Project DEV4 Stratego - Modelization

Authors

Khayre Kadir 54865

Mora Leonardo 54740

Introduction

Stratego is a strategy board game for two players on a board of 10×10 squares. Each player controls 40 pieces representing individual officer ranks in an army. The objective of the game is to find and capture the opponent's Flag, or to capture so many enemy pieces that the opponent cannot make any further moves.

How does Stratego work?

The board game is composed by 100 squares, 8 squares of water clustering into groups of 2x4 squares. The first 4 rows and 4 last rows (depends on player 1 and 2) will be used for placement of pawns, bombs and flags. During each turn, a pawn could move to the following directions : right, left, top or bottom, where he can eat an ennemy's spawn in some cases:

- If the attacking pawn has a better rank, he could attack the targered pawn, this other pawn dies.
- If the pawn attacks a pawn with a better rank, he dies.
- If the pawn attacks a flag, this game is over.
- If the pawn is a miner and attacks a bomb, this bomb will be destroyed, otherwise he dies and the bomb stays at his place.

Here are the different pawns of the game:

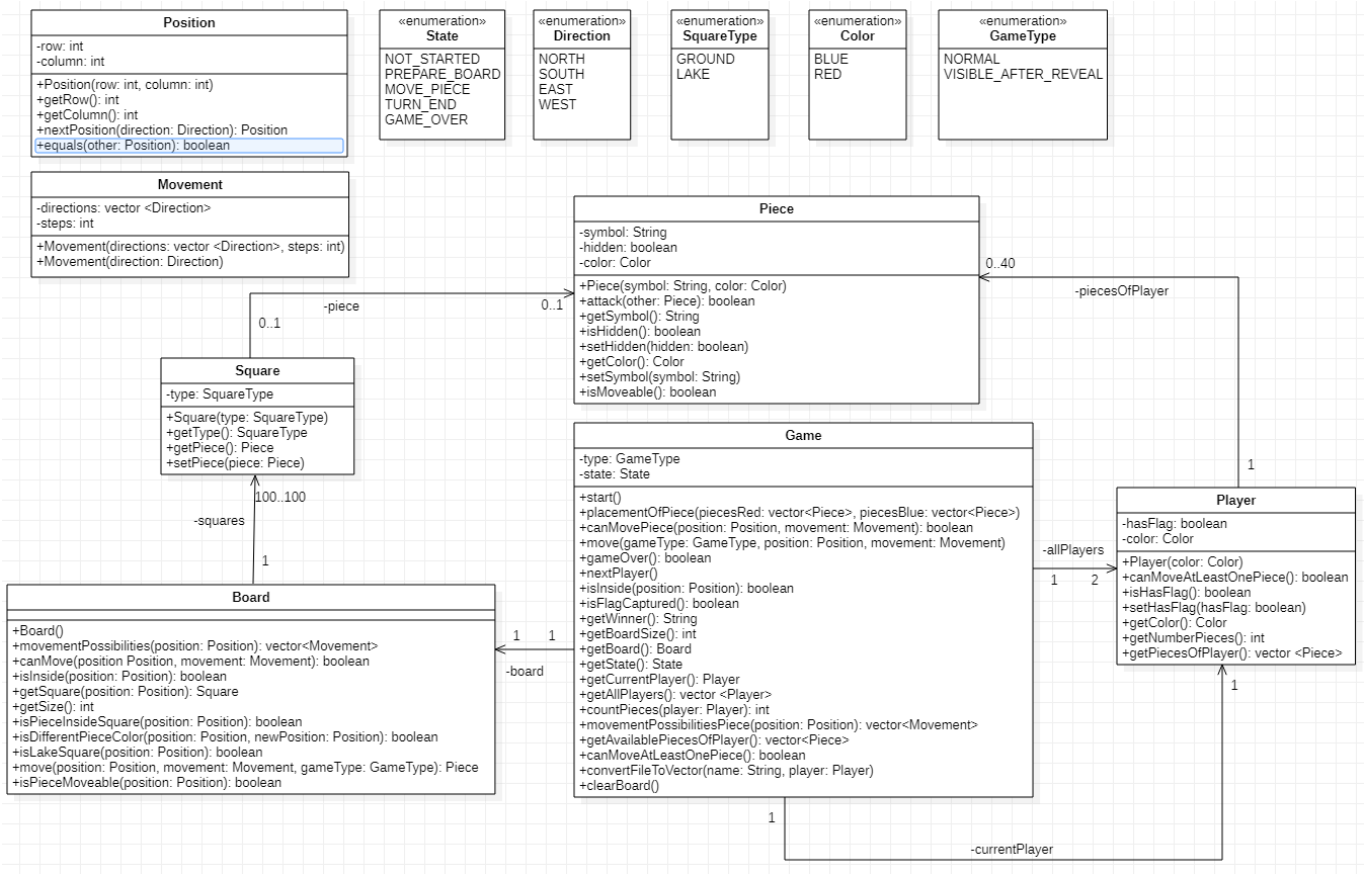
```
Piece       Symbol      Quantity

Marshal       10            1
General       9             1
Colonel       8             2
Major         7             3
Captain       6             4
Lieutenant    5             4
Sergeant      4             4
Miner         3             5
Scout         2             8
Spy           1             1
Flag          D             1
Bomb          B             6
```

Moreover the game has 2 gamemodes:

- The first is normal.

- The second one leaves visibility after verification.

