

SENG300 Final Project

Game Logic Planning

SENG 300

Dr. Steve Sutcliffe

Feb. 25th, 2025

Table of Contents	01
Project Overview	02
Requirements	03
Sprint 1 - Class Diagrams	04
Universal Classes	04
Piece Classes	05
Game Classes	07
Sprint 2 - Use Case Diagrams	08
Use Case Diagrams and Descriptions Split	10
Class Diagrams Split	11
Game Logic Implementation Split	12
Weekly Meeting Notations	13

Project Overview:

We have separated this project into 5 basic sprints in efforts to keep the team agile and dynamic:

1. Sprint 1 ((2025-02-25) - (2025-03-03)) has been designated to develop class diagrams and a skeletal framework for what the game implementation will look like moving forward.
2. Sprint 2 ((2025-03-04)- (2025-03-07)) has been designated to the creation of proper use case diagrams with the involvement of the other core teams of OMG.
3. Sprint 3 ((2025-03-08)- (2025-03-19)) has been set to begin development for the code of the project and beginning to implement the high and medium priority classes for Tic Tac Toe, Connect 4, Checkers . The goal of this sprint is to get most core features completed and have a foundation for the Chess implementation in the final sprint.
4. Sprint 4 ((2025-03-20)-(2025-03-31)) has been designated for implementation of low priority features of TTT, C4, and Checkers, and the final implementation of Chess's core features.
5. Sprint 5 ((2025-04-01)-(2025-04-13)) will be the final code review and full integration with all other teams.

Requirements:

As of *Project_v1.0.1*:

Build a small set of simple games for testing and demonstration purposes. Implement 3 of the following:

- **Chess: Implement the basic game rules and turn-based gameplay**
- **Go: Provide the code mechanics for placing stones and checking for captures.**
- **Tic-Tac-Toe: A lightweight turn-based game to validate basic functionality**
- **Connect Four: For testing multiplayer interactions in real-time**
- **Checkers: Useful for testing both player turns and piece movement logic.**

Planning document(s): Prepare any planning documents that you might need which lists all of the milestones, task completion dates, etc., for the programming part of your project. You should use these documents throughout your project to keep on track. Do not include personal/group info, and submit these to the **public Dropbox.** You should plan to begin working on the code for your project as soon as possible (especially before the P2 deadline).

Structure Diagram: Include a ``class_diagram.png`` or ``class_diagram.svg`` that documents your program's most important, vital or complex parts. Include all the parts you consider vital, but feel free to abstract what isn't essential. Do not include personal/group info; submit this to the public Dropbox.

Use Case Descriptions/Diagram: Create a ``use_case_descriptions.pdf`` that includes a diagram and the use cases that cover your most important interactions. Do not include personal/group info; submit this to the public Dropbox.

Sprint 1 - Class Diagrams:

We divided the classes into three general categories to help with planning those being:

- **Universal:** Classes that will be used by all games and act as the skeleton of all current and future games.
- **Piece:** Classes that represent a unique piece in a game.
- **Game:** Classes that encompass the game loop of supported games.

Universal Classes:

Board:

- Priority: **High**
- Assigned: M6
- Description: This class will represent the game board, holding the pieces and their positions. It aims to create a 2d string array that will serve as the board for all games and create objects of each piece and populate said board with them as a default state for all games. The size of the board will change depending on the game, as well as the type and number of pieces.

Player:

- Priority: **High**
- Assigned: M4
- Description: This class will represent a player in the game. It will create an instance of the player that we can bind to each user. This class will be responsible for the “ownership” of pieces, and potentially hold the score(Win/loss ratio).

GameState:

- Priority: **High**
- Assigned: M6
- Description: This class will be an enumerated class that represents the state of the game. The values are projected to be P1_TURN, P2_TURN, P1_WIN, P2_WIN, DRAW, CHECK, SETUP. These values will be updated throughout the game loop, each value is named to reflect the state the game is in. SETUP will account for creating the game board, and while players are determining other factors for the game.

Game:

- Priority: **High**
- Assigned: M1
- Description: This will be a super class that serves as the base for all games in the system. Creating the instance of the games classes Chess, Checkers, TicTacToe and Connect4. Creating 2 instances of Player, and calling on the board class to create the proper board.

