Rapport du projet java sur le 2048

Binôme : Bruit Gabriel et Neang Thomas
L3 Info

Sommaire

Introduction

Le but de ce projet est de programmer le jeu du 2048 en intégrant un système de sauvegarde et une intelligence artificielle capable de gagner le 2048 en respectant la programmation objet.

Conception

Avant de commencer le projet nous avons jouer au jeu du 2048 afin de cerner correctement les règles du jeu pour savoir quelles classes nous allons avoir besoin et quelles méthodes devront être implémentées.

Voici les étapes du jeu :

-Initialisation d'un plateau constitué de case avec 2 valeurs placer aléatoirement (90 % de chance pour un 2 et 10 % pour un 4)
→ Ici nous avons vu une classe Board qui comprend un tableau 2 dimensions de valeur
Nous avons décidé de créer un type Value qui contient la valeur plus ses coodonnées car nous avons pensé au format de sauvegarde et à l'interface graphique.
→ Coordonnée = Classe Point.
→ Nous avons mis la largeur d'une case la encore pour la partie graphique.

-Attente de l'action du joueur qui doit presser une touche directionnelle
→ Ici nous avons penser à la classe Event qui récupère un événement et fait l'action.

-Si un événement a lieu alors on génère une autre valeur à une position aléatoire avec toujours la même probabilité de 2 et de 4.
→ Nous avons vu la nécessité de mettre un booléen dans Event pour savoir si le joueur a fait quelque chose sous peine d'ajouter des valeurs n'importe comment. De plus cela nous permet de sauvegarder uniquement les actions pertinentes du joueur.
Sauvegarder un mouvement vers le bas alors qu'il ne fait rien est superflue par exemple.

→ La valeur aléatoire est généré parmi les cases vides

→ Un événement est une modification du plateau donc nous savons que c'est au

plateau de se modifier et pas a la classe Event de le faire.
C'est la que nous avons vu que les méthodes moveLeft, moveRight… était dans la classe Board.
Nous pensions qu'il suffisait de déplacer chaque case dans la direction voulu si on avait un 0 ou la même valeur, on faisait la somme mais l'effet voulu n'était pas le bon car plusieurs case pouvait fusionner alors que le jeu ne le permettait pas normalement.

-Si deux cases fusionnent le score est mis à jour
→ Une méthode update et un champ score dans Board

-Retour à l'étape 2 et attente de l'action du joueur si le jeu n'est pas fini.

Question : Quand le jeu est-il fini ?
Le jeu se termine quand le plateau est rempli et que plus aucun mouvement n'est possible.

→ Donc nous avons deux méthodes pour déterminer ces réponses isFull et canMove.
Le joueur gagne quand il une valeur supérieur ou égale à 2048 dans le plateau.
On a donc une méthode contains2048.
Ici plutôt que de parcourir le plateau à la fin on a décidé de mettre un champ highValue et de le mettre à jour à chaque fusion.

Ensuite vient la classe Game qui permettra de gérer les options et qui est la structure principale du programme.

Grâce à ce raisonnement nous avons pu déterminer les classes et méthodes nécessaire pour finir l'étape 1.

Nous avons ensuite regarder les services que le programmes devait fournir donc il s'agit de la sauvegarde et de l'IA et regarder si leur implémentation était compatible avec notre conception.

Pour l'IA, nous n'avons pas eu de soucis il s'agit d'un Player qui joue de la même manière que le joueur humain.

L'IA possède la méthode takeDecision qui consiste à appliquer l'algorithme alpha beta pour les 4 mouvements disponible afin de choisir le meilleur.

Pour la sauvegarde par contre nous avons constaté que si les premières valeurs du tableau ainsi que les prochaines valeurs était stocker directement dedans nous ne pouvions pas savoir laquelle a été ajouter. Si on ajoute un 2 et qu'il y en avait déjà dedans lequel est le bon ?

Nous avons donc décidé de mettre 3 champs Value(firstValue,secondValue et nextValue) dans Board pour faciliter la sauvegarde et de créer une classe Save qui s'occupera de faire la sauvegarde.

Pour mettre l'option de replay, le plus dur était de gérer les erreurs du au format du fichier sinon la classe Event permet de choisir la direction à partir du fichier et le champ nextValue dans Board permettait aussi de rajouter correctement la prochaine valeur.

Pour l'option -a et -n nous avons fait une seule classe qui est Aléa et qui change son format de lecture suivant l'option.

