AlphaDice

Andréas Pastor, Antonin Arquey, Julien Triau, Laouenan Le Corguillé

Sommaire

I) Introduction

Présentation du projet

Dans le cadre du mini-projet C du semestre 6 à Polytech Nantes, nous avons dû développer un jeu sur la base de DiceWars (https://jayisgames.com/review/dice-wars.php). DiceWars est un jeu au tour par tour créé par Taro Ito de GameDesign au Japon. Ce jeu se base sur le contrôle de territoires par des dés donc se basant sur le hasard mais aussi sur la stratégie. Le jeu d'origine est composé d'une carte faite d'hexagones. Le gameplay est similaire au jeu Risk, tout en étant plus simple et plus rapide à jouer. L'objectif du jeu est de conquérir toute la carte en attaquant les territoires adverses et en gagnant au lancer de dé de chaque attaque.

Le jeu se joue de 2 à 8 joueurs, à chaque tour le joueur attaque autant qu'il le souhaite et qu'il le peut jusqu'à ce qu'il clique sur End Turn. Le joueur perd lorsqu'il ne contrôle plus aucuns territoires.

Les règles du combat entre deux territoires adjacents sont simples, tous les dés du territoire attaquant sont lancés, ainsi que tous les dés du territoire défendant. Si le joueur attaquant fait un meilleur score que le joueur attaqué, il laisse un dé sur son territoire et met le reste sur le territoire qu'il a conquis tandis que les dés du joueur défendant sont perdus. Si le joueur attaquant fait un moins bon score que le joueur attaqué, le joueurs attaquant perd tous ses dés sur son territoire sauf un. On ne peut pas attaquer avec un territoire si celui-ci n'a qu'un seul dé. On ne peut pas déplacer de dés entre ses propres territoires.

A la fin du tour d'un joueur, il gagne autant de dés que son plus grand regroupement de territoire adjacents, ces dés sont distribués aléatoirement entre ses territoires. Un territoire ne pouvant pas avoir plus de 8 dés. Chaque joueur peut obtenir une réserve de dé jusqu'à 40 dés si toutes ces cases sont déjà pleines de dés.

Contraintes du projet

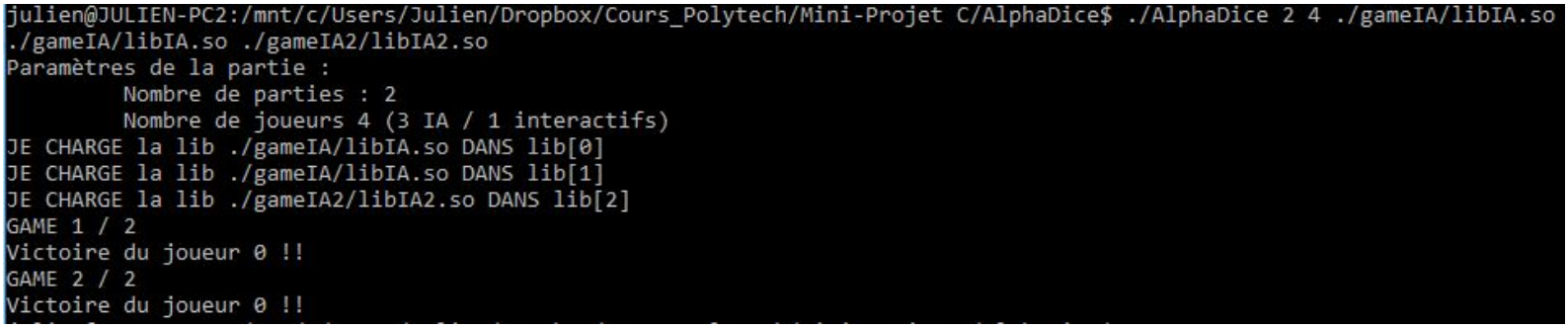
Afin de réaliser ce projet nous avions des contraintes imposées par M. Picarougne. Tout d'abord, la taille de l'équipe doit être de 4 personnes ni plus ni moins, la notre est composée de Andréas Pastor, Antonin Arquey, Julien Triau et Laouenan Le Corguillé. Nous avions l'obligation d'utiliser certaines technologies :

- Git et notamment gitlab afin de collaborer et suivre l'avancement de notre projet.
- SDL afin de faire l'affichage graphique
- Le langage C uniquement

D'autres contraintes plus techniques afin d'avoir une interface commune entre tous les groupes participant à ce projet :

- Doit se lancer entièrement en ligne de commande avec certains paramètres obligatoires comme le nombre de parties à jouer et le nombre de joueurs. On pourra aussi rajouter l'url des librairies de stratégie à utiliser.
- Doit pouvoir ressortir les résultats de la partie dans un fichier texte
- Implémenter une IA sous forme d'une librairie dynamique
- Si une IA renvoie un mauvais tour, elle passe son tour
- Fournir un Makefile compilant sous Linux.

