DIABY Yacouba
MOHAMED-MOUSSA Elmokhtar
KANE Amadou
LE BIAN Raphaël

# PROJET DE PROGRAMMATION : DICEWARS

## I- Introduction

Nous sommes un groupe de 4 étudiants en 1ère année du cycle ingénieur informatique de l'école Polytech Nantes. Dans le cadre de nos études, et plus particulièrement du module de programmation en langage C, nous avons réalisé un projet consistant à reproduire le jeu Dice Wars.

Nous allons dans ce rapport vous présenter la manière dont nous avons abordé ce projet, les problèmes que nous avons rencontrés, les choix que nous avons faits pour enfin conclure sur le résultat final. Nous ne nous étendrons pas ici sur la partie technique concernant le code, étant donné qu'il est lui-même composé de commentaires permettant sa compréhension.

## II- Premier contact avec le sujet

Afin d'aborder ce projet, nous avons eu le même réflexe que la plupart des autres groupes : nous avons testé le jeu. Durant cette phase de test, nous avons pu essayer l'ensemble des fonctionnalités du jeu, ce qui nous a permis d'avoir une première vue globale de ce dernier.

Ainsi, nous avons établi un partage des tâches de la manière suivante : un étudiant était chargé de la partie back-end et un autre de la partie front-end, avec à leur côté un troisième étudiant faisant le relais entre les deux domaines. Le travail du quatrième étudiant consistait à générer une ou plusieurs IA et de les optimiser.
Cependant, il faut noter la participation de tous les étudiants à la résolution des problèmes majeurs que nous avons pu rencontrer.

Suite à cela, nous avons dans un premier temps chercher à produire une interface graphique extrêmement simple, ne reproduisant pas le jeu, mais affichant simplement une fenêtre graphique. L'objectif était de commencer à expérimenter la création d'une carte ou « map ».

D'autre part, nous avons commencé à penser à la modélisation du jeu, à sa manipulation et également à une intelligence artificielle (nous sommes restés sur une IA très simple sur la quasi-totalité du projet, étant donné que cette dernière, malgré sa simplicité, s'est révélée performante).

## III- Création de la map

Dans cette partie, la première question qui s'est posée a été de savoir comment générer une map, et ce de manière aléatoire. Nous avons dans premier temps pensé reprendre un algorithme permettant de générer un graphe aléatoire, pour ensuite générer une map à partir de ce dernier. Étant

donné le nombre très important de ce type d'algorithme, l'idée nous est parue évidente. Or, en creusant un peu, nous nous sommes rapidement rendu compte que ces algorithmes sont complexes, et donc difficiles à implémenter.

Après une courte période de flou à rechercher une nouvelle solution pour générer une map, nous sommes tombés sur un algorithme inverse à notre idée de départ : l'algorithme de Voronoï.

Pour faire simple, cet algorithme prend en paramètre une matrice de la taille de l'écran d'affichage souhaité (par exemple, si on affiche une matrice de 1280*720 pixels, l'algorithme prendra en paramètre une matrice 1280 colonnes et 720 lignes), et le nombre de territoire que l'on veut créer. La première étape de l'algorithme consiste à injecter des « germes » dans la matrice à des positions totalement aléatoire. Il y a autant de germes que de territoire à créer. Suite cela, l'algorithme détermine pour chaque point de la matrice la germe la plus proche. Une fois ces deux étapes faites, on obtient une matrice dans laquelle chaque élément (correspondant à un pixel) appartient à un territoire.

Nous avons ensuite créé le graphe associé à cette matrice : chaque sommet correspond à un territoire, et ces sommets sont reliés entre eux s'ils sont voisins sur la map. C'est à partir de ce graphe que nous avons pu commencer toute la partie backend, que nous allons détailler par la suite. En effet, ce graphe est une modélisation de la map que nous avons appris à manipuler.
Parallèlement à cela, nous avons mis en place une interface graphique afin d'afficher notre map et donc la partie. On y trouve également des également des volets d'affichage contenant le nom de joueur qui doit jouer, les scores et valeurs des stacks (piles) quand il y a une attaque et un bouton pour passer son tour. L'ensemble des détails d'affichages n'ont été ajouté que vers la fin du projet.

Si l'on revient sur la map que nous pouvons générer, nous pouvons voir que cette dernière ne possède pas de « trous » comme dans le jeu original. Nous avons donc accordé un intérêt tout particulier à ce détail, que nous avons régler de la manière suivante :

Étant donné que sur une map, il doit y avoir un chemin entre tous les territoires (ou encore un chemin entre chaque sommet du graphe), nous ne pouvons pas remplacer n'importe quel territoire par un vide. En effet, si on le faisait, on pourrait se retrouver avec deux territoires non-connexes, et la partie serait donc injouable.