Implémentation

Après avoir eu une idée général des méthodes et des classes qui devraient être présente dans le projet nous avons réfléchie à la manière d'assembler le tout. Cette phase du projet était la plus délicate et il fallait attribuer correctement les responsabilitées à nos classes pour éviter de dédoubler du code.

Vu que nous connaissions pas l'interface KeyListener nous pensions que la classe Event devait être une JFrame pour récupérer l'événement, mais un événement n'est pas une fenêtre et nous avons donc réfléchie à une autre implémentation respectant mieux le concept de programmation objet.
Après quelques recherche nous avons décidé de créer une classe Window qui sera une Jframe et de faire une classe Player correspondant au joueur qui implémente KeyListener que l'on pourra ajouter a Window via la méthode addKeyListener.
C'est une manière de « donner la main au joueur » pour qu'il puisse jouer.
A tout moment nous pouvons lui retirer la main pour éviter qu'il ne créer des problèmes, comme pendant un replay par exemple.

La classe Game se charge du bon déroulement du jeu et de la gestion des options.

Donc dans un jeu nous avons un joueur, la fenêtre du jeu avec dedans le plateau, des événements et des options de jeu détecter par des booléens.

Une nouvelle partie est créer grâce au constructeur de Game.
Nous en avons deux le premier étant celui par défault si on ne veut pas d'option et le deuxième prend en paramètre les arguments de la ligne de commande.

On a ensuite une méthode parse qui se charge d'activer les bons booléens et de crée l'objet qui fera les actions associer et qu'on appel dans le constructeur.

On a une méthode start qui est notre game loop et qui se termine si le joueur a terminé ou si la partie est terminé.
On a plusieurs if dans start pour faire les actions associer aux options choisies par l'utilisateur.
Ces if sont un peu lourd pour la lecture mais nous n'avons pas trouver de manière plus propre pour gérer correctement les options.

On a ensuite une méthode isFinish pour détecter la fin du jeu.

Puis une méthode end pour savoir si le joueur a gagné.

On a décidé de faire un popup en début de partie pour savoir si la partie doit se lancer normalement ou en mode IA.

La classe Board est un Jpanel ce qui nous permet de la rajouter dans la fenêtre pour faire l'affichage.

La création d'un plateau se fait en créant un tableau de Value vide en générant les deux premières valeurs.
Cette classe est chargée de faire les modifications du plateau demandé par la classe Event. Chaque méthode permettant de faire un mouvement renvoie un booléen pour en informer Event. De cette manière la classe Game saura s'il faut générer la prochaine valeur, faire une sauvegarde ou juste attendre la prochaine action du joueur.

Nous avons aussi décidé de mettre le score dans le plateau et non dans Game car le score est mis à jours lors d'une fusion entre deux Value du tableau qui se fait dans Board.
Puis nous avons la classe Value qui correspond à une case du plateau.

Nous avons décider de créer cette classe au lieu de faire directement un tableau d'entier pour permettre un affichage différent suivant la valeur. De plus avec la valeur nous devions stocker les coordonnées de la case ce qui permet de faire des sauvegardes et des replays plus facilement.

Nous avons créer une classe Point pour stocker les coordonnées et avons décider de rendre les champs non mutable car les cases de notre plateau ne bouge pas ce sont les valeurs qui sont mis à jour.

Ensuite vient la classe Event qui se charge d'exécuter l'action voulu par le joueur.
Cette classe se résume à un switch et suivant l'input, l'action est effectué, son fonctionnement est simple.

Vient ensuite l'implémentation du joueur.
Au début nous avions deux classe une pour l'IA et une pour le joueur normal.
Cela impliquer une duplication de code dans Game avec une partie pour le mode IA et une pour le joueur normal.
Nous voulions donc que le jeu se lance de la même manière que ce soit l'IA ou le joueur et enlever cette redondance de code.
Nous avons donc fais une classe abstraite Player et deux classes filles IA et Human car une IA est bien un joueur ce qui nous permet de faire ce qu'on voulait.

Donc un Player c'est un entier qui correspond à la touche saisie par le joueur.
Et deux méthode abstraite takeDecision et getKey.
Pour récupérer les touches on a donc Player qui implément KeyListener.
Pour un joueur normal (Human) takeDecision ne fait rien car ce n'est pas au programme de décider et getKey renvoie la touche pressé par le joueur.
Il faut après réécrire les méthodes de KeyListener.

Pour l'IA takeDecision permet d'utiliser les algos de minmax et d'alpha béta pour choisir l'action à effectuer donc les méthodes de KeyListener ne font rien et l'action est transmis directement.

