Stratego

Sami Slim, 56081
Gabriel Espinosa, 54720

Table des matières

1.Introduction

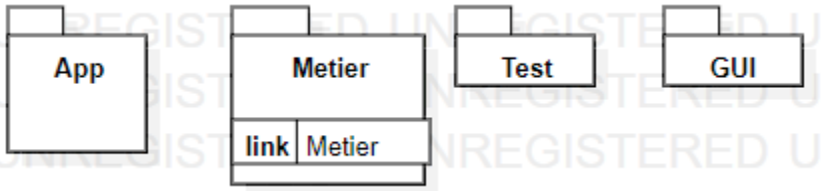
Ce projet organisé par les élèves Sami Slim et Gabriel Espinosa du groupe D111 pour la création du Stratego. Il a été encadrer avec les spécification abordé dans le document d'énoncé du projet[1] ainsi que les modalités[2] exigé par le professeur Marcelo Burda .

Le projet a été structuré et découpé en sous-système. Console : La version console du projet. Métier : Le model du projet, énumérant chaque élément visible et invisible du jeu. Test : Permet d'effectuer des tests dans le projet. GUI : Est la version graphique et interactif du projet.
Mais également implémenté les désignes patterns :
1.Modèle Vue Controller(MVC).
2.Observer observé, permet d'avoir une actualisation directe entre ce qui est fait graphiquement ainsi qu'avec les classes modèle en établissant une relation « d'inscription » entre elles.

[1] https://poesi.esi-bru.be/mod/resource/view.php?id=3023
[2] https://poesi.esi-bru.be/mod/resource/view.php?id=3069

2. Structure

La mise en place du découpage en séparant les package, permet d'apporter plus facilement la notion de Modèle Vue Controller et de réduire la dépendance entre les différente package.



a. App

Permit d'afficher via la console de jouer une partie de Stratego, le contrôleur qui

est dédié au fonctionnement de la console.

1) Controler

-La méthode 'loop ' lance la boucle principale entre le lancement du programma jusqu'à la fin de celui-ci. Lance la phase de spawn, la partie jusqu'à la victoire d'un de joueurs et sa fin.
-La méthode 'init' permet d'initialisé le board pour le début de parti de chaque joueur.
-La méthode 'createFileMap' utilise le tableau de string crée dans un fichier placé dans un dossier map permettant de choisir un placement de début par défaut.

2) View

Donne les différents affichages utiles pour la bonne visualisation du programme.

-la méthode 'toUpper' transforme un string en majuscule.

2) Color

Permet d'afficher des couleurs dans le terminal.

b. Métier

Le model du Project. Qui contiendra pour l'instant toute les classes créent pour le modèle Métier.[3]

1) Entity

-Est la représentation d'un pion qui peut avoir un rôle définit dans l'énumération ROLE et peut être Flag,Spy,Bomb,etc.

[3] DiagrammeClasse.svg

-La méthode ‘initEntity’qui permet d’initialisé l’attribut ‘attack,distance,symbole’ dépendant de son ROLE.
-L’attribut inGame, permet d’indiqué s’il est dans la partie.
-L’attribut distance, est la représentation des mouvements possible du pion 0(immobile) à 10(Scout).
-La méthode « attack » comparer deux entity s’ils ont le même propriétaire et compare leur attaque pour renvoyé un nombre dépendant de celui qui l’emporte.
-La méthode « operator == » check si deux entity sont les mêmes grâce a leur adresse mémoire.

2) Case

La Case indique s’il contient un pion. Sinon indique si celui-ci est disponible ou si celui-ci est un trou.

3) Board

-Son attribut cases_ est un tableau de Case auquel 3 état peuvent être possible.
1. La case d’un tableau fait référence à une’ Case ‘ayant une entité.
2. La case d’un tableau fait référence à une’ Case’ vide.
3.La case d’un tableau fait référence au vide (si c’est une case troue).
- Deux vecteurs de types entité est attribué à chaque joueur, dont chaqu’un est la liste d’entité que les joueurs possèdent.
-La méthode ‘move’ permet de déplacé l’entité d’un endroit donné à un autre.
-La méthode ‘destroy’ enlevé l’entité à la position donnée.
-La méthode ‘put’ permet de mettre des entités à des positions données.
-La méthode ‘checkDash’ permet la bonne application du comportement du scout pour les dash.

4) Game

Initialise le jeu, donne quel joueur est courant ainsi que l’état du jeu.
-La méthode ‘isEnd’ vérifie si le jeu est finis.
- La méthode ’put’ permet de mettre une entité se trouvant dans la liste des entités du joueur dans une position donnée en début de partie (uniquement si STATE est SPAWN).
-Méthode ‘turnPlayer’ change le joueur courant (Uniquement si STATE est TURN ou SPAWN).
-Méthode ‘moveTo’ permet de déplacer l’entité dans une position donnée (Uniquement si STATE est TURN).

5) MapReader

Ici cette classe permet de lire un fichier texte et de retourner un tableau de string délimité par des espaces.

6) ROLE

L’énumération qui donne le rôle de chaque pion ainsi que son nombre total par joueur permet de contrôler la limitation de rôle.

7) STATE

Donne les états du jeu possible du début jusqu'à la fin du programme.

c. Test

Les tests effectuer dans le projet pour le bon fonctionnement de celui-ci.

d. GUI

L'implémentation graphique du projet auquel l'utilisateur effectué des interactions hors commande.

1) Controller

-Est principalement là pour connecté entre la vue et le model.
Pour chaque mise à jour effectuer dans le model celui-ci notifie à la vue.

2) ViewGui

Met en forme la vue graphique totalement effectué en code brute.
-La méthode initView : met en forme la fenêtre ainsi que chaque élément contenu.
-La méthode initRightSide : initialise la partie droite de la fenêtre.
-La méthode display : met à jour le plateau de la partie en regardant le model.
-La méthode update : met à jour la vue en recevant la notification du model en cas de changement.
-Le SLOT moveToPut : permet de si l'état du jeu est Spawn, permet de mettre des pions dans le Plateau mais si l'état est Turn, le joueur courant peut déplacer ces pions.
-Le SLOT spawnSelect : à l'état Spawn permet de sélectionner un pion.
-Le SLOT beginStart : permet rentre l'état du jeu en Spawn et le transforme en bouton reset pour abandonner. Et si cliqué redeviens le bouton start.
-Le Slot beginLoad :si la partie n'a pas encore commencer, on peut charger le fichier pour le Spawn de la partie.

3) Pawn

La classe Pawn est un QPushButton représentant soit un pion, soit une case vide ou soit un trou.
La gestion des événements se fait à partir de SLOT.