Afin donc de pallier ce problème, il nous a fallu déterminer quels sont les sommets du graphe que nous pouvons supprimer. Pour cela, nous avons trouvé un algorithme qui réalise la tâche inverse, à savoir trouver les sommets d'articulations. Une fois les sommets d'articulations trouvés, nous avons alors pu choisir, de manière aléatoire, les sommets (et donc les territoires) à supprimer pour obtenir des vides.

## IV- Initialisation de la partie

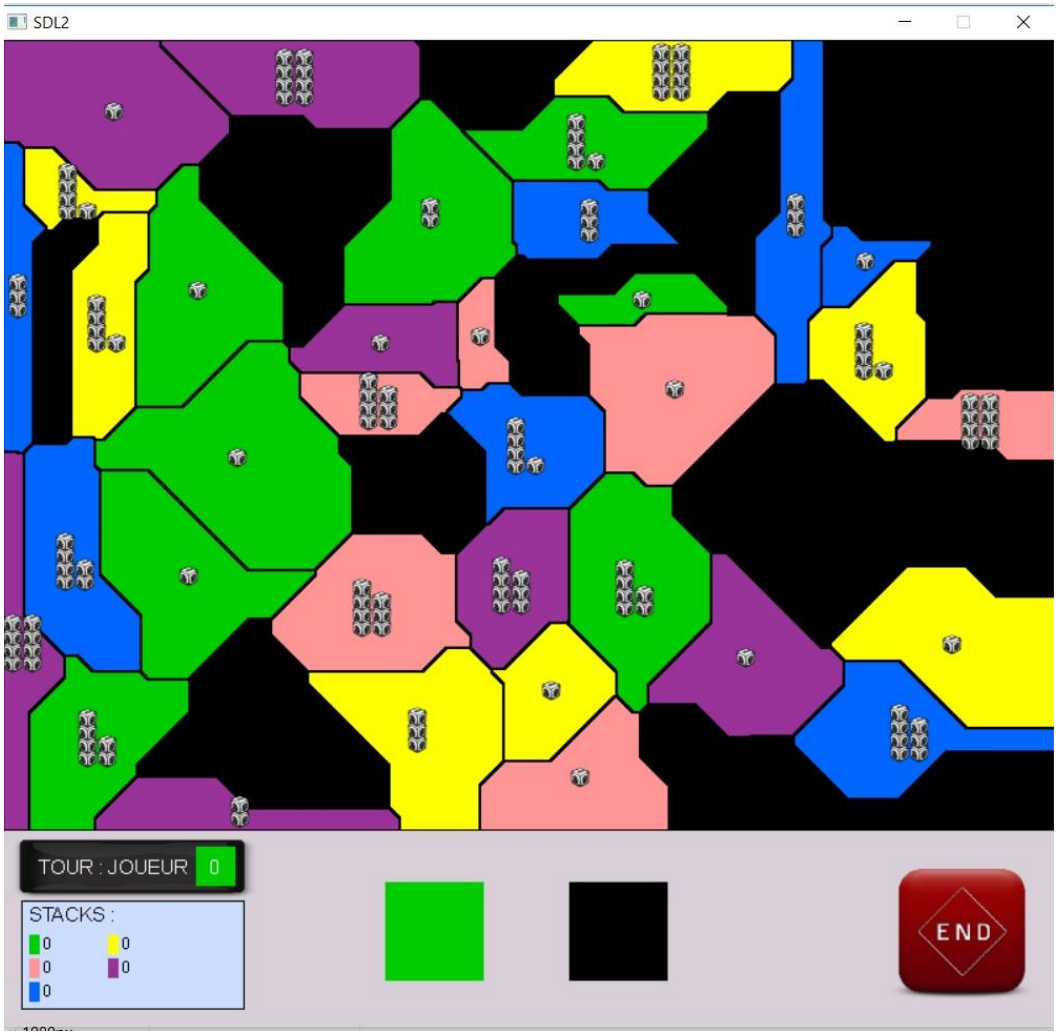
Afin d'initialiser une partie avec une map créée, il faut quels sont les paramètres de lancement du jeu : il comprend le nombre de partie que l'on veut faire, le nombre de joueur ainsi que les

différentes IA (nous reviendrons sur les IA plus tard).

Immédiatement, le premier problème qui se pose est d'attribuer à chaque joueur des territoires. Étant donné que le nombre de territoire à attribuer n'est pas forcément égal à un multiple du nombre de joueur, tous les joueurs n'auront pas le même nombre de territoires. L'ensemble des territoires est donc distribué de manière totalement aléatoire à l'ensemble des joueurs, de manière à ce que la plus grande différence du nombre de territoires entre deux joueurs soit égale à un.

