

Développement d'une méthode de recherche arborescence pour un jeu à deux joueurs : application au jeu 7 Wonder-Duel

TER

CAMBRESY Florian
CHALAUD Jean-Christophe
LE DENMAT Mickaël



Université de Versailles Saint-Quentin-en-Yvelines

DATE ???

Table des matières

1 Développement d'une méthode de recherche arborescence pour un jeu à deux joueurs : application au jeu 7 Wonder-Duel	2
1.1 Sujet	2
1.2 Description du travail attendu	2
1.3 Description du jeu : 7 wonders-Duel	2
1.3.1 7 wonders - Duel	3
1.3.2 Règles du jeu	3
2 Rappels généraux	5
2.1 Théorie des jeux	5
2.2 Catégories de jeux	5
2.3 Représentation d'un jeu	6
2.3.1 forme extensive	6
2.3.2 forme normale	7
2.3.3 forme caractéristique	7
2.4 Recherche arborescente et intelligence artificielle	7
2.5 Méthode alpha-bêta	7
2.5.1 Méthode min-max	7
2.5.2 Amélioration alpha-bêta	8
2.5.3 Coupure Alpha	8
2.5.4 Coupure Bêta	9
3 Travail effectué	9
3.1 La fonction d'évaluation	9
3.2 Implémentation	10
3.2.1 Carte	11
3.2.2 Jetons	11
3.2.3 Joueur	11
3.2.4 Plateau	12
3.2.5 Stratégie	12
3.3 Interface	15
3.4 Résultat obtenu	15
4 Conclusion	15
4.1 Objectif(s) atteint/ non atteint	15
4.2 Suggestion d'amélioration(s)	15
4.3 Difficultée(s) rencontrée(s)	15

5 Annex	15
5.1 Code	15

1 Développement d'une méthode de recherche arborescence pour un jeu à deux joueurs : application au jeu 7 Wonder-Duel

1.1 Sujet

Il s'agira tout d'abord pour les étudiants de se familiariser avec les méthodes de recherche arborescente appliquées aux jeux, en particulier la méthode alpha-bêta.

Dans un second temps, il conviendra d'implémenter une telle méthode pour le jeu intitulé 7wonders - Duel. Ce jeu est décrit sur de nombreux sites, certains proposent même une petite vidéo tutorielle expliquant les règles de ce jeu. Il est à noter que l'encadrant offrira ce jeu aux étudiants qui choisiront ce sujet.

Pour que le programme "joue bien", il conviendra en particulier de réfléchir à des fonctions d'évaluation pertinentes des positions de jeu.

1.2 Description du travail attendu

L'objectif du projet est d'implémenter une fonction d'évaluation ainsi que la méthode alpha-bêta pour faire en sorte que qu'un algorithme puisse jouer. Ce dernier va donc prendre en entrée une situation de départ, les cartes de chaque joueur, leurs monnaies, ect et va fournir un coup à faire. De plus ce coups doit être pertinent et digne d'intérêt. Nous discuterons de cette définition par la suite.

Afin d'apporter une solution à ce sujet, nous découperons le projet en trois parties.

La première sera une partie descriptive concernant les notions contenues dans la méthode alpha-bêta pour le programme joue correctement en insistant sur les points qui devront être étudiés afin d'appliquer la méthode à notre exemple précis. Ensuite, nous expliquerons les règles du jeu et évoquerons le déroulement d'une partie. Nous développerons le système de victoire et, afin de faire une transition sur la partie qui suivra, nous débattrons concernant la force des "positions" au cours d'une partie.

La deuxième partie montrera le chemin de pensée que nous avons eu afin d'arriver à une ou des solutions pour appliquer la méthode alpha-beta sur notre jeu. Plus particulièrement nous décrirons la fonction d'évaluation que nous avons trouvé afin que l'algorithme suggère des coups pertinents.

Enfin la dernière sera une présentation des solutions techniques que nous avons mis en place.

1.3 Description du jeu : 7 wonders-Duel

7 Wonder est un jeu de plateau sorti dans les années 2010, créé par Antoine Bauza et publié par Repos Production en Belgique. Le nombre de joueurs est entre 3 et 7. On y joue une ancienne civilisation avec ses conflits militaires mais aussi ses activités commerciales. Il est connu et très apprécié par la communauté ayant remporté plus de 30 prix et souvent cité comme l'un des jeux

de société les plus influents de la dernière décennie[10]. Le jeu choisi afin d'appliquer l'algorithme min-max est une variante de celui-ci, le 7 wonders - Duel.

1.3.1 7 wonders - Duel

Le jeu 7 wonders - Duel est comme son nom l'indique un 7 wonders mais à deux joueurs. Un jeu sorti en 2015 par la même société.

