6.4D – D Level Custom Program

Design Overview for <<Tic Tac Toe game>>

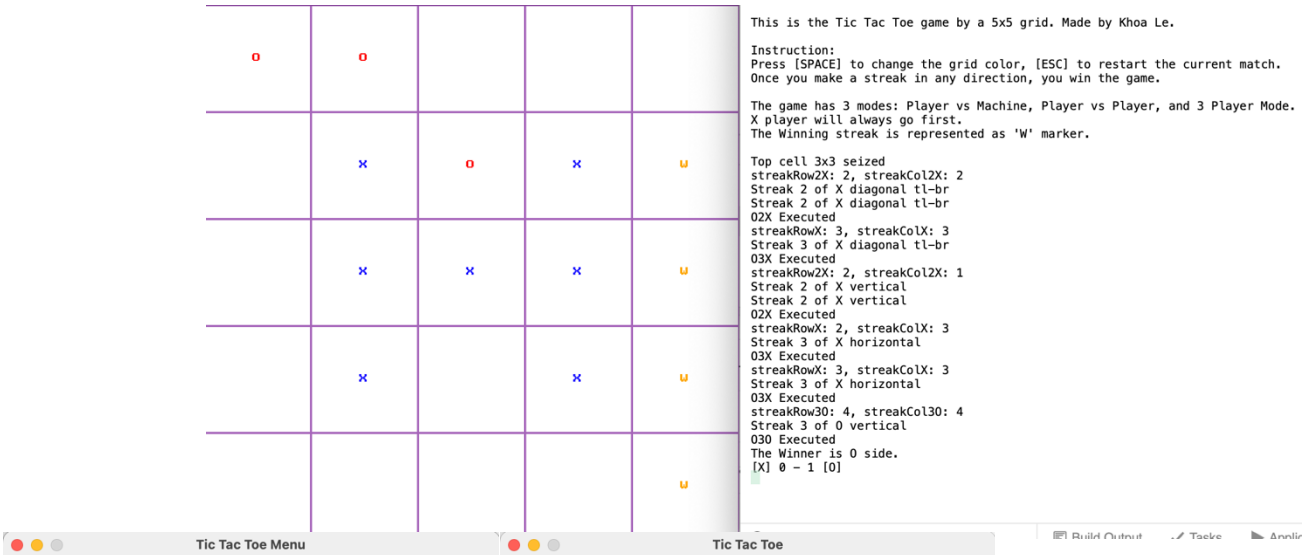
Name: Dang Khoa Le
Student ID: 103844421

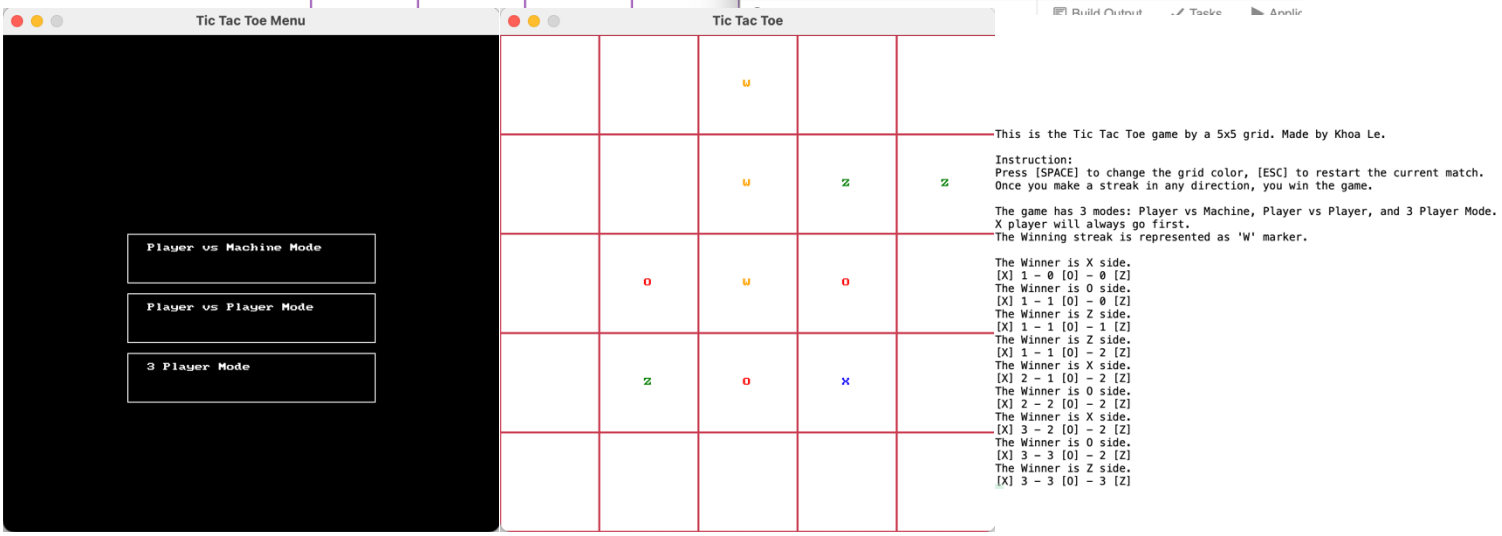
Summary of Program:

