1. **Statement of completion**

Game requirements:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Detailed information | Completed | Comments |
| 1 | C# console application on .NET 6 providing a text-based command-line interface | Yes | * The program has been tested with the commands: dotnet clean, build, and run * .NET version 6.0.201 |
| 2 | The framework is extensible and reusable for a wide variety of board games | Yes | Template Method Design Pattern used as a template board game.  Abstract class: GameController  Concrete class: WTTTGameController |
| 3 | 2 modes of play, including:   * Human versus human * Human versus computer | Yes | Abstract Factory Design Pattern used.   * Abstract class: Player * 2 concrete classes: HumanPlayer & ComputerPlayer |
| 4 | Implementing one game between Numerical Tic-Tac-Toe and Wild Tic-Tac-Toe (**chosen**) | Yes | * Symbol ‘X’ for player1 & ‘O’ for player2. * Winning rules for Wild Tic-Tac-Toe implemented. |
| 5 | For human players, the system must check the validity of moves | Yes | MakeMove() in the concrete class HumanPlayer has modified |
| 6 | For computer players, the system can randomly select a valid move | Yes | MakeMove() in the concrete class ComputerPlayer has modified |
| 7 | The game can be played from start to finish and can be saved and restored from a saved file | Yes | GameState can be saved to a file and be loaded from a file at any time. |
| 8 | All moves from both players can be undoable and re-doable | No | Some errors were found but still not able to fix properly. |
| 9 | Online help system | Yes | Help class implemented. |

1. **Overview of final design**

Summary of some change

|  |  |  |  |
| --- | --- | --- | --- |
| No | Preliminary design | Final design | Reasons for this change |
| 1 | GameStart class | Program class | * C# application starts with the Program class * The Main function within the Program class will execute, initiate the game, and create players |
| 2 | WildTicTacRule class | WTTTGameController class | * WTTTGameController is a concrete class from the abstract class GameController when applying Abstract Factory Design Pattern. * GameController is the framework for general board games:   + Start the game, creating a new Game State   + End the game   + Create 2 players   + Make turns for players |
| 3 | BoardRender class |  | This class was removed because Game State has own display function |
| 4 | GameState class | GameState class | Updated important attributes   * board: char[] * playerMove: int [] * turn: int * currentPlayerID: int |
| 5 | Player class | Player class | Updated important attributes   * playerID: int * symbol: char |
| 6 | giveCommand() in Player class | * giveCommand() in HumanPlayer class * giveCommand() in ComputerPlayer() | * Modify giveCommand() for concrete class HumanPlayer, user will type a command * Modify giveCommand() for concrete class ComputerPlayer, a valid move randomly generated. |
| 7 | Board game with positions in row and column | Board game with positions from 1 to 9 | A single dimensional Array is easier to operate than a multidimensional array. |