1.3.2 Règles du jeu

Le jeu se présente comme suit dans le livret des règles[4].



Il est constitué de :

- 1 plateau de jeu.
- 66 cartes Âge.
 - 23 cartes pour l'Âge I.
 - 23 cartes pour l'Âge II.
 - 20 cartes pour l'Âge III.
- 7 cartes Guilde.
- 12 cartes Merveille, avec un nom, un coût en ressource et un effet (un bonus).
- 4 jetons Militaire, donnant une certaine somme de monnaie.
- 10 jetons Progrès.
- 1 Pion Conflit, indiquant quel joueur a l'avantage militaire.
- 31 pièces de monnaie
 - 14 de valeur 1.

- 10 de valeur 3.
- 7 de valeur 6.
- 1 carnet de scores.

Les cartes Âge et Guilde sont des bâtiments. Elles peuvent avoir un coût (monetaire) afin d'être utilisées, qui est placé en haut à gauche. Elles peuvent donner des effets, placé en haut au centre. Les effets sont multiples comme par exemple une production de matière, une réduction de coût de construction d'un bâtiment ou d'une merveille, ainsi que d'autres. Enfin elles ont aussi un nom, placé en bas. Les cartes ont des couleurs différentes indiquant à quel type de bâtiment elles appartiennent.

- Les Cartes marrons sont des bâtiments de matière première.
- Les Cartes grises sont des bâtiments de produit manufacturé.
- Les cartes bleues sont des bâtiments civils, elles donnent des points de victoire.
- Les cartes vertes sont des bâtiments scientifiques, elles donnent aussi des points de victoire et
- Un symbole scientifique. Lorsqu'un joueur a deux symboles scientifique identique il peut prendre un jeton Progrès.
- Les cartes oranges sont des bâtiments commerciaux, elles ont des objectifs multiples comme donner des pièces au joueur, produire des ressources, modifier les règles de commerce.
- Les cartes rouges sont des bâtiments militaires, elles augmentent la puissance.
- Les cartes violettes sont des bâtiment de Guilde, elles donnent des points de victoire en fonction de certains critères.

De plus le joueur peut acheter des ressources à la "banque" via le commerce comme par exemple si le joueur souhaite construire un bâtiment mais ne dispose pas de toutes les ressources. Pour acheter cette ressource il faut prendre en compte les ressources produites par le joueur adverse (carte marron et grise) y ajouter 2 et remettre la somme dans la banque puis construire le bâtiment. Concernant les bâtiments, certains octroient un symbole de chaînage (en blanc en haut de la carte). Si le joueur souhaite construire un bâtiment et si il possède une carte donnant le symbole de chaînage il peut alors le construire gratuitement, dans le cas contraire il doit payer son coût en ressources et/ou monetaire.

Une fois la présentation faite, nous allons pouvoir parler de la préparation du jeu avant de débuter une partie. La préparation du plateau se fait ainsi :

1. Placez le plateau entre les deux joueurs.
2. Placez le pion Conflit sur la case neutre au milieu du plateau.
3. Placez les quatres jetons Militaire sur leurs emplacements.
4. Mélangez les jetons Progrès et placez-en cinqs au hasard sur la plateau.
5. Chaque joueur prend 7 pièces à la banque.

Puis vient la sélection des Merveilles, pour cela il faut désigner le premier joueur et mélanger les Merveilles, ensuite :

- Disposez 4 Merveilles aléatoires entre les joueurs.
- Le premier joueur choisit une Merveille.
- Le deuxième joueur en choisit deux.
- Le premier joueur prend la dernière.

- Disposez les quatres autres Merveilles.
- Le deuxième joueur choisit une Merveille.
- Le premier joueur en choisit deux.
- Le deuxième joueur prend la dernière.

Pour construire une Merveille, le joueur doit mettre une carte face cachée sous la Merveille (peu importe la carte, elle sert juste à indiquer que la Merveille est construite) et doit payer le coût de la construction. Il y a un maximum de sept Merveilles construit par partie, le joueur dont la dernière Merveille n'est pas construite doit la remettre dans la boîte.

Enfin il faut placer les cartes d'Âge dans une structure particulière en fonction de l'âge. Les cartes face cachées sont retournées lorsque les cartes visibles posées dessus sont enlevées par les joueurs. Les joueurs peuvent choisir aussi de défausser une carte en échange de 2 pièces et d'un bonus d'une pièce pour chaque carte jaune qu'il possède.

