Dossier de validation - Tablut

Groupe 3 - B. Berkati, E. Berthier, A. Cannone, M. Dufrenoy, M. Duplan, L. Postic

June 6, 2014

1 Préambule

Ce dossier est réalisé dans le cadre du projet de fin de troisième année de licence informatique au sein de l'Université Joseph Fourier basée à Grenoble. Il permet notamment de monter et justifier les choix qui ont été fait pour ce qui concerne l'IHM et l'IA, mais aussi de mettre en lumière les résultats obtenus et l'efficacité du travail réalisé.

2 IHM

2.1 Introduction

Cette section contient l'ensemble des informations concernant l'interface et la validation de cette dernière. La section est divisée en 4 parties.

Dans un premier temps, nous explicitons certains concepts requis pour comprendre le fonctionnement technique de la libgdx. Ensuite, nous justifierons les choix effectués pour améliorer l'ergonomie de l'application. La troisième partie couvre les procédures de validation utilisées en interne pour garantir la stabilité de l'interface. Enfin, nous terminons par décrire les méthodes de validations par des utilisateurs externes au projet.

2.2 Structure de l'interface

L'interface graphique est completement géré par une librairie externe appelé LibGDX. Cette libraire affiche ce qu'on appelle un Screen qui contient toutes les informations de la scène. Un seul Screen est actif simultanement. C'est à travers l'utilisation de la méthode setScreen(Screen screen); que les changement sont effectués.

Sur une scène, il est possible d'afficher des sprites ainsi que divers Widget. Tout les Widgets chargent une apparence en fonction de ce qu'on appelle un Skin. Le Skin contient toutes les informations en ce qui concerne l'ensemble des Widgets. Cela permet d'uniformiser l'apparence de ces derniers.

L'ensemble des scènes comprennent donc un ensemble de Widget. Ces informations sont donc redondantes. Nous avons donc décidé de créer une abstraction d'une scène contenant toutes ces informations redondantes.

2.3 Prototypage

L'interface du jeu doit allié ergonomie et attractivité visuelle. Le meilleur moyen d'obtenir une ergonomie suffisante est de s'inspirer d'autres interfaces de jeux de plateau déjà existant. Ainsi, nous avons testé plusieurs jeux de Tablut en ligne mais nous avons également recherché l'inspiration du coté des jeux d'echecs ainsi que du jeu "42 jeux indémodables" sur Nintendo DS.

— image

Voici une comparaison entre la première et la dernière version du menu principal. Les améliorations ne sont que visuelle. De manière analogue, le menu "Options" a subi les mêmes mutations.

2.4 Validation interne

La validation interne a pour objectif de vérifier le bon fonctionnement technique de l'interface.

2.4.1 Procédure

La procédure est très simple pour les widget au comportement simple (boutons, label, ...). Une fois le widget integré, on lance l'application et on teste le widget. Si le comportement effectué par le widget est celui attendu, c'est validé.

Par contre, pour les widgets complexes, il peut y avoir plusieurs paramètres differents possbiles pour une plage de résultat très varié. La procédure est donc plus strict dans ces cas. Voici la liste des Widgets complexe utilisé pour ce programme :

- OptionPane
- PlayerSelection
- GameStatusWidget
- HistoryWidget

Pour les valider, des jeux de tests sont mis en place. Toutes les possibilités doivent être couverte.

2.4.2 Bilan

– Tableau a venir

2.5 Validation externe

La validation externe a pour objectif d'améliorer l'experience de l'utilisateur en receuillant les idées et critiques de personnes externes au projet.

2.5.1 Panel d'utilisateurs

La validation dite "externe" est divisé en 2 catégories : la validation durant le developpement et la vaidation en bêta-test.

Durant le développement, 4 personnes ont permis les améliorations successives de l'ergonomie de l'interface : le responsable de l'audit IHM, notre tuteur ainsi que 2 personnes exterieur au projet. Cette validation est necessaire pour corriger rapidement les erreurs de conception trop graves très vite durant le développement.

La beta-test est prévu pour le week-end du samedi 7 juin à 10h au dimanche 8 juin à 18h. Actuellement, 12 personnes sont prévues mais le chiffre augmente encore. 58L'echantillon comporte 50

—Cela sera remplacé partiellement par des images

Ce panel varié nous permettra de recueillir des informations variés. Nous pourrons ainsi utiliser ces informations afin d'optimiser l'ergonomie du jeu pour le public le plus large possible.

2.5.2 Déroulement de l'évaluation

Durant le développement, l'evaluation est surtout faite via des retours directs. Lors de discussions sur Skype ou via des audits/rencontres avec le tuteurs. Lors des audits, nous faisons tourner le jeu et le professeur emet ses interrogations et avis durant la présentation.