A. Overview

This program represents a classic Tic Tac Toe game within an engaging 5x5 grid cell board. Its foundation lies in the effective application of Object-Oriented Programming (OOP) principles, ensuring a logical and seamless gaming experience. By carefully integrating various classes, variables, and game parameters, this game provides a structured and coherent framework, ensuring user to have a good experience.

One of its distinguishing features is the incorporation of an Artificial Intelligence (AI) player, elevating the gaming experience by offering challenging competition. The game encompasses a rich amalgamation of player interactions, strategical depth, and thoughtful design.





B. Gameplay

1. Modes

The game offers a versatile selection of modes that can be selected from the Menu window. Players can engage in classic Player vs. Player (PvP) matches, challenge a formidable AI opponent in Player vs. Machine (PvM) mode, or partake in the 3 Player (3P) showdown.

2. Markers

This game contain 3 main marker types, including “X” displayed in blue, “O” displayed in red, and “Z” displayed in green.

There will be an additional a dynamic "W" marker in striking orange symbolizes the victorious player's winning streak.

In Player vs Player (PvP) and Player cs Machine (PvM) mode, there will be 2 marker types, including X and O, meanwhile, in the 3 Players mode (3P), three players can each choose a unique marker from X, O, and Z.

3. Instructions

Players can seamlessly control the game using key commands, pressing[SPACE] to change the grid colour and [ESC] to restart the current match at any time.

In all game modes, the "X" player takes the first move, and the progression of a winning streak is signified by the formidable "W" marker.

4. Winners

In PvP and PvM mode, the winning streak is detected by a 4-a-kind marker streak (either X or O marker), while 3P mode’s winning streak is only identified by a 3-a-kind streak (either X, O or Z marker).

Once there is a winner, the game board will be freezed (players can’t no longer make moves until pressing [ESC] key), while the terminal console will print out the scoresheet and identify the winning player in these format:

+ For PvM and PvP game modes:
“The Winner is {winner} side."
“[X] {XScore} - {OScore} [O]"
+ For 3P game modes:
“The Winner is {winner} side."
“[X] {XScore} - {OScore} [O] - {board.ZScore} [Z]”

5. Machine

The Tic Tac Toe game also consist of a machine player (AI), which can interact with users in the Player vs Machine mode (PvM). This player machine will play their O marker and trying to beat the player by using different conditional event testing and move prediction. Like a human player, the machine employs a combination of offensive and defensive tactics, always striving for victory while avoiding defeat.

6. Strategies

The machine player game moves is constructed under these logical strategies:

Strategy 1 (Moderating): Seize the central cell of the board if possible. (Default).

Strategy 2 (Moderating): Seize the 3x3 grid area surround the central cell in first 3 moves if possible. (Priority level 5).

Strategy 3 (Attacking): Once having a 2-a-kind streak of O, try to make the third. (Priority level 3).

Strategy 4 (Attacking): Once having a 3-a-kind streak of O, try to make the fourth. (Priority level 1).

