Custom Program Design/Usage Document

6.6HD – HD Level Custom Program

Name: Nguyen Duc Manh

ID: 105547489 / SWH02701

I. Program Overview

- Game title: Cờ Tư Lệnh (Commander Chess)

- Game overview: Commander Chess is a chess-inspired board game made by a Vietnamese military officer named Đoàn Ngọc Hải. This game is turn-based, desgined to simulate a modern war that contains strategic military elements between two opposing players.

- Why I chose this topic:

+ It demonstrates best the principles of Object-Oriented Programming since the game itself is the interaction between "objects" (game pieces, timer, board, sound etc.).

+ The game requires me to use .NET components like WPF and User Control.

II. Key Features

1. Unique pieces with special movement logic:

Unlike the traditional chess game, the purpose of this game is to stimulate a modern war, so the logic and the rules are much deeper and complicated. There are 11 different types of game pieces in Commander Chess: Commander, Head Quarters, Air Force, Militia, Navy, AAG, AAM, Tank, Artillery, Infantry and Engineer.

For example, the HQs can not move because they are "building", their role is the hiding place for the Commander to hide in and the enemy must destroy the HQ outside first to capture the Commander later.

Another example is the Navy, they carry AAG and AAM, therefore their movements logic is that they move can move on the ocean (just like any ships) but they can shoot inside the mainland and capture enemies.

2. Board

The board game is 12x11. It has 3 different components: land, ocean and the "bridge" or "shallow rivers". It creates a real reality scenario is that for the heavy "pieces" like artillery or missile, it can't pass the river and require Engineer to carry them.

Shallow rivers or bridges can be used to pass to the other side by lighter pieces (like Tank) and this is the border that stops the Navy from going.

3. Turn-based gameplay

Inside the logic is the "game manager" that decide which turn it is and prevent the player of the other side to move the pieces when it's not their turns.

4. Time control

The UI has game clocked implemented to track the time players have used and therefore adding the "time-limit" loosing condition.

5. Graveyard and Points counting

Each captured pieces will be sent to another board for players easier to see and also since each piece types has different points so there is also a point counting system.

6. Victory conditions

For now, I've only finished "Lose Commander" and "Out of Time" conditions

- A side loses when their commander is captured.
- Their clock ran out of time.

III. How to Play

1. Start screen: players choose to start the game or exit, or else they can choose if they want to keep the game sound during the game.

2. Main game screen: the "RED" side is always the first turn, after their turn, the opponent (BLUE) can play. Players can undo their moves by clicking the "R" key.

3. Game over: when the game ends, an end-game menu is turned on and players can choose between restarting the game or exit. On the menu, the winner side will be shown up and the end-game reason will also be called out. For example, "Red Wins, Blue ran out of time!"

IV. System Architecture

Here is how my program is structured:

***UI classes:** These classes are responsible for the UI of the WPF components

- MainWindow (.xaml and .cs): the main game window

- StartMenu.xaml: User Control

- GameOver.xaml: User Control

- PieceImages.cs: to render out the images

***Logic classes:** Responsible for the logic behind the scenes and how objects interact with each other

- GameManager.cs
- GamePieces.cs
- GameClock.cs
- GameResult.cs
- Board.cs
- Direction.cs
- Move.cs
- Player.cs
- Position.cs
- SoundControl.cs

***Design Patterns (HD requirements)**

1. MVC (Model-View-Controller): Here I seperated the logic into 3 types of main functions:

- Model: GameManager, Board, Move and GamePieces are the main classes that store data and the game logic.

- View: BoardView, LogView and GraveyardView to render out the board, hiding the logic to the UI.

- Controller: MainWindow – where UI and logic interact.

2. Observer: The MainWindow subscribes to events (Selected, PieceClicked) and then return to the UI to show up.

3. Command Pattern: all the “Move” will be stored and then “undo” when user clicked “R”.

4. Strategy: Inside the Move folders are the subsclasses like NormalMove, HideHQ, UnhideHQ => encapsulated different move types.

5. Singleton: Single point of control for audio, reused across game states.

6. Factory Pattern: render out all the different pieces using only one class (Board.cs): Board.Initialize()

V. Known Limitations

- I haven’t finished all the pieces’ movements and their logic: “Bombing” mode on AF pieces where they can attack and come back their place.

- The UI hasn’t been polished to the version like I want.

- I want to add more features: players can change the time setting during the match or they can change their names and profile pictures,

- Save the game to later open it.