Enfin il reste les classes d'entrées-sorties qui sont constituées du fichier et d'un flux d'écriture ou de lecture dans ce fichier.
L'utilisation de ces classes implique de gérer une IOException qui peut se créer lors du chargement d'un fichier ou de problème de lecture et d'écriture.
On a décidé de faire remonter l'exception et de l'attraper dans le main ce qui

nous évite de faire un try catch dans chaque méthode.

Pour la sauvegarde nous n'avons pas eu de difficultés particulières, on a une méthode qui sauvegarde les deux premières valeurs du plateau en prennant un Board en paramètre et une méthode qui prend un Event plus une Value pour sauvegarder un événement et la prochaine valeur.

Pour le replay nous avons considéré que le format était respecter car il est généré par le programme lui même.
Nous avons donc une méthode loadValue qui renvoie un objet Value en lisant dans le fichier et en ignorant les espaces, sauts de lignes et commentaires.
Elle renvoie la Value ou null si on a la fin du fichier ou une erreur.
Les méthodes loadBoard et loadEvent utilise loadValue pour la lecture.
LoadBoard se charge de charger les deux premières lignes du fichier de replay et loadEvent lis la suite.

Pour la classe Aléa on a suivi le même principe que le replay sauf que loadValue renvoie directement un entier.
De plus le constructeur prend en paramètre un booléan pour savoir le format de lecture.

IA

Pour la partie IA nous avons commencer par faire celle de la partie 4 car on croyait que l'ordre des niveaux avait changer.

L'algo de l'alpha beta consiste à descendre dans l'arbre avec la profondeur que l'on a choisi et d'évaluer les feuilles avec la fonction heuristique. Ensuite au retour de la fonction récursive on maximise la note pour un nœud joueur et on minimise pour un nœud jeu. Cela nous permet d'évaluer les meilleurs mouvements possibles parmi les cas ou l'apparition aléatoire a le moins de chance de casser notre plateau. Pendant l'alpha beta on essaye de prévoir ou les prochaines valeurs peuvent appraître et qu'elle valeurs elles peuvent prendre. Donc notre complexité augmente beaucoup en fonction des cases libres et de la profondeur.

Pour la fonction heuristique on a décider d'attribuer une note suivant plusieurs critères. Donc chaque critère possède un poids plus ou moins conséquent

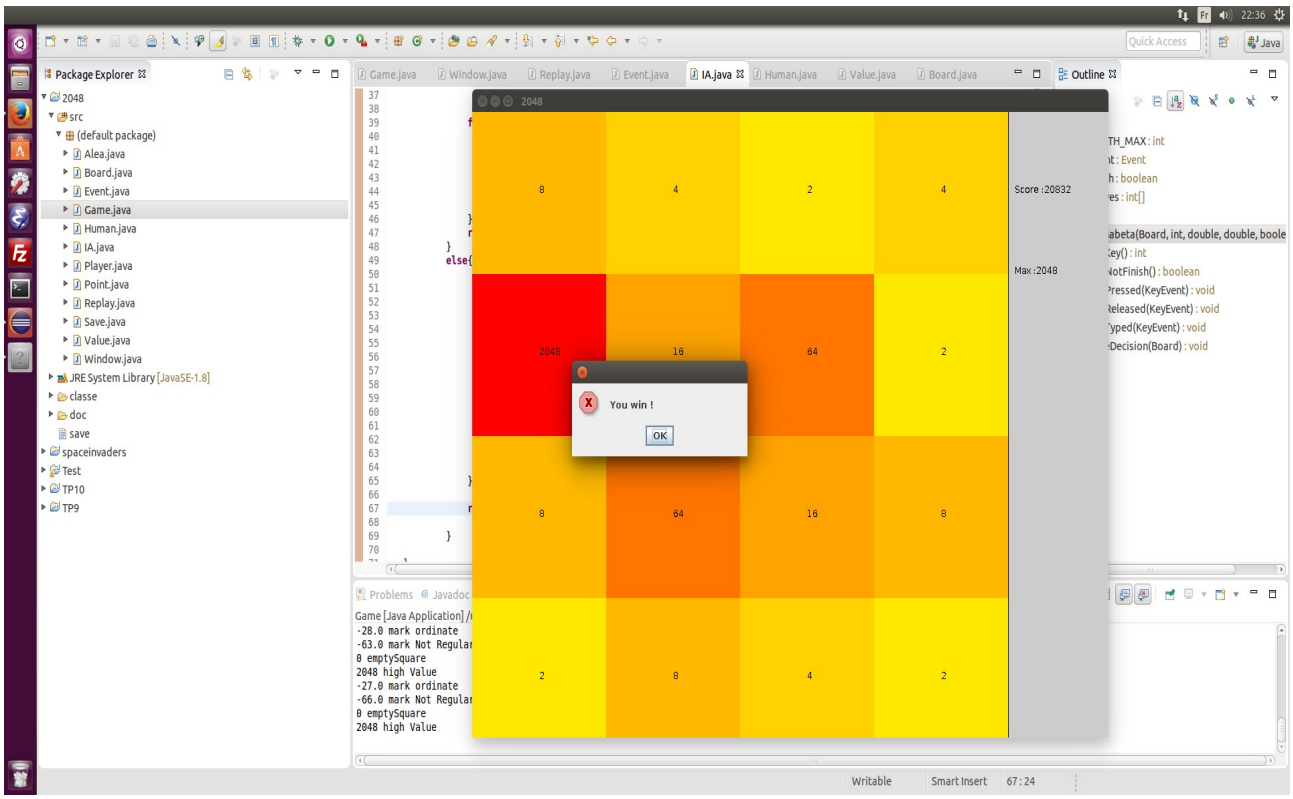
suivant son importance pour avoir un bon plateau.