Strategy 5 (Defensive): Once confronting a 2-a-kind streak, try to avoid the third. (Priority level 4).

Strategy 6 (Defensive): Once confronting a 3-a-kind streak, try to avoid the fourth. (Priority level 2).

Strategy ∅ (Moderating): Make random moves if it doesn't meet any other strategies. (Default).

Alongside with implementing the game logics and strategies, a priority level consideration also has been remarked, which ensure player machine always prior their moves strictly for lower-numbered level (Priority level 1 should be met beforehand to the Priority level 5). These Priority level is set, in order to prevent machine player met different strategies at once and accidentally violate these conditional checkers, also assuring they can utilize the best move each time to win the game.

7. Board

The game board is drawn by using SplashKitSDK extension, creating a squared window interface with the width and height of 500 units. There are 6 horizontal straight lines and 6 vertical straight lines to create a 5x5 grid inside the game window, these gird is black at default, however, their colour can be changed using “RandomRGBColor” function from SplashKit. In the Menu window, 3 buttons indicating 3 game modes also being drawn by SplashKit.

Upon starting the game, two windows materialize. The Menu window emerges initially, where players select their preferred game mode. After the choice is made, this window gracefully transitions to the board window, providing a platform for players to engage with the Tic Tac Toe game according to their chosen mode.

The game also keep tracking on the user’s mouse position and detect their clicks interaction through the intuitive SplashKit framework.

C. Method

1. Menu

From the user mode selection that have been premade in the Menu window, the game use boolean variables to assign which mode has been chosen, which return the corresponding private bool to true, this boolean value will be used in PlaceMarker to redirect user to their chosen gamemode.

2. Draw

The game's graphical user interface is crafted with the SplashKitSDK extension, tt manifests as a 500x500 unit square window. Integer parameters for the number of ‘rows’ and ‘cols’ (rows and columns) is defined, the squared cell size of the board is initialized by dividing the width and height to the number of rows and cols (500/5 = 100 units wide). The cell’s grid colour is black at default, which can be changed using SplashKit’s RandomRGBColor(255) method. Initially, the cell insider is set to be ‘PlayerType.Empty’ (virtual type of player indicating the cell is empty).

3. PlaceMarker

This method checks for user's mode selection and initialise the game corresponding to the game mode.

In PvP mode, X marker is set to be placed at odd-numbered moves and O marker is set to be placed at even-numbered moves. Game users can place their markers onto the board only if the board is empty.

In PvM mode, player uses X marker, which is set to be placed at odd-numbered moves and machine uses O marker, which is set to be placed at even-numbered moves. Game users can place their markers onto the board only if the board is empty. Player will go first and if they place marker elsewhere apart from the central cell, machine will place O marker there (Strategy 1). Within the first 3 moves of the machine, they will try to take advantage of the inner 3x3 grid central of the game board neglecting all player's moves unless they are able to perform an early streak (Strategy 2). After the first 3 rounds, the game will ignite their intensity, a set of 4 conditional strategies is implemented to attack and defense against player (Strategy 3 to 6). Else, machine makes random moves by "MakeMachineMove" method.

In 3P mode, we define the current player as var (variable) type _currentPlayer of the PlayerType enum, for each moves by the user player, the integer value of _currentPlayer will be increase by 1 amongst the three of them.

4. FindStreak

There are FindStreak2, FindStreak3, StopStreak2, and StopStreak3 methods, which are used as the streak checker conditions, while each of these are allocated, their boolean return true. The checker works by seeking for the same kind of marker X and O within the same direction (vertical, horizontal, diagonal top-left-bottom-right, and diagonal top-right-bottom-left), if they may have a streak of 2 and/or 3 markers. Once there are a streak allocated, the corresponded target cell (rows and cols) value will be sent back to PlaceMarker method to execute the machine move.

5. WinnerCheck

The CheckForWinner and ThreePlayerWinnerCheck check for the winning streak in this game. Similar to the FindStreak, they seek for the streak of 4 or 3 markers with the same kind and set the winning streak to the MarkWinningStreak and ThreePlayerMarkWinningStreak methods to execute the "W" streak. The winner side of the game also being returned to the Main() program to print out the scoresheet to the terminal console.

