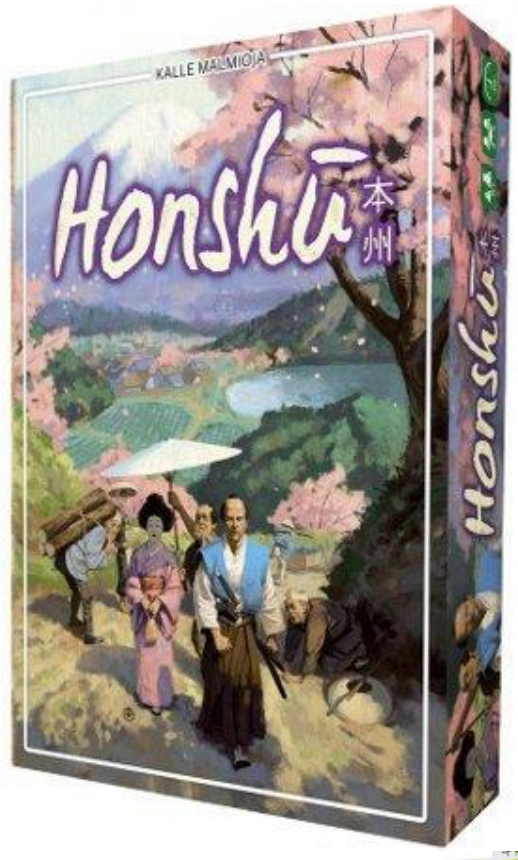
Equipe STRANGER C
Jiahui XU, Kevin XU, Heykel Rajhi, Benoît SCHOLL

Projet Honshu

Rapport LOT C



Sommaire

Introduction

Tâche à effectuer

Pour le lot C, les tâches étaient les suivantes :

- Faire toutes les fonctions pour compter les points d'une grille.
- Intégrer l'affichage du score durant une partie
- Rappel des moyens de gagner des points affichés par l'exécution d'une commande
- Faire un solveur glouton
- Faire un solveur plus efficace

Répartition des tâches

Kevin, Jiahui : Modification de la boucle de jeu, Solveur
Benoit, Heykel : Calcul du plus grand village et du score de village, calcul du score total.

1. Calcul du Score

1.1 ScoreForet

Dans cette fonction le principe est simple : on calcule d'abord le nombre de Foret dans la grille de jeu, ensuite on renvoi le nombre de Foret multiplié par 2.

1.2 ScoreLac

On calcule d'abord le nombre de Lacs dans la grille de jeu, puis on applique la formule 3 * (nombre de lac(s) -1), qui donne le résultat à renvoyer.

1.3 ScoreRess

On calcule le nombre de Ressources et le nombre d'Usines dans la grille de jeu. On choisis le plus petit nombre des 2 (soit on a trop d'usine et pas assez de ressources, soit l'inverse) que l'on multiplie par 4 pour obtenir le résultat.

1.4 ScoreVillage

On utilise l'algorithme de remplissage par diffusion pour calculer le score de chaque village.
Pour une case donnée, on vérifie s'il s'agit du type Ville. Si c'est le cas, on regarde la case au-dessus, en dessous, à gauche et à droite. Afin de s'assurer de ne pas recompter une case déjà compter pour le village en cours, nous avons dû rajouter un entier dans la structure case.

Ce n'est pas dans cette fonction que l'on réinitialise l'entier « comptage » car on ne veut pas recompter plusieur fois le même village dans la fonction « plusGrandVillage ».

1.5 Score_plus_grand_village

On utilise la fonction précédente pour calculer les scores de tous les villages, et on ne garde que le score le plus grand. C'est dans cette fonction, une fois que l'on a le plus grand village, que l'on réinitialise chaque entier « Comptage » de chaque case de la grille.

1.6 calculScore

Ainsi, pour obtenir le score total, il suffit de faire la somme des 4 fonctions au-dessus.

2. Solveur

2.1 Honshu_Opt

Tout d'abord, nous avons implémenté le code fourni en annexe afin d'obtenir un solveur glouton. En effet, il va tester toutes les possibilités de placements de tuiles jusqu'à ce que le deck soit vide. Par conséquent, le temps de calcul augmente très rapidement en fonction du nombre de Tuiles.

De fait, cet algorithme n'est pas optimisé si l'on veut obtenir une solution globale.

2.2.1 Solveur pour un tour

En prennant l'algorithme précédent comme base, on peut définir une nouvelle fonction solveur_tour qui permet de trouver la meilleure tuile à placer pour un tour. On va alors tester pour toutes les tuiles et pour toutes les dispositions possibles, la façon de jouer qui maximisera le nombre de points.

Elle va alors renvoyer un type tuilePosee qui contient les informations de l'action qui fera gagner le maximum de point pour un tour.

De plus, elle prend en paramètre un pointeur vers une fonction Calcul qui correspond à une fonction de calcul de points. De fait, on pourra choisir si l'on veut maximiser le nombre de lacs ou de forêt etc.

2.2.2 Solveur global utilisant le solveur pour un tour

On peut avoir un solveur global mais qui ne va pas forcément donner le maximum de point en utilisant la fonction solveur_tour. Cependant, le temps de calcul est largement réduit par rapport au solveur Honshu_Opt.

Par conséquent, nous n'aurons pas le score maximal assuré, mais une bonne approximation pour un temps de calcul tout à fait raisonnable.

En outre, on pourra aussi choisir ce qu'on veut maximiser.

3 Modification dans la boucle de jeu

3.1 Rappel des moyens de gagner des points et affichage du score

Pour afficher les règles de jeu, on a créé une nouvelle fonction qui affiche la grille, le score et les règles en même temps. Donc l'utilisateur pourra appuyer sur la touche 'h' lorsqu'il voudra afficher les règles.

3.2 Retrait de la dernière tuile posée

Pour sauvegarder les différentes tuiles posées avec leur orientation et leurs coordonnées, nous avons dû créer une nouvelle structure :

tuilePosee	
int	ID
int	X
int	Y
int	Orientation

De fait, à chaque fois que le joueur pose une tuile, les informations sont enregistrées dans un tableau « Historique de tuilePosee ».
De plus, dans notre structure grilleDeJeu, il y a deux autres informations primordiales : un tableau d'entier tuilesDejaPosees qui contient les id des tuiles posées et nbTuilePosees représentant le nombre de tuiles posées.

Ainsi, grâce à toutes ces informations le retrait de la dernière tuile posée est grandement facilité : il suffit de réinitialiser toute les cases de la grillle de jeu à 'E' (Empty) puis on rejoue toutes les tuiles présentes dans le tableau tuilesDejaPosees sauf la dernière.

L'utilisateur aura alors la possibilité à chaque début de tour d'annuler sa dernière action.

3.3 Utilisation du solveur

À chaque début de tour d'une partie de Honshu, le jeu demandera maintenant au joueur s'il veut utiliser le solveur pour l'aider à choisir la meilleure tuile à jouer et aussi la meilleure position.

3.4 Mode solveur

Après avoir choisi un mode jeu entre partie avec aléatoire et partie avec deck préconçu, nous avons ajouté un mode où l'ordinateur joue tout seul. Ce mode va nous servir principalement à tester les solveurs afin de comparer leurs résultats.

4. Test Unitaire

Pour le lot C, nous n'avons fait que le test du calcul de la taille du village. C'est grâce à ce test que nous nous sommes rendu compte que notre fonction échouait dans le cas où le village descendait puis repartait vers la gauche. Grâce au Test unitaire de cette fonction, nous avons pu corriger le problème.