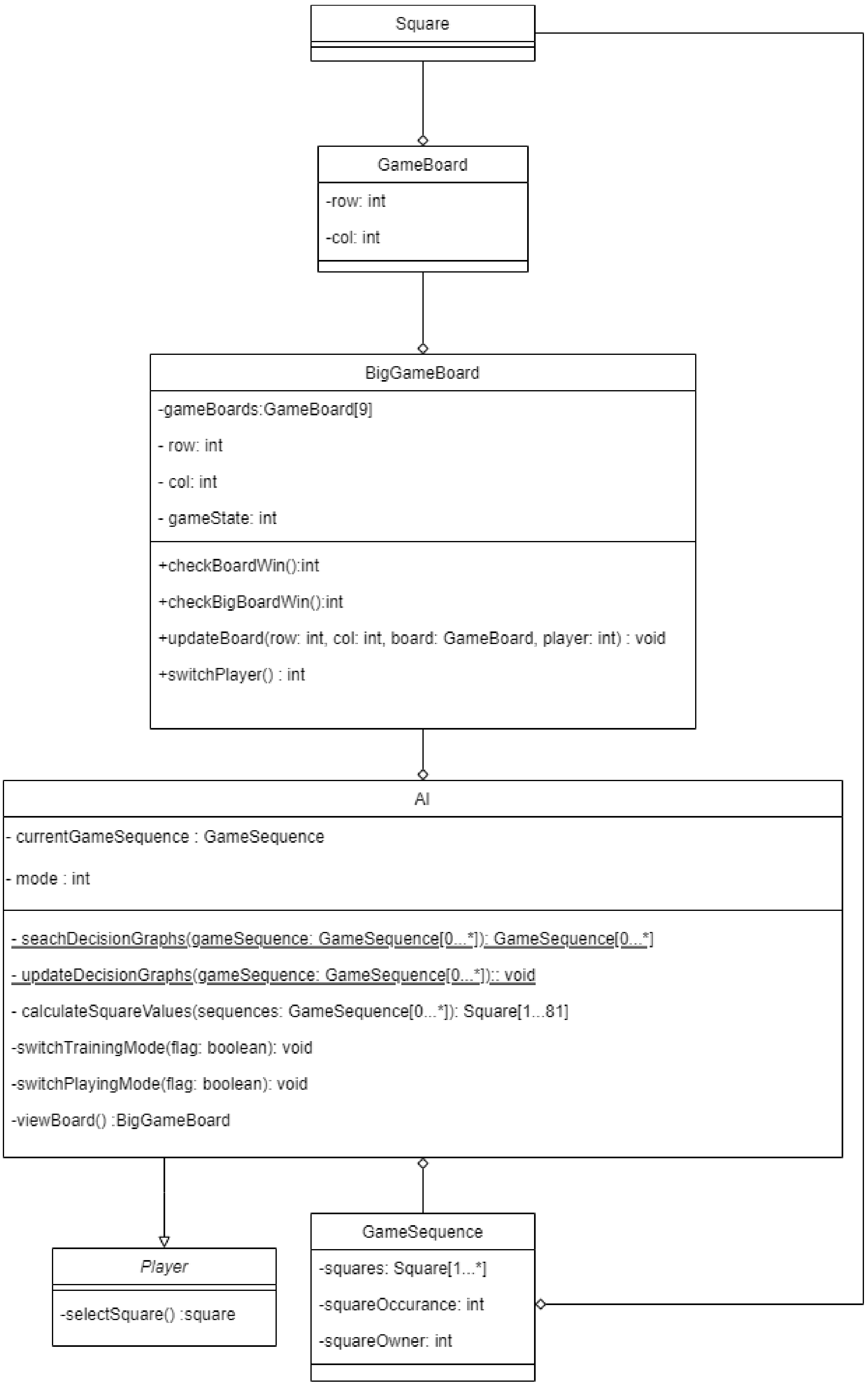
4.2.1 Use Case: Start Game

- **Input:** N/A
- **Output:** The game starts
- **Process:**
    1. The user clicks the “Start Game” button.

