JEU DOMI'NATION

Projet Java A1

Janvier 2019

BAI Yuhe
CHEN Weiyi
LU Chuanhan

Sommaire

I) Introduction 2
II) Présentation du jeu Domi'Nations 3
III) Présentation en java 4
1) Diagramme d'UML et les classes importantes 4
2) Organisation du programme 6
Processus du jeu 6
Préparation 7
Premier tour 7
Un tour normal 8
Règle de placement 8
Fin de la partie et calcul du score + vainqueur 9
IV) Calcul du score : algorithme de parcours en profondeur 10
V) IA 12
Choisir un domino : 12
Placer un domino : 13
Trouver la meilleure position : Algorithme glouton 14
VI) Interface 15
VII) Problèmes rencontrés 17
Réalisation d'IA 17
Algorithme pour le calcul du socre 17
Détection de dimension 18
Réalisation de l'interface 18
VIII) Conclusion 19

I) Introduction

Domi'Nations est un jeu de plateau amusant.

Le but de ce projet est de reprogrammer le jeu sur l'ordinateur avec JAVA afin de permettre aux joueurs de le jouer plus pratiquement.

Nous avons également construit une IA capable de jouer selon les règles afin de permettre aux joueurs de jouer tout seul.

Une interface graphique est également construite à fin de optimiser l'expérience des joueurs.

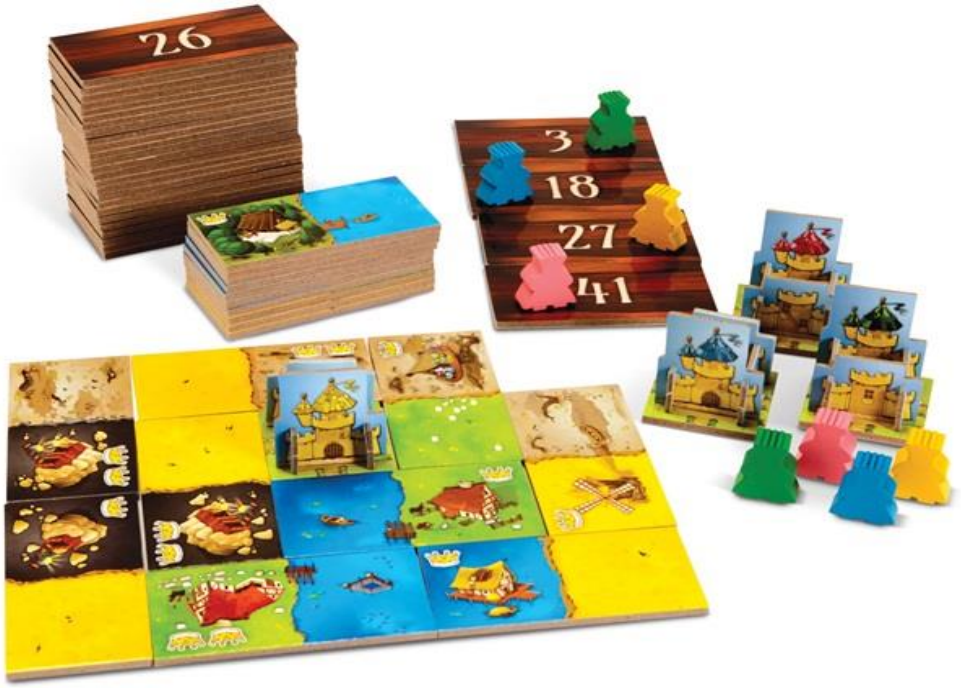
Les parties suivantes vont vous introduire les règles et les méthodes du jeu, ainsi que la manière dont nous avons programmé ce jeu sur l'ordinateur en JAVA.

II) Présentation du jeu Domi'Nations

Un jeu de Domi'Nations permet de 2 à 4 joueurs de participer. Chaque joueur utilisant des dominos pour construire son royaume de taille 5x5.

Le matériel de jeu est composé de :

- 4 tuiles de départ
- 4 châteaux à poser sur les tuiles de départ (rose, jaune, vert, bleu)
- 48 dominos (avec une face paysage, une face numérotée)
- 8 rois en bois (2 de chaque couleur)



Ces dominos comporte chacun deux portions de terrain, certains portions possède un certain nombre de couronne.

Un certain nombre de dominos est utilisés selon le nombre de joueur, chaque joueur possède un ou deux rois, pendant chaque tour, les joueurs choisissent des dominos et les placent dans leur terrain. Lorsque tous les dominos ont été placés, le joueur qui a le plus haut score a gagné.

Le processus plus détaillé sera développé dans les parties suivantes.

III) Présentation en java

1) Diagramme d'UML et les classes importantes

Il y a deux versions de notre projet : une version console et version graphique. Ici on prend la version console pour faciliter l'explication du diagramme de l'UML.