1. **Detailed Design Documents**

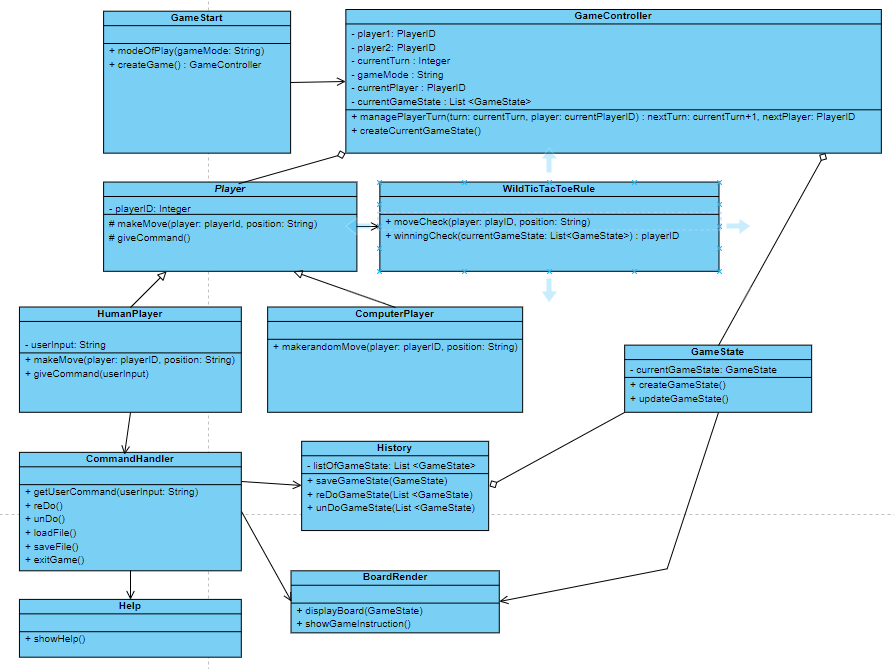
****

Diagram 1.1 Preliminary Class Diagram