4.2.1 Use Case: Start AI Training

- **Input:** N/A
- **Output:** The AI training starts
- **Process:**
    1. The user clicks the “Start AI Training” button.
    2. The AI trains against a random generator, calling the “update decision graphs” use case at the end of each game state.

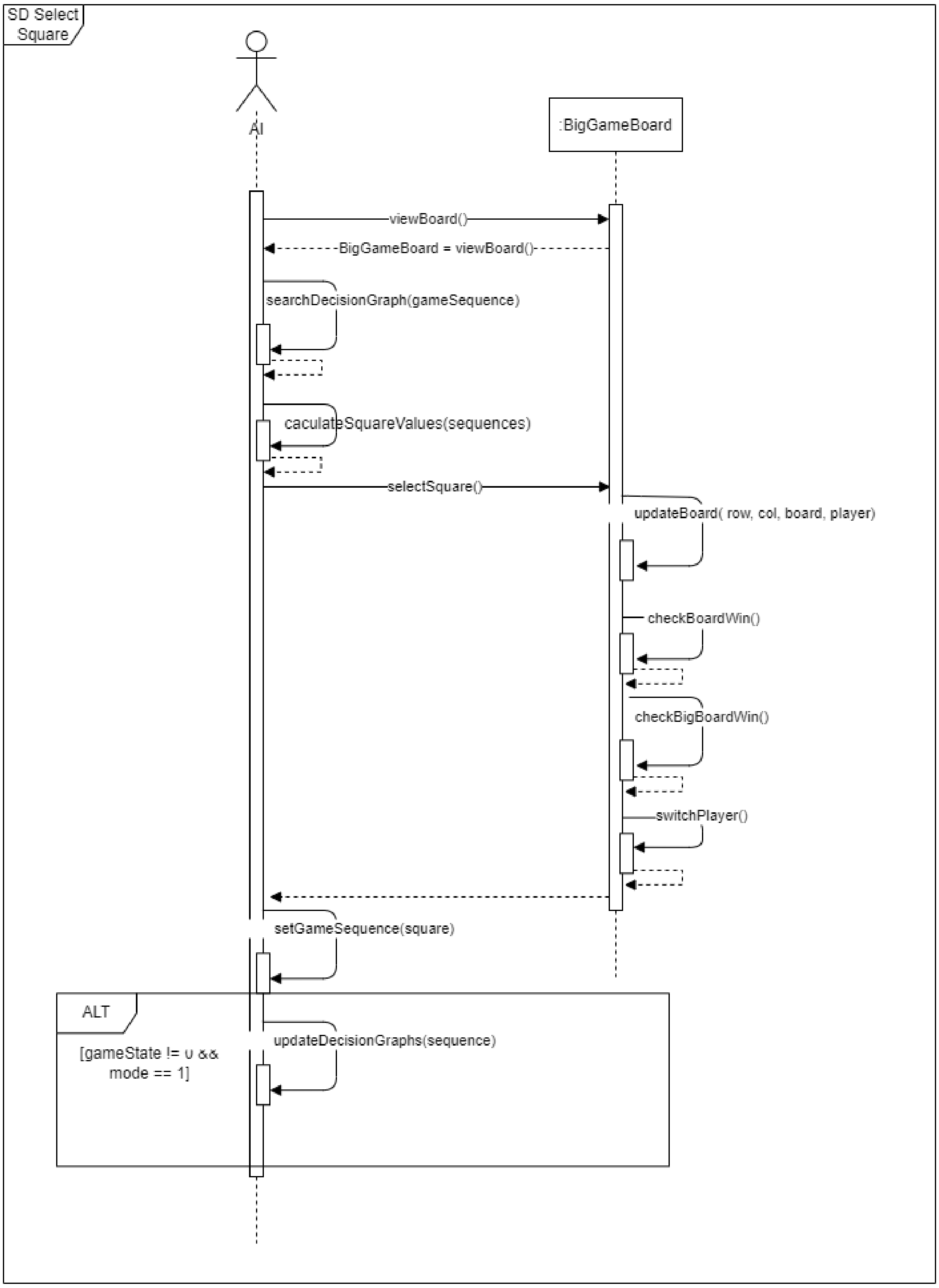
5. Class Diagrams


Square
GameBoard
-row: int
-col: int
BigGameBoard
-gameBoards:GameBoard[9]
- row: int
- col: int
- gameState: int
+checkBoardWin():int
+checkBigBoardWin():int
+updateBoard(row: int, col: int, board: GameBoard, player: int) : void
+switchPlayer() : int
AI
- currentGameSequence : GameSequence
- mode : int
- seachDecisionGraphs(gameSequence: GameSequence[0...*]): GameSequence[0...*]
- updateDecisionGraphs(gameSequence: GameSequence[0...*]):: void
- calculateSquareValues(sequences: GameSequence[0...*]): Square[1...81]
-switchTrainingMode(flag: boolean): void
-switchPlayingMode(flag: boolean): void
-viewBoard() :BigGameBoard
Player
-selectSquare() :square
GameSequence
-squares: Square[1...*]
-squareOccurance: int
-squareOwner: int

