RAPPORT DIABALLIK

DE SOUSA PEREIRA, ANDRE FILIPE - 51999 | TOUDA LACHIRI, ANAS - 43256

Table des matières

Description du Projet

Tout d'abord, une brève description du jeu s'impose, Diaballik c'est un jeu de plateau de stratégie conçu par Philippe Lefrançois. Il tente de simuler un sport où 2 équipes tentent simultanément de placer leur ballon sur la ligne de but des équipes opposées.

Le but du projet est donc d'implémenter le jeu en utilisant le langage de programmation C++. La première partie du projet, celle dont nous allons parler dans ce rapport, vise l'implémentation en mode console.

Précisions relatives au projet

Une fois que l'on exécute le programme, des options de jeux, c'est à dire la taille du plateau de jeu ainsi que l'utilisation ou non du mode « variante », seront proposées aux utilisateurs :

```
Welcome to the Diaballik Game!
Enter the board size: [5, 7 OR 9]:
5
Play with variant mode? (yes/no):
no
```

Après que les utilisateurs font le choix des options de jeux, le jeu commence.

```
The game has been started !
Write help for help with the commands!
     0   1   2   3   4
 0  (N )(N )(N*)(N )(N )
 1  (  )(  )(  )(  )(  )
 2  (  )(  )(  )(  )(  )
 3  (  )(  )(  )(  )(  )
 4  (S )(S )(S*)(S )(S )

> Enter a command :
```

L'interaction avec utilisateurs se fait via des commandes précises, pour prendre connaissances de ceux-ci les utilisateurs n'auront qu'à utiliser la commande : « help »

```
> Enter a command :
help

===========HELP===========
> SELECT line column
> PASS line column
> MOVE line column
> PASSTURN
> HELP
> EXIT
 ========================
```

Les actions incorrectes faites par les utilisateurs, seront signalées via des messages d'erreurs :

```
> Enter a command :
move 3 3
[ERROR]: No position selected!
> Enter a command :
select 9 9
[ERROR]: Position out of the board!
> Enter a command :
```

Points Particuliers :

-Gestion des Erreurs :

Dans l'utilisation de l'application, soit dans une interface graphique soit dans la console, l'utilisateur peut demander des actions qui ne sont pas d'accord avec les règles du jeu. Par exemple essayer de déplacer une pièce qui ne l'appartient pas, dépasser le nombre de mouvements possibles etc…

Afin de gérer ce genre d'erreurs, nous avons décidé de mettre en place un système de flags. Vu que les méthodes qui provoquent ce genre d'erreurs, devrait être des méthodes avec un type de retour void, nous avons décidé de mettre ce type de retour un entier qui retourne un nombre négatif identifiant l'erreur, ou un nombre positif qui décrit ce qui a été fait.

Cette décision est due au fait de simplifier le code, afin de ne pas lancer une dizaine d'exceptions dans une méthode et de nous permettre facilement retrouver le type d'erreur. En utilisant des exceptions, pour pouvoir détecter exactement le type d'erreur provoqué par l'input de l'utilisateur, on aurait du soit créer une exception pour chaque type d'erreur, soit une exception générale contenant par exemple une enum qui identifiait le type d'erreur. Donc vu la petite quantité de méthodes qui provoquent ses erreurs nous avons décidé d'adopter ce système.

Les flags des erreurs peuvent être consultées dans le fichier log.txt, et les messages d'erreur dans le fichier du code source ErrorMessages.cpp. Les flags sont gérés dans la classe Diaballik et Board.

Les prototypes des méthodes qui adoptent ce système sont :

- `int movePiece(const Position & pos);`
- `int throwBall(const Position & pos);`
- `int select(const Position & pos);`

En ce qui concerne d'autre types d'erreurs (ex : passer une lettre à un argument de commande quand ce devrait être un entier), le système traditionnel d'exceptions est utilisé.

-Gestion Antijeu :

Dans cette première version console, l'anti jeu est évalué en fin de chaque commande avec la méthode isOver() de la classe Diaballik. Donc le joueur n'a pas la possibilité de déclarer antijeu ou pas, celui-ci est automatiquement déclaré en cas d'antijeu, et la partie se termine.

Afin de détecter l'anti-jeu, la classe Board possède la méthode :

```
-bool checksAntiGame(Team antiGameVictim) const
```

Cette méthode parcourt chaque ligne de la première colonne afin de trouver une pièce du joueur adversaire. Dans la version originale du jeu, le nombre de pièces d'un joueur est égale au nombre de colonnes dans le plateau, donc pour former une ligne d'antijeu, il faut que toutes les pièces adversaires soient distribuées sur des colonnes différentes, sans

avoir d'espace entre elles. Donc dès qu'on trouve une pièce adversaire, la méthode *verifyLineAntiGame* dans Board est appelé, et vérifie si les pièces adversaires font une ligne d'antijeu, en comptant le nombre de pièces qui ont été bloquées. La méthode s'arrête lorsqu'elle trouve une pièce qui a déjà dépassé la ligne d'antijeu, ou s'il n'y a pas de ligne antijeu. Dés que la méthode arrive a la dernière colonne, elle vérifie si y a au moins 3 pièces qui ont été bloquées pour la ligne, sinon la ligne d'antijeu n'est pas prise en compte.

-Gestion Fin de jeu :

La fin du jeu peu avoir 3 causes :

- Une pièce arrive a la ligne d'objectif avec la balle. Pour cela la méthode checkGameIsFinish de Board vérifie s'il y a une pièce qui arrive à la ligne objective. Cette méthode reçoit une référence d'optionnel<Team> qui est en principe vide, afin de pouvoir informer le joueur gagnant en cas de fin de partie. Un attribut int objectiveRow a été ajouté dans la classe Piece afin de faciliter la vérification dans le mode Variante.
- Le joueur courant est victime d'anti jeu. Seulement le joueur courant peut être victime d'anti jeu, il ne peut pas faire de ligne antijeu vu qu'il peut encore déplacer ses Pieces tant que le joueur adversaire est impossibilité de bouger. Des que le joueur courant termine son tour, il est vérifiée si l'adversaire est victime d'anti-jeu.
- Le joueur courant abandonne la partie. Le joueur courant est le seul capable d'exécuter des commandes donc il est le seul à pouvoir abandonner la partie. Il perd la partie en cas d'abandon.

Écarts/bogues par rapport à l'énoncé :

Le projet a été testé exhaustivement, aucun erreur/bug a été détecté. Les points plus sensibles et plus testés sont :

-**Input :** aucun type d'input peut compromettre le bon fonctionnement du programme où le bon déroulement du jeu.

-Fin du jeu : Le jeu se termine bien quand il faut terminer (fin normale, fin par ligne-anti jeu), et identifie bien le gagnant de la partie, en tenant compte aussi de la variante.

Problèmes rencontrés :

Aucun problème en ce qui concerne le langage C++ ou l'implémentation du jeu.
Plus tôt, des problèmes de analyse/conception sur l'architecture MVC en essayant d'arriver à un polymorphisme entre une Vue console et une Vue graphique. Vu qu'une Vue en console se déroule sur une boucle dans le Controller au lieu des Signaux, et nécessite d'un input par l'entrée standard.

Estimation du temps passé sur cette partie de l'énoncé :

Le temps passé sur la partie console du projet Diaballik est d'environ 4 semaines. Nous estimons en moyen 25h par personne donc un total de 50h.