Il y a trois manières de gagner, la première est la victoire militaire, l'un des joueurs aura avancé le pion Conflit dans la case capitale de son adversaire. La deuxième étant la victoire par suprématie scientifique, l'un des joueurs réunit 6 symboles scientifiques différents. Enfin, si aucun des joueurs n'a gagné à la fin de l'Âge III, il faut alors faire la somme des points de victoire acquis durant la partie, celui qui en possède le plus l'emporte.

2 Rappels généraux

2.1 Théorie des jeux

Dans le monde des mathématiques il y a un domaine qui s'intéresse aux interactions stratégiques qui existent entre plusieurs agents. Ces interactions peuvent se faire dans un jeu, dans les sciences sociales, politiques ainsi que dans d'autres exemples. Ce domaine est la théorie des jeux[8]. Il faut noter que ici le mot "jeu" a un sens plus large qu'un jeu de société ou autre.

Dans ce projet nous porterons attention uniquement aux interactions entre deux joueurs dans un jeu de société à travers un exemple que nous développerons par la suite.

2.2 Catégories de jeux

Afin de connaître quelle approche doit être utilisée pour étudier les stratégies qui existent au sein d'un jeu, la théorie repartie les jeux en différentes catégories[9].

1. Coopératifs ou non.
2. Somme nulle ou non.
3. Simultanés ou séquentiels.
4. Information complète ou non.
5. Mémoire parfaite ou non.
6. Déterminé.

7. Finis.
 8. Répétés.
- (1) Un jeu coopératif permet d'analyser la formation de coalitions afin d'améliorer le gain potentiel.
- (2) Un jeu à somme nulle (ou jeu strictement compétitif) est un jeu dont les gains de certains joueurs sont les pertes des autres afin d'avoir la somme des gains et des pertes nulle, comme dans les échecs, dans le poker ou dans d'autres. A l'inverse dans un jeu à somme non nulle, tous les joueurs peuvent perdre, ou gagner, voir même certains peuvent gagner un gain moins (réciproquement plus) important que la perte totale d'autres joueurs.
- (3) Un jeu est dit "simultané" si les joueurs jouent en même temps, dans le cas contraire c'est un jeu séquentiel.
- (4) Un jeu peut être à "information complète", c'est-à-dire que chaque joueur connaît l'ensemble des informations qui influent leur prise de décision.
- (5) Un jeu peut être aussi à "mémoire parfaite", dans ce cas chaque joueur peut se rappeler des coups qui ont été joués soit en les notant a part, soit ils sont visibles au cours de la partie (par exemple les cartes qu'a choisie le joueur sont à coté de ce dernier, face visible).
- (6) Un jeu "déterminé" est un jeu particulier à somme nulle sans intervention du hasard.

2.3 Représentation d'un jeu

Maintenant que nous savons comment définir un jeu, nous allons étudier comment représenter ce dernier et les interactions entre les joueurs. Il existe globalement trois représentations[7], la forme extensive et la forme normale, qui sont utilisées pour les jeux non coopératifs et la forme caractéristique pour les jeux coopératifs.

2.3.1 forme extensive

La forme extensive est la représentation sous forme d'arbre de décision où les noeuds d'une même hauteur sont les choix possibles (qui ne sont pas en dehors des règles prévues par le jeu) d'un joueur. Plus précisément chaque noeud est une situation produite au cours d'une partie (pas nécessairement le début de la partie) et les fils de ce noeud sont les coups que peut faire le joueur suivant. Par exemple dans les échecs, la racine sera l'état du plateau de jeu à la fin du tour d'un joueur (la position des pions de chaque joueur) et les fils de cette racine seront les déplacements des pions de l'adversaire. Dans cet exemple, ainsi que dans tous les jeux à deux joueurs, les hauteurs paires sont les coups du joueur A tandis que les hauteurs impaires sont les coups du joueur B. De plus, les feuilles de cet arbre sont des noeuds indiquant la fin de la partie, soit il y a victoire d'un des deux joueurs, soit plus aucun coup n'est possible. C'est la forme que nous allons utiliser durant tout ce projet, c'est pour cela que nous passerons rapidement sur les autres formes.

2.3.2 forme normale

La forme normale (ou stratégique) est définie par :

- L'ensemble des joueurs (de taille finie).
- L'ensemble des stratégies possibles pour chacun des joueurs (fini ou infini).
- Les préférences de chacun des joueurs, soit un sous-ensemble de stratégies parmi l'ensemble des
- combinaisons stratégiques possibles soit via une fonction d'utilité ou un fonction de gain.

2.3.3 forme caractéristique

La forme caractéristique est utilisée, comme dit précédememnt, pour les jeux coopératifs. Elle est représentée sous forme d'un graphe $G=(N,v)$ avec N l'ensemble des joueurs et v la fonction caractéristique. Cette dernière associe à chaque sous-ensemble de joueurs (noté S) qui forme une coalition, la valeur $v(S)$, la gain obtenu par la coalition.

