BURDIN Kevin 11507706 – INAYA Victor 11510092

Projet théorie des jeux

I- L'objectif de ce projet

Le jeu présenté par ce projet repose sur un « Troll », un élément du jeu qui est disposé sur une rangée de cases, sur la colonne du milieu en général. Le premier et le dernier élément de cette séquence de cases sont matérialisés par des châteaux : ce sont des cases appartenant à des joueurs. Par convention, le joueur 1 va se situer sur la première case de cette séquence, et le joueur 2 sur la dernière. Chaque joueur a pour objectif de repousser le troll dans le château adverse. Pour cela, chaque joueur dispose d'un certain stock de pierres, qu'ils devront simultanément jeter : chaque joueur ne connait donc pas le nombre de pierres que l'adversaire a jeté. Au cours d'un tour, le troll se déplacera d'une case vers le joueur qui a jeté le moins de pierres. La partie se termine lorsqu'un joueur n'a plus de pierre, et un gagnant est élu selon la position du troll et le nombre de pierres restantes au joueur qui dispose encore des siennes.

L'idée derrière ce jeu est de trouver une stratégie optimale qui va permettre au joueur la jouant d'avoir l'ascendant sur l'autre joueur.

Nous allons, d'abord modéliser le jeu sous forme d'un graphe et calculer la forme normale qui en découle. Pour cela, nous pouvons assimiler une situation de jeu à un vecteur qui a 3 dimensions, à savoir le nombre de pierres qui reste au joueur 1, le nombre de pierres qui reste au joueur 2, et la position du troll. De là, nous pouvons construire une forme normale qui va répertorier tous les vecteurs qui découlent d'une situation donnée. A chacun de ces vecteurs sera associée une valeur de gains qui va déterminer quel joueur a l'avantage sur l'autre, trouvée soit en étant dans une situation où le jeu se termine, soit en résolvant un simplex de la sous-matrice de taille (i-1) * (j-1) de la situation (i,j,t). L'issue de ce simplex nous donne le gain garanti en suivant la stratégie prudente dans la situation (i,j,t), ainsi qu'une distribution de probabilités nous indiquant la stratégie à suivre, i.e. le nombre de pierres à lancer pour un tour donné.

II- Le code

Notre projet a été réalisé en Python, sous Visual Studio. Il est divisé en plusieurs modules que nous allons expliciter :

Un premier module, dans le fichier JeuDuTroll.py, qui comporte l'ensemble des fonctions qui permettent de créer des instances de parties de jeu du troll : une classe Plateau, qui va contenir les différentes informations du jeu, donc le nombre de cases, le nombre de pierres pour chaque joueur, la position du troll. Ensuite, nous avons une fonction « Partie () », qui permet de lancer une instance du jeu du troll selon différents paramètres : le mode de jeu (il est possible de jouer à deux joueurs humains, un joueur humain, ou seulement faire affronter des IAs), les paramètres pour créer le plateau de jeu, ainsi que deux paramètres optionnels, en référence, qui peuvent permettre d'extraire les historiques de jeu des deux joueurs. Cette fonction va prendre en compte la saisie des différents joueurs, soit par l'intermédiaire d'une stratégie, soit par l'intermédiaire d'un input, afin de faire évoluer la situation de jeu. A la fin d'une partie, une fonction va choisir un joueur gagnant en fonction de la situation de jeu. Enfin, si l'on souhaite paramétrer notre partie dans un terminal, nous avons une fonction dédiée.

Un second module, dans le fichier Solveur.py, qui va comporter une fonction qui va résoudre le problème d'optimisation, ainsi qu'une fonction qui va générer la matrice de gains du jeu du troll. La matrice de gains étant différente pour chaque joueur, nous avons calculé les matrices de gains dans deux fonctions distinctes. Il comporte également un ensemble de fonctionnalités qui permettent de résoudre la maximisation sans utiliser de simplex. Il s'agit de notre ancien solveur, pour notre première version de stratégie prudente, avant d'utiliser la bibliothèque de programmation linéaire de scipy. Ce dernier comporte une classe « Solveur » qui va instancier tout ce qui sera requis pour maximiser une fonction objectif, un générateur de fonctions objectifs et de fonctions contraintes en fonction du nombre de variables que l'on cherche, et enfin les calculs des différentes matrices de gains.

