Dollar Game

Un jeu réfléchi sur les graphes

Rapport de projet T.E.R.
Projet Informatique - HLIN601

Étudiants :

Corentin Teyssier
Rayan Derrouiche
Steven Lamerly
Valentin Peron
Allan Crista

Encadrant : Mr Stéphane Bessy

Année : 2020-2021

Remerciements

Nous tenons à remercier notre encadrant, monsieur Stéphane Bessy pour l'aide précieuse qu'il nous a apporté. Son accompagnement, sa bienveillance et ses conseils précieux tout au long de ce projet nous ont permis de le porter à son état actuel.

Table des matières

Partie 1

Introduction

Dans le cadre du TER de notre troisième année à la faculté des sciences de Montpellier nous avons pris le sujet concernant le Dollar Game, un mini jeu consistant à partager de l'argent entre divers individus afin que ces derniers aient en fin de partie un solde positif ou nul. Le but de ce projet est la réalisation du jeu lui même et de l'implémentation de stratégies afin d'approcher la stratégie de résolution la plus efficace.

Le groupe de projet est composé de cinq personnes, Allan CRISTA,Corentin TEYSSIER, Rayan DERROUICHE, Valentin PERON et Steven LAMERLY. Nous sommes encadré par Mr Stéphane BESSY.

1.1 Présentation du sujet

Le Dollar Game est un jeu de solitaire qui se joue sur un graphe. Chaque sommet du graphe va recevoir un certain nombre de dollars, positif ou négatif. A chaque tour de jeu un sommet est choisi et donne 1 dollar à chacun de ses voisins. Le but du jeu étant d'obtenir une configuration où chaque sommet possède une valeur positive ou nulle. Ce jeu a été introduit par M. Baker en 2007.

1.2 Approches

Le projet se scinde donc en deux parties distinctes :

- La création d'un jeu de Dollar Game avec interface graphique
- L'implémentation de différentes stratégies afin d'analyser les résultats pour tendre vers la stratégie de résolution la plus optimale

Nous avons donc décidé de travailler ensemble sur le développement d'une base nous permettant de créer des parties de Dollar Dame pour ensuite répartir le travail en deux pour réaliser les deux objectifs cités précédemment.

1.3 Cahier des charges

Notre projet s'est donc déroulé en 5 parties, que nous vous présentons ci-dessous :

- Tout d'abord, l'étude du point de vue mathématique et algorithmique du jeu. Il s'agira ici d'appréhender les différentes définitions et propriétés du Dollar Game. Pour nous aider, notre encadrant, Mr Stéphane BESSY, nous a fourni un rapport de master sur le sujet intitulée "A graph theoritical approach to the dollar game problem", rédigée en 2020 par Rachel KOCH à l'université Virginia Commonwealth University
- Ensuite, la deuxième étape de notre projet consistera en la création d'une interface graphique permettant aux utilisateurs et visualiser et d'interagir avec le jeu en vue de résoudre le Dollar Game. Le choix du langage et des technologies utilisées étant libre, nous avons décidé de développer l'interface visuelle du jeu en Html et CSS, et le jeu du Dollar Game en Javascript.
- La troisième étape consistera à implémenter l'algorithme de résolution sur les arbres donnés dans le mémoire ainsi qu'un algorithme permettant de générer des arbres.
- Une fois ces étapes réalisées et effectives, nous pourrons élaborer et implémenter différentes stratégies en vue d'optimiser la résolution des graphes. Ce qui nous permettra par la suite de déterminer la stratégie de résolution la plus efficace et la plus otpimale pour résoudre le jeu du Dollar Game, et de comparer ces différentes stratégies entre elles.
- Enfin, la dernière étape étant la rédaction du mémoire en LateX

Partie 2

Technologies utilisées

2.1 Langages

Le choix du langage étant libre nous avons donc réfléchi à la meilleure solution qui nous permettrait de réaliser notre projet. Au fil des discussions, nous avons choisi de réaliser et développer une application web. En effet, une application web est idéale pour les jeux légers comme le Dollar Game. Cela nous permettra également de pouvoir héberger le jeu sur un serveur afin que les utilisateurs puissent accéder au jeu simplement et rapidement au site via leur navigateur web, et sur différents supports (ordinateur, tablette et mobile). Nous avons donc développé notre projet avec HTML pour le contenu, CSS pour le style et Javascript pour le moteur du jeu.

2.2 Outils

Tout au long du projet nous avons utilisé différents outils afin de faciliter nos échanges et le développement du projet.

Étant donné que notre projet a été développé sous forme d'application web, l'outil principal a été l'éditeur de texte Sublime Text 3. Il est très pratique, a beaucoup de fonctionnalités et est très performant. Ensuite bien entendu pour interpréter notre code il fallait un navigateur web. Nous avons donc utilisé principalement Google Chrome mais également Mozilla Firefox pour vérifier que notre code fonctionne sur différents navigateurs.

Nous avons utiliser github.com pour le travail collaboratif afin de faciliter le partage des documents et de notre code.D'autant plus que la situation sanitaire nous empêchait de nous voir en présentiel régulièrement.

Nous communiquions à l'aide d'un groupe Whatsapp, créé spécialement pour ce projet. Nous pouvions avec ce groupe par exemple nous prévenir quand nous avions avancé sur le projet en proposant une nouvelle version sur github.com mais également pour se fixer des rendez-vous pour nos réunions entre nous ou pour les comptes rendus hebdomadaires avec notre encadrant.
Nous faisions ces réunions à l'aide du logiciel de visiophonie Zoom.

Partie 3

Approche théorique du dollar game

3.1 Theorème et définitions

A l'aide d'un mémoire réalisé et rédigé par Rachel KOCH en 2020 dans la Virgnia Commonwealth university, intitulé "A graph theoritical approach to the dollar game problem" nous avons pu en apprendre plus sur le Dollar Game afin de mieux appréhender les définitions et propriétés simple du Dollar Game.

Établissons des maintenant une définition simple d'un partie de dollar game : Soit G un graphe. Soit $D(v)$ la valeur en dollar pour chaque sommet v. On dit quun sommet a $D(v)$ dollars et on appelle une configuration du graphe donné $D(G)$.

A la base le Dollar Game possède deux types de mouvement : le lending move et le borrowing move. Le premier permet de donner un dollar a chacun des voisins du sommet choisi et le second permet de prendre un dollar a chacun des sommets voisins du sommet choisi. Cependant il a été prouvé dans le mémoire qu'un lending move pouvait être effectué grâce à une série de borrowing move, ainsi on n'utilisera que le lending move dans le cadre de ce projet.

On apprend également que les mouvements sont commutatifs, c'est a dire que l'ordre n'a pas d'importance. On obtient le même résultat si l'on clique, par exemple, sur les sommets dans l'ordre A->B->C que dans l'ordre C->A->B.

