PARADUX Design

Game rules

Paradux is a two player game where the objective is to align 4 tokens of your colour in a straight line (Can be Diagonally, horizontally, and vertically). The game starts with placing tokens around the perimeter of the hexagonal board (alternating colours/tokens) and the remaining tokens in the middle of the board as shown in the picture. The players or in this case the program will decide who will go first.

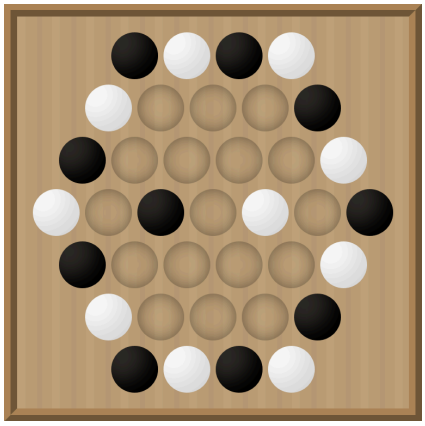
On each turn, a player selects one of their own tokens and one adjacent opponent player token. You cannot pick two of your tokens! The tokens chosen are moved as a single unit by one space in any horizontal, vertical or diagonal direction. The player can also choose to swap the tokens places as well which would account as a move as well.

Some constraints are that a player cannot move a single token by itself, cannot move two of the same token, or make a move that reverses the previous turn.

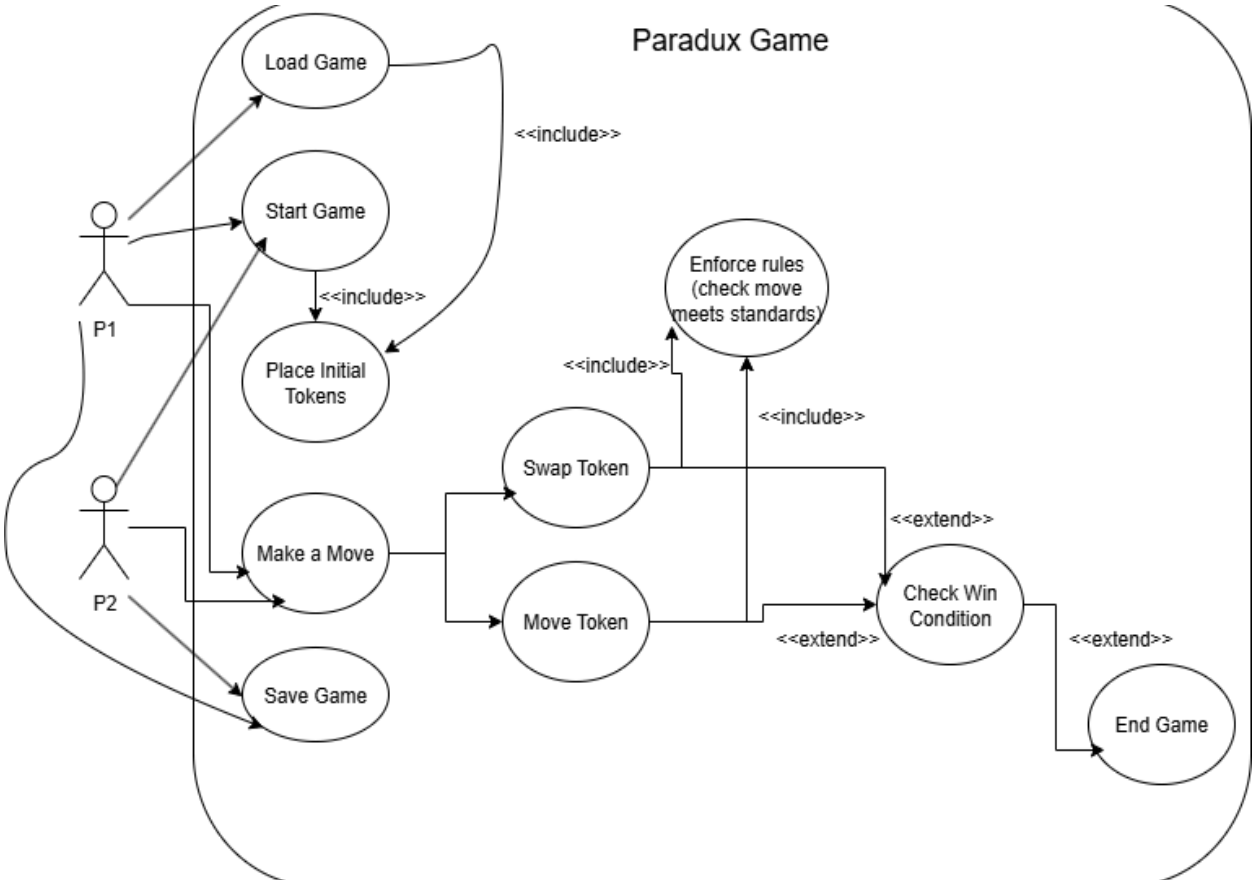
The game will continue, with players doing alternate turns until one player can successfully line up four tokens vertically, horizontally, or diagonally, thereby winning the game.

Equipment: Hexagonal Board, and two sets of 10 tokens (different colours)

You can set the board up in these two options:



Use Case Diagram



Use Cases

UC1: Start Game

Primary actor: *Player 1, Player 2*
Purpose: *Starting a new Paradux game*
Stakeholders and interests: Players (want a fair start), The actual game and board **(**need to be initialised**)**
Preconditions: Players are ready
Postconditions: Turns are decided, board is initialised, and game is started.
Initiating event: A player initiates start game, other player agrees
Main flow:
1. Players agree to start the game
2. Player 1 goes first
3. The game has started, so the system sets the game state “in progress”
4. UC2 invoked.
Alternate flows (extensions):
2a. *Players disagree to start the game*
2b. Don’t do coin toss, don’t place tokens and end the use case here.

UC2: Place Starting Tokens

Primary actor: *Game System*
Purpose: *Set up paradux game board by placing tokens according to the official rules*
Stakeholders and interests: Players (want a fair start), The actual game board/tokens
Preconditions: UC1
Postconditions: All starting tokens are placed, board is ready for first move.
Initiating event: Board is initialised
Main flow:
1. System loads initial board layout
2. Tokens are placed in their starting positions
3. System confirms the setup
Alternate flows (extensions):
2a. *Failed to load initial board layout*
2b. End game.

UC3: Make Move

Primary actor: *Current Player*
Purpose: *move pieces into winning position.*
Stakeholders and interests: *Player 1, Player 2*
Preconditions: *Tokens placed and tokens selected for the move.*

Postconditions: *If move is valid, then the board gets updated with the tokens in the new spots/coordinates.*
Initiating event: *Start of game or a switch of turns*
Main flow:

1. Player makes a move
2. System validates the move
3. The board is updated to reflect the move

Alternate flows (extensions):

1. Player makes move
2. System invalidates the move
3. The move is disallowed.

UC4: Enforce Game Rules (Check that move meets standards)

Primary actor: *Game System*
Purpose: Check that the requested action is legal and allowed within Paradux ruleset
Stakeholders and interests: Player 1, Player 2
Preconditions: A move action or a swap action needs verification
Postconditions: The Game System will allow the swap or move to occur
Initiating event: Player 1 or Player 2 provided a move or swap input
Main flow:

1. Player provides a move or swap action
2. System passes input to UC6 to validate input
3. System allows action to occur

Alternate flows (extensions):

1. Player provides a move or swap action
2. System passes input to UC6 to validate input
3. System doesn't allow action to occur

UC5: Check win condition

Primary actor: *Player 1, Player 2*
Purpose: *To conclude if the win state has been achieved by a player.*
Stakeholders and interests: *Player 1 and Player 2*
Preconditions: *Start of game, Move of tokens.*
Postconditions: *Winner declared, Game is ended.*
Initiating event:*Move of tokens*
Main flow:
1. Boards state is analysed
2. Win State confirmed
3. End game

Alternate flows (extensions):
1. Boards state is analysed
2. Win State confirmed absent.
3. Change turn.