2.4 Recherche arborescente et intelligence artificielle

La recherche arborescente est une recherche qui se base sur la forme extensive afin d'utiliser l'arbre de décision[2]. Cette recherche s'appuie aussi sur une fonction d'évaluation, qui associe à toutes situations du jeu une valeur indiquant si elle est favorable à une victoire. Bien évidemment plus cette valeur est grande plus la probabilité de victoire est grande. Cette fonction ne s'applique qu'aux feuilles de l'arbre de décision, ces feuille peuvent être la fin de la partie ou un état de la partie dans lequel nous évaluons le rapport de force entre les deux joueurs afin de déterminer qui est en meilleur poste pour gagner. Ainsi l'objectif de cette recherche est de choisir la suite de noeud à prendre (donc de coups à faire) afin d'arriver à la valeur d'évaluation maximale.

2.5 Méthode alpha-bêta

2.5.1 Méthode min-max

Avant d'évoquer la méthode alpha-bêta, nous allons expliquer comment fonctionne la méthode min-max, la méthode alpha-bêta en est une amélioration. La méthode min-max est un algorithme utilisé dans la théorie des jeux, plus précisément dans les jeux à deux joueurs, à somme nulle et à information complète[3].

L'objectif de cette dernière est de minimiser la perte dans le pire cas pour un joueur est de maximiser la perte pour l'adversaire, d'où son nom "min-max". Pour cela l'algorithme va simuler tous les coups possibles et leur donner une valeur, représentant combien ce coup est intéressant ou non. La simulation de tous les coups possibles passe par la construction de l'arbre de décision et le calcul de la valeur des feuilles est l'application d'une fonction d'évaluation. Nous obtenons donc un arbre avec des valeurs aux feuilles uniquement. Ensuite, nous allons devoir faire "remonter" la valeur du jeu des feuilles jusqu'à la racine et choisir parmi tous les coups celle qui a la valeur la moins basse. Pour calculer cette valeur nous allons passer par une approche récursive comme suit[5] :

- $\text{minmax}(p) = f(p)$ si p est un feuille de l'arbre où f est la fonction d'évaluation.

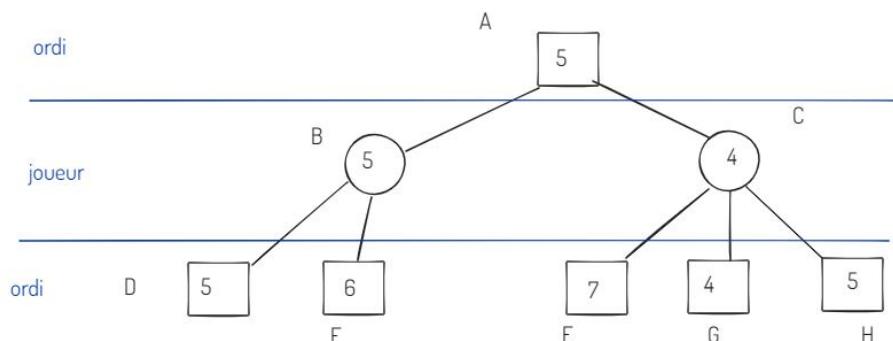
- $\text{minmax}(p) = \max(\text{minmax}(O_1), \dots, \text{minmax}(O_n))$ si p est un noeud de l'ordinateur avec les fils O_1, \dots, O_n .
- $\text{minmax}(p) = \min(\text{minmax}(O_1), \dots, \text{minmax}(O_n))$ si p est un noeud du joueur avec les fils O_1, \dots, O_n .

Le souci de cette méthode est le nombre de cas à étudier. En effet comme la simulation produit un arbre que nous devons parcourir en entier afin de choisir le meilleur coup à jouer nous devons faire un parcours de cet arbre. Cependant comme cela passe soit par un algorithme récursif, qui doit simuler un nombre important de coups et de réponses ce qui fait augmenter la profondeur de l'arbre. En plus de faire augmenter le nombre d'appels récursifs cela surcharge le tas d'appel. Soit, dans l'autre cas, nous pouvons passer par un algorithme itératif avec une file ou une pile. Or si le nombre de réponses possibles à un coup du joueur devient trop important cela nous oblige à utiliser beaucoup de mémoire. Enfin, même si nous prévoyons le fait que le programme utilise beaucoup de mémoire, nous ne voulons pas que le programme continue de calculer une suite de coups qui est moins avantageuse une suite qu'il a déjà calculé. Cela sera contre-productif. Ainsi c'est pour répondre à toutes ces problématiques que l'élagage alpha-bêta a été conçu.