Pour finir ce mémoire nous avons appris des définitions et théorèmes sur la solvabilité ou non d'une partie. Assez logiquement on sait qu'un montant de dollars total sur une partie est nécessairement positif ou nul pour que la partie soit gagnable, cependant ce n'est pas une condition suffisante. C'est grâce au nombre de Betti que nous allons pouvoir déterminer si une partie est gagnable ou pas, il se calcule en soustrayant le nombre de sommets plus un au nombre darêtes.

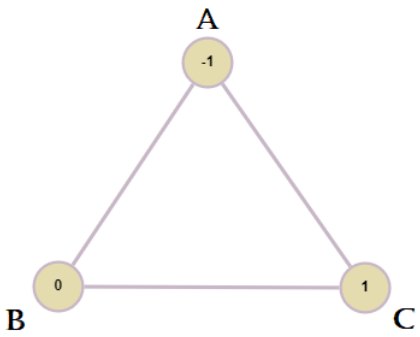
$$Betti = E - V + 1$$

Theorem 1 *Si la somme total de dollars d'une partie $D(G)$ est supérieur ou égal au nombre de Betti de G, alors la partie est gagnable. C'est sur cette définition que nous débuterons notre projet afin de mieux appréhender la création des différentes stratégies.[1]*

La preuve de ce théorème repose sur des outils mathématiques profonds (géométrie algébrique) et actuellement aucune preuve algorithmique 'simple' de ce théorème n'est connue. C'est cette absence de preuve qui motive l'étude des stratégies dans la suite de notre projet.

3.2 Exemple théorique

La première idée que l'on pourrait avoir en voyant un Dollar Dame est que si la somme des sommets est positive ou nul, alors la partie sera forcément gagnable. Nous avons donc essayé de trouver un graphe qui remplie la condition de la somme des poids des sommets positives ou nulles mais qui n'est pas gagnable et avons tenté de le prouver.



Soit le graphe G à 3 sommets suivant dont la somme des poids est égal a 0. D'après le lemme vu précédemment nous savons que les coups sont commutatifs, donc peut importe l'ordre d'exécution, le résultat sera la même. On considère donc que l'on joue 'a' fois sur le sommet A, 'b' fois sur le sommet B et 'c' fois sur le sommet C quel que soit l'ordre. On considère donc qu'a la fin des coups, quel qu'ils soient, la partie est gagné si tout les sommets sont à 0 ou plus. Nous pouvons donc traduire cela en une équation pour chaque sommet, suivant la forme : poids de départ - 2*coup sur le sommet + coup sur voisin 1 + coup sur voisin 2. Ce qui nous amène aux inégalités suivantes :

$$\begin{aligned} -1 - 2a + b + c &\geqslant 0 \\ 1 - 2c + a + b &\geqslant 0 \\ 0 - 2b + a + c &\geqslant 0 \end{aligned} \tag{3.1}$$

En les simplifiant ou obtient ensuite les inégalités suivantes :

$$\begin{aligned} b + c - 2a &\geqslant 1 \\ a + b - 2c &\geqslant -1 \\ a + c - 2b &\geqslant 0 \end{aligned} \tag{3.2}$$

On simplifie ensuite en ajoutant les inégalités 1 et 2 et en multipliant la 3 par -1 :

$$\begin{aligned} 2b - a - c &\geqslant 0 \\ 2b - a - c &\leqslant 0 \end{aligned} \tag{3.3}$$

Ce qui nous amène à l'égalité suivante :

$$\begin{aligned} 2b = a + c &\geqslant 0 \\ b = \frac{a+c}{2} & \end{aligned} \tag{3.4}$$

On utilise maintenant cette valeur de b sur les deux premières inégalité obtenue :

$$\begin{aligned} &(1) \\ b + c - 2a &\geqslant 1 \\ \frac{a+c}{2} + c - 2a &\geqslant 1 \\ a + c + 2c - 4a &\geqslant 2 \\ 3c - 3a &\geqslant 2 \\ \\ &(2) \\ a + b - 2c &\geqslant -1 \\ a + \frac{a+c}{2} - 2c &\geqslant -1 \\ 2a + a + c - 4c &\geqslant -2 \\ 3a - 3c &\geqslant -2 \\ 3c - 3a &\leqslant 2 \end{aligned} \tag{3.5}$$

Ce qui nous amène à notre équation finale, qui nous permet de prouver que cette partie n'est pas gagnable :

$$\begin{aligned} 3c - 3a &= 2 \\ c - a &= 2/3 \\ a \vee c &\notin N \end{aligned} \tag{3.6}$$

On a donc un résultat incohérent car nous ne pouvons pas effectuer des fractions de coup, par conséquent cette partie est bien impossible. On obtient le même résultat si l'on utilise le nombre de Betti. En effet le nombre de Betti de cette partie est égal a 1 alors que la somme des poids total est égal a 0 ce qui montre bien l'impossibilité de cette partie. Résoudre l'inégalité (3.1) revient à trouver la solution à un "programme linéaire en nombre entier" (PLNE). Pour chaque partie, on peut écrire un tel ensemble d'inéquations. Malheureusement il n'existe pas de méthode algorithmique efficace pour résoudre les PLNEs.

Partie 4

Conception du Dollar Game

4.1 Les étapes de conceptions

Concevoir un jeu, même relativement modeste comme le Dollar Game nécessite de l'organisation. C'est pourquoi nous y sommes allés par étapes, nous avons découpé le travail en plusieurs petites parties. Voici ci-dessous les étapes principales.

4.1.1 Génération de graphes aléatoires

Premièrement, pour pouvoir jouer au Dollar Dame il nous faut un graphe. C'est pourquoi nous avons commencé par créer une fonction permettant de créer des graphes aléatoires d'une taille de sommets donnée en console. Avec ce que nous avons observé sur les jeux de Dollar Game existants et sur les conseils de notre encadrant nous avons choisi de nous limiter à des graphes de taille maximum 3x3 soit 9 sommets disposé sur une grille carré. Les 9 sommets sont reliés par une arrête à chacun de leurs voisins. Pour que ce soit jouable il faut évidemment que le graphe soit connexe. Pour palier ce problème de connexité nous avons décidé que le sommet du milieu (le numéro 4 de coordonnée [1,1]) qui est voisin de tous les autres sommets soit toujours présent dans le graphe.
Pour représenter le graphe dans le code nous avons décidé d'utiliser un tableau à deux dimensions. Le tableau est donc composé d'autant de "sous-tableau" que de sommets. Les sous-tableaux sont composés tout d'abord du numéro du sommet (de 0 à 8), puis du poids du sommet et enfin de tout ses voisins. Voici un exemple de représentation d'un graphe à 3 sommets :

[[3, -4, 4, 7],[4, 3, 3, 7],[7, 2, 3, 4]]

On voit donc qu'il y a dans ce graphe le sommet numéro 3 de poids -4 et il a 4 et 7 comme voisins, puis il y a le sommet numéro 4 de poids 3 et il a 3 et 7 comme voisins etc. La fonction prend en paramètre un entier qui définit le nombre de sommets du graphe, puis les sommets sont choisis aléatoirement à l'aide d'une fonction random. Les poids sont aussi définis de manière aléatoire sauf le dernier. En effet si ils étaient tous défini de manière aléatoire il n'est pas garanti que le graphe soit résolvable. Les premiers sont donc définis aléatoirement entre [- le nombre de sommets, + le nombre de sommets] et le dernier est défini de sorte que la somme total de tout les sommets soit égale au nombre de Betti.