UC6: End Game

Primary actor: *Game System*
Purpose: *Finish the game and declare a winner*
Stakeholders and interests: *Players (They want to know the outcome of who won)*
Preconditions: *Win Condition is Satisfied*
Postconditions: *Game ends, no further moves possible. The system puts the state of the game as done.*
Initiating event: *Win Condition Detected*
Main flow:

1. **System announces winner**
2. **System locks the board**
3. **Option to restart the game, by resetting tokens or quit the game.**

Alternate flows (extensions):

UC7: Save Game

Primary actor: *Player 1, Player 2*
Purpose: *Preserve the current game state for later continuation in a secure format.*
Stakeholders and interests: Players (want to preserve the game state)
Preconditions: Game state set to “in progress”
Postconditions: Current board state and the Player's turns are stored successfully.
Initiating event: A player selects to “save game” in the quit menu.
Main flow:
1. Players choose “save game”
2. System captures current game state
3. Game data is written in a local file.
4. System confirms successful save
Alternate flows (extensions):
3a. *Save operation fails and aborts.*
3b. System alerts Player of failure

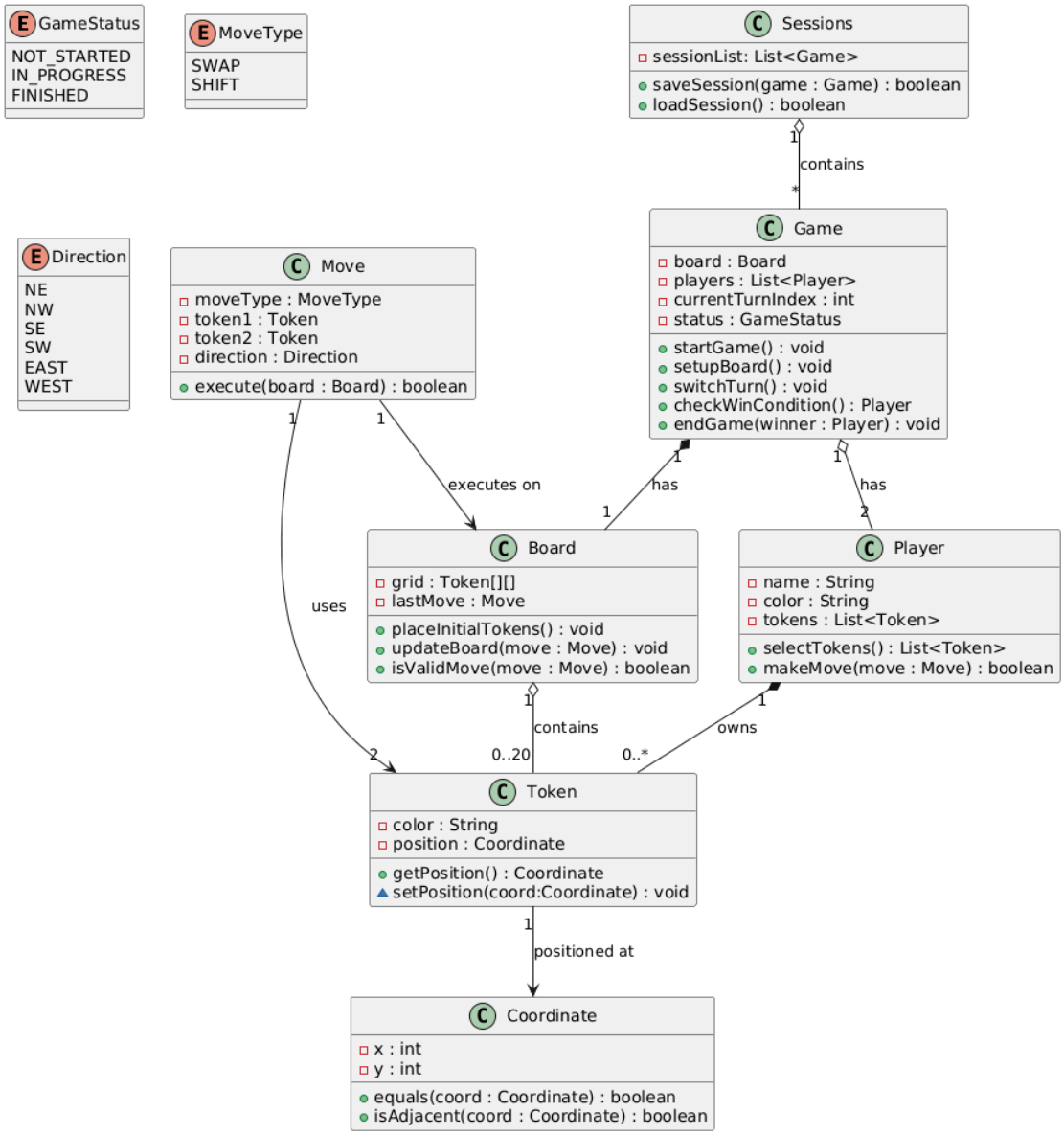
UC8: Load Game

Primary actor: *Player 1, Player 2*
Purpose: *Retrieve and restore a previously saved game session*
Stakeholders and interests: Players (want to resume a stored game session).
Preconditions: Valid save file exists.
Postconditions: Game board, players and turn order restored, with game state being set to “in progress”
Initiating event: Player selects the “load game” option in the main menu.
Main flow:
1. Players choose “load game”
2. System loads and displays all available saved sessions
3. Player selects a file
4. System retrieves and loads save file
5. Game Board, turns and players are restored.
Alternate flows (extensions):
3a. *Save files not loading or corrupted.*
3b. System alerts Players of failure.

Class diagram

Classes:

- Board
- Tokens
- Player
- Game (handles the logic)
- Move
- Coordinate
- Sessions



Class list

```
' ========================
' ENUMS / SUPPORT TYPES
' ========================
enum GameStatus {
  NOT_STARTED
  IN_PROGRESS
  FINISHED
}

enum MoveType {
  SWAP
  SHIFT
}

enum Direction {
  NE
  NW
  SE
  SW
  EAST
  WEST
}
```

Class name: *Game*
Description: *This class controls the overall game by initiating the setup for the game and board. As well, it is where the code will switch turns between Player 1 and 2. Lastly, it also ends the game and checks if a player has won in the game.*
Instance variables: • *board : Board* • *players : List<Player>* • *currentTurnIndex : int* • *status : GameStatus*
Constructor: • *Initializes the Game class, players and runs setupBoard() automatically*
Private methods: • *[method_name1(args): return type – short description]*

Public methods:

- *StartGame() : void - Starts the game, showing the board to the players and gives first turn to Player 1*
- *setupBoard() : void - Sets up the board*
    - *Initializes tokens, giving each token a coordinate, and places all tokens in their correct position on the board*
- *switchTurn() : void - switches currentTurnIndex between 1 and 2; for each player's turn respectively*
- *checkWinCondition() : Player - Checks if either player satisfies the win condition (4 in a row)*
- *endGame() : void - Ends the game if either player wins*