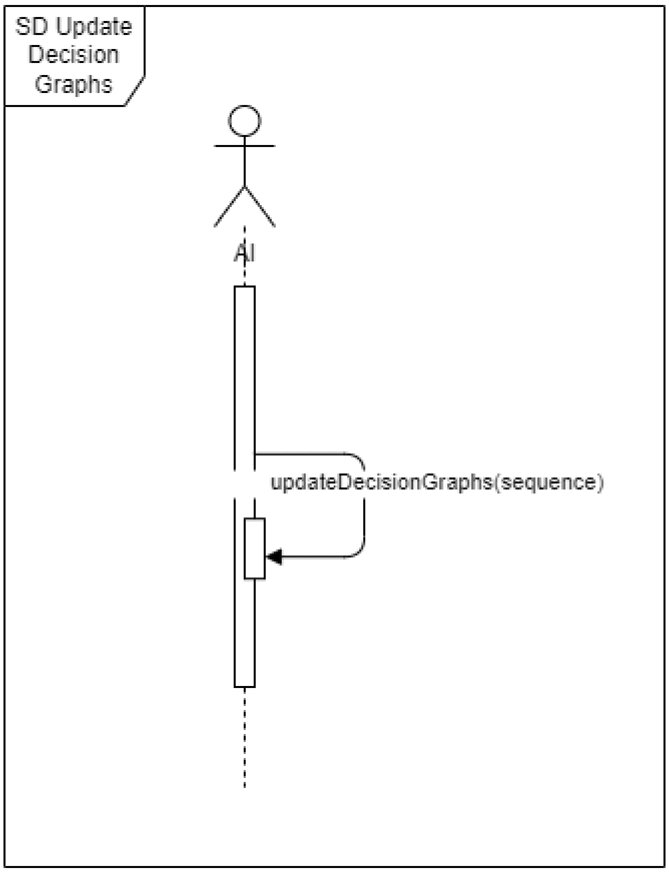
6. Interactive Sequence Diagrams

6.1 Player Management Subsystem

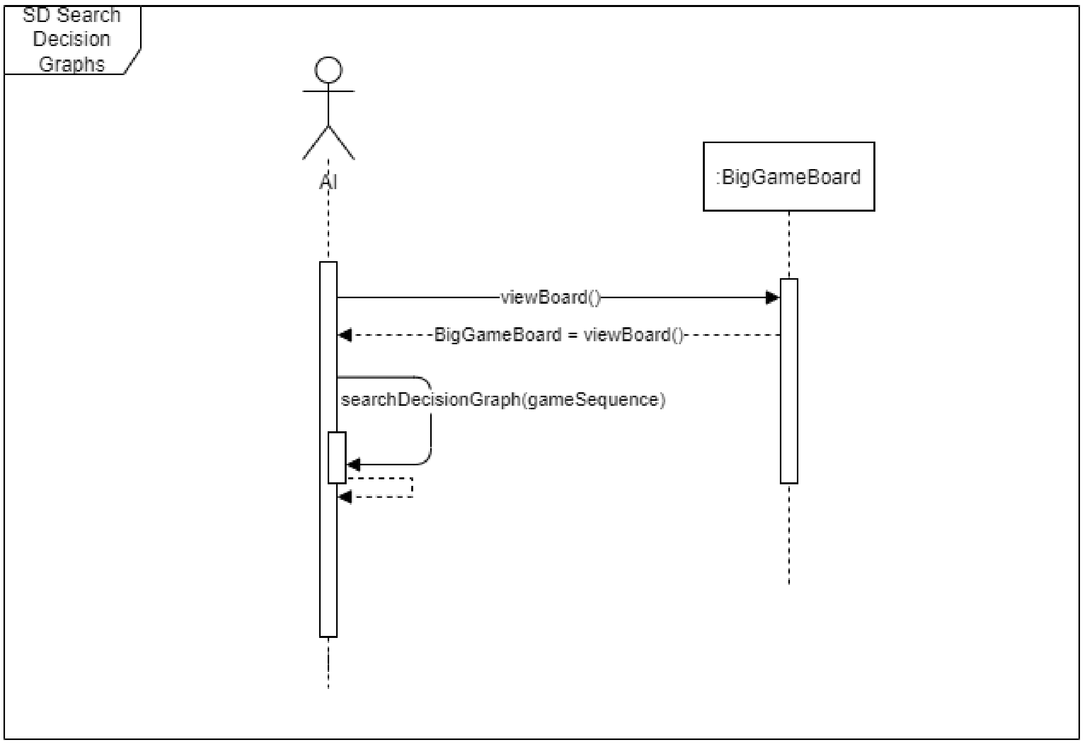
6.1.1 ISD: Select Square


SD Select Square
AI
:BigGameBoard
viewBoard()
BigGameBoard = viewBoard()
searchDecisionGraph(gameSequence)
caculateSquareValues(sequences)
selectSquare()
updateBoard(row, col, board, player)
checkBoardWin()
checkBigBoardWin()
switchPlayer()
setGameSequence(square)
ALT
[gameState != u && mode == 1]
updateDecisionGraphs(sequence)