4.1.2 Affichage du graphe

Ensuite, une fois le graphe créé, il faut l'afficher sur la page avec ses arêtes et le poids sur chacun de ses sommets. Nous avons commencer par les sommets. Tout d'abord nous avons implémenter 9 paragraphes dans le code HTML, un pour chaque sommet puis avec le CSS nous les avons disposés en forme 3x3. Ensuite une fonction javascript récupère le graphe créé et donne le bon poids sur chaque paragraphe du sommet correspondant. Si le sommet n'existe pas car le graphe à un nombre de sommet < 9, alors on le laisse tout simplement vide.
Pour les arrêtes nous avons d'abord créé une fonction *draw(xabs,xord,yabs,yord)* qui prend en paramètre les coordonnées de deux points x et y pour tracer un trait entre ces deux points. Nous avons donc placé tous les traits, un pour chaque arrête. Ensuite nous vérifions pour chaque sommet quels sont ses voisins et nous affichons les bons traits avec les bonnes coordonnées. Si deux sommets ne sont pas voisins nous n'affichons donc pas le trait.
A ce stade, nous avons donc une fonction qui crée un graphe aléatoire de taille donnée et l'affiche avec ses arêtes et avec les poids de ses sommets. En actualisant la page nous pouvons donc voir graphiquement les différents graphes générés.

4.1.3 Détection des clics

Maintenant, pour que notre Dollar Game soit jouable il faut que nous puissions cliquer sur les sommets. Pour ce faire, nous avons créé une variable pour chaque sommet dans le DOM. Ensuite à l'aide de la fonction *onclick()* associée à chacune des variables, quand l'utilisateur clique sur un des sommets, nous pouvons exécuter une fonction. Le jeu reconnaît maintenant les clics et sait sur quel sommet nous cliquons.

4.1.4 Incrémentation et Décrémentation

Pour que le jeu soit enfin jouable il nous reste à créer notre fonction *click(numsommet)*. Cette fonction prend en paramètre le numéro du sommet cliqué. Elle va faire deux choses, tout d'abord elle appelle la fonction *décrémente(numsommet)* qui va récupérer le nombre de voisins du sommet cliqué, puis décrémenter de ce nombre le sommet cliqué. Ensuite, avec une boucle for elle va pour chacun des voisins appeler la fonction *incremente(numsommet)* qui va tout simplement incrémenter de 1 les sommets.

4.1.5 Divers options

Pour rendre le jeu plus attrayant nous avons rajouté certaines fonctionnalités.
Tout d'abord une fonction permettant de détecter quand le jeu est résolu. Cette fonction *gagner()* est appelée après chaque clic et vérifie si le poids de chaque sommet est supérieur ou égale à 0, si c'est le cas nous félicitons le joueur et lui proposons de rejouer.
Pour rendre le jeu plus clair nous avons implémenté une fonction permettant de colorer les sommets. Elle est appelée pendant l'affichage du graphe et après chaque incrémentation et décrémentation. Elle colore en rouge le poids du sommet si il est négatif et en noir si il est positif. Ce qui rend le jeu plus agréable pour le joueur, qui peut ainsi repérer les sommets de poids négatifs plus facilement.
Résoudre un Dollar Game en 10 coups ou en 150 coups n'est pas la même performance. C'est pourquoi il nous paraissait important de rajouter un compteur de coups. Pour cela rien de plus simple il suffit à chaque clic de l'utilisateur d'incrémenter de 1 le compteur.

Nous avons également laissé à l'utilisateur le choix du nombre de sommets qu'il veut pour son graphe aléatoire. Pour cela nous avons dû faire un menu.

4.1.6 Le menu

L'envie de faire un menu dans notre jeu nous est vite venue à l'esprit. Nous avons donc fait une page web spécialement pour le menu. Ce dernier se compose de différents boutons qui redirigent vers les pages correspondantes. Il y a un bouton d'aide qui explique les règles du jeu, un bouton pour jouer au dollar game avec un graphe aléatoire de taille donnée, un bouton pour accéder aux différents niveaux et un boutons pour tester les différentes stratégies implémentées. Nous avons mis aussi un bouton retour dans le jeu pour que l'utilisateur puisse revenir au menu si il le souhaite.

4.2 Problème de conception

Le Dollar Game est un jeu simple et donc relativement modeste à développer. Néanmoins nous avons fais face à quelques problèmes, qui n'étaient certes pas très importants mais qui nous ont fait perdre quand même pas mal de temps. En effet, dans le développement informatique nous passons beaucoup de temps à résoudre des problèmes et fixer des bugs.
Voici un exemple de problème que je vais vous détailler. Le problème s'est produit lorsque nous voulions cliquer sur un sommet. Nous mettions des évènements sur les sommets pour qu'au moment du clic sur un sommet, un message s'affiche en console. Sauf que le message ne s'affichait jamais, cela veut dire que le navigateur ne détectait jamais les clics sur les sommets. Sur une page test cela fonctionnait mais pas sur notre jeu. Le problème est en faites au niveau de l'affichage des arrêtes. En effet, les arrêtes sont dessinées sur un canevas transparent qui recouvre toute la grille ! Donc quand nous cliquions sur une arrête, nous cliquions en réalité sur le canevas transparent. Nous avons donc dû rendre ce canevas insensible et impointable aux clics grâce à la propriété CSS "pointer-events : none ;". Du coup nous pouvions enfin cliqué sous le canevas et donc sur les différents sommets. Comme le canevas est transparent nous n'avions pas vu tout de suite que l'on cliquait en réalité sur celui-ci et non sur les sommets. Ceci est un des quelques petits soucis qui nous ont fait perdre du temps.

Partie 5

Stratégies de résolution du dollar game

5.1 Réflexion sur les stratégies

Le but de cette partie est donc de pouvoir tester différentes stratégies sur un grand nombre de graphes afin de pouvoir approcher la stratégie qui résout le plus de graphes avec le moins de coups possibles. Nous devions donc développer des méthodes permettant de générer un grand nombre de graphes aléatoires afin de tester un certain nombre de fois chaque stratégie dessus. Dans cette optique nous avons donc réfléchi en premier à l'aspect des graphes aléatoires qui composeront notre échantillon de test. Le but étant de pouvoir générer des graphes en fonction de certains critères comme le nombre d'arêtes ou la densité sans pour autant diriger trop l'aléatoire pour obtenir des résultats plus cohérent.