Enfin, il y a un troisième module « stratégies » dans le fichier « Strategies.py », qui consiste en une suite de fonctions prenant différents paramètres, et sortant un entier, le nombre de pierres que l'IA devra lancer. Parmi elles se trouvent la stratégie prudente, mais également plusieurs autres stratégies, issues de nos réflexions.

Notre fonction main se trouve dans un fichier python dédié. Notre main va lancer une instance de jeu de troll pour chaque itération, puis stocker et afficher la stratégie qui a gagné le plus de manches.

Pour exécuter le projet, il suffit d'exécuter le fichier main.py. Les bibliothèques additionnelles utilisées sont numpy et scipy.

III- Difficultés liées à ce projet

Durant ce projet, nous avons rencontrés plusieurs difficultés :

Tout d'abord, même si le jeu du troll est un jeu conceptuellement très simple, sa stratégie mixte prudente ne nous est pas naturelle du tout. Nous avons donc commencé par imaginer différentes stratégies pures qui nous paressaient pertinentes dans certaines situations. Cependant, ces stratégies pouvaient se faire contrer en connaissant comment elles marchaient. Nous avons, par la suite, commencé l'implémentation d'une stratégie mixte.

L'implémentation de la stratégie mixte n'est pas complexe à implémenter en théorie. Cependant, il s'agit d'une stratégie imprévisible, et il est difficile de toujours savoir ce qu'un simplex va nous donner comme résultat. Il n'est donc pas toujours aisé de savoir si telle ou telle distribution de probabilités est bonne. De plus, si une valeur de la matrice est erronée (même si elle est à peu près juste), cela va complètement nuire à la matrice de gains qui s'en retrouvera altérée.

Enfin, nous avions plusieurs idées de stratégies que nous n'avons pas pu implémenter, que nous expliciterons dans la partie Analyse.

Finalement, nous avons une implémentation d'une stratégie prudente qui n'est pas optimale et qui pourrait être affinée.

IV- Analyses de la stratégie prudente du jeu du Troll et des autres stratégies

Avant l'implémentation de la stratégie prudente du jeu du troll, nous avons réfléchi à des implémentations de stratégies pures pour ce jeu, qui pourraient être pertinentes.

Une idée de stratégie pure était d'envoyer un nombre maximal de pierres, sur un nombre de tours égal au nombre de cases que le troll doit parcourir pour faire gagner le joueur de la stratégie. Un exemple : sur un jeu à 15 pierres par joueur, et 7 cases, le troll doit, en théorie, avancer de trois cases pour qu'il atteigne le château d'un des joueurs. L'idée de la stratégie est donc d'envoyer le maximum de pierres tout en se laissant la possibilité de jouer 3 fois. Il va donc envoyer 5 pierres. Cette stratégie se trouve dans le module des stratégies : la fonction StrategieAgressive ()

L'inconvénient de cette stratégie est que si l'on sait comment elle marche, il est tout à fait possible de la contrer totalement. Cette dernière est, de plus, très prévisible (nombre de pierres lancées constant)

Une autre façon de penser serait de se dire que si le troll se trouve à proximité du château d'un joueur, et que ce dernier dispose des pierres suffisantes, il devrait empêcher l'adversaire de faire bouger le troll de nouveau en envoyant le nombre de pierres dont l'adversaire dispose.

Le problème est que cette stratégie ne s'applique que dans le cas où un joueur se trouve en difficulté. De plus, elle est également très facile à contrer si l'on connait son implémentation.

Dans ce jeu, on peut donc en conjecturer que si un joueur joue une stratégie pure, il se fera contrer par un joueur adverse connaissant la stratégie. Ainsi, nous pouvons penser qu'il n'y a pas de façon optimale de jouer en stratégies pures, ni d'équilibre de Nash en stratégie pure puisque toutes les stratégies pures se font contrer par une autre stratégie pure. Nous avons donc implémenté une stratégie prudente, ou optimale, en stratégie mixte.