Les classes importantes : Console(Main), Player, AI qui hérite de la classe Player, Domino, King et Area.

Console : Il contient tout le processus du jeu, il contient également des listes pour stocker les objets des autres classes, par exemple dominoList, kingList, playerList etc. Toutes les fonctions et variables des autres classes sont appelées et utilisées dans Console.

Player : Il contient les informations des joueurs, il possède un Area de taille 9*9 et des rois, il possède également des fonctions en lien avec son royaume : par exemple mettre un domino dans le royaume, calculer le score de son royaume etc.

AI : elle hérite de la classe Player parce que l'IA est également un joueur, mais il possède des fonctions en plus : par exemple trouver la meilleure position, choisir le meilleur domino.

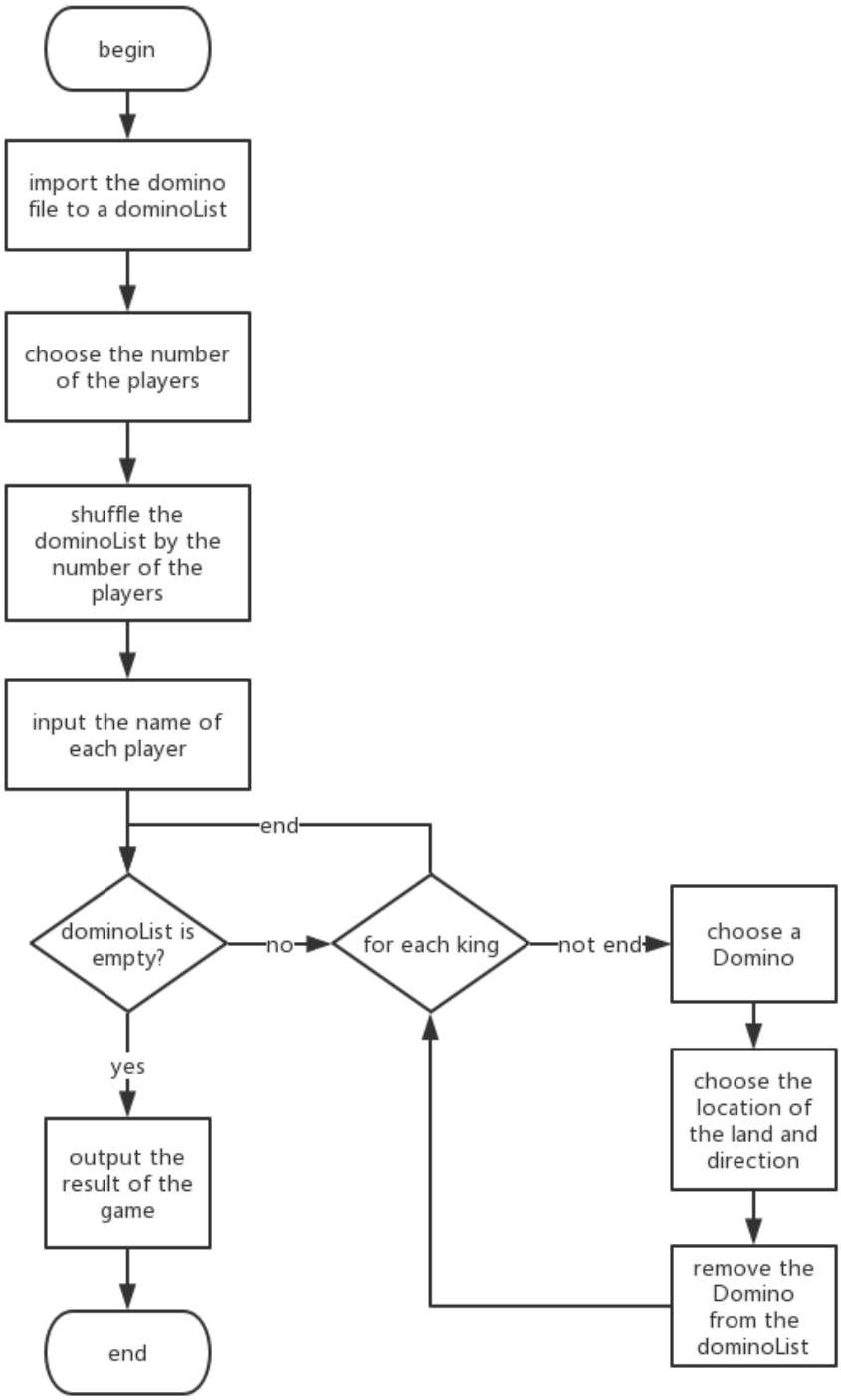
Domino : il possède des informations des dominos, par exemple le type, le numéro de couronne, le numéro de domino et la direction.

King : il possède la couleur, l'Id et une fonction permettant de choisir le domino puisque c'est les rois qui choisissent le domino.