5.2 Création d'une partie de dollar game aléatoire

5.2.1 Génération du graphe

Le squelette de l'interface graphique étant déjà développé au moment de commencer cette partie nous avons du réfléchir au maintient ou pas de la méthode actuelle de génération de graphe. Nous sommes vite arrivés à la conclusion que la méthode était efficace pour l'interface graphique mais avait peu de pertinence pour la partie stratégie. Nous avons donc orienté la génération de graphe vers quelque chose de plus libre et modulable tout en gardant la structure en matrice pour représenter le graphe. Nous avons donc gardé le nombre de sommets en paramètres cependant le graphe n'était plus restreint a la grille 3x3 utilisée précédemment. Cette fois ci le sommet 1 pourrait avoir une arrête potentielle avec tout les autres sommets. Pour ce faire nous avons utilisé le degré comme second paramètre. Compris entre 0 et 100 il représente la probabilité que chaque arrêtes existe, ainsi lors de la création d'un graphe, pour chaque couple de sommet on génère un nombre aléatoire entre 0 et 100 et si il est inférieur au degré, on ajoute l'arrête au graphe. Pour finir afin d'éviter tout problème, on vérifie la connexité du graphe pour assurer que la partie soit cohérente. On utilisera l'algorithme composante afin de vérifier la connexité.

5.2.2 Remplissage du graphe

Vient maintenant la question du remplissage du graphe afin de simuler une partie de dollar game. L'idée était de trouver une méthode qui remplirait de manière aléatoire la partie sans pour autant trop 'guider' l'aléatoire. La grande problématique était de trouver un algorithme qui réparti les poids de manière assez homogène, c'est-à-dire que l'on n'obtienne pas, par exemple, de sommets avec une valeur de -75, sans trop diriger la chose. Pour rappel on doit obtenir un poids total supérieur ou égal au nombre de Betti, mais dans le cadre de ce projet nous allons utiliser des graphes dont le poids total est égal au nombre de Betti.

La première solution que nous avions trouvé consistait à donner à chacun des sommets un poids compris entre $-n$ et n, n étant le nombre de sommet, puis de compléter le dernier sommet afin de donner au graphe un poids égal au nombre de Betti. Cependant nous nous sommes vite aperçus que cette méthode donnait des graphes trop inconstants, ce qui faussait donc les résultats.

Nous avons donc opté pour une nouvelle stratégie de répartition du graphe. Cette fois-ci nous avons attribué des valeurs comprises entre -(moyenne+4) et moyenne+4, la moyenne étant l'arrondi du nombre de Betti divisé par le nombre de sommets. Cette fois-ci pas de remplissage du dernier sommet différent, cependant en fonction du poids total du graphe nous ajustions les valeurs de certains sommets aléatoires. En effet si le poids total du graphe était supérieur au nombre de Betti nous devions retirer du poids de certains sommets, de ce fait nous avons retiré de manière aléatoire du poids à certains sommets afin que le poids total du graphe soit égal au nombre de Betti.

Nous voila donc avec une méthode permettant de générer un graphe aléatoire de taille E et de degré D, de manière aléatoire et homogène. Nous allons donc pouvoir commencer à implémenter les stratégies afin de les faire tourner sur un grand nombre de parties de dollar game.

Algorithm 1 Création d'une partie de dollar game

Input : nombre de sommet n et degré d

```
graphe ← []
for i=0 ;i<n ;i++ do
    graphe[i] ← [i, 0]
end for
for i=0 ;i<n ;i++ do                                   ▷ Initialisation du graphe
    for j=i+1 ;j<n ;j++ do
        rand ← randomInt(1, 100)
        if d>rand then                                                    ▷
            graphe[i].push(j)
            graphe[j].push(i)
        end if
    end for
end for
if !estConnexe(graphe) then                            ▷ Test de connexité du graphe
    return generation_graphe(n, d) ;
end if
nbrBetti ← nombreBetti(graphe)                         ▷ Calcul du nombre de Betti
moyenne ← round(nbrBetti/taille)                       ▷ Calcul de la moyenne par sommets
total ← 0
for i=0 ;i<graphe.length ;i++ do                       ▷ Remplissage du graphe
    val ← randomInteger(−(moyenne + 4), moyenne + 4)
    graphe[i][1] ← val
    total+ = val
end for
if total<nbrBetti then                                 ▷ Ajustement des poids du graphe
    for i=0 ;i<nbrBetti-total ;i++ do
        graphe[getRandomInt(n)][1] + +
    end for
else if total>nbrBetti then
    for i=0 ;i<total-nbrBetti ;i++ do
        graphe[getRandomInt(n)][1] − −
    end for
end if
```

5.3 Stratégies détaillé

5.3.1 Liste des stratégies

Vint à ce moment là du projet une phase de réflexion afin de trouver un maximum de stratégies réalisables afin de pouvoir obtenir les résultats les plus pertinents et les plus optimisés. Nous avons donc trouvé 9 stratégies et nous avons créé une méthode pour chacune d'entre elles. L'idée étant ensuite de pouvoir déterminer le taux de réussite d'une stratégie appliquée n fois sur m sommets.

1. Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prend le premier par ordre des numéros de sommet

2. Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité, prend le premier trouvé

3. Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité prend celui au plus grand degrés

4. Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité prend celui au plus petit degrés

5. Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre le sommet au plus grand degré

6. Trouve le numéro du sommet qui a le plus faible poids cumulé de ses voisins quel que soit sont poids

7. Trouve le numéro du sommet qui a le plus faible poids cumulé de ses voisins quel que soit sont poids, si il y a égalité prend le plus grand

8. Choisis un sommet aléatoire parmi les positifs

9. Trouve le numéro du sommet étant le voisin le plus grand du sommet le plus petit

5.3.2 Stratégie 2 détaillé

Afin de mieux comprendre le fonctionnement d'une stratégie, voici ci-dessous l'algorithme représentant la stratégie numéro 2 en pseudo-code :

- La variable max représente la valeur du plus gros sommet rencontré, il est initialisé a la valeure du premier sommet.

- La variable maxsommet représente le numéro du sommet de poids le plus élevé. Il faut bien distinguer numéro et poids, le numéro est le nom du sommet et le poids est sa valeur en dollar.

- La variable sommevoisinsmax représente la somme des poids des voisins du sommet le plus grand.

- La variable grapheCourant[][] représente le graphe lui-même.

