VEITH Fil TESEI Roméo S2A

Rapport « Gomoku »

M2103 2021

VEITH Fil et TESEI Roméo, groupe S2A

Ce rapport décrit la conception du jeu Gomoku.
Dans ce jeu, que l'on peut comparer au morpion ou au Puissance 4, le but est de poser des pions sur un plateau, en alternance avec son adversaire, afin d'aligner 5 de nos pions.

Sommaire

VEITH Fil TESEI Roméo S2A

1. Analyse

1.1) Spécification des besoins

Le programme doit fonctionner en mode texte dans la console.

1.2) Conception

Pour ce jeu, quelques objets semblent évidents au premier abord :

→ Nous aurons besoin d'un écran (que l'on affiche et qui s'occupera d'une grande partie du code), de joueurs et d'une interface avec le/les utilisateurs.

Nous avons aussi fait le choix de créer une classe Case, afin que le plateau ne soit qu'un tableau de ces objets.

1.3) Programme principal

Le programme principal est donc composé de ces parties :

→ Interroge le.s joueur.s sur le mode de jeu, leurs noms, la taille du plateau.
→ Initialise le plateau.
→ Enfin, une boucle tournant tant que la partie n'est pas terminée qui demande aux joueurs, chacun leur tour, leur action.

2. Réalisation

Nous avons fait le choix de créer qu'un package général, dans lequel nous avons coder plusieurs classes différentes, représentant chaque objet et fonctionnalités nécessaires, et un package test, dans lequel nous ne testons que nos classes les plus complexes.

2.1) Package src

C'est dans ce package que se trouve toutes nos classes, dont voici les descriptions :

2.1.1) Classe Screen

Cette classe est la plus complexe car c'est là que la majeure partie de la logique du jeu s'effectue et que les différentes fonctionnalités sont implémentées.

Je n'expliquerai ici que les méthodes les plus importantes, occultant volontairement les autres, n'étant qu'utilitaires pour les méthodes principales.

Tout d'abord, un Screen possède :

→ un Scanner *in*, pour pour lire les entrées de l'utilisateur.
→ une taille *size*, qui représente la taille du plateau.
→ un tableau de Case *image*, le plateau en lui-même.
→ un tableau *alphabet*.
→ un tableau *direction*, qui contient les 8 directions cardinales.

Pour les méthodes principales, nous avons :

2.1.1.a) *checkUserInput*

Cette méthode prend en argument un caractère *letter* et un entier *chiffre*, qui représente l'input du joueur. Elle va ensuite vérifier que l'input est valide, mais aussi qu'il représente une action possible (c'est à dire que la Case est bien vide et aux alentours d'une autre case)

2.1.1.b) *setPoint*

Cette méthode prend en argument un caractère *letter* et un entier *chiffre*, qui représente l'input du joueur, mais aussi un char *nature.* Cette méthode permet de « physiquement » posé un pion sur le plateau.

2.1.1.c) *clear*

Cette méthode sert à vider le tableau, mais aussi à l'initialiser.

2.1.1.d) *display*

Cette méthode affiche le plateau et le nécessaire pour le joueur, et peux prendre une *command* en argument pour customiser le display.

2.1.1.e) *getAllPossiblePlays*

Cette méthode calcule et renvoie une liste de tout les coups jouables sur le plateau au moment de son appel. Elle prend un booléen *afficher* en argument, au cas où l'utilisateur voudrait voir s'afficher la liste au moment de la liste.

2.1.1.f) *checkIfWin*

Cette méthode va tout simplement, en appelant la méthode récursive *checkIfWinFromCase,* vérifier si un des joueurs est actuellement en train de gagner ou non.

2.1.1.g) *restart*

Cette méthode demande au joueur s'il décide de redémarrer une partie après un avoir fini une.

2.1.2) Classe Joueur

La classe Joueur représente évidemment un joueur. Chaque joueur possède :
→ un nom *nomJoueur*
→ un numéro *numJoueur*
→ une nature *natureJoueur* (‘X’ ou ‘O’)
→ un booléen *isIA* qui indique si le joueur est une IA.
→ et enfin, une liste de tout les coups joués *coupJoues.*

Chaque attribut possède les get/set nécessaires.