Nous avons donc considérer les critères suivants :
-Avoir un plateau qui a des valeurs ordonnées
-Avoir un plateau avec le plus de cases vides
-Avoir des valeurs équilibré pour permettre des fusions plus facilement
-Avoir une valeur haute sur le plateau

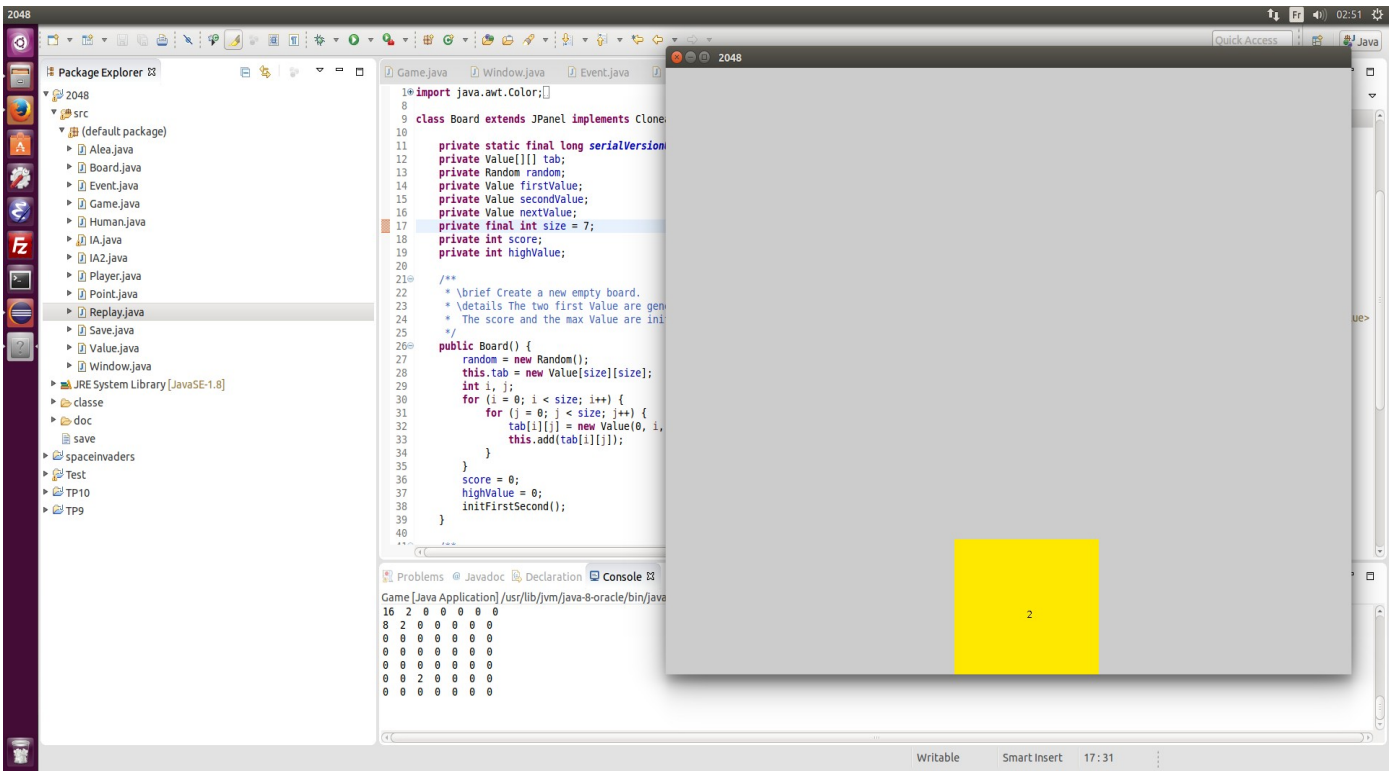
Nos critères les plus important pour notre fonction heuristique ont été le nombre de case vide et le fait d'avoir un plateau équilibré.
Pour l'évaluation de ces caractéristiques nous avons du utilisé Math.log pour équilibrer notre notation pour ne pas utiliser les valeurs direct du plateau car cela créait un déséquilibrage pour des valeurs plus élevée .