Class name: *Move*
Description: *This class controls the player's actions by moving the tokens on the board according to the Player's instructions.*
Instance variables: ● *moveType : MoveType* ● *token1 : Token* ● *token2 : Token* ● *direction : Direction*
Constructor: ● *No/empty constructor*
Private methods: ● *[method_name1(args): return type – short description]*
Public methods: ● *execute(board : Board) : boolean - Moves the tokens according to the Player's input*

Class name: *Player*
Description: *This class acts as the player and allows them to make moves and select tokens.*
Instance variables: ● *name : String* ● *color : String* ● *tokens : List<Tokens>*
Constructor: ● *Assigns both players name and color along with all tokens that they control* ○ *Defaults to "Player 1" and "Player 2" if no given name*

Private methods:
- *[method_name1(args): return type – short description]*

Public methods:
- *selectTokens() : List<Token> - Returns the selected tokens (2 of them)*
- *makeMove(move : Move) : boolean - Returns true or false if move succeeds*

Class name: *Board*

Description: *This class is responsible for storing and updating the position of all the tokens on the board, as well as keeping track of the last move.*

Instance variables:
- *grid : Token[][]*
  - Fill empty spots with null
- *lastMove : Move*

Constructor:
- *Runs placeInitialTokens()*
  - *Fills grid[][] with Token objects*

Private methods:
- *[method_name1(args): return type – short description]*

Public methods:
- *placeInitialTokes() : void - Places tokens on board*
- *updateBoard(move : Move) : void - Updates board after Move is ran*
  - *Runs isValidMove() during this function*
- *isValidMove(move : Move) : boolean - Checks if move is valid and according to game rules*

Class name: *Token*

Description: *This class acts as a token on the board, allowing it to be moved across the board and logging its current position*

Instance variables:
- *color : String*
- *position : Coordinate*

Constructor:
- *Gives a color and assigns a coordinate based on input*
  - *Defaults to the same color and to the same coordinate if no input*

Private methods:
- *[method_name1(args): return type – short description]*

Public methods:

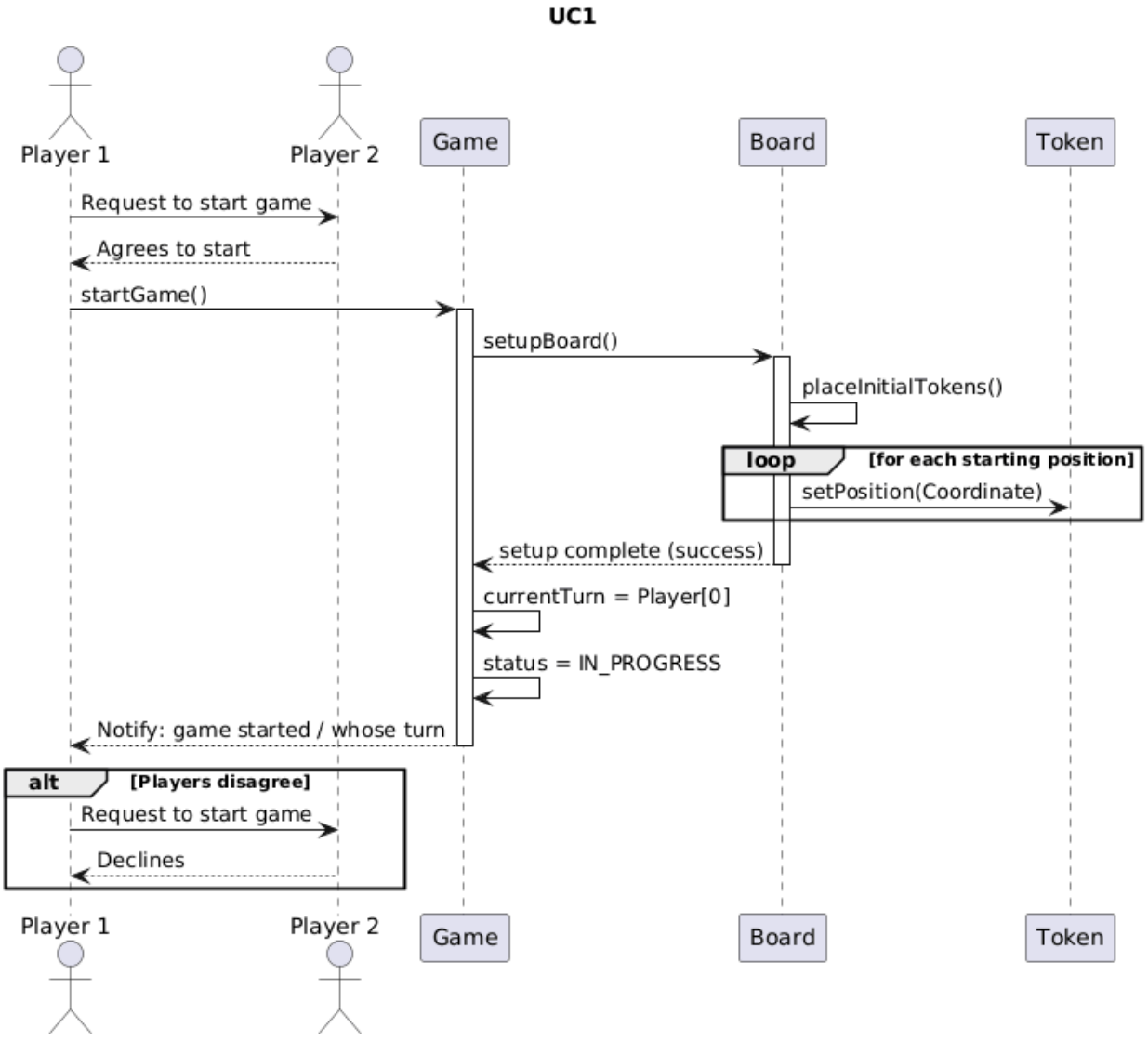
- *getPosition() : Coordinate - Returns the position of the token*
- *setPosition(coord : Coordinate) : void - Sets the position of the token*

Class name: *Coordinate*
Description: *This class is used to determine where on the board each token is located.*
Instance variables: ● *x : int* ● *y : int*
Constructor: ● *Pulls the load files*
Public methods: ● *equals(coord : Coordinate) : boolean - checks if coordinates are equal to the inputted value* ● *isAdjacent(coord : Coordinate) : boolean - checks if given coordinate is adjacent to the current coordinate*

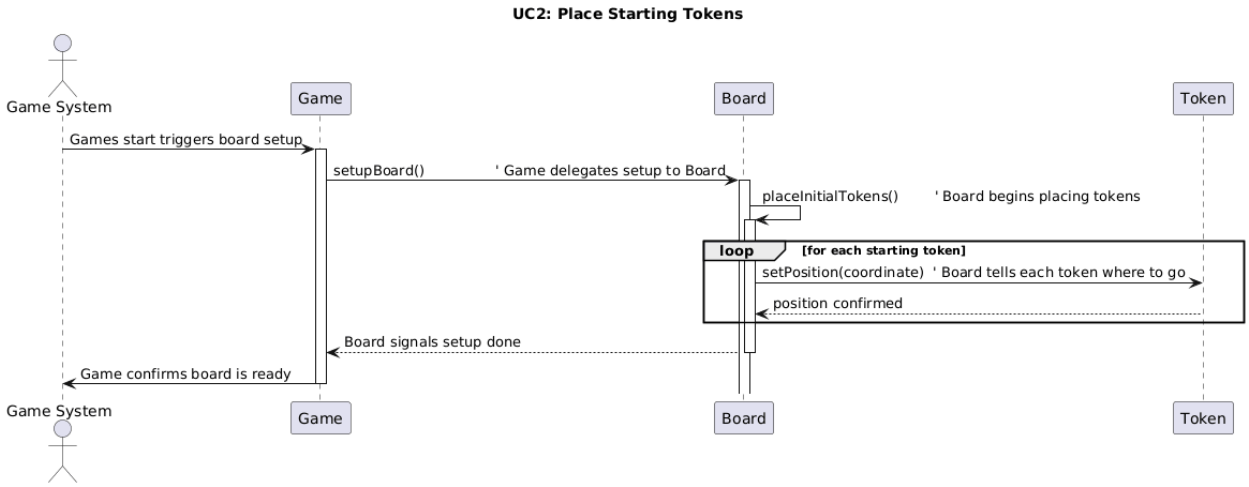
Class name: *Session*
Description: *This class is used to save or load Games.*
Instance variables: ● *sessionList : <Game>*
Constructor: ● *Checks local storage and saves list of save files.*
Private methods: ● *[method_name1(args): return type – short description]*
Public methods: ● *saveSession(game : Game) : boolean - saves the currently running Game session.* ● *loadSession() : boolean - provides a list of visible game session saves and loads the selected one*