Une fois que nous avons pour chaque joueur des territoires associés, il faut distribuer les dés sur les territoires. La principale condition de la distribution des dés est que tous les territoires doivent au minimum avoir 1 dé, et au maximum 8. Nous avons donc dans un premier temps attribué à chaque territoires un dé, puis ensuite nous leur avons attribué à chacun un nombre aléatoire entier de l'intervalle [0 ;7] (nombre de dés).

## V- Déroulement de la partie




Une fois que la partie est initialisée, nous possédons une map visualisable dans une interface graphique, où les territoires sont attribués aux différents joueurs, et chacun d'eux possèdent un certain nombre de dés. Nous pouvons donc commencer à voir de quoi se compose le tour d'un joueur (une

partie étant une suite de tours). Comme nous l'avons dit précédemment, les joueurs peuvent être des personnes physiques ou bien des IA. Nous allons donc dans un premier temps voir comment nous avons abordé le tour d'une personne physique, puis nous parlerons en détail de nos IA.

Dans un premier temps, nous pouvons voir qu'un tour de jeu se divise en deux phases très simples : attaquer et passer son tour. Alors qu'attaquer est une phase qui elle-même se décompose en plusieurs sous-parties, il suffit de cliquer sur un bouton de l'interface ? Mais comment est gérer ce click ?

Étant donné que nous connaissons chaque pixel de la fenêtre graphique, nous somme capable de récupérer la position du click de l'utilisateur. Suite à cela, il nous est possible de savoir sur quoi il a voulu cliquer (Nous savons également à quelle cellule appartient chaque pixel de l'écran), et donc nous pouvons répondre avec l'action associée. Le fonctionnement est le même pour tout type d'application utilisant une fenêtre graphique.

Maintenant que l'on sait comment interpréter le click de l'utilisateur pour qu'il passe son tour, on peut appliquer ce principe à la phase d'attaque.

Il faut donc récupérer les cliques et faire une série de vérification. En effet pour attaquer, le joueur doit sélectionner une cellule lui appartenant avec plus d'un dès avant de cliquer à nouveau sur une cellule adjacente ne lui appartenant pas. Lorsque le joueur a fini ses coups il a un bouton pour indiquer la fin de son tour. A noter qu'à chaque coup, le fichier log est mis à jour: il y est inscrit les id du joueur attaquant, du joueur attaqué et, du joueur gagnant sur une ligne séparée par des espaces.

Voici le fonctionnement global pour une personne physique. Qu'en est-il pour notre intelligence artificielle?

## VI- Intelligence artificielle

Tout d'abord nous avons voulu commencer par une IA simple. La stratégie était la suivante:

L'IA commence par chercher toutes les cellules appartenant au joueur qui a la main, avant de chercher parmi les voisins de ces derniers la cellule "ennemie" qui présent la différence de dès la plus avantageuse.

Dans le cas d'une égalité, si le joueur possède des dès dans sa stack et dans ce cas il tente sa chance, sinon il ne joue pas.

Ensuite, il fallait créer des fonctions capables de faire la liaison entre l'animation graphique et les coups de l'IA. C'est à dire faire appel aux fonctions déjà implémentées pour un jeu entre personnes physiques.

Nous avons également implémenté une autre IA que nous n'avons pas pu tester en raison de bugs dans le reste du programme que nous nous devions de corriger, qui plus est, connaissant déjà l'efficacité de la première.

L'idée consiste à rechercher la plus grande composante connexe appartenant au joueur faisant le coup. En cas d'égalité, celle qui présente la différence de dès la plus avantageuse par rapport aux voisins à attaquer, est retenue.

Ensuite, la deuxième phase consiste à attaquer la cellule adjacente à l'ensemble présentant le meilleur avantage en terme de nombre de dés.
Dans le cas échéant cette IA devait tout simplement attaquer la cellule présentant le plus grand écart de dés favorable au joueur attaquant.

## VII- Bilan

Ce projet nous a permis de continuer à travailler des points abordés lors du premier projet d'algorithmique tant au niveau de l'organisation des tâches et du respect des délais d'exécution de celles-ci, de la mise en commun au sein d'un groupe (maîtrise de gitlab), qu'au niveau algorithmique malgré le bug présent sur notre version finale( une fuite mémoire fait planter le programme lorsqu'il tourne pendant longtemps.)
Mais également, nous avons pu aborder d'autres points, en particulier la manipulation de l'interface graphique qui a représenté une part importante du projet et qui n'avait pas été présente dans le précèdent

Par ailleurs, nous avons su tirer profit de notre première expérience afin de ne pas répéter les mêmes erreurs lors de la réalisation de ce projet . Ainsi nous étions alertés sur l'importance du découpage en étapes du projet et de la difficulté de prévoir les durées de celles ci. De ce fait, nous avons multiplié les réunions afin de mettre en commun l'avancement de chacun dans le but d'avancer dans la même direction mais également de résoudre ensemble les problèmes majeurs tels que les choix de structures.