D. Testing

Different debug purposes messages has been implemented in this code, printing different testing messages to the terminal (if enabled), for developers to ensure the machine player works properly. These debugging message are used to allocate any streak position, type of streak and whether a corresponding move has been made thoughtfully.

Required Roles:

Table 1: <<Drawing>> (class)

Responsibility	Type Details	Notes
Manages the game board, player move, machine move strategies, and checking for winner method.	**Fields**: board (2D array of PlayerType), markers (List of Player), _gridColor (Color), _gridSize (int), _vsMachineMode (bool), _vsPlayerMode (bool), _threePlayerMode (bool), _currentPlayer (PlayerType), XScore (int), OScore (int), ZScore (int), _horizontalStreak2O (bool), _horizontalStreak2O (bool), _verticalalStreak2O (bool), _TLBRStreak2O (bool), _TRBLStreak2O (bool), _horizontalStreak3O (bool), _verticalalStreak3O (bool), _TLBRStreak3O (bool), _TRBLStreak3O (bool), _horizontalStreak2X (bool), _verticalalStreak2X (bool), _TLBRStreak2X (bool), _TRBLStreak2X(bool), _horizontalStreak3X (bool), _verticalalStreak3X (bool), _TLBRStreak3X (bool), _TRBLStreak3X(bool), canCompleteStreak2O (bool), canCompleteStreak3O (bool), canStopStreak2X (bool), canStopStreak3X (bool), streakRow2O (int), streakCol2O (int), streakRowO (int), streakColO (int), stopRow2X (int), stopCol2X (int), stopRowO (int), stopColO (int), **Methods**: ChangeGridColor(), Draw(), PlaceMarker(), FindStreak2(), FindStreak3(),StopStreak2(),StopStreak3(), MakeMachineMove(), CheckForWinner(), ThreePlayerWinnerCheck(), MarkWinningStreak(),ThreePlayerMarkWinningStreak(), SwitchToPlayerVsMachineMode(), SwitchToPlayerVsPlayerMode(), SwitchToThreePlayerMode(), UpdateScores(), UpdateScores3P(), Reset()	This class stores most of the game's fundamental set-ups and functionalities, logics. It also contain the machine player logics and function, storing conditions and checkers components.

Table 2: << PlayerType >> (enumeration)

Value	Notes
X, O, Z, W, Empty.	This enumeration represents the types of players and/or markers on the board.

Table 3: << Player>> (class)

Responsibility	Type Details	Notes
Represents a player with a specific player type (X, O, or Z).	**Fields**: Type (PlayerType).	Represent the real player (human player)

Table 4: << Machine>> (class)

Responsibility	Type Details	Notes
Represents a machine player with a specific player type (marker O).	**Fields**: Type (PlayerType).	Represent the machine player (AI player)

