# Design Overview for **Commander Chess – Cờ Tư Lệnh**

Name: **Nguyễn Đức Mạnh**

Student ID: 105547489 – SWH02701

# Summary of Program

My program is a chess-like game called “Cờ Tư Lệnh” or Commander Chess. I choose this game because it implements well the concepts and ideas of Object-Oriented Programming.

Another aspect is that this is a chess game made by Vietnamese – writer Nguyen Quy Hai. This game simulates mordern war and how our past soldiers fought at the warzone. Therefore, the game’s rules are much more complicated than the original Chess so it will be more challenging.

I expect my game to be played by 2 players (no AI or Bot yet) and take turns.

# Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

### GamePieces <<Abstract>>

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Assign name for piece | PieceName Name {get;} <<abstract>> | Assign name for each piece in board’s initialization |
| Assign color for piece | Player Side {get;} <<abstract>> | Assign which side does the piece belong to |
| Assign piece’s point | Int Point {get;} <<abstract>> | The lose/win rule does need points to be tracked |
| Check if the piece has moved yet | Bool HasMovedYet {get; set;} <<abstract>> | Some move required piece to not move before |
| Get all possible move in that turn | IEnumerable<Move> GetMoves (Position from, Board board, Direction direction) <<abstract>> | Use this to check the possible or illegal move later on |
| Check the possible moves | IEnumerable<Position> MoveInDirectionsLimited(Position from, Board board, int maxSteps, params Direction[] directions) | The maximum steps do affect the possible move. Just move here, NOT CAPTURE |

PieceName -> GamePieces

|  |  |
| --- | --- |
| Value | Notes |
| Commander | Tư lệnh – 100 point |
| Infantry | Bộ binh – 10 point |
| Tank | Xe tăng – 20 point |
| Militia | Dân quân – 10 point |
| Engineer | Công binh – 10 point |
| Artillery | Pháo binh – 30 point |
| AAG | Cao xạ - 10 point |
| AAM | Tên lửa phòng không – 20 point |
| AF | Máy bay – 40 point |
| Navy | Hải quân – 80 point |
| HQ | Sở chỉ huy – 10 point |

### Player

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Determine the Opponent | Player Opponent(this Player player) | If player is Red => return Blue and so on |

Player Enum

|  |  |
| --- | --- |
| Value | Notes |
| None |  |
| Blue |  |
| Red |  |

### Board

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Initialize the board to WPF | Board Initialize(): static | If player is Red => return Blue and so on |
| Put the piece to board | GamePieces this[int row, int column] | Use to initialize pieces |
| Put the piece using Position | GamePieces this[Position pos] | Set using Position objects |
| Check if piece is inside the board | Bool InsideBoard(Position pos): static |  |
| Check if that position is empty | Bool EmptyPosition(Position pos) |  |
| Set piece’s coordination | Void InitializePieces() |  |
| Create 2D array to place pieces | readonly GamePieces[,] pieces = new GamePieces[12, 11] |  |

### Position

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Check if that position is in “Ocean” | Bool OceanPosition(int row, int column) | Some pieces’ moves don’t allow to be on Ocean |
| Compare values and not instances | Bool Equals(object obj) => obj: bool |  |
| using Position in hash collections | Int GetHashCode(): bool |  |
| Get Row value | Int Row {get;} |  |
| Get Column value | Int Column {get;} |  |
| Return if 2 pieces have the same position | Bool operator ==(Position left, Position right) |  |
| Ensure consistency with the above method | Bool operator !=(Position left, Position right) |  |
| Use in Direction calculation | Position operator +(Position p, Direction d) |  |

### GameTurn

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Get board after initialized | Board board {get;} |  |
| Control the turn of the game | Player Turn {get; private set;} | Other classes could read and get the value but only the GameTurn class can set this property |

### Direction

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Piece moves forward | Direction Up = new Direction(1, 0) |  |
| Piece moves backward | Direction Down = new Direction(-1, 0) |  |
| Piece moves left | Direction Left = new Direction(0, -1) |  |
| Piece moves right | Direction Right = new Direction(0, 1) |  |
| Piece moves northwest | Direction UpLeft = new Direction(1, -1) |  |
| Piece moves northeast | Direction UpRight = new Direction(-1, 1) |  |
| Piece moves southwest | Direction DownLeft = new Direction(-1, -1) |  |
| Piece moves southeast | Direction DownRight = new Direction(1, 1) |  |
| Detect row’s value changed | Int RowChange {get;} |  |
| Detect column’s value changed | Int ColumnChange {get;} |  |
| Move 1 position at a time | Direction operator +(Direction d1, Direction d2) |  |
| Move more than 2 positions | Direction operator \*(int k, Direction d) | For sliding pieces (like Tank) |

# Class Diagram

A diagram of a data flow

AI-generated content may be incorrect.

# Sequence Diagram

Provide a sequence diagram showing how your proposed classes will interact to achieve a specific piece of functionality in your program.

# **\*HD\*** Design Patterns

Here are 4 initial design patterns that I will use in my program:

1. Command Pattern: Encapsulate each move into an object.

2. Memento Pattern: Save and restore the board => allows players to redo their moves if needed.

3. Observer Pattern: Notify when game state changes.

4. Factory Method Pattern: To create pieces by grouping them with their side (blue or red) instead of hardcoding.

# **\*HD\*** Additional/complex features

- First of all I think the rules and how the pieces move in the Commander Chess are already challenging because of its complexity and strategies. So to implement all of them and make them works together is quite hard.

- Secondly, I use mentioned design patterns to implement important features of a chess-like game (redo moves,…) to make my program run smoothly.