2.5.2 Amélioration alpha-bêta

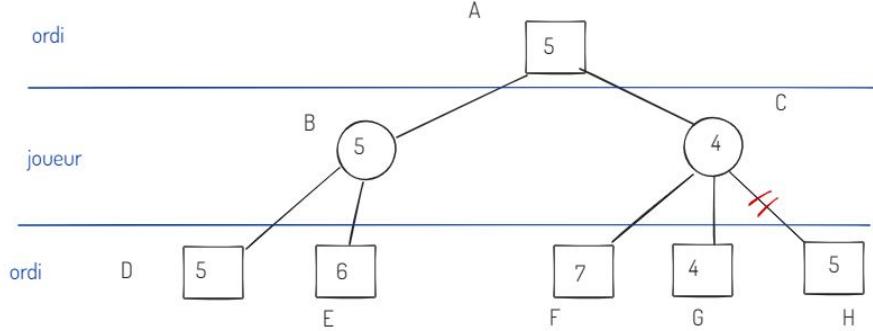
L'élagage alpha-bêta est comme son nom l'indique une méthode qui vise à "supprimer" les branches de l'arbre qui ne sont pas utiles, qui ne change rien au résultat final. Pour élaguer les branches, nous disposons de deux types de coupures, la coupure alpha et la coupure bêta, d'où le nom de la méthode[6]. Cette méthode s'applique uniquement sur des arbres au moins de profondeur trois (la racine ayant un profondeur de un). Une fois que la méthode min-max à construit l'arbre et appliquée la fonction d'évaluation aux feuilles, nous allons ajouter les coupures lors de la "remonter" des valeurs à la racine.

2.5.3 Coupure Alpha



Prenons l'exemple ci-dessus pour expliquer la coupure alpha, elle se place du point de vue du joueur. Une fois que la valeur du noeud B a été initialisé avec le minimum de ces fils (D, F) cette valeur va nous servir de borne pour faire nos coupures. Regardons les fils du frère de B, c'est à dire F, G (supposons que la valeur de H ne soit encore connue) afin d'initialiser la valeur de C. Nous

devons prendre la valeur minimale entre F et G (ici $F=4$) mais nous constatons que cette valeur est plus petite que notre borne ($B=5$), ainsi explorer le noeud H ne sert à rien car peut importe la valeur qu'il aura (plus petite ou plus grande que F) il ne sera pas utiliser pour initialiser la valeur de A puisque $A=\max(B,C)$. Nous pouvons donc élaguer H (et tous les autres fils de C s'ils existent) comme sur l'image ci-dessous.



2.5.4 Coupe Bêta

La coupe bêta est similaire mais se base du point de vue de l'ordinateur, on élague donc les feuilles plus grande que la borne[10].

3 Travail effectué

3.1 La fonction d'évaluation

Comme expliqué précédememnt la fonction d'évaluation nous permet d'associer un nombre à une situation lors d'une partie. C'est-à-dire l'ensemble des éléments (cartes, merveilles, monnaies, position du jeton conflit, jetons militaires, jetons progrès) présents sur le plateau. Pour cela nous avons commencé par réduire le nombre d'éléments que nous prenons en compte lors d'une partie et nous avons travaillé uniquement avec les cartes. De plus nous avons considéré uniquement les cartes de l'age I. Après quelques brain storming nous avons conclu que la solution la plus simple était de donner un "poids" aux cartes en fonction de notre manière de jouer, puis de modifier ce poids en fonction de la situation.

Pour définir ce poids, nous nous sommes tous reunis afin de nous concerter pour trouver la note la plus adequate pour chacune des cartes. Premièrement nous avons défini un intervalle de valeur arbitraire (entre 0 et 20) et nous avons aussi attribué un poids à une carte face cachée (la moyenne de l'intervalle donc 10). Pour les cartes avec une valeur inférieure à 10, il est préférable de tenter de prendre la carte face cachée, avec une chance de 50 % d'avoir une meilleure carte. De cette manière nous avons pu prendre en compte l'évolution des âges au cours d'une partie. Nous avons défini un objectif par âge, le premier doit permettre au joueur de collecter des ressources simples donc du bois, de la pierre et de l'argile. L'âge II doit permettre au joueur de collecter des ressources rares,

donc verre et papyrus mais aussi attaquer et obtenir des symboles scientifiques et construire des merveilles. Durant le dernier âge le joueur doit viser les cartes donnant des points militaires. Cette solution nous permet aussi de prendre en compte le coût des cartes, par exemple les cartes "Chantier" et "Exploitation" donnent toutes les deux du bois mais "Exploitation" coûte une pièce, si nous avons le choix entre les deux il est préférable de prendre "Chantier" et d'utiliser la monnaie pour acheter des ressources que nous ne possérons pas.