En ce qui concerne la bêta-test, nous envoyons la première version bêta par mail avec des directives et plusieurs documents. Les documents en question sont un sondage sur l'ergonomie du jeu et un tableau de rapport de bug. Nous demandons à tous les participants d'être le plus honnêtes possibles lors du remplissage de ces documents.

2.5.3 Bilan

Beta test non terminé ...

3 Intelligence Artificielle

3.1 La classe IA

La classe IA hérite de la classe Player. Elle possède sept attributs :

- Un entier représentant la profondeur à vérifier après la première couche. Ainsi, si une profondeur de quatre est donnée, l'IA calucle sur cinq coups.
- Un type enuméré IaType représentant quelle intelligence est sélectionnée. Les types implémentés sont facile, agressif, défensif et difficile.
- Deux PawnType, camp et adversaire, permettant de savoir quel est le camp de l'IA (pour le premier) et de déduire celui de l'adversaire (le deuxième).
- Deux entiers, *vlagagner* et *valperdre*, servant à moduler le nombre et le poids d'une prise/perte de pion.
- Un booléen, *bougerRoi*, sert à favoriser les mouvements incluant le roi en cas d'égalité.

Un algorithme Min/Max avec élaguage Alpha/Bêta est utilisé pour les IA.

3.2 Les différentes difficultés

Comme précisé précédemment, il y a quatre niveaux de difficulté :

- Difficile : un algorithme de profondeur cinq est utilisé. Il favorise les déplacements du roi et un jeu équilibré. Le temps de jeu est un peu long, de quelques secondes à une vingtaine de secondes.
- Agressif : l'algorithme utilisé est de profondeur quatre. Il privilégie les déplacements du roi pour un jeu agressif. Si le sacrifice d'un pion permet d'en prendre un autre, il optera pour cette solution. Cette IA jouera plus rapidement du fait d'une profondeur inférieure.
- Défensif : de profondeur quatre, cet algorithme ne sacrifiera pas de pions pour en manger un.
- Facile : correspond à un jeu équilibré de profondeur trois.

3.3 Les fonctions d'évaluation

3.3.1 Evaluation des coups

Le but est de calculer très rapidement, sur chque noeud de l'arbre, le nombre de pions mangés en cours de route. Ce traitement n'est pas fait sur les feuilles car un pion peut avoir bougé deux fois entre le début et la fin.

private int evalCoup(int mange);

La fonction prend un paramètre, un entier, basé sur un système de masque. De plus, si *bougerRoi* est à *true*, elle ajoute 1 au résultat pour devenir prioritaire.

3.3.2 Evaluation du plateau

private int evalFinal(Board p);

La fonction prend en paramètre l'état final du plateau, et renvois un int qui correspond à un calcul sur le nombre de pions moscovites entourant le roi.

Un algorithme de type Disktra a été envisagé, mais le nombre trop important de feuilles (22 000 au premier tour) ne fait que rajouter une compléxité qui ralentie énormément l'IA.

3.4 Les autres fonctions essentielles

3.4.1 Déplacement sans vérification

public void deplacementsansverif(Move c);

Cette fonction prend en paramètre un coup et effectue le déplacement dans le plateau de jeu. Comparée à la fonction standard, celle-ci n'effectue aucune vérification sur la validité du coup. En effet, tous les coups testés sont générés à partir d'une autre fonction (*getDeplacementPossibles*) et sont donc valides. Cependant, cela permet un gain de temps considérable sur la vitesse de calcul de l'IA.

3.4.2 Vérification des déplacements

Cette fonction permet de récupérer à partir d'une position (x, y) tout les coups possibles pour le pion présent dans la case. Tout les coups générés respectent les règles de déplacement (collision, variantes de déplacement limité pour le roi, variante des forteresses).

public Coup[] getDeplacementPossible(int x, int y);

Elle prend en entrée deux entiers qui correspondent aux coordonnées du pion à déplace, et renvois un tableau de coup qui contient tous les coups jouables par ce pion.

3.4.3 Annuler une prise de pion

Cette fonction permet de remettre les pions qui ont été mangés lors du calcul du meilleur coup pour l'IA.

private void demanger(Block[][] plateau, PawnType dernier, int mange, Move c);

Elle prend en paramètre un tableau à deux dimensions qui permet d'accéder à l'état du plateau. Le PawnType permet de savoir quel joueur a mangé un pion, tandis que l'entier correspond à un masque sur les pions mangés à remettre, ainsi que le coup ayant permis de les manger.