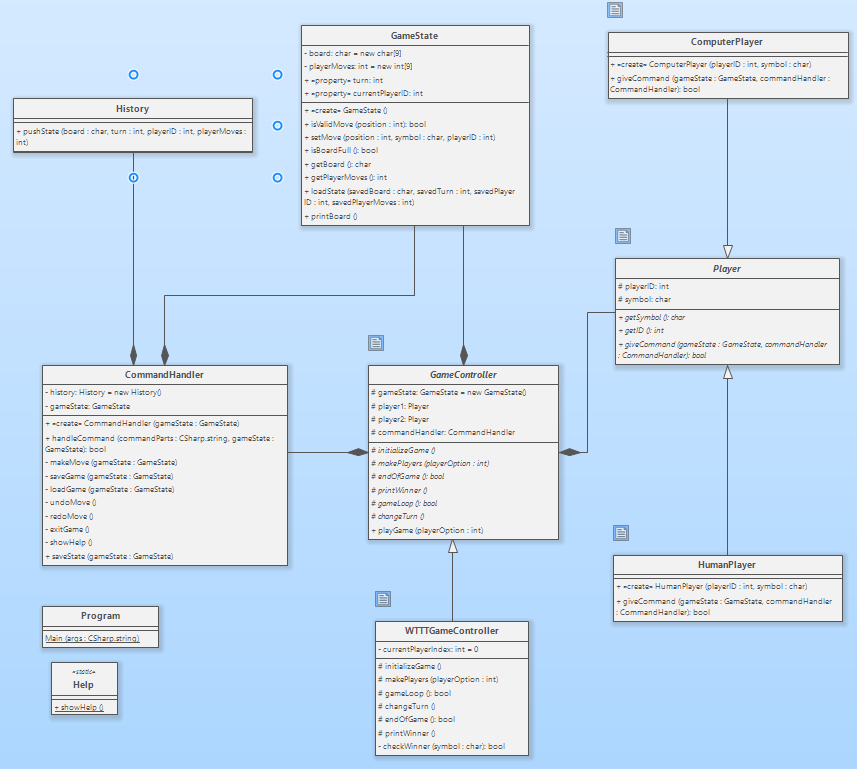
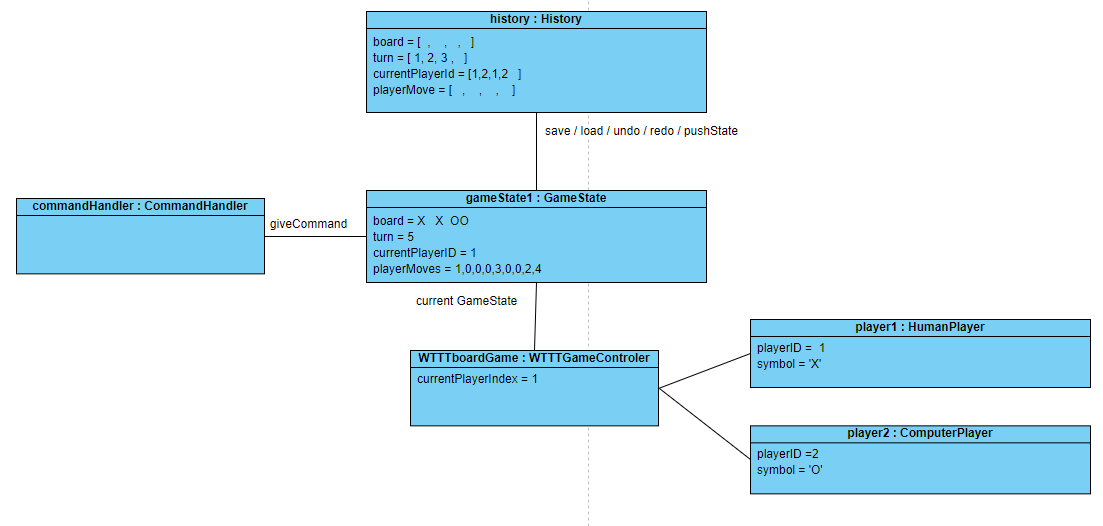
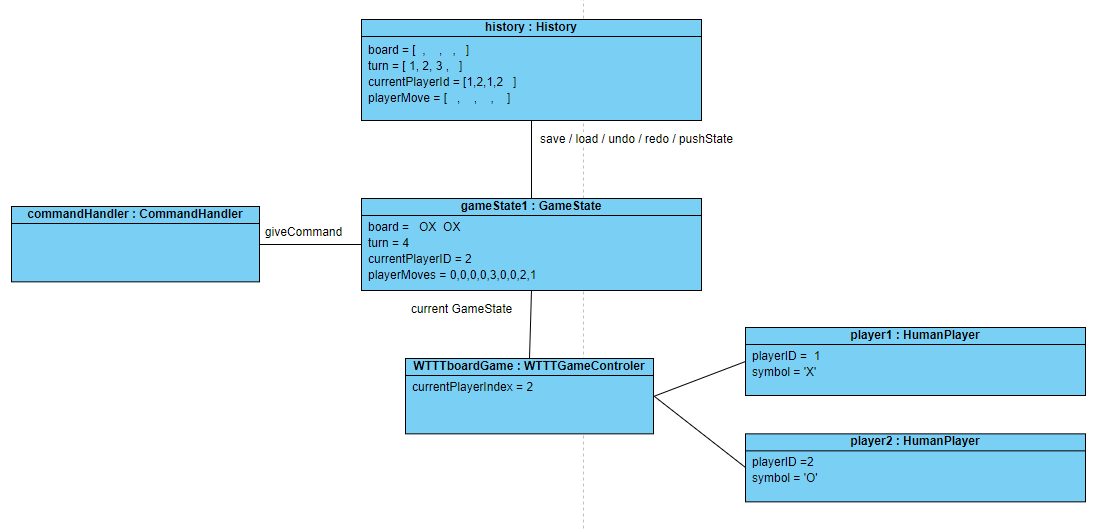