Après l'initialisation des variables, l'algorithme va boucler sur tous les sommets (L3). Pour chaque sommet il va d'abord tester si le poids du sommet courant est égal au poids du sommet max (L4), si c'est le cas on calcule la somme des voisins du sommet courant (L6-8) et on la compare a la somme des voisins du sommet max (L9). Si le somme des voisins du sommet actuel est supérieur a la somme des voisins du sommet max, le sommet courant devient le sommet max (L10-12). On a donc gérer ici la clause 'si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité, prend le premier trouvé' de la stratégie 2. On doit ensuite traiter le cas ou le poids du sommet courant est strictement supérieur au poids du sommet max. Dans ce cas là, pas de test de voisinsmax, on remplace directement le sommet max par le sommet courant (L15-17). Pour finir on actualise la valeur de sommevoisinsmax avec les voisins du sommet courant qui est le nouveau sommet max (L18-20). On retourne maxsommet qui est le numéro du sommet trouvé par la stratégie 2 (L23).

Algorithm 2 Stratégie numéro 2

```
1 : max ← −100000 ;
2 : sommevoisinsmax ← 0 ;
3 : for i=0 ;i>grapheCourant.length ;i++ do
4 :     if grapheCourant[i][1] == max then
5 :         sommevoisins ← 0 ;
6 :         for j=2 ;j<grapheCourant[i].length ;j++ do
7 :             sommevoisins ← grapheCourant[position(grapheCourant[i][j])][1] ;
8 :         end for
9 :         if sommevoisins < sommevoisinsmax then
10 :            max ← grapheCourant[i][1] ;
11 :            maxsommet ← grapheCourant[i][0] ;
12 :            sommevoisinsmax ← sommevoisins ;
13 :        end if
14 :    else if grapheCourant[i][1] > max then
15 :        max ← grapheCourant[i][1] ;
16 :        maxsommet ← grapheCourant[i][0] ;
17 :        sommevoisinsmax ← 0 ;
18 :        for j=2 ;j<grapheCourant[i].length ;j++ do
19 :            sommevoisinsmax+ = grapheCourant[position(grapheCourant[i][j]) ;
20 :        end for
21 :    end if
22 : end for
23 : return maxsommet ;
```

5.4 Comparaison des stratégies

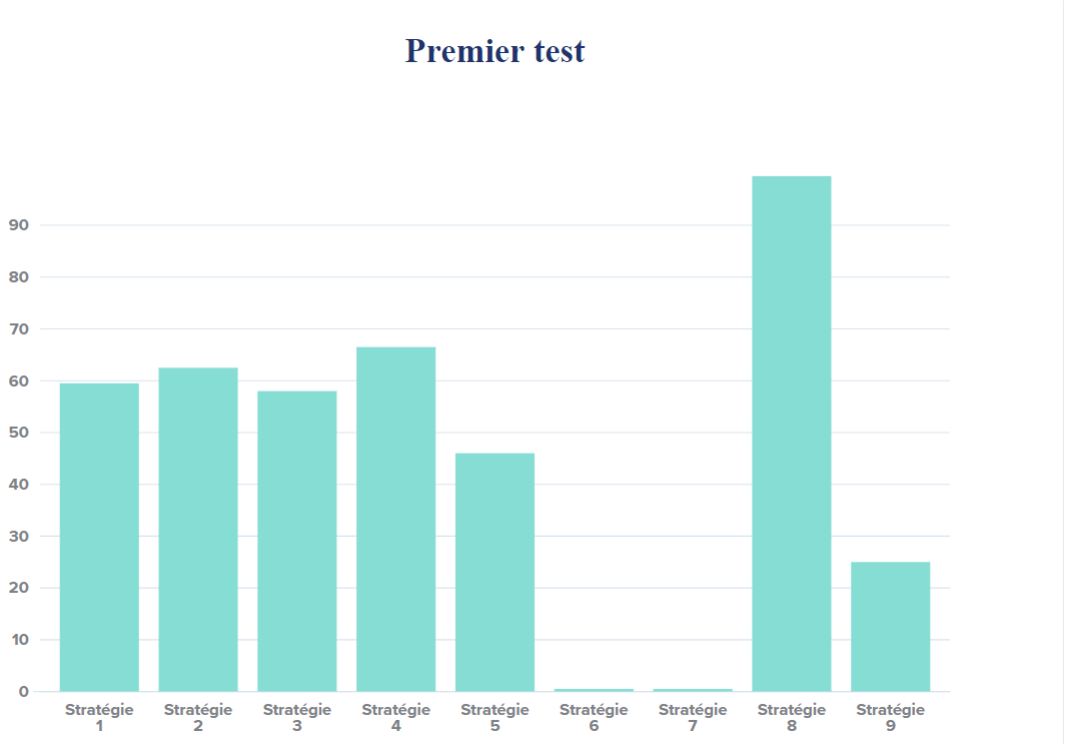
Maintenant que nous avons implémenté des stratégies et que nous pouvons les tester sur un grand échantillon de parties de Dollar Game aléatoires, nous allons pouvoir étudier les meilleurs stratégies et l'impact des différents facteur (degré, nombre de sommet, nombre de coups) sur les résultats. Pour commencer nous allons analyser une situation précise pour avoir un premier exemple. Le rappel des stratégies est fait en annexe du rapport.

5.4.1 Premier exemple

Pour ce premier exemple nous allons essayer de déterminer la meilleur stratégie dans un cadre donnée. Les paramètres de ce premier test sont les suivants :

- 8 sommets
- degré de 0.3
- 10000 graphes
- 1000 coups max

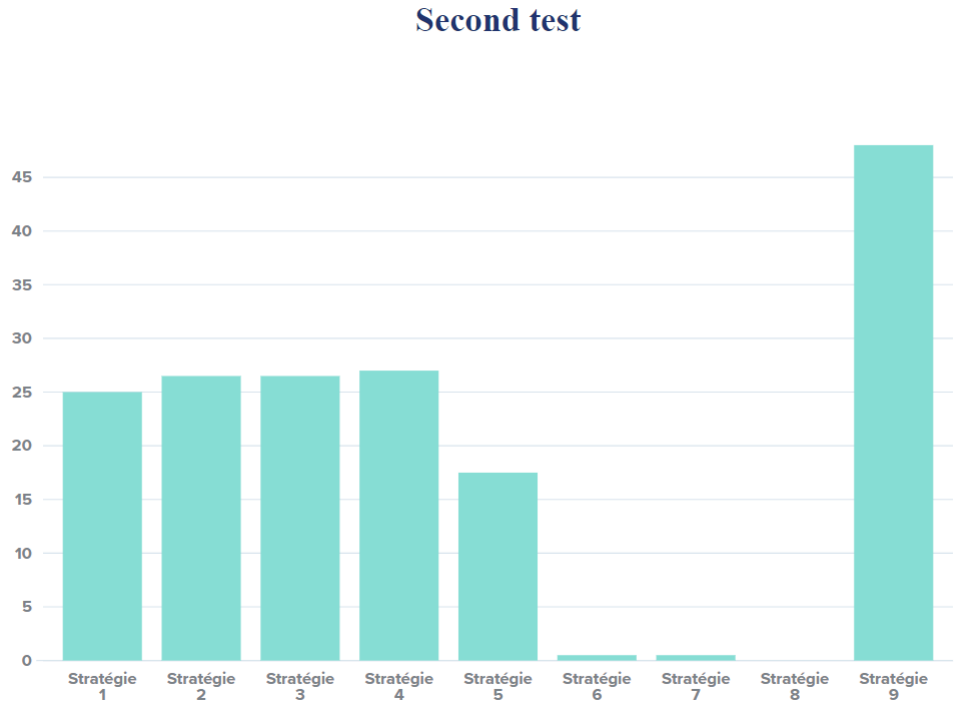
Nous allons donc tester chaque stratégies sur 10000 graphes aléatoires de 8 sommets avec un degré de 0.3 sur 1000 coups au maximum, et obtenir le taux de parties finies. En réalité le nombre de graphe n'influe pas sur le résultats mais permet d'obtenir des valeurs bien plus précise.



