IFN563 - Object Oriented Design

Final Design Report

Part I. Statement of Completion

Part II. Overview of Final Design

Part III. Design Documents

Part IV. Design Pattern Identification

Part V. Program Execution Guide

Student Name: Li Chen Huang

Student ID: n10883789

I. Statement of Completion

Implemented requirements:

1. Connect Four game with a 7x6 board
2. Command line playing system
3. Human to Human play mode
4. Human to Computer play mode
5. Move validation check system. If move is not valid, system will prompt player to re-enter.
6. Save and Load game progress function that is playable after loading
7. Undo and redo functions during play
8. Online help system that can be called during playing

Unimplemented requirements:

None

II. Overview of Final Design

A. Final design:

This board game system is a system that can be expand to including different board games. In this current design, it includes Connect Four and Gomoku. It is consists of abstract classes that appears in most boardgames, while concrete classes for both the connect four and the Gomoku game inherit from the abstract classes.

The abstract classes that's serves as templates include player, game, piece, rules and round, which are universal elements in most board games. There is Concrete classes inheriting from thesis classes. The game class derives into Connect4Game and GomokuGame, the player class derives into Cnct4HumanPLayer, Cnct4ComPLayer, GomokuHumanPlayer and GomokuComPlayer. The player deviation design allows the system to support both computer and human player. Similar class inheriting method can be found in class Round and class Rules. Whenever a new type of game is to introduce into this design, classes can be simply added to the templates rather than create a total new class. In addition to these basic elements of board game, there is an addition class History and an interface ICommand. The History class helps store data to support save and load file functions, and the undo/redo commands. The ICommand interface produces UndoCommand and RedoCommand to provide undo and redo execution.

B. Changes made from preliminary design:

There are several changes of the preliminary design of this system which are listed below:

1. Classes attributes and method is added into class diagram. This is necessary for showing complete relationships between classes and their functioning roles in this system.