Sequence diagrams

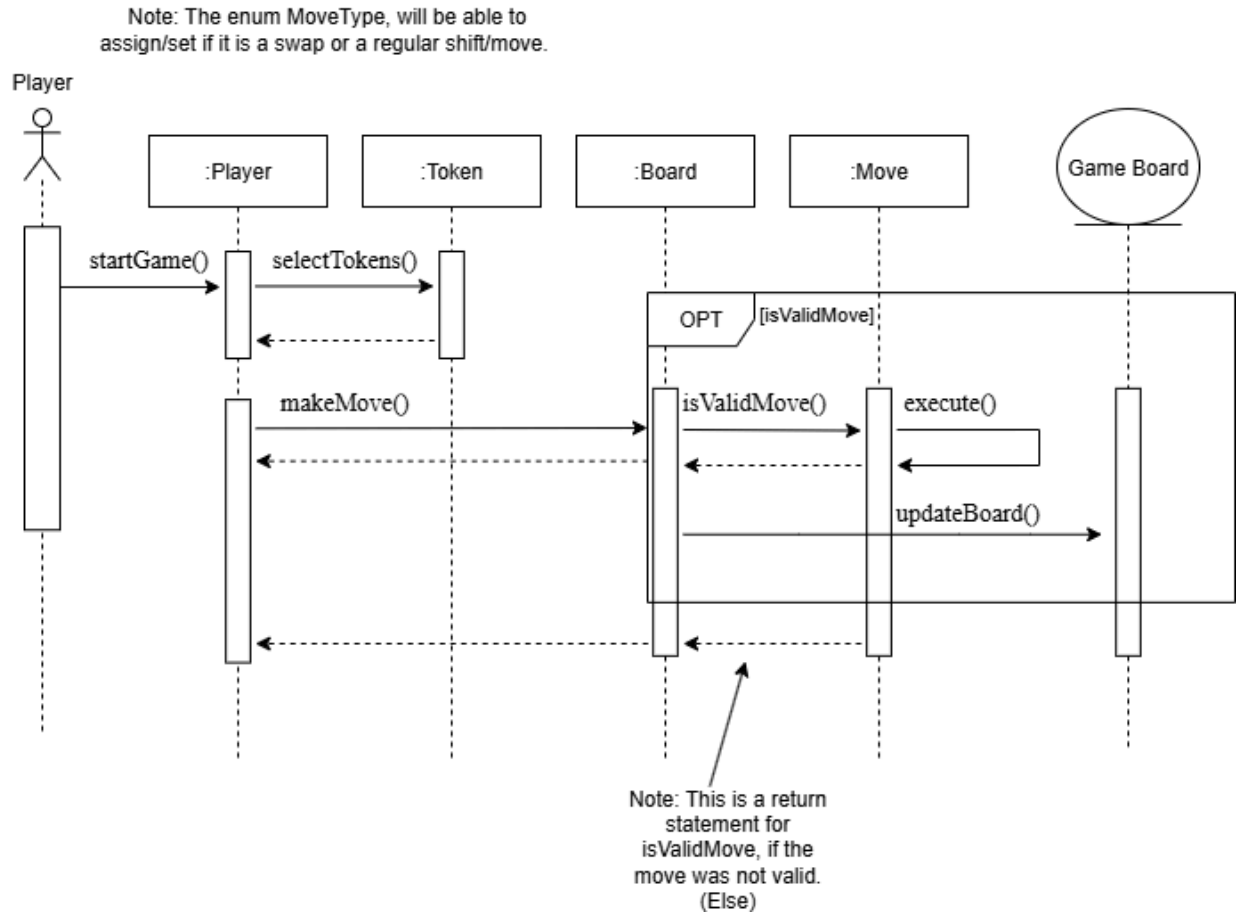
UC1: Start Game



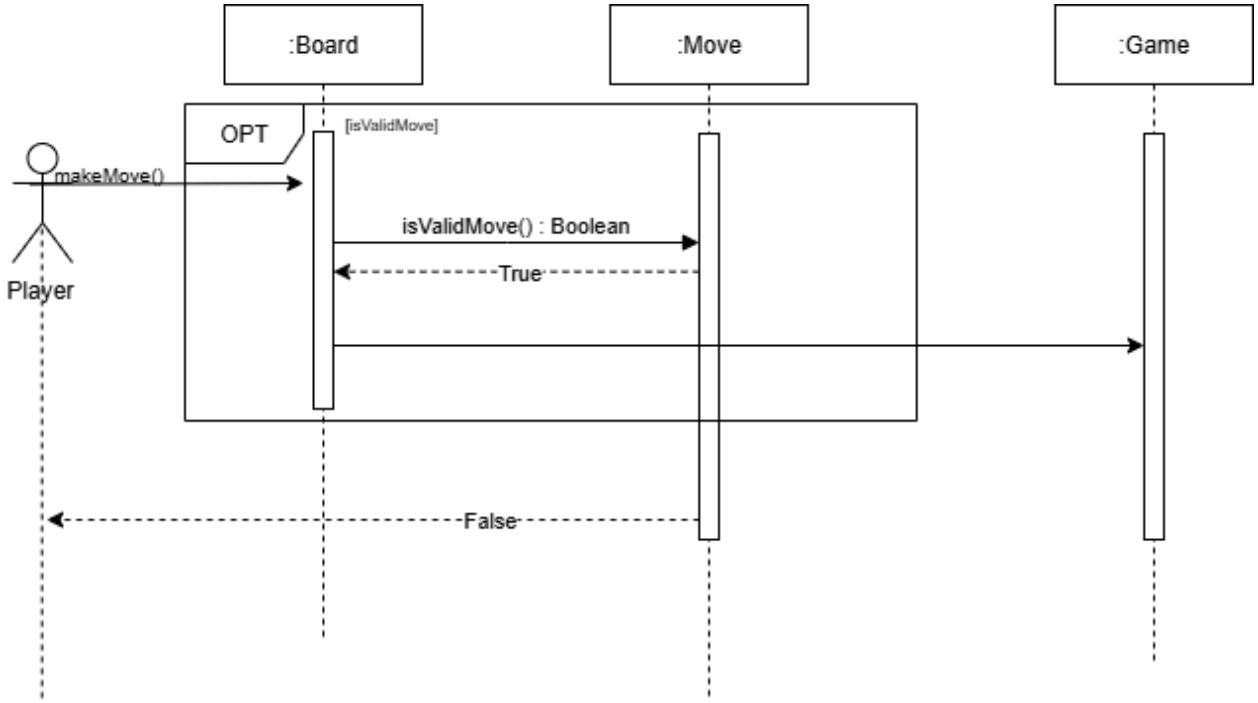
UC2: Place Tokens



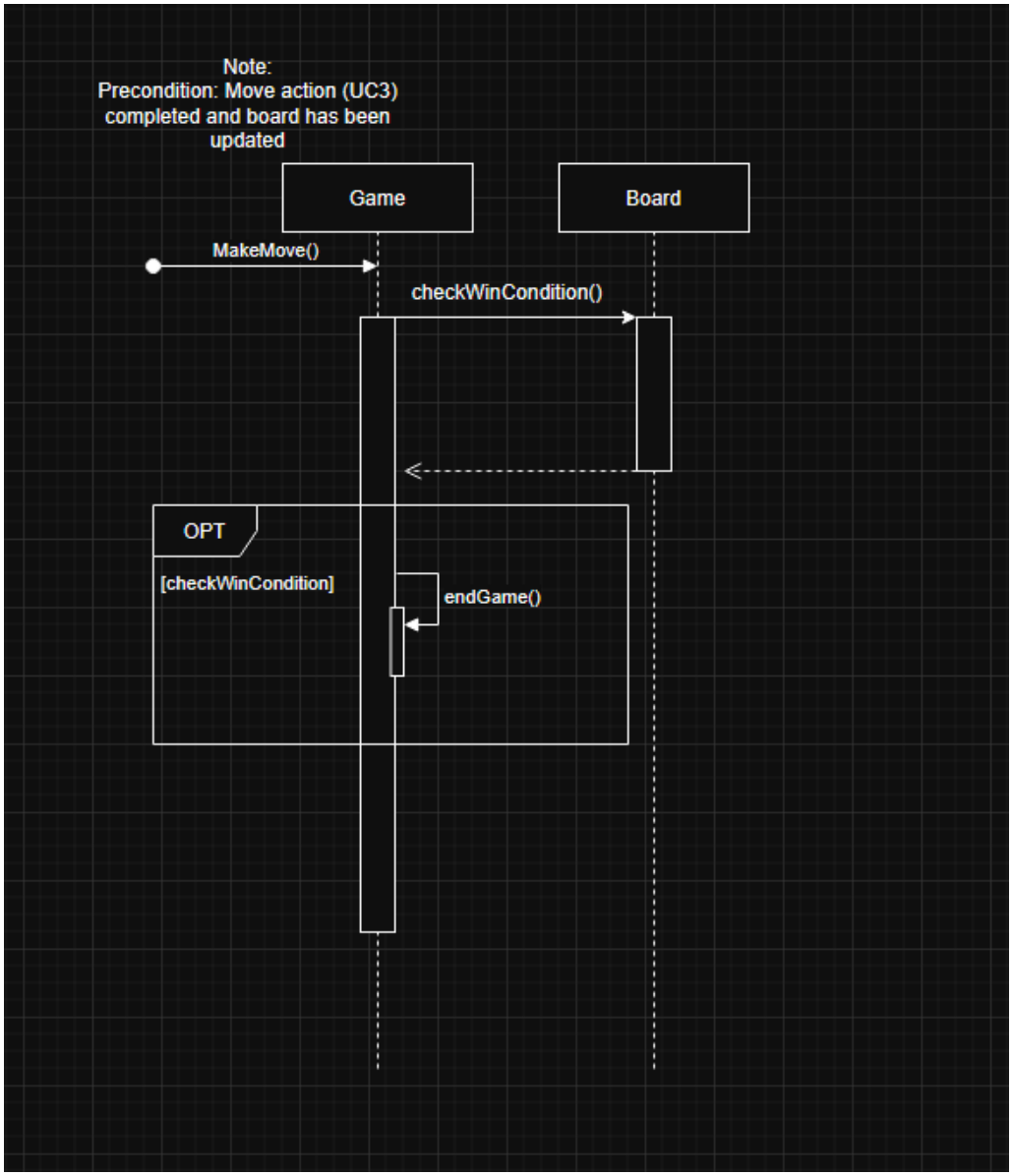
UC3: Make a move



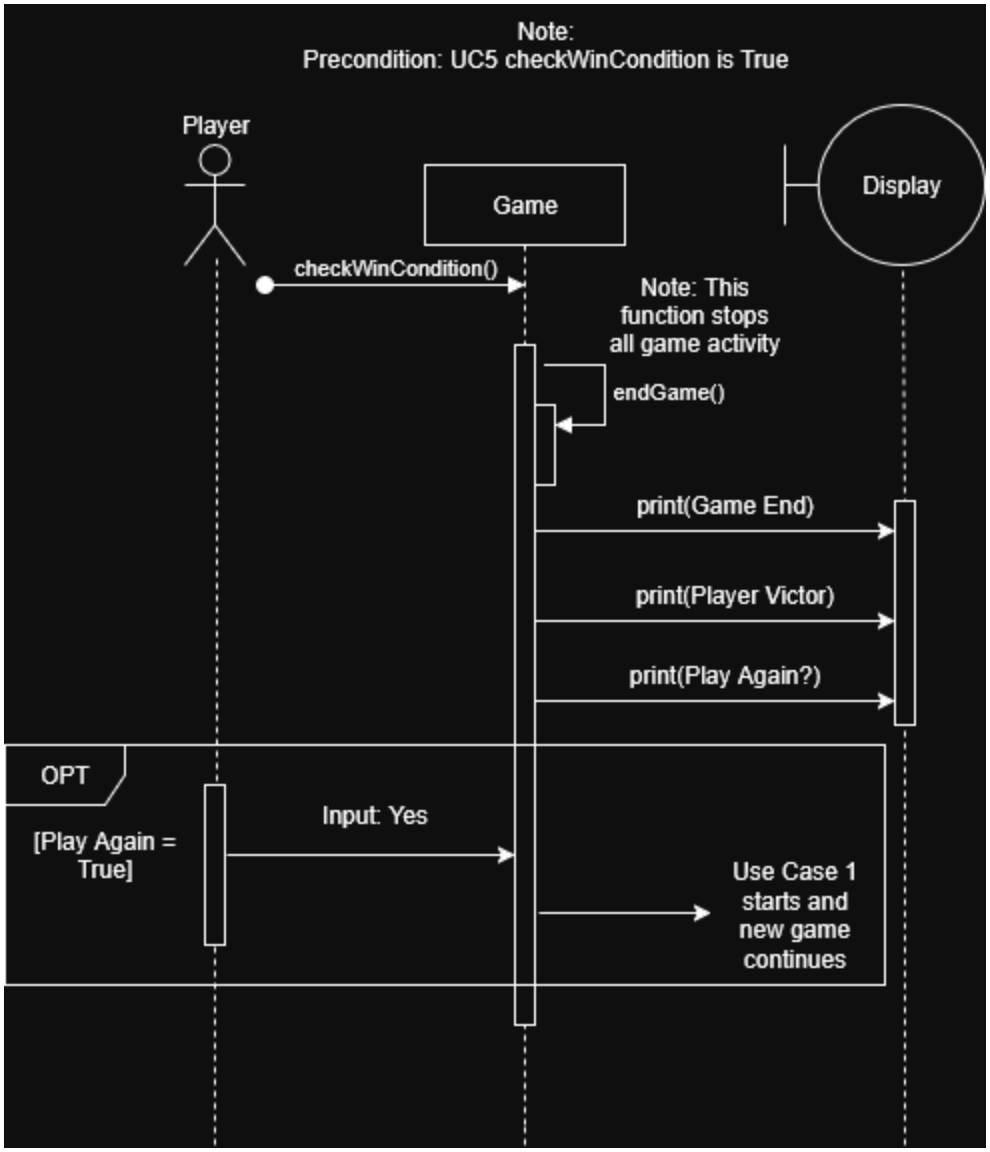