Sur ce premier échantillon de test on observe que la stratégie 8 possède un taux de réussite extrêmement élevé (99.5%). Ce résultat est à prendre avec des pincettes car cette stratégie sera toujours efficace dans la mesure ou le nombre de coups maximum est grand. Cette stratégie choisissant un sommet positif aléatoire, un grand nombre de coup permet toujours de finir la partie. Observons maintenant les résultats sur les stratégies 1 a 4. Ces stratégies appliquent la même idée principale mais avec des petites variantes, on a cherché à savoir l'incidence de rajout de spécifications sur une stratégies sur les résultats. Les résultats sont assez intéressants car l'on se rend compte que rajouter des conditions n'est pas toujours bénéfique. Par exemple le rajout d'une condition pour départager les égalités de la stratégie 1 a la stratégie 2 a été bénéfique. D'un autre côté on observe sur les stratégies 3 et 4 ,qui appliquent la stratégie 2 avec une nouvelle condition pour départager les égalités, qu'approfondir une stratégies qui semble bien n'est pas toujours efficace. On observe bien que la stratégie 3 est moins efficace que sa stratégie 'parente' alors que la stratégie 4 est plus efficace que la stratégie parente.

5.4.2 Second exemple

Observons maintenant les résultats si l'on change uniquement le nombre de sommets. Pour ce second test le nombre de sommets passe de 8 a 30.



La première chose que l'on observe c'est des taux bien plus bas pour les premières stratégies que sur le premier test. La seconde observation est l'évolution de la stratégie 8 qui passent de 99%

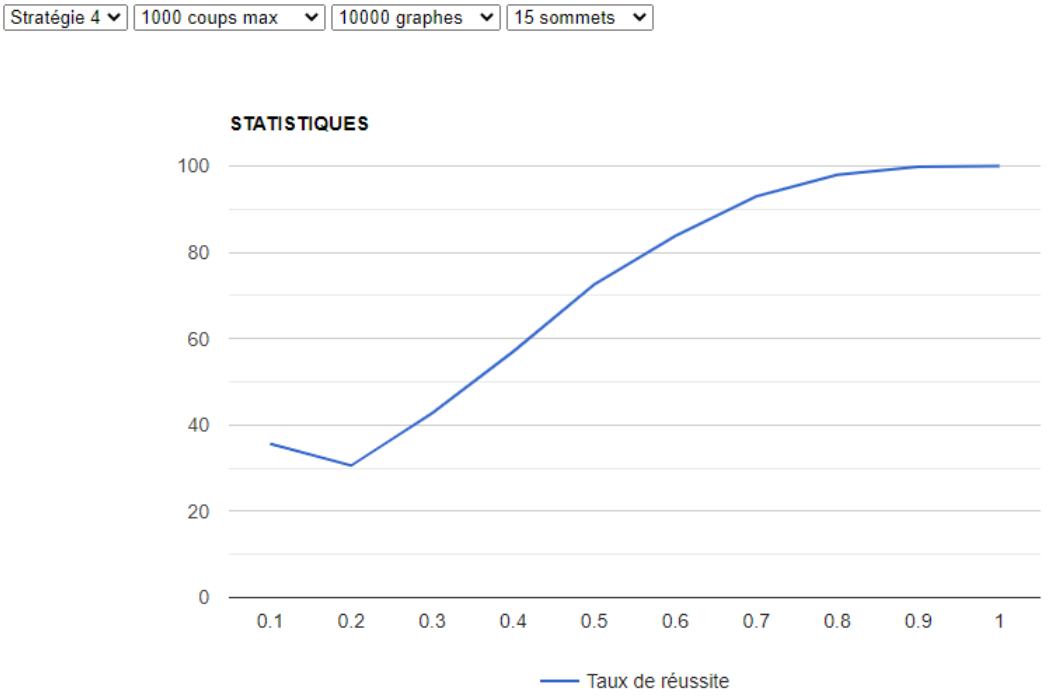
à 0%. Ce qui semble assez logique : plus l'on a des sommets potentiels, plus le hasard devient moins efficace. A l'inverse la stratégie 9 devient bien plus efficace dès lors que l'on travaille sur des parties avec plus de sommets. Les stratégies 6 et 7 semblent ne pas être efficaces quel que soit les paramètres.

5.5 Statistiques sur les stratégies

Afin d'aller plus loin dans l'analyse des différentes stratégies, nous avons implémenté une interface permettant de générer des graphiques. En prenant plusieurs paramètres en compte, l'interface est capable de donner les taux de réussites d'une stratégies en fonction du degré. Nous avons pu donc étudier la pertinence d'une stratégie en fonction du degré.

5.5.1 Premier exemple

Ce premier exemple portera sur la stratégie 4. On travailleras sur 10000 graphes de 15 sommets avec 1000 coups au maximum. Le graphique représente donc l'évolution de la stratégie 4 avec ces paramètres en fonction du degré.



La première chose que l'on observe est l'évolution croissante du taux de réussite en fonction du degré, on atteint même un taux de 100% de réussite avec un degré de 1. Lorsque nous avions obtenu ce résultat, nous avions d'abord pensé que quelque chose n'allait pas dans un de nos algorithmes. Nous avons donc investiguer pour savoir si oui ou non ce comportement étais normal ou pas. Il se trouve que ce comportement est parfaitement normal car il correspond au cas limite du jeu du

dollar game. Prenons en compte les paramètres précédant avec un degré de 1, le nombre de Betti est fixe sur toute les partie car le nombre d'arêtes est maximal. On travail donc sur un graphe complet, le nombre d'arêtes est donc égal a la somme des entiers de 1 a $n-1$ soit :

$$\sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n} i = \frac{n(n-1)}{2}$$

On sait que n est égal a 15 donc les graphes comporteront toujours :

$$\frac{n(n-1)}{2} = \frac{15 * 14}{2} = 105$$

On a donc quel que soit le graphe (avec les paramètres données) :