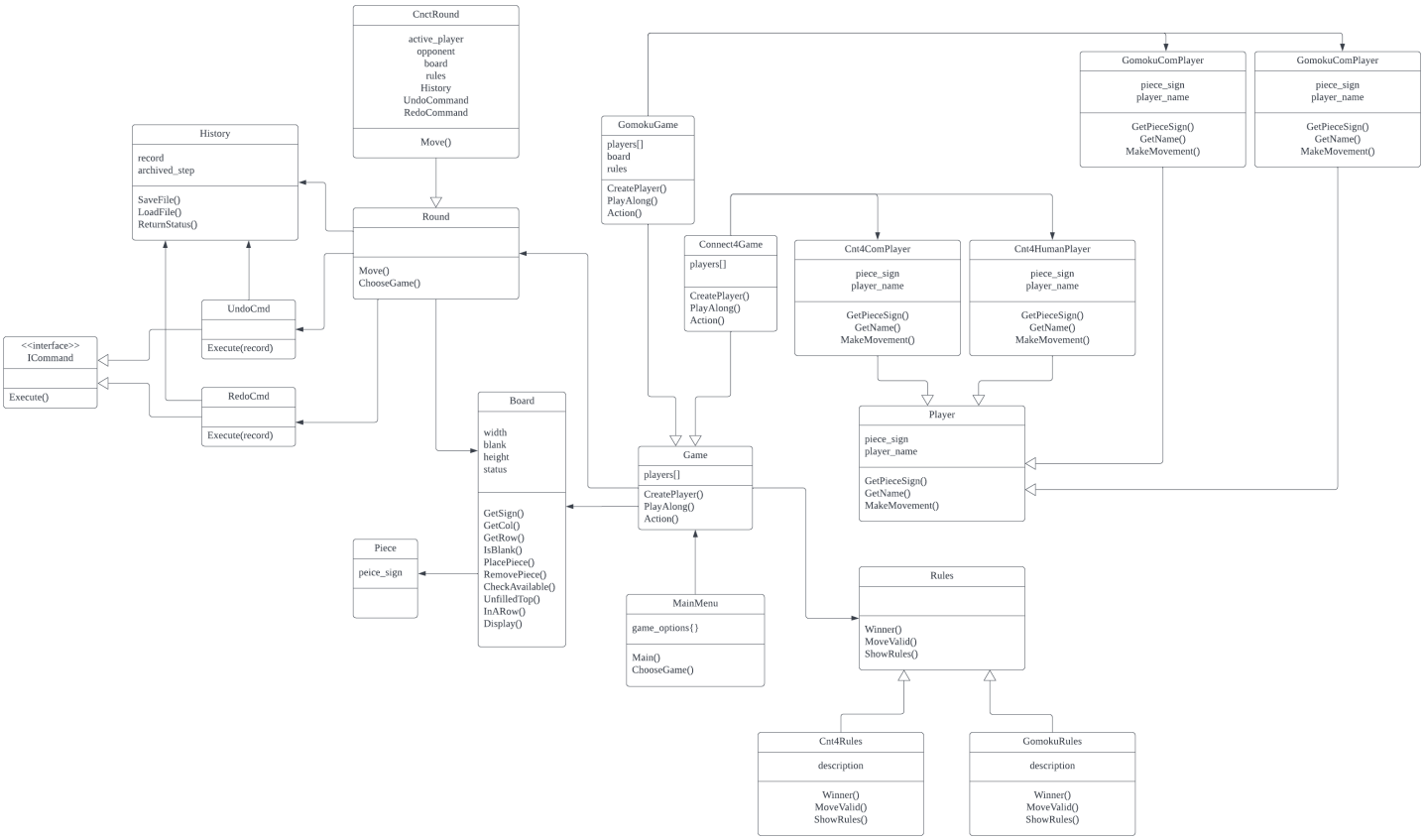
2. A Command interface and concrete commands of undo/redo is added to support undo and redo functions. This change is updated to meet the requirement of the implementation.

3. Class Round and its inherited classes are added into the scope. This round class breaks down the whole gaming logic