Area : il possède un type, un numéro de couronne, et un boolean isOccupied, un objet area est donc une case dans le royaume.

2) Organisation du programme

Processus du jeu



Le processus du jeu est composé de trois parties principales, la partie préparation, les tours du jeu et la fin.

Au cours de la préparation, on importe les dominos dans une liste, on initialise les joueurs ainsi que leurs royaume et rois, on mélange la liste et retirer le nombre de dominos qu'il faut durant le jeu. Pour chaque tour, on pioche un certain nombre de dominos, chaque roi choisit un

domino et place le domino précédent. Une fois le tour terminé, les dominos sont enlevés de la liste. Quand la liste devient vide, ça signifie que tous les dominos ont été piochés donc on arrive à la fin du jeu. Dans la partie fun du jeu, on calcule les scores et on définit le gagnant.

Préparation

Au cours de la préparation, nous avons d'abord importé le fichier dominos.csv dans notre projet, nous avons également créée une classe Domino pour pouvoir enregistrer les informations du fichier csv sous forme de Domino, donc les 48 dominos sont importés comme 48 objets, il y a deux parties, chaque partie contient un type et un numéro de couronnes, donc pour chaque objet, on lui a assigné son type1, crownNum1, type2 et crownNum2. La direction est définie par défaut D1, c'est-à-dire la première portion est à gauche et deuxième portion est à droite.

Après on demande l'utilisateur de saisie le nombre de joueurs et de mettre leur nom, si on saisit "AI" comme nom de joueur, le joueur en question sera un IA.

Des classes Area et King sont créée pour les objets land et king, chaque joueur possède donc un objet (land) Area[][] de taille 9*9, qui contient toutes les possibilités de forme de son royaume final, on place le château au milieu de ce terrain lorsqu'on l'initialise, après les dominos pourront être placés à gauche, à droite, en haut ou en bas comme le joueur souhaite suivant les règles du jeux. Bien sûr, le royaume final sera de taille 5*5 maximum.

Pour les couleurs de rois, on les assigne aux utilisateurs aléatoirement. Si le nombre de joueur est égal à 2, chaque joueur possède donc deux objets rois, donc il y a 4 rois au total ; si le nombre de joueur est égal à 3 ou 4, chaque joueur possède donc un seul objet roi.

Le nombre de dominos utilisés sont aussi en fonction de nombre de joueurs. D'abord on mélange tous les dominos, si le nombre de joueur est égal à 2, on pioche 24 dominos ; si le nombre de joueur est égal à 3, on pioche 36 dominos ; si le nombre de joueur est égal à 4, on pioche 48 dominos donc tous les dominos sont utilisé.

À noter que pour le choix de domino, c'est les rois qui choisissent le domino, pas les joueurs, les joueurs sont possédant des rois. Pour le premier tour, on tire les rois dans un ordre aléatoire.

Une fois que toutes les préparations sont faites, le premier tour peut commencer.