Controller:

- Priority: Medium
- Assigned: M4
- Description: This class will handle user input and interaction with the game. Ie how the user interacts with pieces, and play areas.

GameRules:

- Priority: Low
- Assigned: M6
- Description: This class will provide information about the rules of a specific game. We imagine there will be a sort of get help button that when pressed outputs a very basic overview of the game. Ie, Bishops can move diagonally, Tic Tac Toe is won by getting 3 of your shapes in a row, etc.

Piece Classes:

CheckersPiece:

- Priority: High
- Assigned: M2
- Description: Represents a checkers piece, it will have a unique reachedEnd attribute for the implementation of crowning, and a unique move function in comparison to the chess pieces. It will also have a ReachedEnd method that is called to enable crowned movement

Connect4Piece:

- Priority: High
- Assigned: M3
- Description: Represents a Connect4 piece, it will have a unique color per team, and unique logic to ensure the piece goes to the bottom most slot available.

TTT_Piece:

- Priority: High
- Assigned: M2
- Description: Represents TicTacToe pieces, it will have a unique shape per team.

Piece:

- Priority: Medium
- Assigned: M5
- Description: An interface for moving and stationary pieces and will define the common behavior for all pieces, whether they can move or not. It will contain colour, coordinates, ownership, and score.

Piece(Stationary):

- Priority: Medium
- Assigned: M3
- Description: Represents a piece that cannot move (TicTacToe, Connect 4 pieces). This will have functions to place a piece on the board, see if that space is valid, and act as a template for stationary pieces.

Piece(moving)

- Priority: Medium
- Assigned: M4
- Description: Represents a piece that can move (Checkers, all Chess pieces). This will have functions to move the piece, see if moves are valid, and provide a basis for the movement function. Each piece will need a unique implementation of the feature.

Pawn:

- Priority: Medium
- Assigned: M3
- Description: Represents a pawn, it will have a firstMove attribute, unique score, and uniquely a promotion method, and an enPassant function.

Rook:

- Priority: Medium
- Assigned: M2
- Description:
Represents a rook, it will have a firstMove attribute, unique score, and hasMoved.

Knight:

- Priority: Medium
- Assigned: M3
- Description: Represents a knight, it will have a unique score attribute. All other traits will be inherited.

Bishop:

- Priority: Medium
- Assigned: M2
- Description: Represents a queen, it will have a unique score attribute. All other traits will be inherited.

Queen:

- Priority: Medium
- Assigned: M2
- Description: Represents a knight, it will have a unique score attribute. All other traits will be inherited.

King:

- Priority: Medium
- Assigned: M3

- Description: Represents a king, it will have a unique score attribute, firstMove. All other traits will be inherited.

Game Classes:

Checkers:

- Priority: High
- Assigned: M5
- Description: Simulates a checkers game, initializes a 8x8 tiled board using the board class and fills 12 pieces per player on the board in position. Creates two players (Colour 1 and Colour 2) sets game state to P1_TURN. Finishes if GameState == DRAW, P1_WIN or P2_WIN.

Connect4:

- Priority: High
- Assigned: M1
- Description: Simulates a Connect 4 game, initializes a 6x7 board using the board class and creates 21 same coloured pieces per player to use. Creates two players (Colour 1 and Colour 2) sets game state to P1_TURN. Finishes if GameState == DRAW, P1_WIN or P2_WIN.

TicTacToe:

- Priority: High
- Assigned: M1
- Description: Simulates a Tic TacToe game, initializes a 6x7 board using the board class and creates 21 same coloured pieces per player to use. Creates two players (Colour 1 and Colour 2) sets game state to P1_TURN. Finishes if GameState == DRAW, P1_WIN or P2_WIN.

Chess:

- Priority: Medium
- Assigned: M5
- Description: Simulates a chess game, initializes a 8x8 tiled board using board class and fills it with 8 pawns, 2 rooks, 2 knights, 2 bishops, 1 queens, 1 kings per player in position. Creates two players (White and Black), sets game state to P1_TURN. Finishes if GameState == DRAW, P1_WIN or P2_WIN. Will track how many pieces have been taken, and track the material advantage on each side.