UC4: Enforce Game Rules



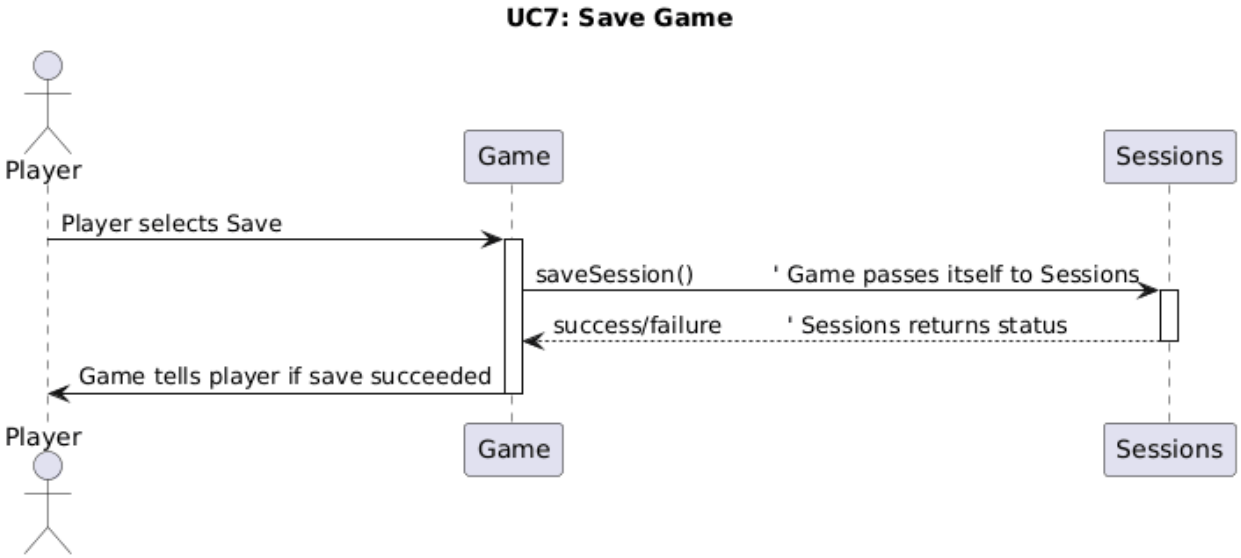
UC5: Check Win Condition


Note:
Precondition: Move action (UC3) completed and board has been updated
Game
Board
MakeMove()
checkWinCondition()
OPT
[checkWinCondition]
endGame()

