Dev4-Projet

Rapport1-remise métier

FOUEFACK ATCHEPONG Vivace (54490)

NGOUMENI NKEUDJEU Jeanny (55015)

Table des matières

Introduction

Stratego est un jeu de stratégie ou deux joueurs (un joueur avec les pièces rouges et un autre avec les pièces bleues) s'affrontent sur un plateau. Il nous est demandé de concevoir et d'implémenter ce jeu en c++. Pour réaliser ce projet, nous allons définit l'environnement de développement, les outils utilisés ainsi que la modélisation des classes métiers.

1.1 Environnement de développement et outils utilisés

Pour la réalisation de ce projet, nous allons implémenter l'architecture MVC (Model View Controller) et utiliser :

- L'IDE (Integrated Development Environment) Qt Creator.
- StarUML : est un logiciel de modélisation UML (unified modeling language) qui a permis de mettre sur pieds le diagramme de classe.

- GitLab for esi : outil qui nous faciliterait la collaboration à distance et nous permettrait de gérer nos différentes versions.

Modélisation UML

Cette section va traiter la modélisation UML des classes métiers qui vont constituer le cœur du projet. Le diagramme de classes est disponible à la fin de ce document.

2.1 Classe Board

La classe Board désigne le plateau de jeu. Cette classe est constituée de :

- un attribut « tab » qui représente un tableau à deux dimensions de type pièce et dont les cellules représentent les différentes possibilités de déplacement. Certaines cellules de ce tableau sont inaccessibles (lacs).

- Une méthode fillBoard (vector<Piece> playerOne, vector<Piece> playerTwo) ->void : initialise le tableau avec le contenu des vecteurs de pièces, respectivement le vecteur de pièce du joueur rouge et bleu reçu en paramètre.

2.2 Classe Player

La classe Player désigne un joueur. Elle est caractérisée par :

- Un attribut Pièce de type vecteur qui va contenir l'ensemble des pièces d'un joueur.

- Une méthode IsMyPiece (Position position) ->Boolean : cette méthode permet de vérifiée si une des pièces du joueur se trouve à une position reçue en paramètre.

- Une méthode RemovePiece (Position position) ->void : soustrait la pièce située à la position reçue en paramètre de la liste des pièces d'un joueur.

2.3 Classe Piece

Elle désigne une pièce sur le plateau de jeu. Cette pièce possède les méthodes et attributs suivants :

- Un attribut position qui représente la position d'une pièce sur le tableau.
- Un attribut symbole qui désigne le rang de la pièce.

- Un attribut dévoilé de type boolean qui permet de savoir si une pièce est dévoilée ou pas. il est mis à vrai lorsqu'un joueur attaque avec une pièce ou est attaqué par l'adversaire. Par défaut, cet attribut est à faux.

- Un attribut lastOccupations représentant un vecteur d'anciens positions qui permettrait de retenir les différentes positions d'une pièce ce qui contribuerait à la gestion de la contrainte de non-allers-retours (entre deux positions) consécutifs de plus de 3 fois sur une pièce.

- Une Méthode move (Position nextPosition) ->void : déplace une pièce de sa position actuelle vers la position reçue en paramètre.

- Une méthode roundTripCheck () ->boolean : vérifie si la pièce a déjà effectué trois allers-retours entre 2 positions, retourne vrai si c'est le cas et faux dans le cas contraire.

2.4 Classe Position

Permet de situer une pièce dans l'espace à deux dimensions du plateau de jeu, par conséquence elle est constituée de :

- Un attribut « x » qui représente l'axe x.
- Un attribut « y » qui représente l'axe y.

Cette classe serait particulièrement utile puisse qu'elle nous permettrait d'assurer la gestion des déplacements.

2.5 Classe Game

Il s'agit probablement de la classe métier la plus importante puisqu'elle permettrait une communication plus aisée entre le contrôleur et le modèle. On y trouvera :