Sprint 2 - Use Case Diagrams:

playGame:

- Assigned: M1
- Description: A player starts and plays a game (chess, checkers, connect 4, tic tac toe) against another player. High-level use case that encompasses many of the other actions
- Actor: Player
- Precondition:
- Postcondition:

movePiece:

- Assigned: M1
- Description: A player moves one of their pieces on the game board according to the rules of the game.
- Actor: Player
- Precondition:
- Postcondition:

placePiece:

- Assigned: M2
- Description: A player places a piece on the game board (this might be more specific to games like Connect 4 or Tic-Tac-Toe where pieces are placed rather than moved).
- Actor: Player
- Precondition:
- Postcondition:

Surrender:

- Assigned: M2
- Description: A player resigns from the game, forfeiting the match to their opponent.
- Actor: Player
- Precondition:
- Postcondition:

inCheck:

- Assigned: M2
- Description: A player's king is under attack
- Actor: Player
- Precondition:
- Postcondition:

enPassant:

- Assigned: M3
- Description: A special capture move in chess
- Actor: player
- Preconditions:
- Postconitions:

Castling:

- Assigned: M3
- Description: A special move in chess involving the king and one of the rooks.
- Actor: Player
- Precondition:
- Postcondition:

Checkmate:

- Assigned: M3
- Description: The king is in check and has no legal move to remove it from check.
- Actor: System
- Precondition:
- Postcondition:

4inaRow:

- Assigned: M4
- Description: A player wins the game by connecting four of their pieces in a row (horizontally, vertically, or diagonally).
- Actor: System
- Precondition:
- Postcondition:

3inaRow:

- Assigned: M4
- Description: A player wins the game by connecting three of their pieces in a row (horizontally, vertically, or diagonally).
- Actor: System
- Precondition:
- Postcondition:

Capture:

- Assigned: M5
- Description: A player captures an opponent's piece by moving their own piece to the occupied space.
- Actor: System
- Precondition:
- Postcondition:

noPieces:

- Assigned: M5
- Description: A player has no legal moves available.
- Actor: System
- Precondition:
- Postcondition:

Use Case Diagrams and Descriptions Split:

Use Case Description/diagram	Member
playGame	M1
movePiece	M1
placePiece	M2
surrender	M2
inCheck	M2
enPassant	M3
castling	M3
checkmate	M3
4inaRow	M4
3inaRow	M4
capture	M5
noPieces	M5

Class Diagrams Split:

.Class:	.Description:	.Priority:	.Game	Person for class diagrams:
Board	Create, destroy, and maintain board.	High	All	M6
Player Abstract	Player attributes such as win, lose, red team, black team, mainly properties	High	All	M4
Piece(stationary) abstract		High	Connect 4, Tic Tac Toe	M4
Piece(moving) abstract		High	Chess, Checkers	M4
Controller	Input class	High	All	M4
GameState	ENUM class, status of current game (ONGOING, P1_WIN, P2_WIN, QUIT)	High	All	M6
Chess	Envelops the chess game	High	Chess	M5
Checkers	Envelops the checkers game	High	Checkers	M5
Tic Tac Toe	Envelops the tic tac toe game	High	Tic Tac Toe	M1
Connect 4	Envelops the Connect 4 game	High	Connect 4	M1
Pawn	Pawn class, inherit moving piece	High	Chess	M3
Rook	Rook class, inherit moving piece	High	Chess	M2
Knight	Knight class, inherit moving piece	High	Chess	M3
Bishop	Bishop class, inherit moving piece	High	Chess	M2

King	King class, inherit moving piece	High	Chess	M3
Queen	Queen class, inherit moving piece	High	Chess	M2
c4Piece		High	Connect 4	M3
tictacPiece		High	Tic Tac Toe	M2
GameRules	Output rules of the game for player	Medium	All	M6

Game Logic Implementation Split:

M5: Tic Tac Toe/Checkers/Chess

M3: Checkers/Chess

M6: Checkers/Chess

M2: Checkers/Chess

M1: Connect Four

M4: Connect Four

Weekly Meeting Notations:

GUI:

- Displaying captured pieces for chess/checkers?
- Showing next potential moves chess/checkers?
- Generating boards?
- Piece colours?
- Displaying game rules?
- What do you need from us?

Networking

- How are we dealing with players?

Leaderboard

- Demands?
- Clarification:
 - Counter for pieces captured/types?
 - Counter for pieces promoted?
 - Win-Loss ELO per game?
 - Overall ELO?
 - Additional Point systems?