and made the game more executable. Rather than putting the whole playing process in a single game class, playing process us now separated into different rounds.

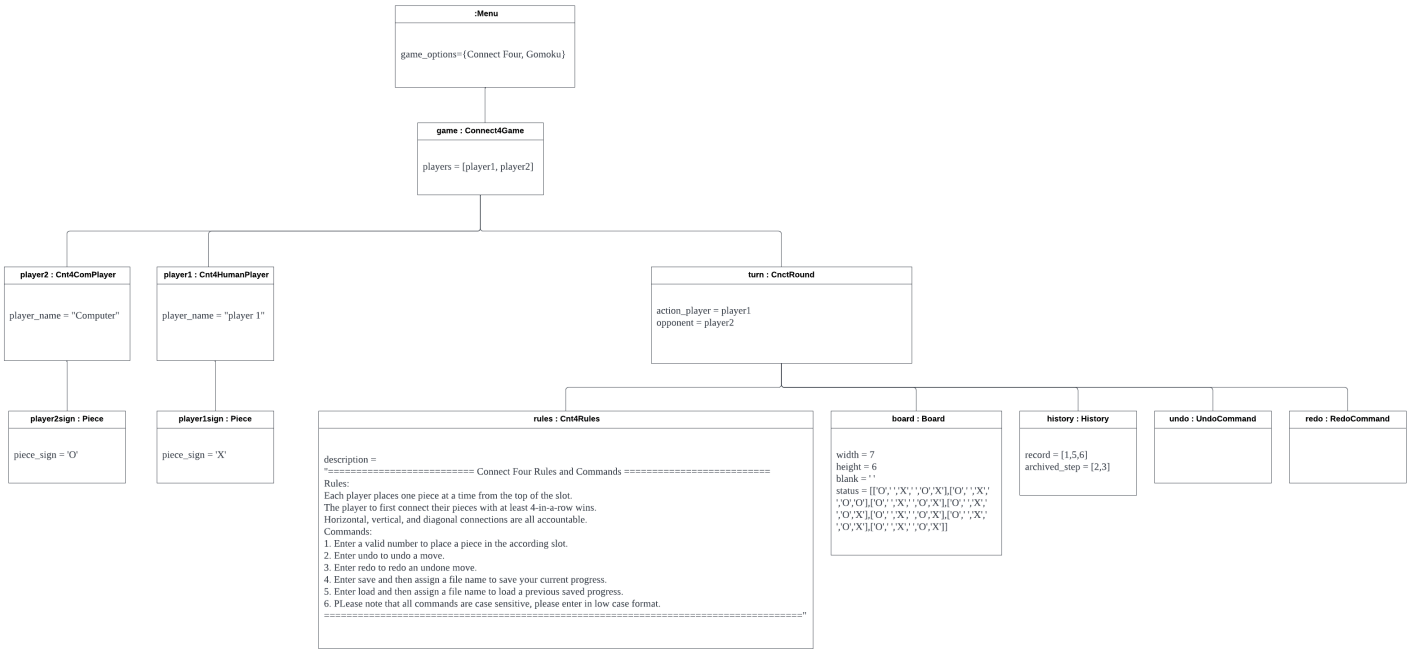
4. History class is created to support loading and saving file, also provide data for the undo and redo command to function.