Nous avons constaté un bug qui arrive de temps en temps. L'IA ne prend plus de décision. Nous pensons que le problème vient du cas où dans les prévisions de l'IA qui dépend de la profondeur, elle a vu que les 4 mouvements pouvaient entraîner une défaite et qu'elle ne fait donc pas de choix, comme les 4 notes pour chaque mouvement donne -INFINI.
Edit :Le problème venait effectivement du retour de l'alpha beta en cas de nœud perdant et a été corriger. Nous avons monter la profondeur de recherche à 4 au lieu de 3. La proba de gagner passe de 3/10 à 7/10 et met en moyenne 10 minutes.

Note: Nous avons coder l'IA pour qu'elle fasse le 2048 en moins de mouvement possible et pas pour aller le plus loin possible.En général quand l'ia fait un 2048 la partie s'arrête car quand on essayer d'aller le plus loin possible l'ia essayait de garder le meilleur plateau possible à chaque fois.
Donc quand on avait un 1024 et 2 valeur a 512, l'ia préférait garder un plateau correct au lieu de faire le 2048 et on pouvait perdre avant.
L'idée d'aller le plus loin possible aurait pu être possible en améliorant nos algos pour augmenter la profondeur de recherche car on ne peut pas aller plus loin que 4. A partir d'une profondeur de 5 cela devient trop long .



Nous avons pensés au niveau 5 du projet depuis le début et le jeu est bien jouable cependant l'interface graphique n'est pas adapté à la fenêtre. Le jeu devient donc jouable mais en terminal car nous avons eu quelques difficultées pour faire la partie graphique.
Pour cela il faut modifier le champs size dans la classe Board pour mettre la taille désiré.



De plus si on utilise le mode IA sur de plus grand plateau la complexité augmente rapidement à cause de la partie ou on essaye de prévoir l'apparition de la prochaine valeur aléatoire.

Pour l'ia du niveau 3 nous avons essayer de ne pas utiliser d'algo similaire au minMax ou l'alpha Beta. Nous avons essayer d'appliquer uniquement notre fonction heuristique en fonction des mouvements possibles et de le refaire sur le plateau engendré jusqu'à a une certaine profondeur. Mais on n'obtient pas les même performance qu'avec notre IA du niveau 4.
Edit:Nous avons compris trop tard qu'il fallait utiliser un Random dans cette IA qui utilise la même graine que le plateau pour prévoir les prochaines apparitions.
On aurait pu utiliser le même algo que notre niveau 4 en prenant en compte les vrai valeur qui vont apparaître pour gagner du temps et de l'efficacité.
Par manque de temps nous n'avons pas pu implémenter notre idée.

Fonctionnement du programme

Nous avons fais un build.xml mais nous n'avons pas d'option pour choisir de lancer le solveur ou une partie normal.
Le programme demande directement le mode de lancement à l'exécution grâce à un popup qui est en fait le menu du jeu.

Le build.xml correspond à celui fourni sur moodle pour faciliter vos tests,il faut juste fournir le fichier aléa pour le niveau 2 qui possède le même nom que dans le sujet. Pour le niveau 4 nous avons mis une graine ou notre IA trouve 2048. Pour le tester avec d'autre valeur il suffit juste de lancer le Niveau 1 qui est en fait le lancement par défault de notre programme et choisir l'ia comme joueur.

Conclusion

Nous avons remarqué qu'en programmation objet il y avait plusieurs manières d'implémenter notre projet et que certaines ne permettait pas de programmer correctement ce que l'on souhaitait faire.
Nous avons eu des problèmes lors de la première version du projet pour faire une sauvegarde à cause d'une mauvaise distribution des taches et des méthodes.
La partie la plus dur fut de trouver une bonne fonction heuristique pour évaluer notre jeu correctement.