Après plusieurs parties contre l'ordinateur, nous avons ajouté la possibilité de construire des merveilles. Nous avons utilisé la même méthode évoquée précédemment.

3.2 Implémentation

Pour l'implémentation nous avons décidé de commencer par réfléchir au découpage du jeu en différent modules ainsi que du design de l'interface.

Nous avons conclus que l'arborescence des fichiers serait :

- Un dossier "interface_graphique" contenant tous les fichiers utiles pour l'interface graphique, c'est-à -dire :
 - Un dossier "ressources" avec toutes les images (cartes, merveilles, jetons, plateau, ...). Mais aussi les musiques.
 - Un dossier "src" contenant les fichiers sources de l'interface, c'est-à-dire :
 - Boutons.py
 - Constantes.py
 - Fenetre.py
 - MonSprite.py
 - SpriteCarte.py
 - SpriteJetonMilitaire.py
 - SpriteJetonProgre.py
 - SpriteMerveille.py
- Un dossier "utils" contenant tous les fichiers utiles au jeu, c'est-à-dire :
 - Carte.py
 - CarteGilde.py
 - Merveille.py
 - JetonMilitaire.py
 - JetonProgres.py
 - Joueur.py
 - notation_carte.csv (un fichier stockant les "poids" des cartes)
 - Outils.py
 - Plateau.py
 - Stratégie.py
- Un dossier "test" contenant tous les fichiers tests, c'est-à-dire :
 - TestCarteJeton.py
 - TestJoueur.py

- TestMerveille.py
- TestPlateau.py
- TestStratégie.py

Comme vous pouvez le constater les extensions de fichiers sont en ".py" cela signifie que se sont des fichiers Python. En effet lorsque nous avons discuté au sujet du langage à utiliser nous avions remarqué qu'ils nous fallait un langage orienté objet et que nous maîtrisons. Nous avions donc le choix entre le Java et le Python. Pour des raisons de simplicité nous avons choisi le Python. Une fois tous les préparatifs terminés nous avons commencé l'implémentation à proprement parlé. Nous avons suivi plusieurs phases, nous travaillons sur un module, une fois fini nous le testons à travers différentes batteries de tests, puis nous passons au module suivant. Une fois le jeu fini nous avons testé le jeu en 1vs1 via le terminal.

Nous allons maintenant expliquer brièvement le contenu des différentes classes constituant le projet et nous détaillerons le contenu de la classe "Stratégie" qui est le cœur du projet.

3.2.1 Carte

Commençons par la classe carte. Une carte est composé d'un nom, d'un effet, d'un cout, d'un nom de carte de chainage (none si la carte n'en possède pas), une couleur, et elle appartient à un age. Concernant les effets et les couts, nous avons utilisé les propriétés du Python en définissant les effets comme des chaînes de caractère respectant un pattern précis. Elle commencent par le type d'effet comme "ressource", "monnaie", "attaquer", "symboleScientifique" puis les paramètres comme "ressource bois 1", "attaquer 3". Les classe "CarteGuilde" et "Merveille" sont des classes filles de la classe "Cartes". Les cartes guildes et les merveilles sont des cartes mais sans carte de chaînage sans couleur et sans âge (elles appartiennent toutes à l'âge III). Les merveilles possèdent un attribut pour savoir si elles sont construit ou non.

3.2.2 Jetons

Passons maintenant aux jetons. Il en existe deux types, le jeton militaire qui possède un nom, un nombre de pièces que perd l'adversaire lorsque le jeton est pris et un nombre de points de victoire que gagne le joueur. Le deuxième type est le jeton progrès qui possède un nom et des effets.

3.2.3 Joueur

Nous avons aussi une classe joueur. Le joueur est défini par un nom, et possède plusieurs listes d'objets visible sur le plateau, c'est-à-dire ces cartes, ces merveilles, et ces jetons progrès, ainsi que de la monnaie. Mais aussi d'autre informations comme un dictionnaire de ressource, des points de victoires un dictionnaire de symbole scientifiques et le nombre de symbole scientifiques différent. Cette classe implémente certaines méthodes qui le jeu utilise pour avoir des informations comme par exemple les coûts manquants pour construire une carte, dans ce cas il faut regarder dans le dictionnaire de ressource du joueur, mais aussi ces monnaies ainsi que dans ces merveilles ou cartes la présence de

carte donnant des ressources au choix ou d'autre effet impliquant la construction de carte ou de merveille. Il existe aussi d'autres méthodes comme savoir si le joueur possède une carte pour faire du chaînage , s'il produit un type de ressources en particulier, ect....