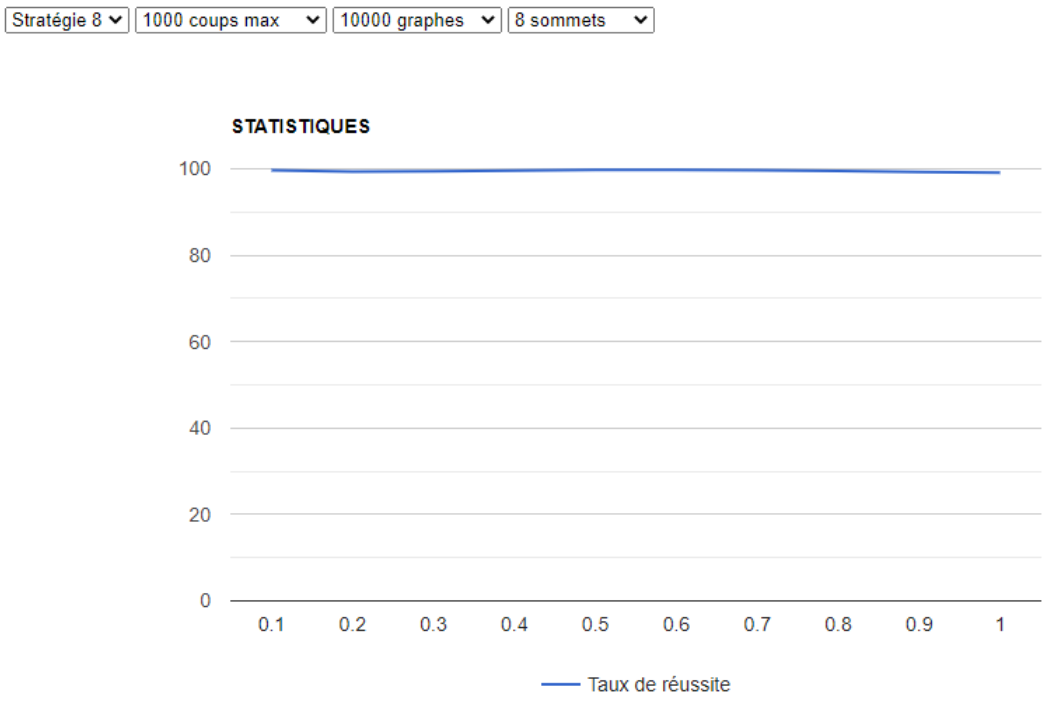
$$Betti = 105 - 15 + 1 = 89$$

Pour que la partie soit gagnable, la somme totale des sommets doit donc être supérieur ou égal a 89. Ce qui équiveut a une moyenne d'environ 6 par sommets. On observe donc que plus le degré augmente, plus le nombre de Betti augmente et donc par conséquent rend la partie plus simple. En réalité quel que soit la stratégie, le taux de réussite avec un degré de 1 est quasiment toujours de 100% car la plupart des graphes générés sont déjà gagné ou presque. De plus, obtient souvent une courbe qui augmente avec le degré, on peut donc faire un parallèle entre degré et taux de réussite. Comme ces cas représente les cas limite du dollar et ne représente pas réellement une partie de dollar game (Ex : 30 sommets, 435 arêtes) observons maintenant les résultats sur des degrés plus faible.

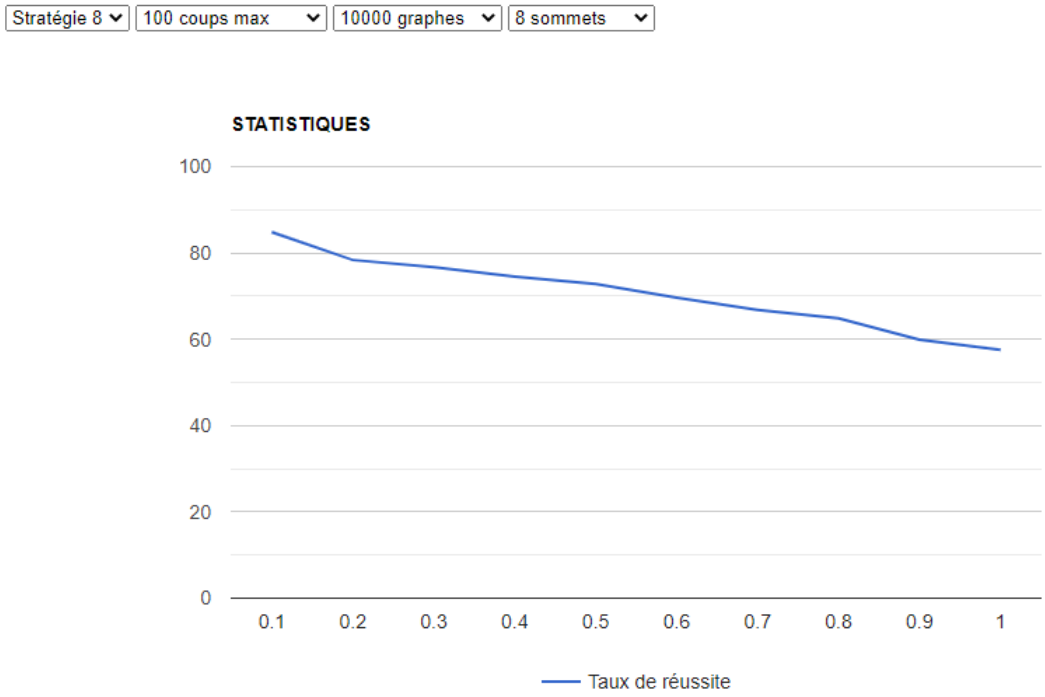
Malgré la courbe ascendante on observe que la courbe part de 0.2 et non 0.1, ce qui implique que la stratégie est plus efficace avec le degré le plus faible. On peut expliquer cela par la structure crée avec un degré de 0.1, en effet un degré si faible implique souvent que le graphe est un arbre. Un arbre ne contenant pas de cycle implique une résolution plus simple de la partie.

5.5.2 Deuxième exemple

Le deuxième exemple portera sur la stratégie 8, qui choisit un sommet positifs aléatoire. On observera encore les résultats sur 10000 graphes de 8 sommets. Nous allons comparer la courbe produite en 1000 coups maximum avec celle produite en 100 coups maximum.



Comme le nombre de coups maximum est très grand, cette stratégie s'avère être efficace en terme de taux de réussite d'une partie quel que soit le degré. Cependant cette efficacité est relative car le nombre de coups pour chaque partie n'est pas du tout optimal.



On observe logiquement une chute drastique du taux de réussite dès que l'on autorise moins de coups maximum. Cette stratégie possédant une grande part d'aléatoire, il est normal qu'un grand nombre de coups soient nécessaire pour finir une partie.