UC6: End Game

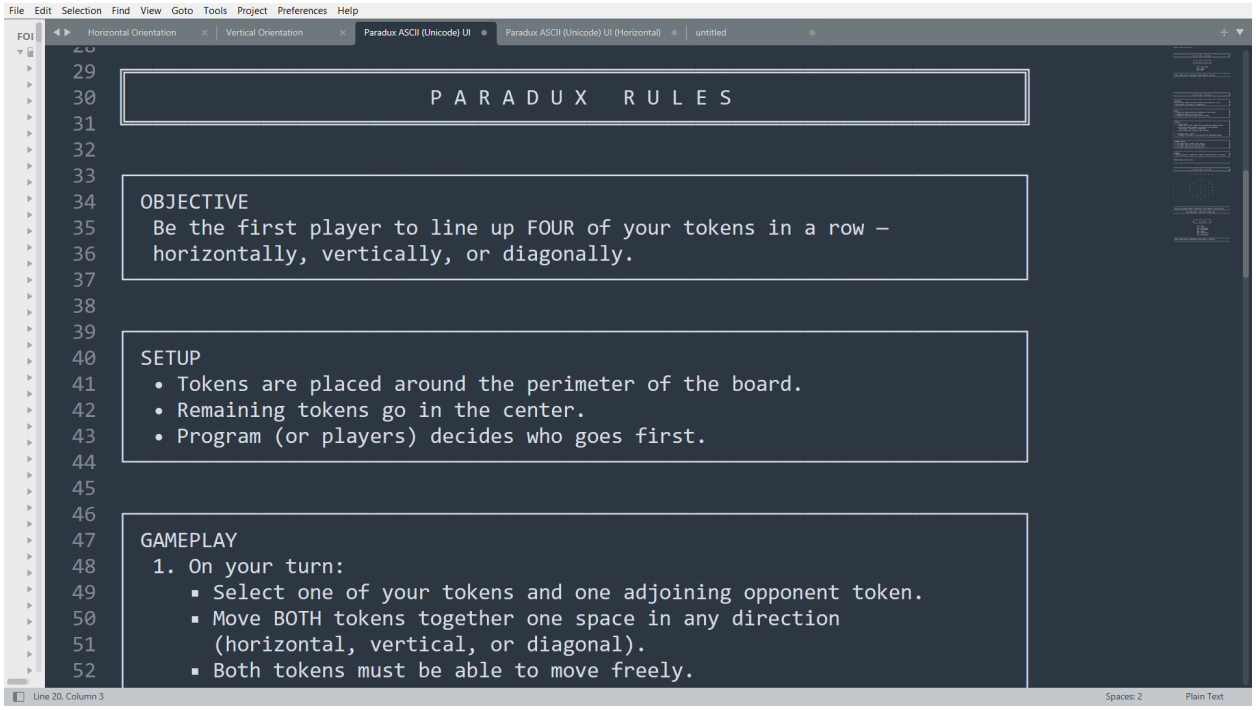


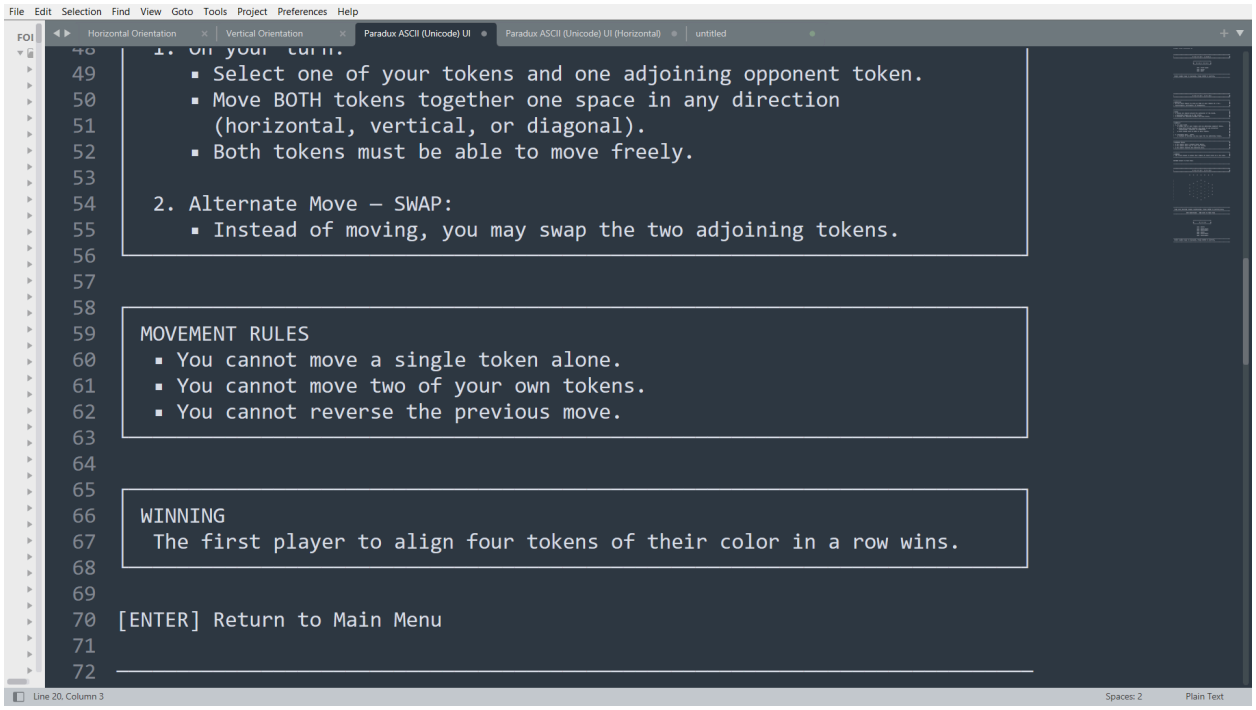
UC7: Save Game

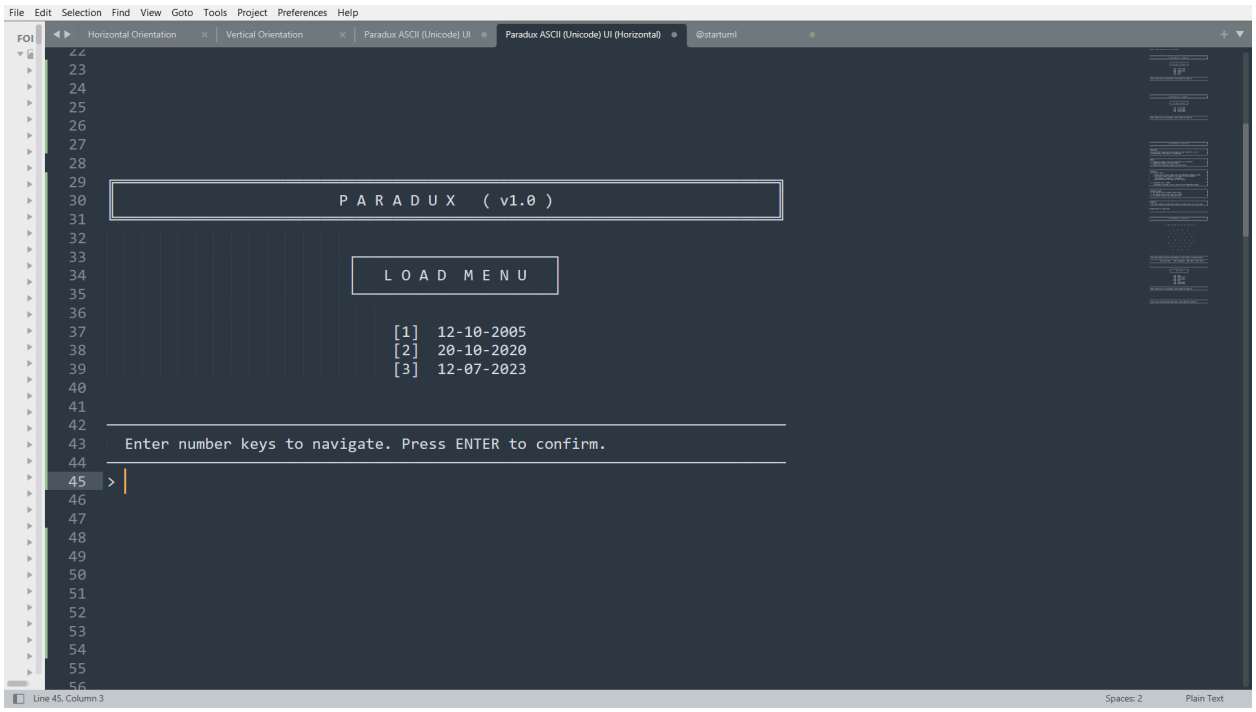


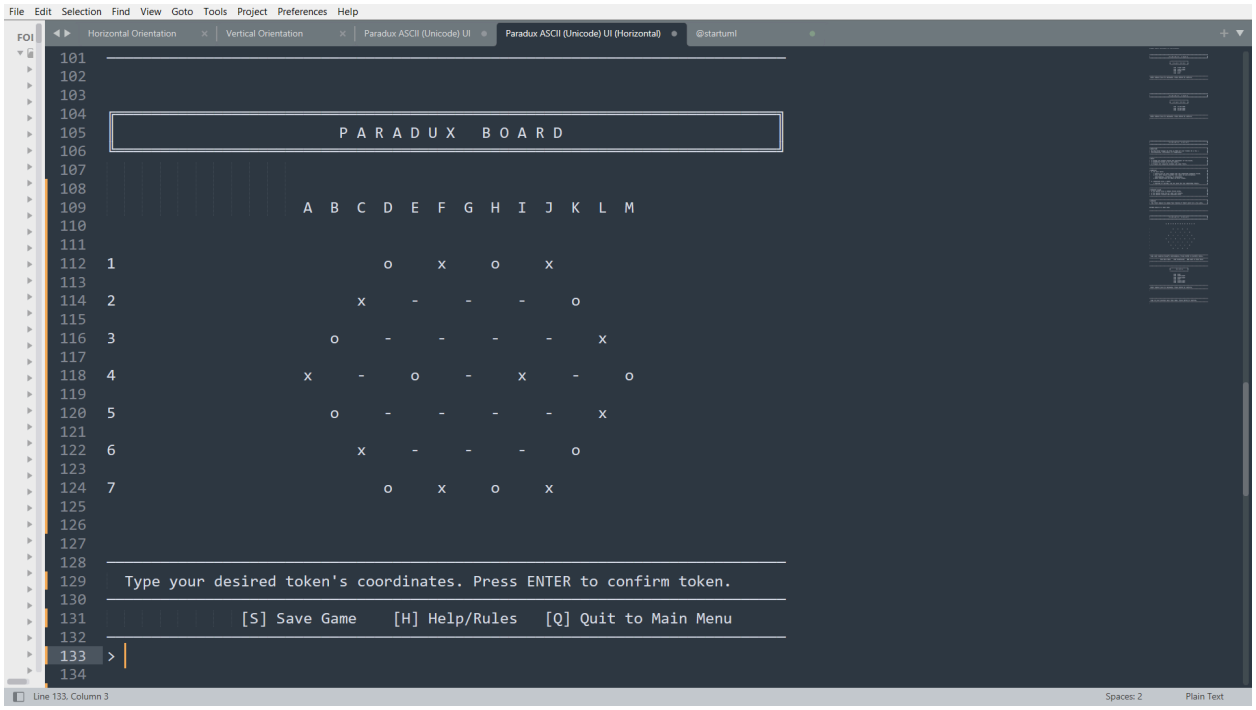
UC8: Load Game

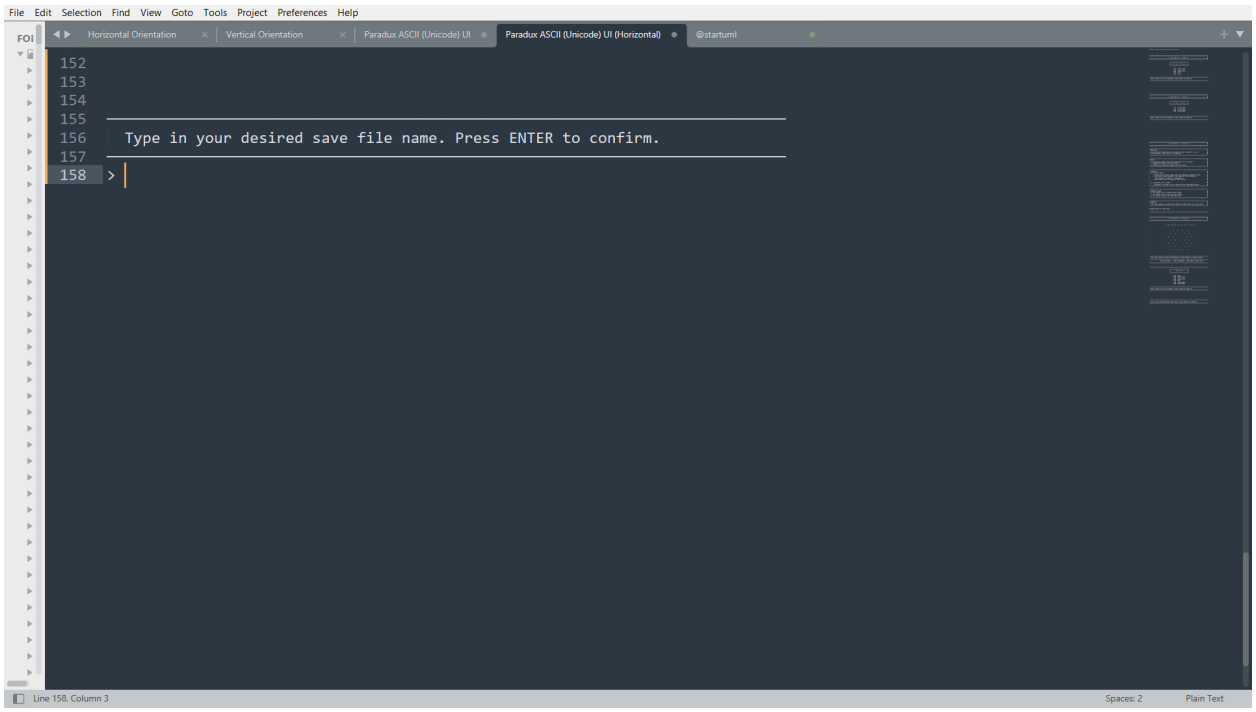
ASCII GUI design

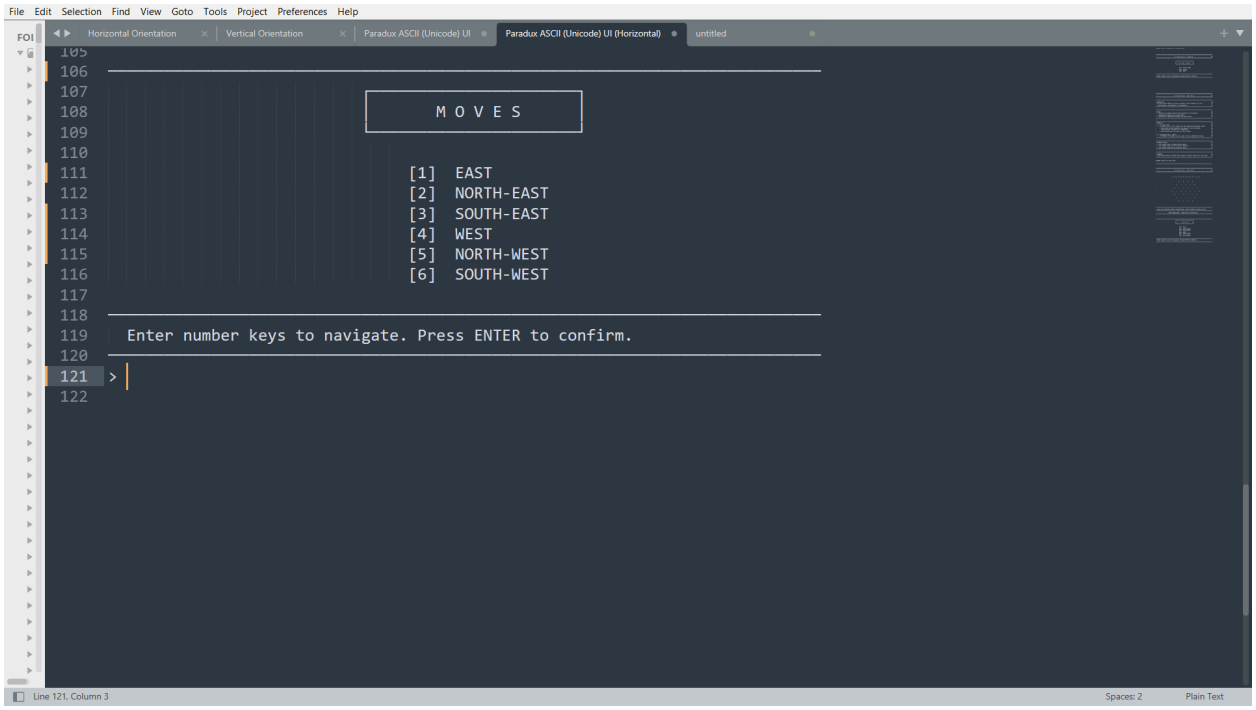



File Edit Selection Find View Goto Tools Project Preferences Help
Horizontal Orientation
Vertical Orientation
Paradux ASCII (Unicode) UI
Paradux ASCII (Unicode) UI (Horizontal)
untitled
1. On your turn:
▪ Select one of your tokens and one adjoining opponent token.
▪ Move BOTH tokens together one space in any direction (horizontal, vertical, or diagonal).
▪ Both tokens must be able to move freely.
2. Alternate Move — SWAP:
▪ Instead of moving, you may swap the two adjoining tokens.
MOVEMENT RULES
▪ You cannot move a single token alone.
