

# COS20007 - Object Oriented Programming

## HD Level Custom Program

Student name: Nguyen Duc Manh  
ID: 105547489



```
1 namespace CommanderLogic;
2
3 public enum PieceName
4 {
5     Commander,
6     Infantry,
7     Tank,
8     Militia,
9     Engineer,
10    Artillery,
11    AAG,
12    AAM,
13    AF,
14    Navy,
15    HQ
16 }
17
18 public abstract class GamePieces
19 {
20     public abstract PieceName Name { get; }
21
22     public abstract Player Side { get; }
23
24     public abstract int Point { get; }
25
26     public bool HasMovedYet { get; set; } = false; // Track if the piece has moved
27
28     public abstract GamePieces CopyPiece();
29
30     public abstract IEnumerable<Move> GetMoves(Position from, Board board);
31
32     protected IEnumerable<Position> MoveInDirection(Position from, Board board, Direction direction)
33     {
34         for (Position position = from + direction; Board.InsideBoard(position); position += direction)
35         {
36             if (board.EmptyPosition(position))
37             {
38                 yield return position;
39                 continue;
40             }
41
42             //GamePieces piece = board[position];
43
44             //if (piece.Side != Side)
45             //{
46                 //yield return position; // Can capture the piece
```

```
47         // yield break; // Stop if we hit a piece
48         //}
49         ////yield break; // Stop if we hit a piece
50     }
51 }
52
53 protected IEnumerable<Position> MoveInDirections(Position from, Board board, params Direction[] directions)
54 {
55     return directions.SelectMany(direction => MoveInDirection(from, board, direction));
56 }
57
58 protected IEnumerable<Position> MoveInDirectionsLimited(Position from, Board board, int maxSteps, params Direction[] directions)
59 {
60     foreach (var direction in directions)
61     {
62         Position position = from;
63         for (int i = 0; i < maxSteps; i++)
64         {
65             position += direction;
66             if (!Board.InsideBoard(position)) break;
67
68             if (board.EmptyPosition(position))
69             {
70                 yield return position;
71             }
72             break;
73             //else
74             //{
75                 // GamePieces piece = board[position];
76                 // //if (piece.Side != Side)
77                 // // yield return position; // can capture
78                 // break; // stop after hitting any piece
79             //}
80         }
81     }
82 }
83 }
```

```
1 using CommanderLogic.Pieces;
2
3 namespace CommanderLogic
4 {
5     public class Board
6     {
7         private readonly GamePieces[,] pieces = new GamePieces[12, 11]; //2D Array, Col and Row + 1 because of zero indexed
8
9
10        public static Board Initialize() //initial the board
11        {
12            Board board = new Board();
13            board.InitializePieces();
14            return board;
15        }
16
17        public GamePieces this[int row, int column] //set a piece with both row and col
18        {
19            get { return pieces[row, column]; }
20            set { pieces[row, column] = value; }
21        }
22
23        public GamePieces this[Position pos] //set a piece's with Position object
24        {
25            get { return pieces[pos.Row, pos.Column]; }
26            set { pieces[pos.Row, pos.Column] = value; }
27        }
28
29        public static bool InsideBoard(Position pos) //prevent pieces outside the board
30        {
31            return pos.Row >= 0 && pos.Row < 12 && pos.Column >= 0 && pos.Column < 11;
32        }
33
34        public bool EmptyPosition(Position pos) //that position is empty or not
35        {
36            return this[pos] == null;
37        }
38
39        private void InitializePieces()
40        {
41            // The coordinate also zero indexed
42            //11,6 means 12th row and 7th column
43            //(doesn't matter the coordinate system, just count by hand)
```

```
44
45         // Blue
46         this[11, 6] = new Commander(Player.Blue);
47         this[7, 2] = new Infantry(Player.Blue);
48         this[7, 10] = new Infantry(Player.Blue);
49         this[8, 5] = new Tank(Player.Blue);
50         this[8, 7] = new Tank(Player.Blue);
51         this[7, 6] = new Militia(Player.Blue);
52         this[7, 3] = new Engineer(Player.Blue);
53         this[7, 9] = new Engineer(Player.Blue);
54         this[9, 3] = new Artillery(Player.Blue);
55         this[9, 9] = new Artillery(Player.Blue);
56         this[8, 4] = new AAG(Player.Blue);
57         this[8, 8] = new AAG(Player.Blue);
58         this[9, 6] = new AAM(Player.Blue);
59         this[10, 4] = new AF(Player.Blue);
60         this[10, 8] = new AF(Player.Blue);
61         this[10, 1] = new Navy(Player.Blue);
62         this[8, 2] = new Navy(Player.Blue);
63         this[10, 5] = new HQ(Player.Blue);
64         this[10, 7] = new HQ(Player.Blue);
65
66         // Red
67         this[0, 6] = new Commander(Player.Red);
68         this[4, 2] = new Infantry(Player.Red);
69         this[4, 10] = new Infantry(Player.Red);
70         this[3, 5] = new Tank(Player.Red);
71         this[3, 7] = new Tank(Player.Red);
72         this[4, 6] = new Militia(Player.Red);
73         this[4, 3] = new Engineer(Player.Red);
74         this[4, 9] = new Engineer(Player.Red);
75         this[2, 3] = new Artillery(Player.Red);