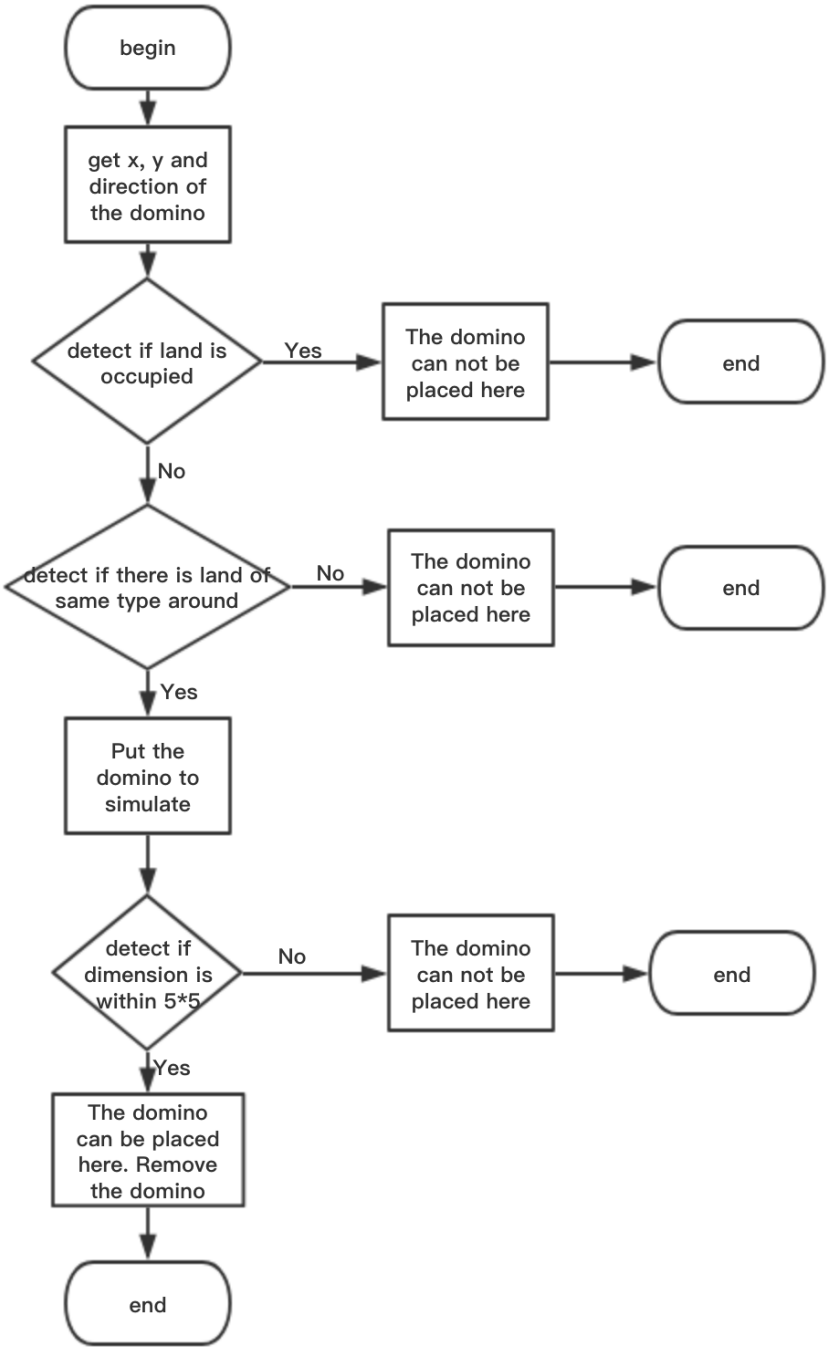
Premier tour

On pioche parmi les dominos en fonction du nombre de rois, c'est-à-dire 4 dominos si on a 2 ou 4 joueurs et 3 si on a 3 joueurs. Ensuite on les place par ordre croissant, à partir de l'ordre prédéfini dans la partie préparation, chaque roi choisit un domino pioché et on passe au tour suivant.

Un tour normal

Pour tous les autres tours, on pioche d'abord parmi les dominos restants le nombre de domino nécessaire en fonction du nombre de rois et on les place par ordre croissant. Cette fois ci l'ordre des rois est selon l'ordre croissant des dominos choisis au tour précédent. Les rois choisissent un nouveau domino, et il pose le domino choisi lors du tour précédent.

Règle de placement



Selon les règles du jeu, le domino doit être placé dans un endroit vide, et il faut qu'il soit à côté d'au moins une case qui contient le même type de domaine ou un château à proximité.

Quand le joueur place un nouveau domino, on vérifie d'abord si la place qu'il a choisi est occupée, sinon, on vérifie s'il y a des terrains du même type autour, ensuite on place le domino dans le terrain pour vérifier si la dimension ne dépasse pas 5*5, si ce n'est pas le cas, on l'enlève du terrain. Seul les choix répondant à ces trois conditions seront acceptés, sinon on refuse son choix et affiche un message.

Fin de la partie et calcul du score + vainqueur

La partie se termine lorsqu'il ne reste plus de domino non placé, au dernier tour les rois ne choisissent plus de domino, ils placent les derniers dominos dans le terrain. À ce moment-là on détermine les différents domaines du royaume de chaque joueur, un groupe de cases qui a les mêmes terrains connectées horizontalement ou verticalement est un domaine.