Our Analysis



Explanation of classes and methods

Class Piece: the piece has a value between 1 and 10 or either represented by B or F

- *Attribute:*
    - String symbol : rank of a piece represented by a number or a letter
    - Color color : piece's color
    - boolean hidden : piece's visibility
- *Method:*
    - Piece (String symbol, Color color) : constructor of Piece
    - vector < Direction> movementPossibilities(Movement movement): shows every possibility for moving this piece.
    - boolean canMove(Movement: movement) : checks if the piece can be moved
    - boolean attack(Piece piece) : checks the winner of the fight by checking ranks
    - String getSymbol() : getter of symbol
    - boolean isHidden() : getter of hidden
    - void setHidden(boolean hidden) : setter of hidden
    - Color getColor() : getter of color
    - void setSymbol(String symbol) : setter of symbol
    - boolean isMoveable() : checks if the piece is not "D" or "B"

Class Square: the square has a type GROUND or LAKE and can own a piece

- *Attribute:*
  - SquareType type : type of the square
  - Piece piece : piece on square
- *Method:*
  - Square(SquareType type) : constructor of Square
  - SquareType getType() : getter of type
  - Piece getPiece() : getter of piece
  - void setPiece(Piece piece) : setter of piece

Class Position: the position is characterized by a row and a column

- *Attribute:*
  - int row : row of the board
  - int column : column of the board
- *Method:*
  - Position(int row, int column) : constructor of Position
  - int getRow() : getter of row
  - int getColumn() : getter of column
  - Position nextPosition(Direction direction): give the next position with the direction
  - boolean equals(Position other): checks if the position equals to the other position

Class Movement: the movement is characterized by a direction and a number of steps

- *Attribute:*
  - vector < Direction > directions : list of different directions
  - int steps : number of steps for the movement
- *Method:*
  - Movement(Direction direction, int steps) : first constructor of Movement
  - Movement(Direction direction) : second constructor of Movement

```
--> Why 2 constructors? The first constructor is used for the scout's movement and
the second constructor is used for others pieces namely a simple movement of one
step.
```

Class Board: the board in which piece is placed on the square

- *Attribute:*
  - Square[][] squares : a two-dimensional array of squares
- *Method:*

- Board() : constructor of Board
- boolean isInside(Position position) : checks if the given position is inside the board
- Square getSquare(Position position) : determines the square with the given position
- int getSize() : determines the size of the board
- vector movementPossibilities(Position position): search each movement possibilities for a piece and put it on a vector of movement
- boolean isPieceInsideSquare(Position position): Check if the square at the Position contain a Piece
- boolean isDifferentPieceColor(Position position, Position newPosition): Check if the color of piece at position is different to the color of piece of newPosition
- boolean isLakeSquare(Position position): search each movement possibilities for a piece and put it on a vector of movement
- Piece move(Position position, Movement movement, GameType gameType): move the piece at the position to a new position with movement
- boolean isPieceMoveable(Position position) : check if the piece is not a "D" or a "B" on the position

Class Player: the player is represented by a color which is associated with pieces of the same color

- *Attribute:*
  - vector < Piece > piecesOfPlayer : pieces of the player
  - boolean hasFlag : the captured flag of the opposing player
  - Color color : color of the player
- *Method:*
  - Player (Color color) : constructor of Player
  - boolean canMoveAtLeastOnePiece() : Checks if at least one piece can be moved
  - boolean isHasFlag() : getter of hasFlag
  - void setHasFlag(boolean hasFlag) : setter of hasFlag
  - Color getColor(): getter of color
  - int getNumberPieces(): get the number of pieces of m_piecesOfPlayers
  - vector getPiecesOfPlayer(): getter of piecesOfPlayer

Class Game: gathers necessary elements for the game and implements different stages of the game

- *Attribute:*
  - Player currentPlayer : current player of the turn
  - vector < Player > allPlayers : list of players
  - GameType type : type of the game
  - State state : state of the game
- *Method:*
  - void start() : intialize a game

- void placementOfPiece(vector < Piece > piecesRed, vector < Piece > piecesBlue): setting up the pieces at the start of the game
- boolean canMovePiece(Piece piece, Movement movement) : checks if the piece can be moved
- void move (Piece piece, Movement movement) : Moves the piece in a given direction
- boolean gameOver() : checks if the game is over
- void nextPlayer() : changes the current player
- boolean isInside (Position position) : checks if the given position is inside the board
- boolean isFlagCaptured() : checks if the flag is captured
- String getWinner() : determines the winner of the game
- int getBoardSize() : determines the size of the board
- Board getBoard () : getter of board
- State getState () : getter of state
- Player getCurrentPlayer() : getter of currentPlayer
- vector < Player > getAllPlayers() : getter of allPlayers
- int countPieces(in player:Player): counts each piece the player actually has
- vector movementPossibilitiesPiece(in position:Position):
- vector getAvailablePiecesOfPlayer(): getter of getAvailablePiecesPlayer()
- boolean canMoveAtLeastOnePiece(): checks if at least one piece can be moved
- void convertFileToVector(String name, Player player): converts a .txt file to players pieces
- void clearBoard(): clear the board

Enumeration Direction:

Enum can move to a specific direction (the 4 cardinal points).

- *Litteral:*
  - NORTH
  - SOUTH
  - EAST
  - WEST

Enumeration State

Enum who determines the status of the game.

- *Litteral:*

  - NOT_STARTED -> the only possible action for the game (on the beginning), on the methode start().
  - PREPARE_BOARD -> setup of pieces placement.
  - MOVE_PIECE -> step for the mouvement of a piece.
  - TURN_END -> the end of the turn, the change of player
  - GAME_OVER -> checks if the game is gone, namely a player caught the flag or they can't move.

Enumeration SquareType

Enum who determines the SquareType, can be a ground or a lake

- *Litteral:*

    - GROUND
    - LAKE

Enumeration Color

Enum who joins for each player, a color

- *Litteral:*

    - RED
    - BLUE

Enumeration GameType

Enum who manage the game mode

- *Litteral:*

    - NORMAL
    - VISIBLE_AFTER_REVEAL