▪ You cannot move two of your own tokens.
▪ You cannot reverse the previous move.
WINNING
The first player to align four tokens of their color in a row wins.
[ENTER] Return to Main Menu
Line 20, Column 3
Spaces: 2
Plain Text


File Edit Selection Find View Goto Tools Project Preferences Help
Horizontal Orientation
Vertical Orientation
Paradux ASCII (Unicode) UI
Paradux ASCII (Unicode) UI (Horizontal)
@startuml
P A R A D U X (v1.0)
L O A D M E N U
[1] 12-10-2005
[2] 20-10-2020
[3] 12-07-2023
Enter number keys to navigate. Press ENTER to confirm.
>
Line 45, Column 3
Spaces: 2
Plain Text


File Edit Selection Find View Goto Tools Project Preferences Help
Horizontal Orientation
Vertical Orientation
Paradux ASCII (Unicode) UI
Paradux ASCII (Unicode) UI (Horizontal)
@startuml
P A R A D U X B O A R D
A B C D E F G H I J K L M
1 o x o x
2 x - - - o
3 o - - - - x
4 x - o - x - o
5 o - - - - x
6 x - - - o
7 o x o x
Type your desired token's coordinates. Press ENTER to confirm token.
[S] Save Game [H] Help/Rules [Q] Quit to Main Menu
>
Line 133, Column 3
Spaces: 2
Plain Text


File Edit Selection Find View Goto Tools Project Preferences Help
Horizontal Orientation
Vertical Orientation
Paradux ASCII (Unicode) UI
Paradux ASCII (Unicode) UI (Horizontal)
@startuml
Type in your desired save file name. Press ENTER to confirm.
>
Line 158, Column 3
Spaces: 2
Plain Text


File Edit Selection Find View Goto Tools Project Preferences Help
Horizontal Orientation
Vertical Orientation
Paradux ASCII (Unicode) UI
Paradux ASCII (Unicode) UI (Horizontal)
untitled
M O V E S
[1] EAST
[2] NORTH-EAST
[3] SOUTH-EAST
[4] WEST
[5] NORTH-WEST
[6] SOUTH-WEST
Enter number keys to navigate. Press ENTER to confirm.
>
Line 121, Column 3
Spaces: 2
Plain Text