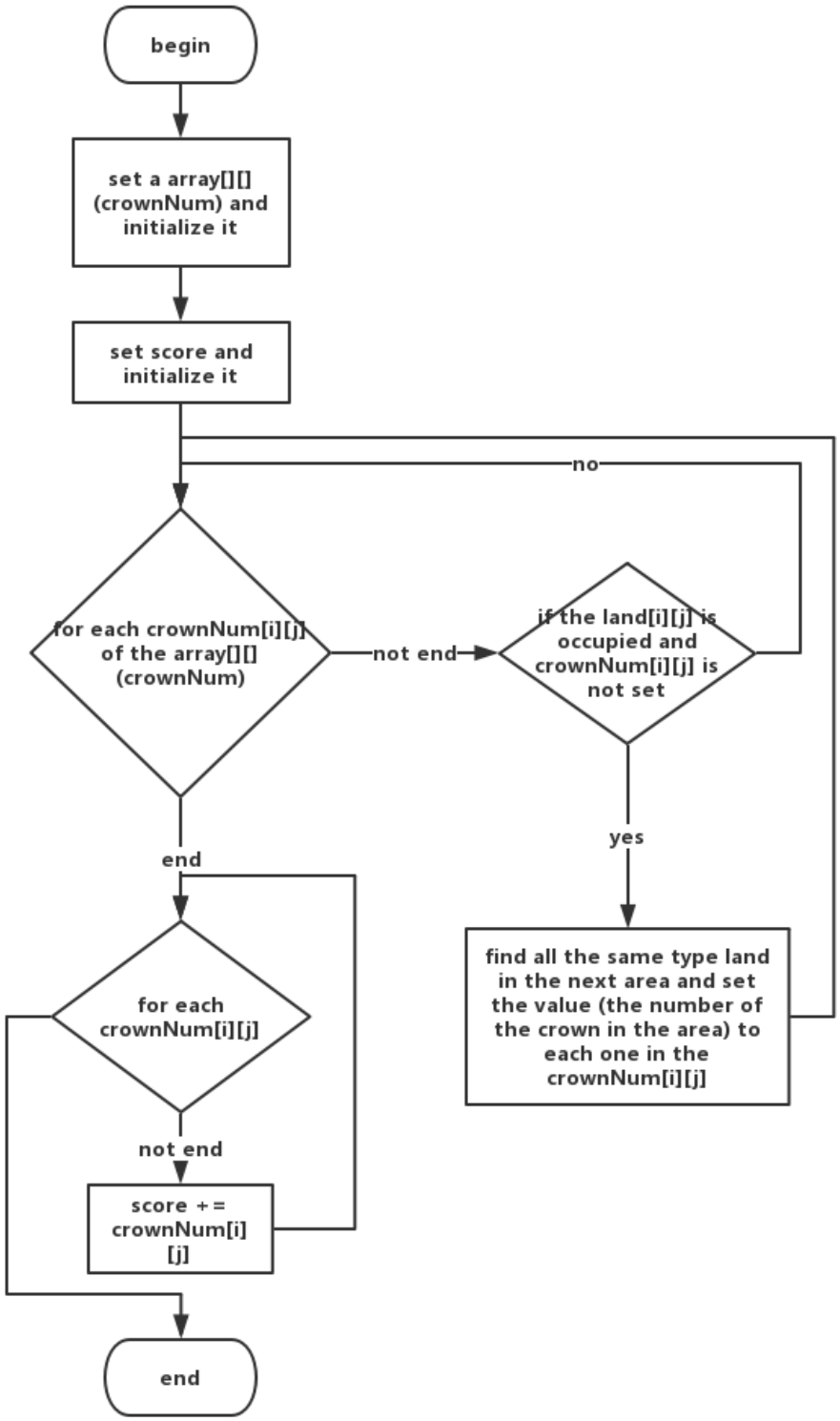
Le score final du joueur est la somme de score de chaque domaine et la formule de score de chaque domaine est：

$$n = N_{cases} \times N_{couronnes}$$

N_{cases} correspond au nombre de cases formant le domaine,
$N_{couronnes}$ correspond au nombre de couronnes présentes sur le domaine.

Le joueur qui obtient le plus grand nombre de points est le vainqueur.

IV) Calcul du score : algorithme de parcours en profondeur



Nous avons utilisé un algorithme de parcours en profondeur (Depth-first search en anglais) pour réaliser le calcul du score. Cet algorithme est très pratique pour déterminer une partie de domaine comme il y a beaucoup de domaines dans le terrain.

Tout d'abord nous avons initialisé un tableau crownNum de taille 9*9 pour indiquer le nombre de couronne totale dans le domaine d'un endroit indiqué, comme ça la somme de crownNum de toutes les cases est bien le score total (par ex. nbCouronne * nbCases = nbCouronne + nbCouronne+…+nbCouronne (nbcases fois)).

Ensuite, nous avons utilisé une pile (stack en anglais) pour trouver tout le domaine connecté d'un endroit donné, et mettre crownNum de toutes les cases dans le domaine au nombre total de couronne.

Trouver tous les domaines connectées d'un terrain donné est le cœur de cet algorithme, on cherche tout d'abord vers le haut jusqu'à il n'y a plus de case de même type en haut, ensuite on cherche vers la droite, À noter que pour chaque case, on cherche toujours vers le haut au début, même s'il est venu par le gauche etc. L'ordre est dont haut - droit - bas - gauche. Une fois qu'une case est parcourue, on la met dans la pile, et on la note 'déjà parcourue'. On recommence la recherche à partir de cette case. Lorsqu'on rencontre une case où dans toutes les directions il n'y a plus de case de même type ou la case est déjà parcourue, on enlève cette case de la pile. Une fois toutes les cases sont enlevées, on obtient un domaine qui contient la case du début.

Une fois que tous les endroits sont vérifiés, on additionne le crownNum de toutes les cases et obtient le score total.