II) Explication de la démarche



Affichage graphique

Pour réaliser l'interface graphique, nous avons bien entendu utilisé la librairie SDL et plus précisément SDL2. Pour cela SDL est assez simple, il suffit de créer une fenêtre dans laquelle on place un rendu que l'on modifie par la suite.

Les dés

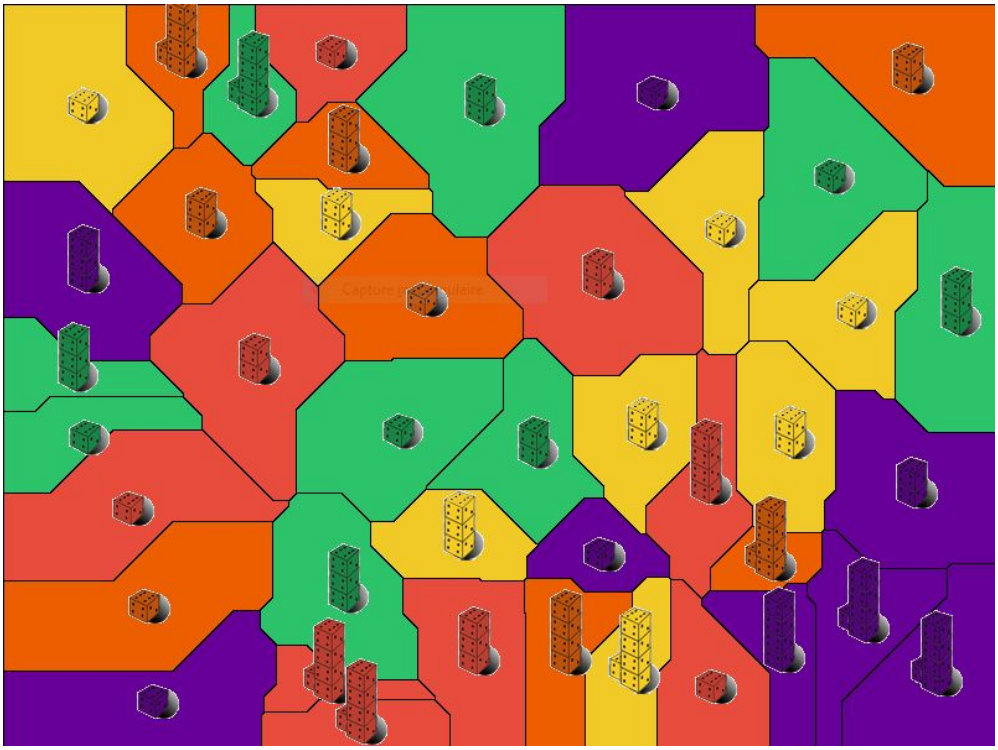
Afin d'afficher les dés nous avons utilisé des images bmp, et pour les avoir de différentes couleurs nous avons créé un jeu d'image de chaque couleurs pour chaque territoire (x8 sachant que le nombre maximum de joueurs dans notre jeu est de 8). Si nous avons utilisé des images bmp et non des png qui sont transparentes et facilitent le changement de couleur des dés ainsi qu'un affichage plus esthétique est pour la simple et bonne raison que SDL ne prends pas en compte les png.

Les lancers de dés sont aléatoire entre 1 et 6 par dé.

La carte

Afin de créer la carte nous avons utilisé la méthode du **diagramme de Voronoï**, cette méthode consiste à placer un certains nombre de points dans un plan appelés germes. A partir de ces germes nous créons des cellules. Chaque cellule renferme un seul germe, et forme l'ensemble des points du plan plus proches de ce germe que de tous les autres. La cellule représente en quelque sorte la « zone d'influence » du

germe. Pour nous un germe est égal au “centre d’un territoire”, et une cellule à un territoire. Le résultat est donc le suivant :



Les IA

Pour la partie que l’ordinateur puisse jouer tout seul nous avons choisi d'utiliser un arbre de recherche permettant ainsi de modéliser l'ensemble des coups possible pour l'ordinateur a un instant donné. Grâce à un arbre de recherche nous avons pu aussi modéliser les coups des adversaires et ainsi prédire Ce qu'il allait jouer dans le futur.

Ce qui est pratique avec cette méthode et que l'on peut définir la profondeur de recherche suivant ce qu'il nous plaît et le temps de calcul que nous voulons consacré à

la recherche. Par exemple avec une profondeur de 1 on va pouvoir jouer plusieurs coups par seconde et avec une profondeur de 3, seulement un coup toutes les 10 secondes car il y a beaucoup plus de combinaison à prendre en compte. Plus la profondeur est importante et plus la qualité du coût choisi sera bonne.

Nous voulions ensuite grâce à un apprentissage à partir d'une base de données de 1000 parties pouvoir avoir une bonne fonction d'évaluation des feuilles de l'arbre de recherche malheureusement notre intelligence artificielle semble avoir des problèmes à finir des parties et donc nous n'avons pas pu étudier cette solution.