3.2.4 Plateau

Toutes les différentes fonctionnalités du jeu sont dans le fichier Plateau.py. Cette classe représente le jeu dans sa globalité c'est-à-dire le plateau avec les jetons progrès choisit, les cartes présentent en dessous du plateau, les deux joueurs, la fausse et la banque. Elle s'occupe aussi des effets des cartes ainsi que des effets des merveilles. C'est aussi la première classe que nous avons développé durant le projet, elle a donc connue plusieurs évolutions. La classe Plateau contient toutes les méthodes faisant fonctionner le jeu. Elle implémente une méthode pour préparer le plateau, c'est-à-dire tirer aléatoirement les jetons, les cartes, ainsi que les merveilles (si le jeu est en mode choix automatique). Elle contient aussi des méthodes pour piocher, pour enlever une carte, pour obtenir la liste des cartes prenables, pour interagir avec la banque, ect

Après tout cela, l'équipe se sépare en deux groupes, l'un crée l'interface pendant que l'autre fait l'algorithme minimax, d'alpha beta ainsi que la fonction d'évaluation.

3.2.5 Stratégie

Concernant la classe "Stratégie", elle contient la fonction d'évaluation, la fonction minimax ainsi que son amélioration la fonction alpha-beta. Elle commence d'abord par lire le fichier contenant le poids des cartes, des merveilles, et des jetons progrès.

```

1      notation_carte = {}
2      with open("src/utils/notations_carte.csv", mode='r') as file:
3          fichier_csv = csv.reader(file)
4          for lignes in fichier_csv:
5              notation_carte[lignes[0]] = int(lignes[1])
6

```

Nous avons rempli un fichier Google Sheet avec le nom de la carte puis son poids. Puis nous l'avons importé sous un format rapide et simple à lire, le format csv. Le Python nous a permis d'exécuter ceci en seulement quelques lignes de code.

Nous avons ensuite la fonction d'évaluation, elle prend en entrée la partie à un instant donné et va renvoyé une valeur, la différence entre l'évaluation du joueur 2, donc de l'ordinateur, et du joueur 1.

```

1  def fonction_evaluation(partie):
2      evaluation_j2 = 0
3      for carte in partie.joueur2.cartes:
4          if carte.est_face_cachée:
5              evaluation_j2 += 10

```

```

6         else :
7             evaluation_j2 += notation_carte[carte.nom]
8
9     for merveille in partie.joueur2.liste_merveilles_construite() :
10        evaluation_j2 += notation_carte[merveille.nom]
11
12    if partie.joueur1.monnaie == 0:
13        evaluation_j2 += 10
14
15    if partie.position_jeton_conflit == 0:
16        evaluation_j2 += 10
17
18    evaluation_j1 = 0
19    for carte in partie.joueur1.cartes:
20        if carte.est_face_cachee:
21            evaluation_j1 += 10
22        else:
23            evaluation_j1 += notation_carte[carte.nom]
24
25    for merveille in partie.joueur1.liste_merveilles_construite() :
26        evaluation_j1 += notation_carte[merveille.nom]
27
28    if partie.joueur2.monnaie == 0:
29        evaluation_j1 += 10
30
31    if partie.position_jeton_conflit == 18:
32        evaluation_j1 += 10
33
34    return evaluation_j2 - evaluation_j1

```

Comme nous l'avons expliqué plutôt la fonction va partir courir les cartes du joueur deuxième et va faire la somme de leur poids. Puis elle va faire la somme du poids des merveilles construites. Elle va de même pour le joueur un. Nous avons ajouté deux trois modifications, notamment si la position du jeton militaire est à 0, donc dans la ville du joueur 1, la situation est très favorable pour l'ordinateur est inversement.

Enfin nous avons la fonction minimax et alpha-beta, nous sommes parties d'un tutoriel sur Youtube [1] que nous avons adapté avec notre jeu. La fonction minimax est une fonction récursive qui prend en paramètre la partie, la profondeur à calculé, un booléen indiquant si c'est à l'ordinateur de joueur et le nombre de noeuds. Elle renvoie l'évaluation du meilleur coup, la carte à prendre, et le nombre de noeuds évalué.