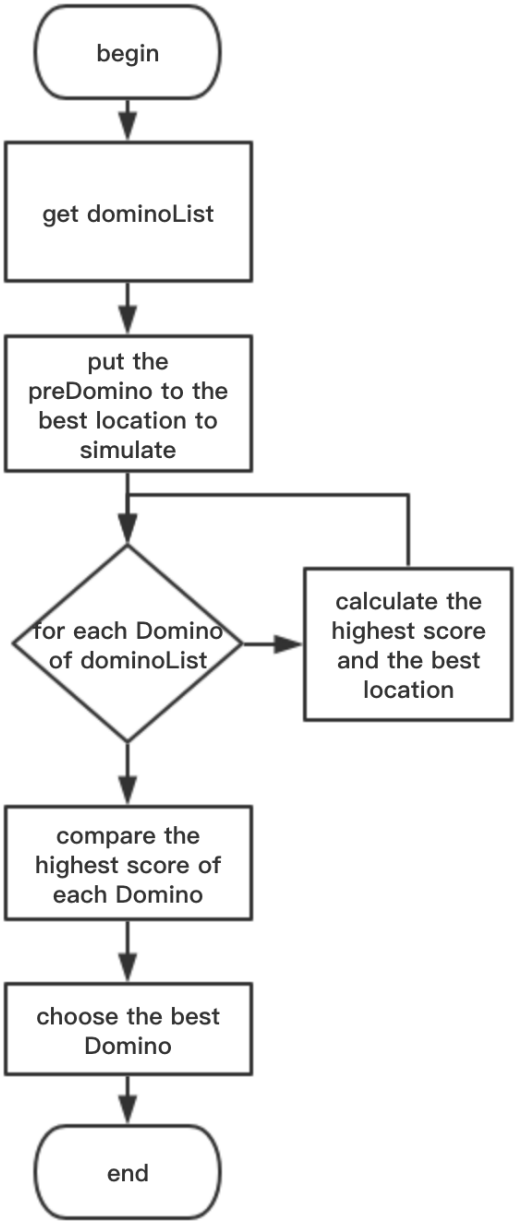
Quelques règles additionnelles sont également impliquées :

Empire du milieu : Ajouter 10 points de bonus si le château se retrouve au centre du royaume.

Harmonie : Ajouter 5 points de bonus si le royaume est complet (c'est-à-dire, fait exactement 5 × 5 cases).

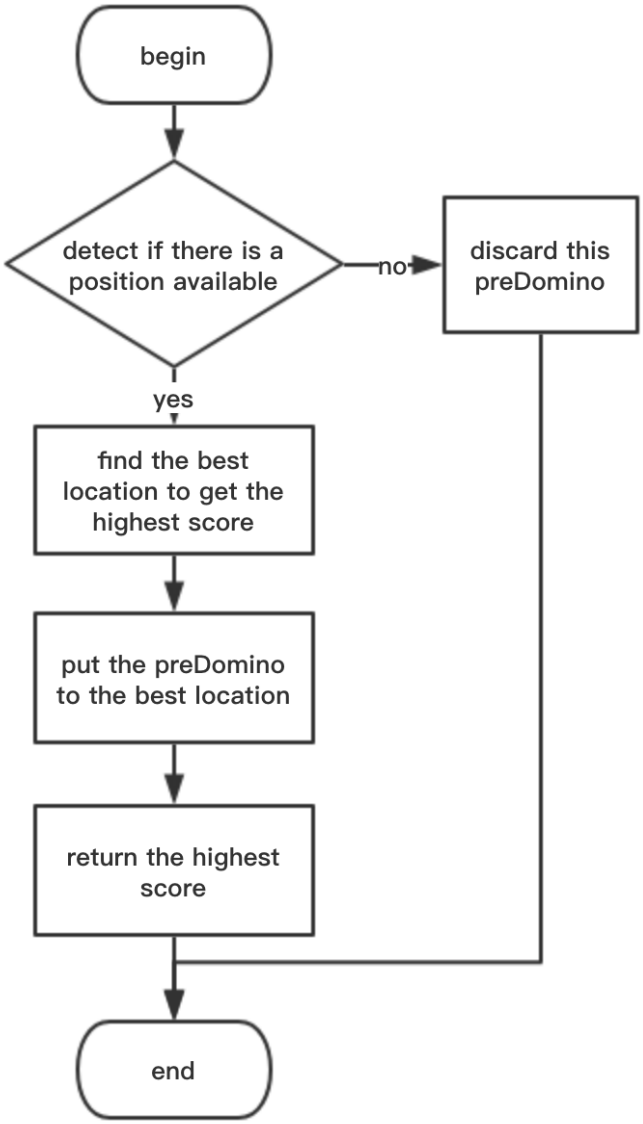
V) IA

Choisir un domino :