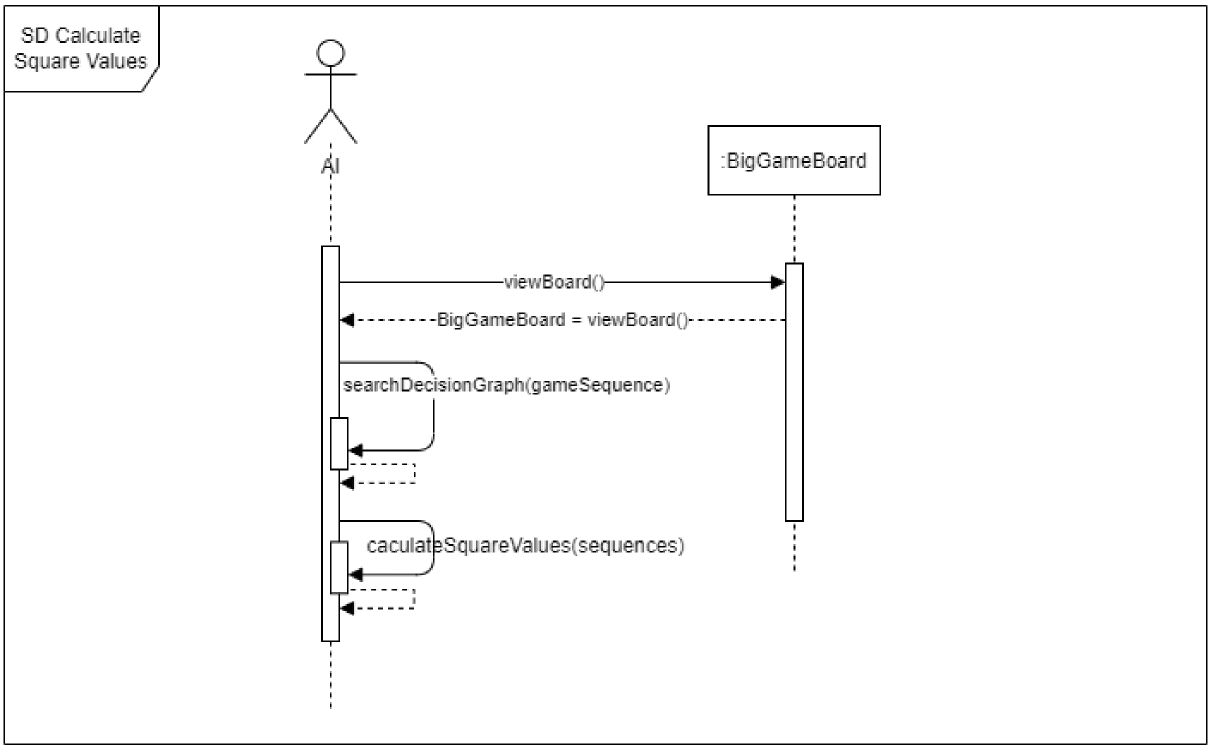
6.1.2 ISD: Update Decision Graphs



6.1.3 ISD: Search Decision Graphs

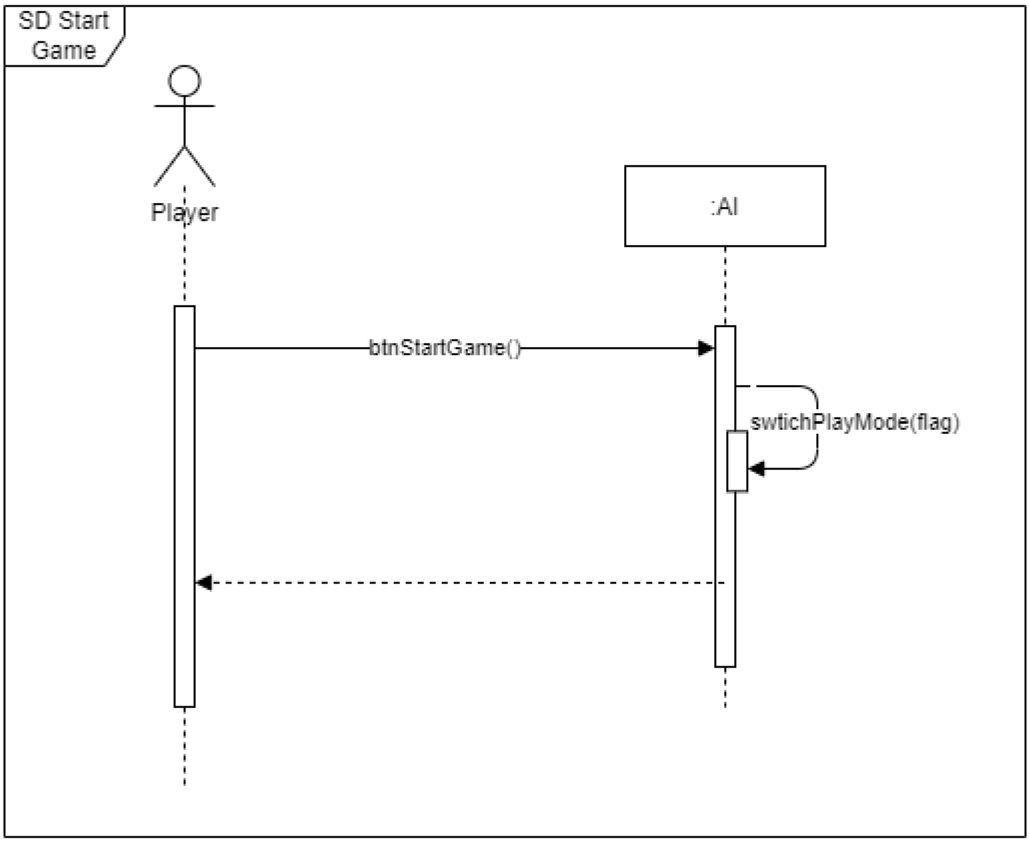

SD Search Decision Graphs
AI
:BigGameBoard
viewBoard()
BigGameBoard = viewBoard()
searchDecisionGraph(gameSequence)