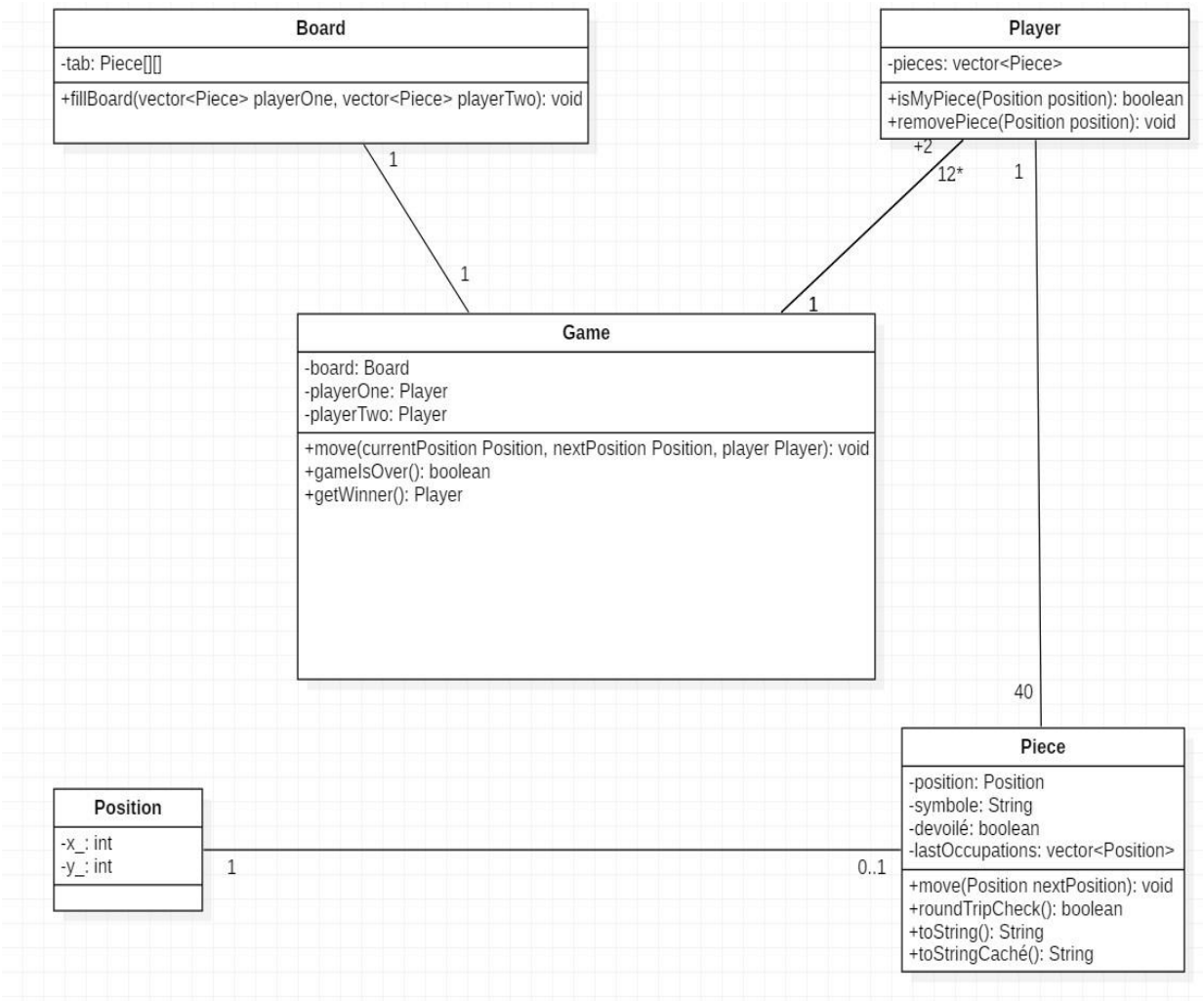
- Un attribut board représentant le plateau de jeu.
- Un attribut playerOne représentant le joueur rouge.
- Un attribut playerTwo représentant le joueur rouge.

- Une méthode move (Position currentPosition, Position nextPosition, Player player) ->void : déplace la pièce d'un joueur reçu en paramètre de sa position actuelle(currentPosition) vers la position suivante (nextPosition).

- Une méthode gameIsOver () ->boolean : vérifie l'état du jeu. si le jeu est dans un état terminé (le drapeau d'un joueur est capturé ou il n'est plus possible pour un joueur de déplacé un de ses pièces), la méthode retourne vrai ou faux dans le cas contraire.

- Une méthode getWinner () ->player : permet de déterminer qui est le vainqueur à la fin d'une partie.

Annexe


Board
-tab: Piece[][]
+fillBoard(vector<Piece> playerOne, vector<Piece> playerTwo): void
Player
-pieces: vector<Piece>
+isMyPiece(Position position): boolean
+removePiece(Position position): void
Game
-board: Board
-playerOne: Player
-playerTwo: Player
+move(currentPosition Position, nextPosition Position, player Player): void
+gameIsOver(): boolean
+getWinner(): Player
Position
-x_: int
-y_: int
Piece
-position: Position
-symbole: String
-devoilé: boolean
-lastOccupations: vector<Position>
+move(Position nextPosition): void
+roundTripCheck(): boolean
+toString(): String
+toStringCaché(): String
1
1
+2
12*
1
1
40
1
0..1