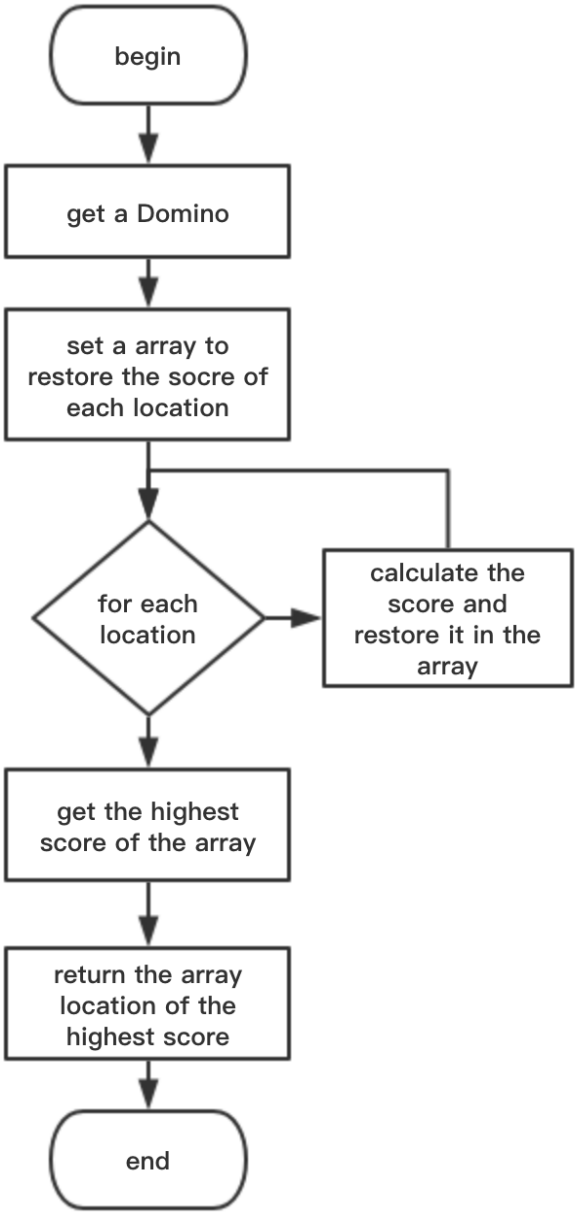
Quand l'IA obtient une liste de domino, elle effectue une simulation en plaçant tout d'abord le domino choisi lors du tour précédent dans la meilleure position calculée, ensuite elle parcourt cette liste, pour chaque domino dans la liste, elle calcule le meilleur score qu'elle pourrait obtenir dans la meilleure position, et elle compare le plus haut score possible pour chaque domino et choisit le domino permettant d'obtenir le plus haut score.

Placer un domino :



L'IA possède maintenant un domino à placer, tout d'abord elle détecte s'il y a au moins une position valable à mettre, sinon elle jette ce domino. Si oui, elle cherche la meilleure position à placer et elle place le domino dans cette position.

Trouver la meilleure position : Algorithme glouton



Pour trouver la meilleure position, nous avons utilisé algorithme glouton (greedy algorithm en anglais). C'est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local.

L'IA possède un domino, elle cherche toutes les positions valables à placer ce domino, elle calcule le score qu'elle peut obtenir en plaçant le domino dans cet endroit, elle met le score dans une liste, une fois toutes les positions calculées, elle cherche la position qui permet d'obtenir le plus haut score. C'est-à-dire si elle place le domino ici, elle peut obtenir le plus haut score pour ce tour, sans considérer les tours suivants.

Petite optimisation : Quand deux ou plus de positions permettent d'obtenir le même score, elle choisit celui plus près du château pour avoir plus de chance d'avoir des points bonus à la fin.

VI) Interface

Pour la réalisation de ce projet, on a utilisé la bibliothèque Java Slick 2D, cette bibliothèque est conçu pour la réalisation des jeux en 2D. Il est inclus dans ce bibliothèque des supports pour image, animations, son etc. On a utilisé dans la réalisation de cette interface, les classes BasicGames, Images, Graphics, GameContainer et les fonctions contenues dans ces classes.

Notre classe Game est une classe fille du classe BasicGames, les classe filles de ce ce classe possèdent trois fonctions particulière : init, update et render. Parmi ces trois fonctions, init est exécutée lors de l'initialisation, on a donc mis à l'intérieur de cette fonction tous les codes à exécuter lors de l'initialisation du jeu. Les fonctions update et render sont exécutés en boucle sans arrêt jusqu'à la fin du jeu, il ne faut donc pas utiliser des boucles while à l'intérieur de ces fonction la car elles peuvent surcharger la mémoire de l'ordinateur. Comme son nom l'indique, la fonction update est la fonction où on met les les code essentiel du jeu (création du joueur, calcule du nombre de rois nécessaire, configuration de la longueur de la liste de domino etc.). La fonction render s'occupe de la partie graphique du jeu, c'est-à-dire l'affichage des dominos, des informations sur le joueur actuel, le terrain du joueur etc.