Diagram 1.2 Final Class Diagram

****

1.3 Final Object Diagram – Human vs Computer



1.4 Final Object Diagram – Human vs Human

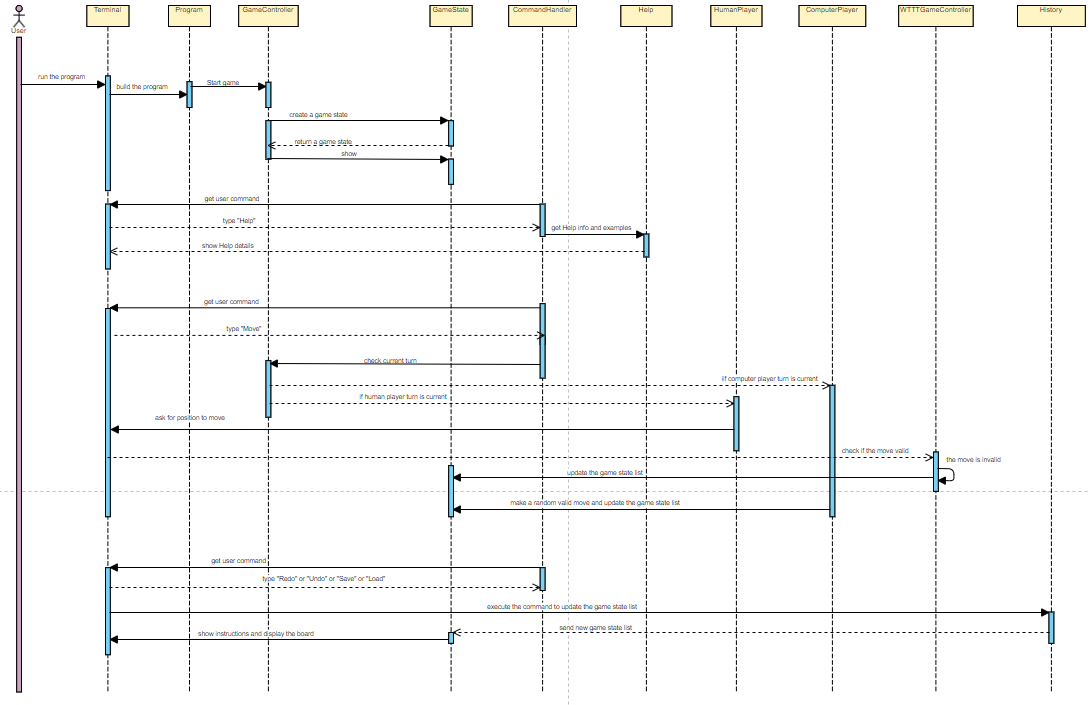
****

Diagram 1.5 Final Sequence Diagram

1. **Design Principles and Patterns**

There are two design patterns used in this application:

* Template Method design pattern
* Abstract Factory design pattern
  1. **Template Method design pattern**

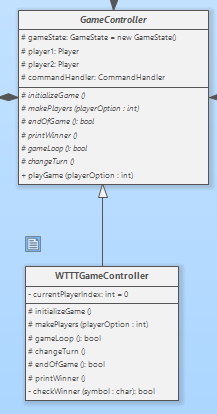
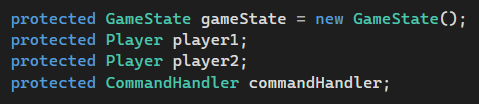
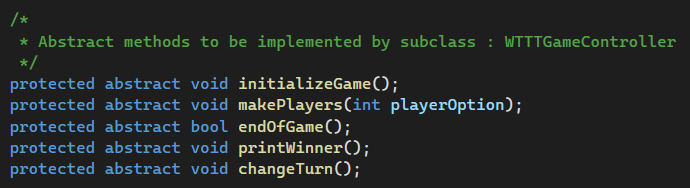


Diagram 1.6 Template Method Design Pattern

* GameController is the Abstract Class as a framework for general board games, that gives a game structure. All abstract functions must be declared empty with protected abstract:
  + Create a new GameState
  + Create a new CommandHandler
  + Create players
  + Make turns for all players
  + Check the winner
  + End game
* The Abstract Class must have the template method which is used by the Concrete Class. Here, playGame function is the template method of the Abstract Class GameController. And it will be used by the Concrete Class WTTTGameController.
* All concrete methods in the Concrete Class WTTTGameController must override to the Abstract methods in the Abstract Class GameController.
* All Abstract attributes within GameController Class:



* All Abstract methods within GameController Class:



* The Abstract Class can be extended to create another board game like Numerical Tic-Tac-Toe. However, in this assignment, the Wild Tic-Tac-Toe board game chosen to build.
  1. **Abstract Factory design pattern**