76         this[2, 9] = new Artillery(Player.Red);
77         this[3, 4] = new AAG(Player.Red);
78         this[3, 8] = new AAG(Player.Red);
79         this[2, 6] = new AAM(Player.Red);
80         this[1, 4] = new AF(Player.Red);
81         this[1, 8] = new AF(Player.Red);
82         this[1, 1] = new Navy(Player.Red);
83         this[3, 2] = new Navy(Player.Red);
84         this[1, 5] = new HQ(Player.Red);
85         this[1, 7] = new HQ(Player.Red);
86     }
87 }
88 }
89
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace CommanderLogic
8 {
9     public class Direction
10    {
11        //preset directions
12        //readonly so that it cannot be reassigned after initialization
13        public readonly static Direction Up = new Direction(1, 0);
14        public readonly static Direction Down = new Direction(-1, 0);
15        public readonly static Direction Left = new Direction(0, -1);
16        public readonly static Direction Right = new Direction(0, 1);
17        public readonly static Direction UpLeft = new Direction(1, -1);
18        public readonly static Direction UpRight = new Direction(-1, 1);
19        public readonly static Direction DownLeft = new Direction(-1, -1);
20        public readonly static Direction DownRight = new Direction(1, 1);
21
22
23        public Direction(int rowChange, int columnChange)
24        {
25            RowChange = rowChange;
26            ColumnChange = columnChange;
27        }
28
29        public int RowChange { get; }
30
31        public int ColumnChange { get; }
32
33        public static Direction operator +(Direction d1, Direction d2)
34        {
35            return new Direction(d1.RowChange + d2.RowChange,
36                                d1.ColumnChange + d2.ColumnChange);
37        }
38
39        public static Direction operator *(int k, Direction d)
40        {
41            return new Direction(k * d.RowChange, k * d.ColumnChange);
42        }
43    }
44
```

```
1 namespace CommanderLogic
2 {
3     public class GameTurn
4     {
5         public GameTurn(Player player, Board board)
6         {
7             Turn = player;
8             Board = board;
9         }
10
11         public Board Board { get; }
12
13         public Player Turn { get; private set; } //GameLogic can set,
14         other can just read
15
16         public IEnumerable<Move> LegalMove(Position position)
17         {
18             if (Board.EmptyPosition(position) || Board[position].Side !=
19                 Turn)
20             {
21                 return Enumerable.Empty<Move>(); // No legal moves if the
22                 position is empty or not the player's turn
23             }
24
25             GamePieces piece = Board[position];
26             return piece.GetMoves(position, Board);
27         }
28
29         public void MakeMove(Move move)
30         {
31             move.MoveExecute(Board);
32             Turn = Turn.Opponent(); // Switch turn to the opponent after a
33             move
34         }
35     }
36 }
```

```
1 namespace CommanderLogic
2 {
3     public abstract class Move
4     {
5         public abstract MoveType Type { get; }
6
7         public abstract Position From { get; }
8
9         public abstract Position To { get; }
10
11         public abstract void MoveExecute(Board board);
12     }
13     public enum MoveType
14     {
15         Normal,
16         HideHQ
17         //other special moves later on
18     }
19 }
20
```



```
1 namespace CommanderLogic
2 {
3     public enum Player
4     {
5         None,
6         Red,
7         Blue
8     }
9
10    public static class PlayerExtensions
11    {
12        public static Player Opponent(this Player player) //return player' s opponent ↗
13        {
14            return player switch
15            {
16                Player.Red => Player.Blue,
17                Player.Blue => Player.Red,
18                _ => Player.None
19            };
20        }
21    }
22 }
23
```

```
1 namespace CommanderLogic
2 {
3     public class Position
4     {
5         public Position(int row, int column)
6         {
7             Row = row;
8             Column = column;
9         }
10
11         public bool OceanPosition(int row, int column)
12         {
13             int[] oceanColumn = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
14
15             if (((row == 0 || row == 1) && oceanColumn.Contains(column)))
16             {
17                 return true;
18             }
19             else
20                 return false;
21         }
22
23         public override bool Equals(object obj) => obj is Position && Row == position.Row && Column == position.Column;
24
25         public override int GetHashCode() => GetHashCode.Combine(Row, Column);
26
27         public int Row { get; }
28
29         public int Column { get; }
30
31         public static bool operator ==(Position left, Position right)
32         {
33             return EqualityComparer<Position>.Default.Equals(left, right);
34         }
35
36         public static bool operator !=(Position left, Position right)
37         {
38             return !(left == right);
39         }
40
41         public static Position operator +(Position p, Direction d)
42         {
43             return new Position(p.Row + d.RowChange, p.Column +
44                                 d.ColumnChange);
45         }
46         // End
47     }
```

48 }

49