Nous allons commencer par expliquer l'implémentation avec la version qui ne prend pas en compte les merveilles et les jetons progrès.

```

1         def minimax(partie , profondeur , coup_bot , nbr_noeuds):
2             if profondeur == 0 or partie_fini(partie):

```

```

3             return fonction_evaluation(partie), None, nbr_noeuds+1
4
5     carte_a_prendre = None
6
7     if coup_bot:
8         partie.joueur_qui_joue = partie.joueur2
9         max_eval = -math.inf
10
11    for carte in partie.liste_cartes_prenables():
12
13        copie_partie: Plateau = partie.constructeur_par_copie()
14        ret = copie_partie.piocher(carte)
15
16        if ret == -1:
17            copie_partie.defausser(carte)
18        else:
19            copie_partie.joueur_qui_joue.cartes.append(carte)
20            copie_partie.enlever_carte(carte)
21
22        evaluation, _, nbr_noeuds = minimax(copie_partie, profondeur
23        - 1, False, nbr_noeuds)
24
25        if evaluation > max_eval:
26            max_eval = evaluation
27            carte_a_prendre = carte
28
29    return max_eval, carte_a_prendre, nbr_noeuds+1
30
31 else:
32     partie.joueur_qui_joue = partie.joueur1
33     min_eval = math.inf
34
35     for carte in partie.liste_cartes_prenables():
36
37         copie_partie: Plateau = partie.constructeur_par_copie()
38         copie_partie.piocher(carte)
39         copie_partie.joueur_qui_joue.cartes.append(carte)
40         copie_partie.enlever_carte(carte)
41
42         evaluation, _, nbr_noeuds = minimax(copie_partie, profondeur
43         - 1, True, nbr_noeuds)
44
45         if evaluation < min_eval:
46             min_eval = evaluation
47             carte_a_prendre = carte
48
49     return min_eval, carte_a_prendre, nbr_noeuds+1

```

Cette fonction permet de simuler le coup de l'ordinateur et le coup du joueur. Pour cela elle parcourt toutes la carte prenable de la partie passée en paramètre, pour chacune la fonction créer une copie de la partie. Dans cette copie de la partie le joueur qui joue pioche la carte, ce qui applique les effets de cette dernière. Puis nous appelons récursivement la fonction avec la partie copie en paramètre. Dans cette copie nous regardons à nouveau les cartes prenables et ainsi de suite nous construisons l'arbre de décision.

Passons maintenant à la version alpha-beta de la fonction.

1
2

3.3 Interface

3.4 Résultat obtenu

4 Conclusion

4.1 Objectif(s) atteint/ non atteint

4.2 Suggestion d'amélioration(s)

4.3 Difficultée(s) rencontrée(s)

Nous sommes rapidement tombés sur une première difficulté lorsque nous rédigions le schéma de classe du jeu. Par exemple, les cartes comportent beaucoup d'effets pouvant avoir un impact sur le joueur, comme un gain de ressource, mais aussi sur le jeu, comme le fait du pouvoir rejouer. L'application d'un effet doit être effectué par le jeu et non dans le module "Carte". Ensuite nous voulions créer des classes (plus précisément des interfaces) pour les effets mais cela rendait compliqué la création de carte avec les effets.

5 Annexe

5.1 Code

Références

- [1] Sebastian LAGUE. *Algorithms Explained – minimax and alpha-beta pruning*. URL : <https://youtu.be/l-hh51ncgDI>.

- [2] Université de LILLE. *Arbre de jeu*. URL : https://www.fil.univ-lille1.fr/~L2S3API/CoursTP/Projets/minmax/algo_minmax.html#arbre-de-jeu.
- [3] *Recherche arborescente dans les jeux*. URL : <https://123dok.net/article/recherche-arborescente-jeux-contr%C3%B4le-raisonnement-hypoth%C3%A9tique-%20environnement.qvlo0jgy>.
- [4] *Regles du jeux 7 wonder duel*. <https://cdn.1j1ju.com/medias/bd/ad/a7-7-wonders-duel-regles.pdf>.
- [5] WIKIPEDIA. *Algorithme minimax - principe*. URL : https://fr.wikipedia.org/wiki/Algorithme_minimax#Principe.
- [6] WIKIPEDIA. *Elagege alphabeta*. URL : https://fr.wikipedia.org/wiki/%C3%89lagage_alpha-b%C3%AAta.
- [7] WIKIPEDIA. *representation theories des jeux*. URL : https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_jeux#Repr%C3%A9sentations_des_jeux.
- [8] WIKIPEDIA. *theorie des jeux*. URL : https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_jeux.
- [9] WIKIPEDIA. *typologie theorie des jeux*. URL : https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_jeux#Typologie.
- [10] WIKIPEDIA. *Wonder*. URL : [https://en.wikipedia.org/wiki/7_Wonders_\(board_game\)](https://en.wikipedia.org/wiki/7_Wonders_(board_game)).