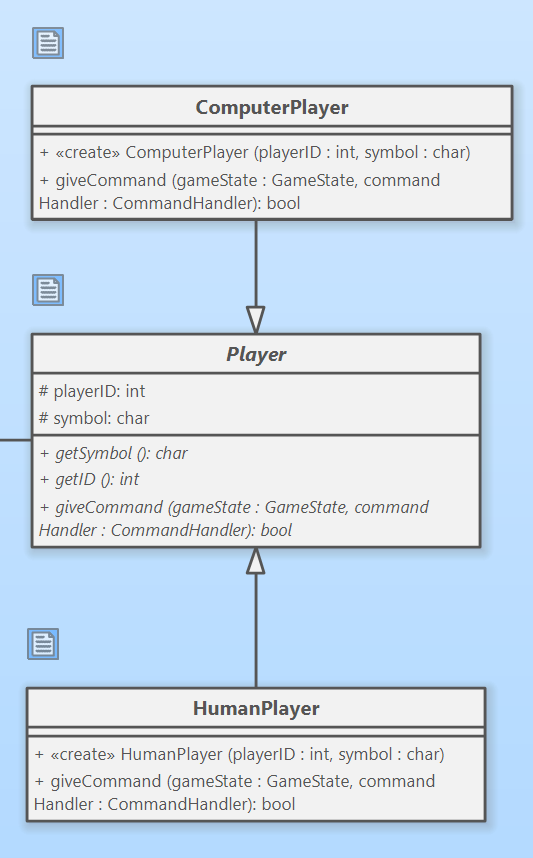
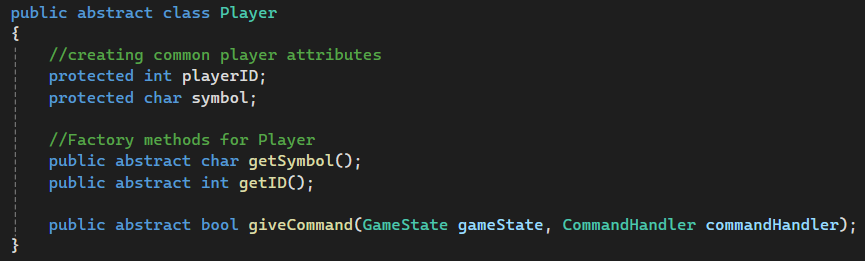
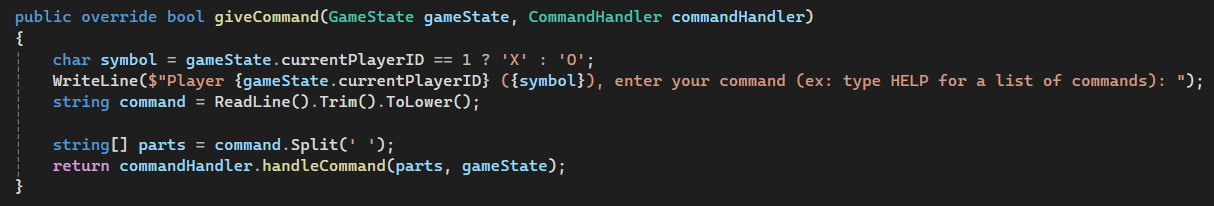


Diagram 1.7 Abstract Factory design pattern

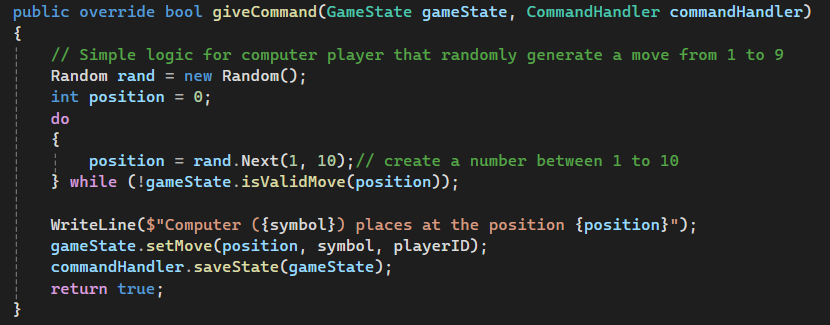
* Abstract Class Player is declared with common attributes and factory methods used by both Concrete Classes: HumanPlayer and ComputerPlayer:



* Each Concrete Class has its function which is different from other Concrete Class, although the function is same.
* For the Concrete Class: HumanPlayer, the function giveCommand will get the input of command form user.

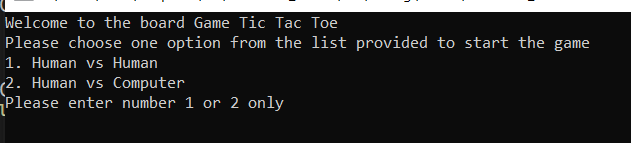


* For the Concrete Class: ComputerPlayer, the function giveCommand implements a random valid move:

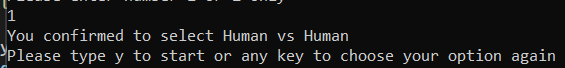


* By applying Abstract Factory design pattern, other type of player can be created without affecting the game structure.

1. **Execution instructions**
   1. The game starts with a welcome statement and a guide to enter one number 1 or 2 according to the mode of play preferred.