On a réalisé les différentes parties de ce jeu grâce aux nombre entier associé à chaque phase, voici le tableau de correspondance des phases de jeu et les nombre donnés :

Phase du jeu	Nombre entier donné
écran principal	0
saisir le nombre de joueur	1
création des joueurs	2
triage des rois par ordre aléatoire (seulement le premier tour)	3
tirage des dominos	4
choix du domino	5
placement du domino	6
fin du jeu	100

Par précaution, on affecte à la dernière phase du jeu (la fin du jeu), un nombre entier assez grand pour prévenir les changements des règles de jeux dans le futur.

Grâce à un switch d'une variable de la phase de jeu, on peut exécuter les codes correspondants à la phase actuelle et donc permet aux joueurs d'interagir avec l'interface.

VII) Manuel d'utilisation

- Importez le projet, s'il y a des erreurs de slick, réimportez slick2D.
- Exécuter la classe Main, le jeu commence. Tapez entrée pour commencer.
- Pour avoir un IA, saisissez "AI" en tant que nom du joueur.
- Pour chaque tour, choisissez un domino en cliquant sur le domino, placez le domino dans le terrain en cliquant dans l'endroit vous souhaitez. Il faut taper entrée à la fin de chaque tour.
- Pour tourner le domino, tapez sur les gauche et droit du clavier ou utiliser la touche droite de votre souris.

VIII) Problèmes rencontrés

Réalisation d'IA

Au début, on a voulu faire une IA qui peut apprendre et prendre en compte le futur pour le choix et le placement des dominos, mais au cours de réalisation on a appris que c'était trop difficile, tout d'abord il faut utiliser machine learning et il faut avoir un grand échantillon de donnée, ce qui est très dur. Ainsi on a choisi d'utiliser un algorithme relativement facile, qui est algorithme glouton, c'est-à-dire qu'on étudie seulement le tour actuel, et mettre le domino dans la position qui permet d'obtenir le plus haut score pour ce tour.

Comme le règle du jeu est de choisir d'abord le prochain domino et ensuite placer le domino précédent, on a voulu étudier en même temps deux dominos pour faire le choix, mais on a appris que la complexité est plus haute : O(n^2). Ainsi on a réalisé de manière simple : mettre le domino précédent à la meilleure position pour simuler, et faire le choix du tour actuel.

Au cours de test, on a constaté que le château était toujours dans le coin, c'est donc presque impossible d'avoir le point bonus, c'est parce qu'il y a beaucoup de positions qui permettent d'obtenir le même score, l'IA a choisit le plus petit index, ainsi nous avons rajouté une condition : si deux ou plus de positions permettent d'obtenir le même score, elle choisit celui plus près du château. Avec quelques lignes de codes, l'IA devient donc beaucoup plus intelligente.

Algorithme pour le calcul du score

Au début on croyait que le calcul du score était facile, mais au cours de réalisation, on a appris que c'était difficile, selon le règle du jeu, le score de chaque domaine est le nombre de cases de ce domaine fois le nombre de couronne total, pour simplifier ce problème, on a utilisé un tableau crownNum pour mettre le nombre de couronne total du domaine de chaque case. A la fin le score est juste la somme de crownNum de toutes les cases. Ça nous permet de transmettre le produit en somme, et de transmettre un grand problème à deux petits problèmes.

Le problème le plus compliqué consiste maintenant à trouver tous les domaines. Par l'oeil c'est facile, mais si l'on veut réaliser par ordinateur, c'est relativement difficile. Après des recherches sur internet, nous avons décidé d'utiliser algorithme de parcours en profondeur. Étant donné une position du début, grâce à l'algorithme de parcours en profondeur, nous pouvons trouver le domaine qui contient cette position sans manquer ni répéter.

Détection de dimension

Lors du placement d'un domino, il faut détecter si la position choisie va conduire à dépassement de dimension, on a perçu que c'était difficile de détecter la dimension sans mettre le domino dans le terrain. Ainsi pour la détection de dimension, on met d'abord le domino dans le terrain, et détecte si cela dépasse 5*5, après on retire le domino du terrain.

Réalisation de l'interface

Le problème principal rencontré lors de la réalisation de l'interface concerne les fonctions update et render, comme ces deux fonctions sont exécutées en boucle sans arrêt, les boucles while ne sont pas utilisables. L'exécution de ces deux fonction en boucle nous impose également le problème des variables locaux : certain d'entre eux ne peuvent pas être initialisés dans ces fonction-là, et doit donc être remplacés par des variables globaux.

IX) Conclusion

Nous avons appris beaucoup de choses : réalisation d'IA, beaucoup d'algorithmes (par exemple algorithme de parcours en profondeur et algorithme glouton. et une interface graphique slick2D etc.) On a aussi rencontrés des difficultés de rechercher en JAVA et de travailler en équipe, mais ça n'a changé pas la joie de travailler, nous avons également obtenir beaucoup d'expérience précieuse.

Si nous devons encore faire un travail de groupe similaire, nous pensons pouvoir le faire plus rapidement et plus efficacement que cette fois-ci.