Partie 6

Gestion du Projet

6.1 Organisation et planification

Le diagramme de Gant du projet ce situe à la prochaine page. Il y retrace les tâches réalisées, à quel moment et durant combien de temps. Les parties en vert foncé ont été réalisées par Corentin Teyssier tandis que les parties grises correspondent au travail de Allan Crista. Le mémoire en vert clair a été réalisé à deux.

6.2 Changements majeurs

Malheureusement le projet ne s'est pas passé comme prévu. Nous avons dû faire face a trois abandons dès les premières semaines du projet, par conséquent nous avons du réaliser le projet a deux : Corentin Teyssier et Allan Crista. Nous avons par conséquent partager tout le travail à deux, chose qui n'était évidemment pas du tout prévue. Malheureusement, la partie du cahier des charges qui concernent les arbres n'a pas pu être réalisé.

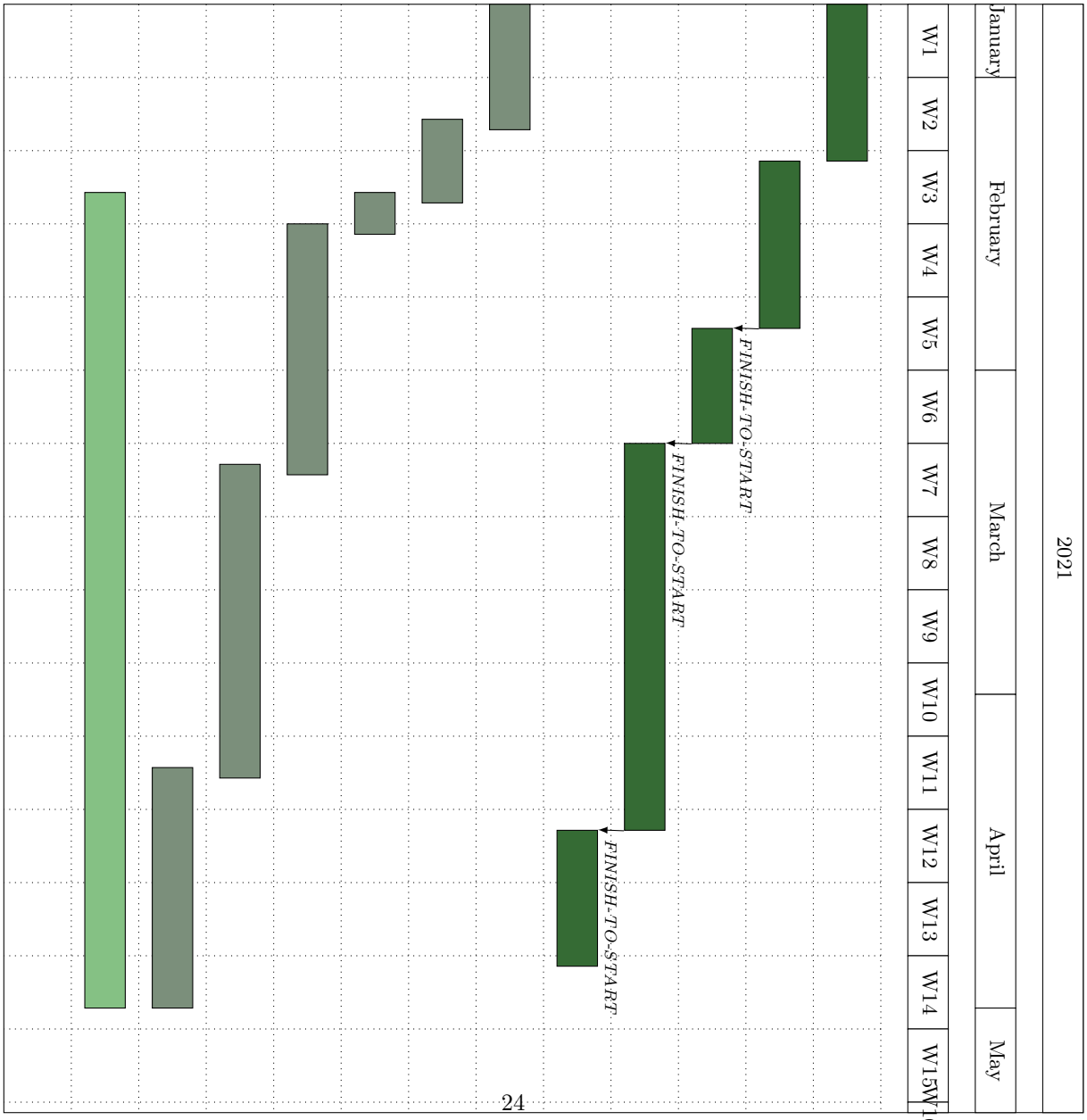


Figure 6.1 – Diagramme de Gant du projet

Partie 7

Bilan et Perspectives

7.1 Bilan

Pour conclure, à l'issue de ce projet nous avons pu produire une interface graphique jouable et des algorithmes de stratégies applicable sur un grand échantillons de graphe aléatoire. Ce TER nous aura permis d'approfondir nos connaissances en javascript et en algorithmiques des graphes. Malgré les aléas nous avons pu respecter une grande partie de notre cahier des charges, la plus grande partie manquante étant celle sur les arbres.

7.2 Perspectives

- Améliorer l'interface afin de proposer plus de niveaux et des difficultés.
- Approfondir les stratégies afin de trouver une stratégie adaptée à toutes les configurations.
  - Trouver de nouvelles stratégie en s'inspirant des résultats obtenues pour améliorer le taux de réussite.
  - Développer un algorithme 'intelligent' qui choisit la bonne stratégie en fonction du graphe donné afin d'obtenir le meilleur résultat.
  - Améliorer les algorithmes pour permettre de traiter plus rapidement des grands échantillons de graphe
- Développer la partie non-traitée sur les arbres.
  - implémenter l'algorithme de résolution sur les arbres
  - Implémenter un algorithme permettant de générer des arbres.

Annexes

Annexe A

Bibliographie

[1]- M. Baker and S. Norine, Riemann-roch and abel-jacobi theory on a finite graph, Advances in Mathematics vol 215, no. 2 (2007) 766788

Annexe B

Stratégies

1	Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prend le premier par ordre des numéros de sommet
2	Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité, prend le premier trouvé
3	Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité prend celui au plus grand degrés
4	Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre celui avec le plus faible poids cumulé de ses voisins, si il y a égalité prend celui au plus petit degrés
5	Trouve le numéro du sommet avec la plus grande valeur, si il y en a plusieurs, prendre le sommet au plus grand degré
6	Trouve le numéro du sommet qui a le plus faible poids cumulé de ses voisins quel que soit sont poids
7	Trouve le numéro du sommet qui a le plus faible poids cumulé de ses voisins quel que soit sont poids, si il y a égalité prend le plus grand
8	Choisis un sommet aléatoire parmi les positifs
9	Trouve le numéro du sommet étant le voisin le plus grand du sommet le plus petit

Annexe C

Interface