2.1.3) Classe IA

Cette classe se charge du comportement de l’IA. Chaque IA a un nom, et comme ce projet ne demander pas d’IA très développée, celle ci ne peut que jouer un coup au hasard parmi les coups possibles.

2.1.4) Classe Case

La classe Case représente une case. Chaque Case possède seulement une nature, accompagnée de ses get/set.

2.1.5) Classe Conversion

C’est grâce à cette classe que nous effectuons nos conversions, comme par exemple transformer une String en caractère ou en entier.

2.1.6) Classe UI

C’est dans cette classe que la majeure partie de l’interaction avec le joueur hors-partie se fait.

C’est ici que l’on demande la taille du plateau avec *taillePlateau et* les noms des joueurs avec *choixNomJoueur.*

C’est aussi cette classe qui s’occupe de demander au joueur quelle action il veut effectuer, et détermine quoi faire à l’aide de *userInterface.*

2.2) Package test

C’est ici que nous avons nos classes de test. Nous ne testons que les classes qui en ont besoin, c’est pourquoi nous n’avons que des classes test pour nos classes Conversion, Screen et évidemment Main.

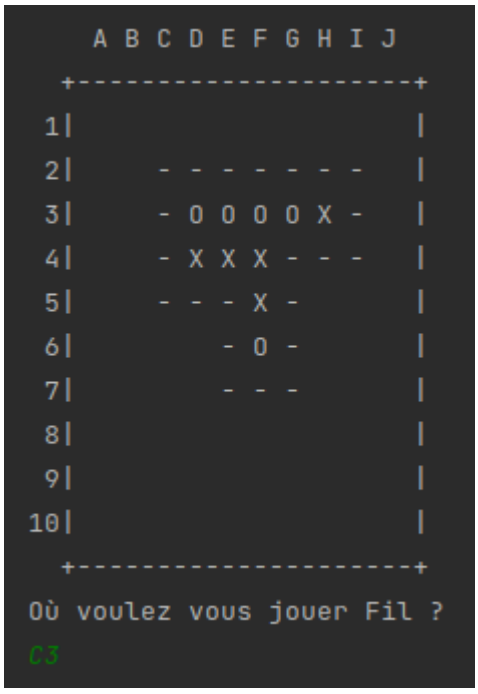
3. Conclusion

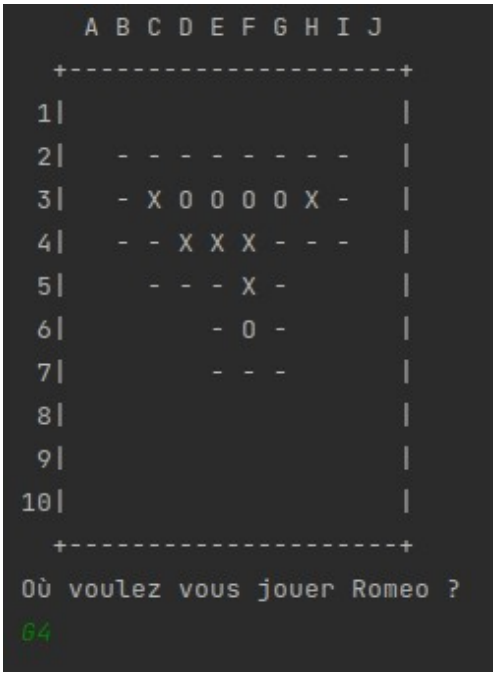
3.1) Exemple d’exécution

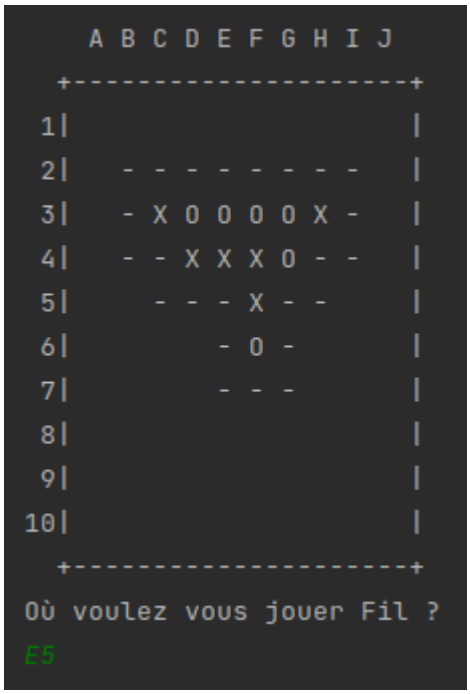
Un exemple de début de partie :

VEITH Fil TESEI Roméo S2A

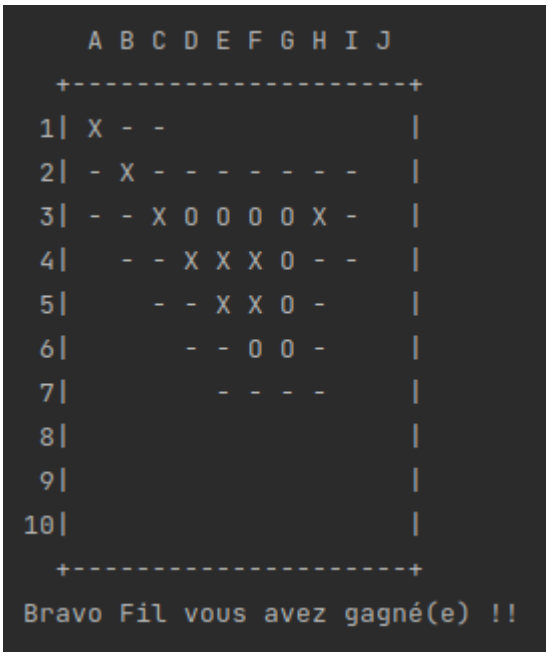
Le « mid-game » :

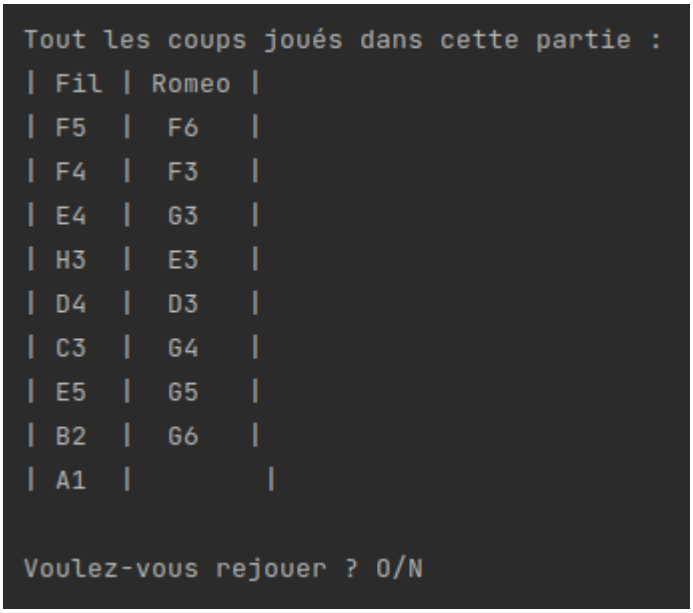






Une fin de partie :





3.2) Remarques

-Notre résultat final comporte tout ce qui nous était demandé, et nous avons aussi rajouté quelques fonctionnalités, comme la commande l’affichage des coups jouables sur le plateau par exemple.

- Il aurait peut être été plus judicieux de découpé le projet en plus de package, mais ce fonctionnement nous convenait davantage.

- Nous aurions peut être pu éviter d’avoir une classe comme Screen, qui est très chargée et effectue beaucoup du travail, en répartissant mieux les fonctionnalités.