6.1.4 ISD: Calculate Square Values

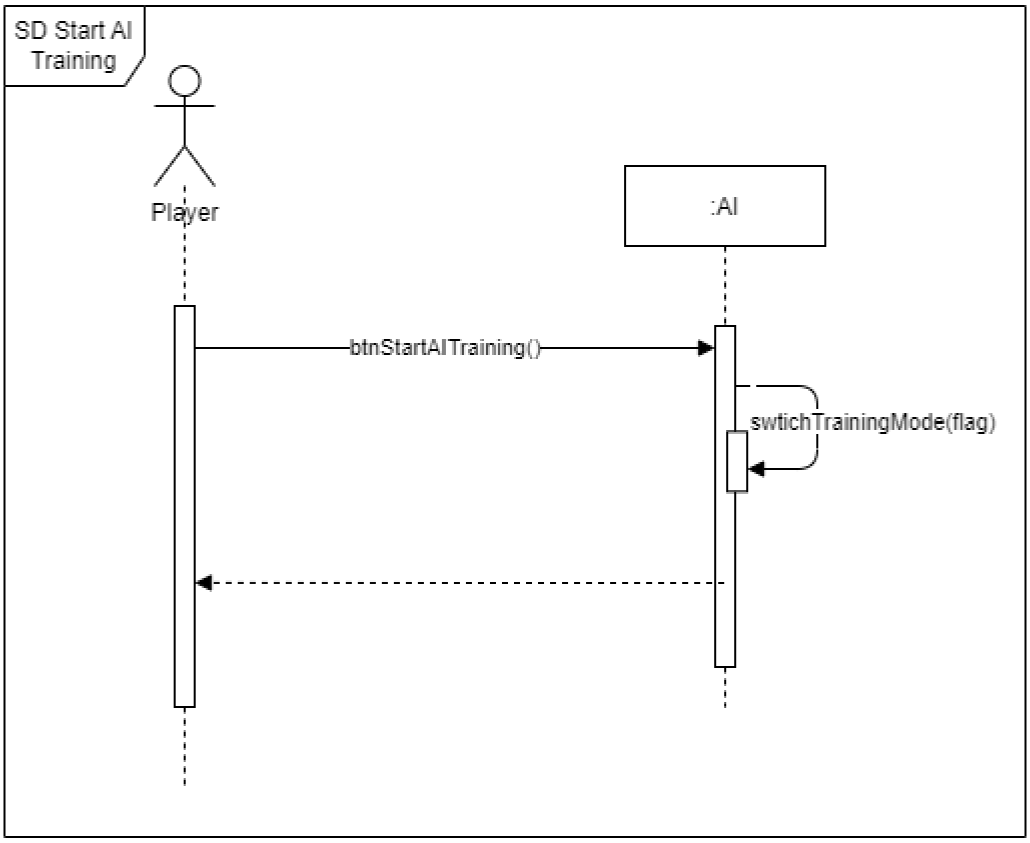

SD Calculate Square Values
AI
:BigGameBoard
viewBoard()
BigGameBoard = viewBoard()
searchDecisionGraph(gameSequence)
caculateSquareValues(sequences)

6.2 Setup Management Subsystem

6.2.1 ISD: Start Game



6.2.2 ISD: Start AI Training



7. Intelligence of Agent

In Summary, at the end of the day, the agent will not truly be intelligent. It will only need to make choices that seem logical to a human. To accomplish this, the agent will use a multitude of previous training games to help base its decisions on.

While brute force methods such as storing every combination that you could possibly play works for small games, in this case 81! combinations is impossible to store as such. Therefore, during the training phase, storing a smaller subset of winning, losing and drawing game states that the agent can refence when an unknown game board state occurs, will allow the agent to still make intelligent choices. The agent will be able to accomplish this by using them to determine if there is a possible winning action, or if an action would lead to a loss or misplay.

As a means of measuring the success and intelligence of the agent, I will use two methods: Firstly, I will pit it against a random generator, and record the number of wins, draws and losses. Secondly, I will play against the agent, recording how many obvious misplays and intelligent choices it makes. The recordings made will be based off of the performance measures seen on the "2.2 PEAS Analysis" table.

8. References

Next Strike Games. (2019, 03 27). *Stategic Tick Tack Toe*. Retrieved 03 12, 2024, from Next Strike: https://www.nextstrike.com/games/1/Strategic-Tic-Tac-Toe