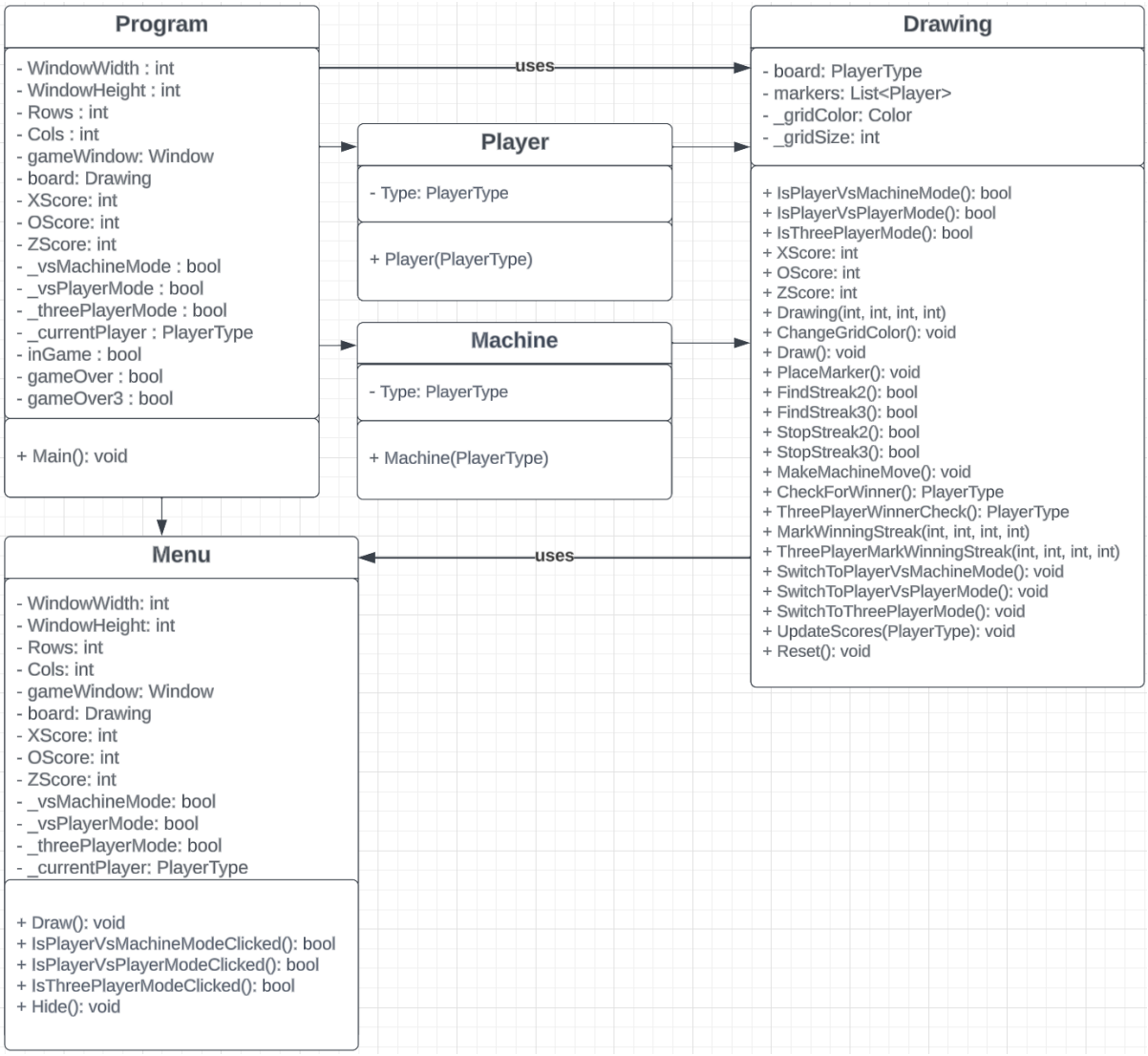
Table 5: <<Menu>> (class)

Responsibility	Type Details	Notes
Manages the game menu and user interface to select game modes.	**Fields**: _window (Window), _PlayerVsMachineModeButton (Rectangle), _playerVsPlayerModeButton (Rectangle), _threePlayerModeButton (Rectangle). **Methods**: Draw(), IsPlayerVsMachineModeClicked(), IsPlayerVsPlayerModeClicked(), IsThreePlayerModeClicked(), Hide().	This class stores the game modes menu, explicitly differs each mode after entering any given modes.

Table 5: <<Program>> (class)

Responsibility	Type Details	Notes
Field type, parameter, and return types. A general set-up for game sketch, defines modes and print game instruction.	**Fields**: WindowWidth: int, WindowHeight: int, Rows: int, Cols: int, gameWindow: Window, board: Drawing, XScore: int, OScore: int, ZScore: int, _vsMachineMode: bool, _vsPlayerMode: bool, _threePlayerMode: bool, _currentPlayer: PlayerType, inGame: bool, gameOver: bool, gameOver3: bool **Methods** Main(): void	A general Main class for generic parameters, types, controlling handle inputs from user.