* 1. When a valid number is entered, an instructed statement will be shown to confirm a game started by typing ‘y’



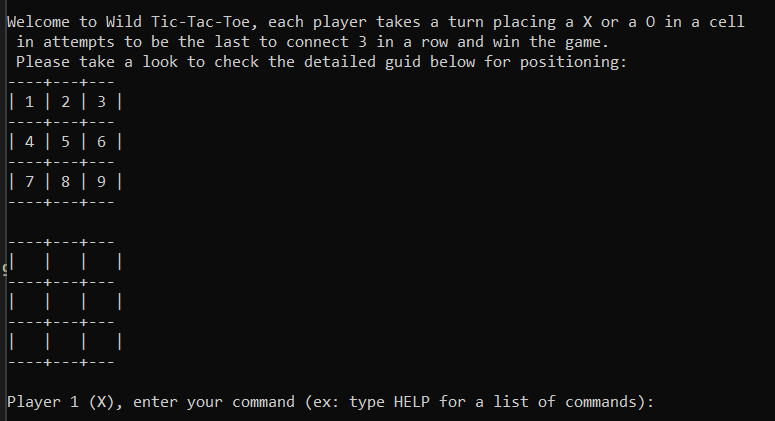
* 1. If user enters invalid option, user is required to enter again



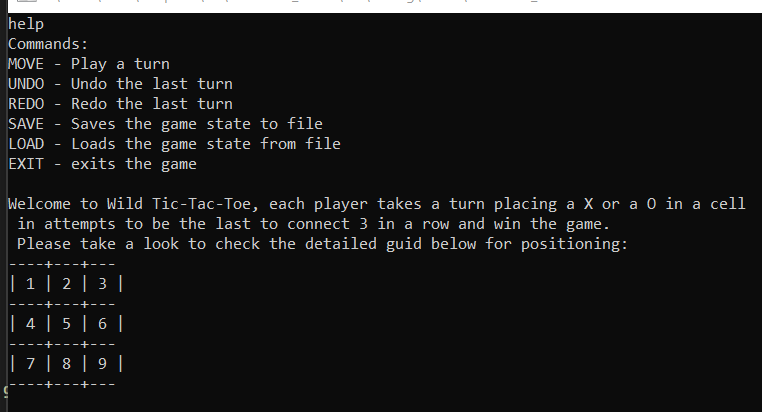
* 1. When the game starts, the screen will show a confirmation of play mode that the user chooses between Human vs Human or Human vs Computer.



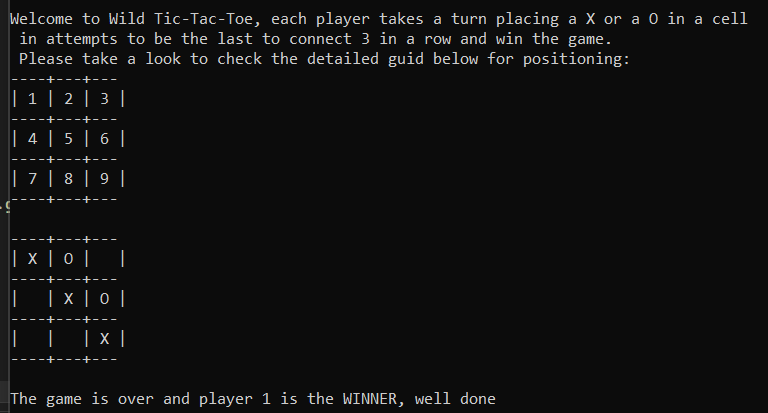
* 1. Detailed instructions and a position map are shown to assist the user to know their symbol and how to place the position



* 1. user can type ‘help’ to get a detailed explanation of all commands



* 1. When the player has one winning pattern satisfied, he will win. The game will inform that.



1. **Summary of classes/interfaces**

|  |  |  |
| --- | --- | --- |
| No | Libraries & Frameworks | Classes |
| 1 | .NET 6 framework | System.Console |
| 2 | .NET 6 framework | FileStream Class (System.IO) |
| 3 | .NET 6 framework | System.Linq |
| 4 | .NET 6 framework | Stack<T> Class (System.Collections.Generic) |