III. Design documents:

A. Class Diagram:

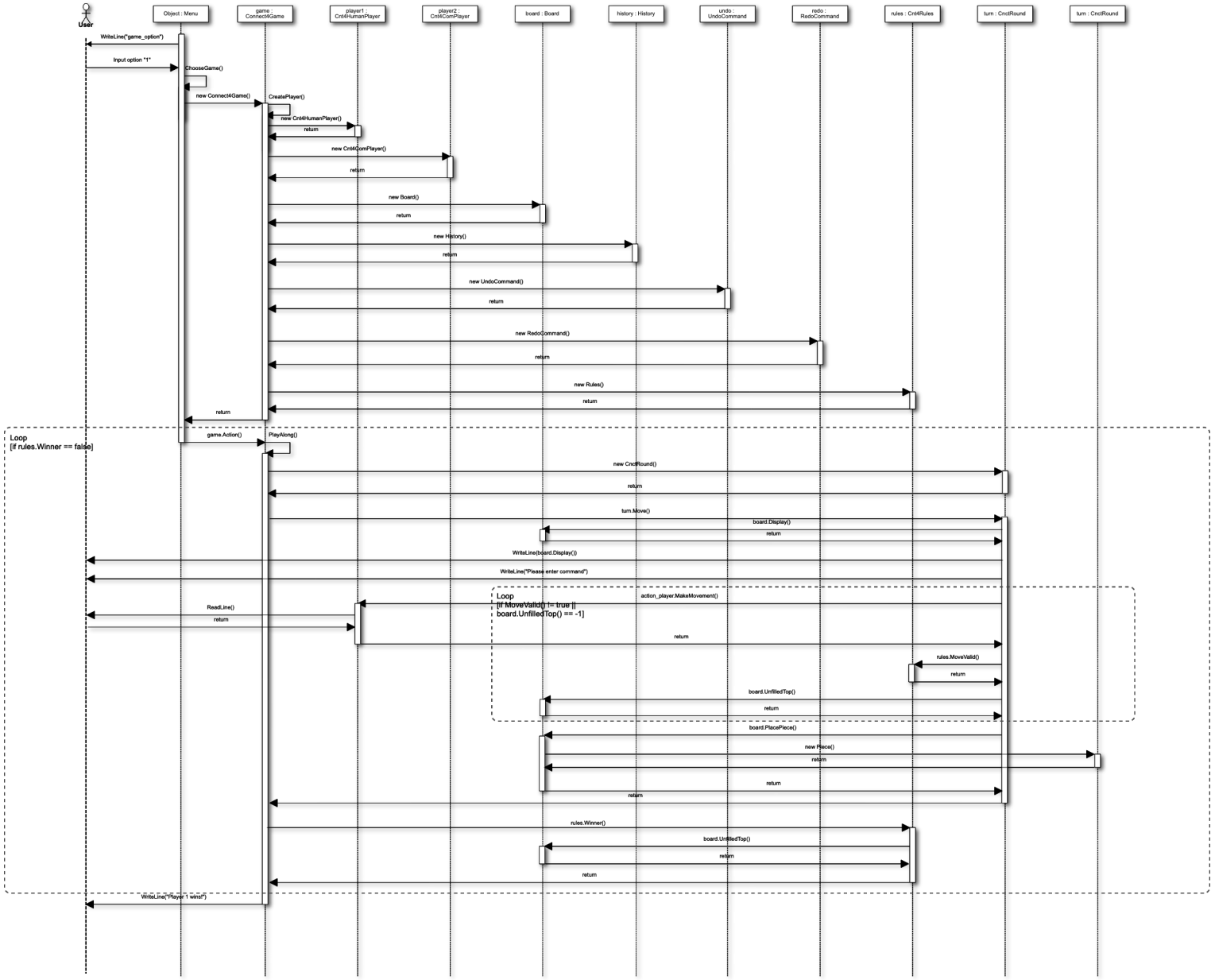


B. Object Diagram:



The object diagram above demonstrates a gaming timeframe which is played by human versus computer.

C. Sequence Diagram:



This sequence diagram demonstrates the progress from the very beginning (when a player enters the system) to declaring the winner of the game.

IV. Design Pattern Identification:

This system uses three design patterns, which are **template method, command pattern and factory pattern**. Below is the description of each patterns in this system.

1. Template Method:
   The template method refers to the method of a concrete class inheriting from an abstract class. In the case of this system, template method are used in Player, Game and Rules. They are declared as abstract classes to let concrete class such as Cnt4HumanPlayer, Cnt4ComPlayer, Connect4Game, Cont4Rules to inherit. These concrete class will add attributes and method context. Template method allows this system to be flexible when adding new game in.

2. Command Pattern:
   Command pattern in this game is implemented in the undo/redo class. First of all there is a ICommand class, which is an interface that contains only an Execute() method. This class is inherited by two concrete command classes- UndoCommand and RedoComand, which has there concrete method of execution. So when the player would like to execute these commands, the system has these commands as object and could call them.

3. Factory Pattern:
   The factory pattern in this system is implemented in the Game class with the CreatePlayer method. This method take in arguments and creates new Player class referencing different classes. With the factory method, we do not need to manually create player each time we called the new Player.

V. Program Execution Guide:

The program is executed automatically when the files are compiled. It starts with the menu prompting the user for choosing a game type. The options are connect four and Gomoku. If the user enters “1” the program will call a new Connect Four game. If the user enters “2”, it will return a message that tells the user that Gomoku is currently unavailable.

After entering the game, the user have to choose between human or computer for both players. The program will create new players according to the user’s command. After that, a board will be displayed and the user can start placing pieces by entering according slot numbers. The user can undo/redo, save/load file or call the inline help system during this stage. The system will declare the winner and end the program once there is a board status that fulfills the winning condition.

VI. Summary of classes/interface from Existing Libraries

1. Console.WriteLine():
   - CnctRules
   - Connect4Game
   - ContRound
   - History
   - Menu
   - RedoCommand
   - UndoCommand

2. Convert():
   - Cnt4ComPlayer
   - Connect4Game
   - ContRound
   - History
   - RedoCommand
   - UndoCommand

3. System.IO.FileStream():
   - History

4. System.IO.File():
   - History

5. TryParse():
   - ContRound