Les logs

Pour avoir un historique de la représentation graphique de la map et des coups joués par les joueurs, il nous faut un fichier de log. Pour cela nous avons créé un fichier game.log qui a l'architecture suivante :

Ligne 1 : Nombre de pays n
n lignes suivantes : idCellule, idOwner, coordX, coordY, nbDices

Symbole /Fin Map/ : Fin de la répartition des territoires début de la partie

Symbole /Fin répartition/ : Fin de la répartition des dés, Debut d'un tour
idAttaquant, idCelluleFrom, idCelluleTo, ScoreAttack, ScoreDefense

Symbole /répartition dés/ : fin du tour d'un joueur et début de la remise des dé
idCell (on rajoute un dé a cette cellule)

Symbole ### : ligne suivante -> gagnant de la partie (jusqu'au prochain symbole ###)

III) Architecture du code

Déroulement du main

Dans notre main, nous avons commencé par déclarer certaines variables comme le nombre de joueurs et le nombre de parties à faire. Ensuite on vérifie les arguments qui ont étés passés en paramètre et ainsi que leur nombre. On charge ensuite les éventuelles libraires d'IA qui ont étés passées en paramètre. On initialise les variables d'affichage c'est-à-dire la fenêtre avec son contenu (la carte avec les territoires du jeu, etc…). Se lance alors la partie avec une boucle for bouclant sur le nombre de parties à réaliser.

A chaque partie :

1. On refait une carte et on l'affiche
2. Tant que la fenêtre n'est pas fermée et qu'il n'y a pas de gagnant
    a. On lance le tour d'un joueur
    b. Si c'est un joueur interactif
        i. On le laisse jouer
        ii. Fini son tour
    c. Si c'est une intelligence artificielle
        i. On crée une copie de la carte que l'on envoie à l'IA
        ii. On fait jouer l'IA correspondant
        iii. On vérifie son coup
        iv. On affiche son tour
    d. On vérifie que le joueur n'a pas gagné
        i. Si il a gagné on change une variable qui fait sortir de la boucle
        ii. Sinon on notifie de la fin du tour et on change de joueur
3. Fin du jeu
4. On libère les variables (Dont les libraires)
5. On ferme la fenêtre

Description des fichiers du jeu

Arbitre.c : Gère les fonctions d'arbitrage du jeu, déplacement et combats de dés
Libloader.c : Charge les librairies dynamiques nécessaires aux IA
Log.c : Se charge de l'input et de l'output pour remplir les fichiers de logs
Map.c : Créer et affiche la carte du jeu

Player.c : Gère le tour d'un joueur humain
Renderer.c : Gère le renderer (Sa création et les fonctions liées à celui-ci)
Window.c : Crée la fenêtre et ses objets associées

Description des fichiers de l'IA

Arbre.c : Gestion de l'arbre de recherche
gameIA.c : Coeur de l'intelligence artificielle
interface.c : Fournit les fonctions nécessaires pour que la librairie s'exécute

IV) Difficultés rencontrées

Nous avons rencontré certaines difficultés le long de notre projet, que nous avons soit contournée soit résolue. En l'occurrence voici une liste non exhaustive des problèmes rencontrés et des solutions mises en oeuvre.

Affichage des images avec une transparence sur les territoires : SDL ne prenant pas nativement les images transparentes, il faudrait installer SDL_Image, soit une librairie supplémentaire non acceptée par le correcteur. Nous avons donc créé un jeu d'image bmp afin de pouvoir les afficher nativement avec SDL.

Affichage des scores et des informations de partie : Une fois de plus, SDL ne prend pas en compte l'affichage de caractères nativement, en effet, il faut installer l'extension TTF de SDL pour pouvoir écrire dans la fenêtre, nous nous sommes donc une fois de plus résigné à utiliser des images car le correcteur n'a pas forcément la librairie et ne pourra compiler le programme sans.

Dans la partie intelligence artificielle nous avons rencontré de nombreux problèmes comme des fuites de mémoire intempestives. Ainsi que des problèmes pour débugger l'arbre de recherche, car le C n'est pas un langage qui nous donne de bonne information sur l'origine des problèmes.

V) Conclusion

Pour conclure, nous trouvons que le travail que nous avons réalisé est fonctionnel et agréable à utiliser. Nous avons refait le jeu DiceWars en adaptant notamment la carte avec un visuel graphique différent. Nous avons appris à bien utiliser le langage C, avec les allocations mémoire et les libérations de mémoire. De plus nous avons appris l'utilisation d'une librairie externe et de librairies statiques.

Nous avons retenu de ce projet que le C est un langage très performant mais qui se veut très exigeant notamment avec la mémoire, que ce soit les pointeurs, les librairies, les allocations, les libérations de mémoire.