Class Diagram:



Abstraction:

In this Tic Tac Toe game, abstraction is implemented through the use of classes that model the domain, generating a user interface, gaming space and encapsulate the relevant data and behaviours. Abstraction in this context simplifies the complexity of the game by defining clear, high-level interfaces and structures, and hiding the internal details. Here's how abstraction is implemented:

PlayerType (enum):

Abstraction is evident in the PlayerType enumeration. This enum abstractly represents the different player types in the game, including 'X', 'O', 'Z', 'W' (for the winning streak), and 'Empty' to indicate empty cells on the board. By using this enum, the actual player types are abstracted into simple, meaningful values.

Player and Machine classes:

Abstraction is demonstrated in the creation of the Player and Machine classes. These classes define the type of human and machine markers. They both have a PlayerType field, which abstractly represents their role in the game.

Drawing class:

The Drawing class abstracts the game board, including logic for drawing and updating it. It uses a 2D array of PlayerType to represent the board's state. Besides, this class also stores the logic and gameplay of the game, containing conditions for machine player moves and the method of checking streaks and winner. This abstraction simplifies the representation of the game's board and functionality by storing player markers, conditions and parameters in a structured manner.

Menu class:

The Menu class abstracts the game menu and user interface. It manages different game modes, such as Player vs. Player, Player vs. Machine, and 3-Player mode. Users interact with this abstract menu to select their preferred mode.

Main Program:

The Main program serves as the entry point for the game and abstracts various parameters and game states. It abstractly manages the current player, game modes, and the game's window dimensions.

These abstractions make it easier to work with the game logic and user interface. The specific details of how the game board is drawn, how players interact, and how the gamemodes' functions are abstracted into their respective classes. This separation of concerns enhances code readability and maintainability. It also allows for easy modifications or extensions of the game without affecting other parts of the code, adhering to the principles of Object Oriented Programming (OOP) and abstraction.

Inheritance:

Inheritance is a fundamental concept in OOP, and this Tic Tac Toe game demonstrates it effectively. There is a strong connection relationship between the Player and Machine classes, at which the Player class serves as the base class, and it's inherited by the Machine class (machine player also use PlayerType).

Besides, the Menu and Drawing classes also shares their properties and behaviours among themselves as well as inherited the others from the Main Program such as the winner and scores variables, window's configuration, chosen modes, PlayerType, and boolean values. This demonstrates inheritance, which reduces code redundancy.

Polymorphism:

Polymorphism allows objects of different derived classes to be treated as objects of the base class. In the Tic Tac Toe game, polymorphism is primarily used to handle player and machine moves.

Player Moves: Regardless of whether the player is a human or a machine, the game can handle their moves uniformly. This is made possible through polymorphism. Both human players and

the AI machine use a common method, PlaceMarker(), which is defined in the base class Player.

Overriding Methods: In the Machine class, it can override the PlaceMarker() method. The Machine class has its own implementation of this method to provide AI moves. This is an example of polymorphism, as it allows different behaviours for the same method name based on the specific class instance.

C# Coding Conventions:

Naming Conventions:
The program's code follows C# naming conventions for variables and classes. For example, it use PascalCase for class names (Player, Machine) and camelCase for local variables (_currentPlayer). This consistency adheres to C# naming conventions.

Indentation:
Proper indentation is used throughout the code, which enhances readability. The game program consistently use tabs or spaces for indentation, following C# coding standards.

Additional implementations from Initial Plan:

Within this version of the Tic Tac Toe game, different upgrades have been implemented to ensure the game can be able to perform with minimal errors and provide a better interaction to the user.

1. MarkWinningStreak

The 2 methods of marking the winning streak have been assigned, which allows user to visualise their winning/losing streak of marker easily, enhance user interaction to the game. The winning streak of marker will be allocated in orange "W" marker, also, the game will be frozen and wouldn't allow user to interact until continue the game, this has also allowed them to learn from failures and enjoy their times with other players, especially in PvM and 3P modes.

2. FindStreak conditions

The streak detections have been upgraded and extend to the total of 4 methods, which enable the machine player to detect different streak of their and/or user's marker in various direction. This also allows them to allocate the next move based on the conditional checkers.

3. PlaceMarker

PlaceMarker method, which control the game interface has been upgraded, which allows the machine player to execute their moves more logical and understand which condition they should put in prior.