Pour cela, nous nous sommes appuyés sur une matrice de gains, et avons calculé, pour chaque case, une valeur comprise entre -1 et 1, qui représente le gain garanti, à l'aide, soit de cas triviaux (situations gagnantes / perdantes), soit de résolutions de problèmes de programmation linéaire.

Une fois la stratégie optimale implémentée, nous avons fait différents tests, basés sur un grand nombre parties, afin d'avoir une proportion de victoires pour chaque partie qui est parlant :

Les temps d'exécutions pouvant être très longs, certains tests seront effectués sur un nombre limité d'itérations, dans le but que l'on se fasse une idée de l'efficacité de notre stratégie pour une situation donnée.

Test 1 : Stratégie prudente et Stratégie aléatoire avec la configuration initiale (15,15,0) avec 7 cases

Nos premiers tests se sont basés sur la confrontation d'une stratégie aléatoire face à la stratégie prudente.

Avec Strategie1 qui correspond à notre stratégie optimale et Strategie2 qui correspond à une stratégie aléatoire, voici les résultats que nous avons obtenu :

```
Score :  Strategie 1 :  1098  Strategie 2 :  408  Matchs nuls :  94
Finished in :  6205.77  seconds
```

Observations :

On remarque que la stratégie prudente gagne dans 60% des cas environ. On peut donc affirmer que la stratégie prudente est meilleure que la stratégie aléatoire mais n'est pas infaillible.

Nous pensons également que notre implémentation de stratégie prudente n'est pas optimale, et que la stratégie prudente devrait, en théorie, avoir un score encore meilleur.

Cependant, à cause de l'aléatoire, la stratégie prudente ne peut pas gagner dans 100% des cas.

Test 2 : Stratégie prudente et Stratégie agressive avec la configuration initiale (15,15,0) avec 7 cases

Ensuite, nous avons voulu comparer notre « stratégie pure prudente » avec la stratégie optimale.

Avec Strategie1 qui correspond à notre implémentation de la stratégie mixte optimale et Strategie2 qui correspond à notre « stratégie agressive », voici les résultats que nous avons obtenu :

```
Score final :  Strategie 1 :  100  Strategie 2 :  0  Matchs nuls :  0
```

Observations :

On remarque que la stratégie prudente gagne dans 100% des cas . La stratégie prudente domine notre stratégie agressive. Cela est dû au fait que si la stratégie agressive joue toujours la même chose.

Test 3 : Stratégie prudente pour les deux joueurs avec la configuration initiale (15,15,0) avec 7 cases

Nous voulions voir comment réagirait notre stratégie si elle est jouée contre elle-même. Avant de voir les résultats, nous nous attendons à ce que les deux stratégies gagnent un nombre équivalent de manches, voire qu'il y ait systématiquement match nul.

```
Score final :  Strategie 1 :  0  Strategie 2 :  0  Matchs nuls :  10
```

Observations :

C'est bien ce que nous observons. Cela implique également que le jeu est équitable si les deux joueurs jouent optimalement.

Test 4 : Stratégie prudente contre une stratégie aléatoire avec la configuration initiale (5,5,0) avec 7 cases

Nous allons voir comment va se comporter la stratégie prudente avec un nombre de pierres très faible. Avec Strategie 1 qui correspond à notre stratégie prudente et stratégie 2 qui correspond à une stratégie aléatoire, voici le résultat :

```
Score final :  Strategie 1 :  35  Strategie 2 :  0  Matchs nuls :  65
```

Observations :

On remarque que le nombre de matchs nuls est beaucoup plus élevés lorsque le nombre de pierres par joueur diminue.

Test 5 : Stratégie prudente contre une stratégie aléatoire avec la configuration initiale (10,10,0) avec 7 cases

Nous allons voir le comportement de la stratégie avec un nombre de pierres supérieur à l'exemple du dessus mais inférieur à l'exemple par défaut (15)

```
Score final :  Strategie 1 :  58  Strategie 2 :  26  Matchs nuls :  16
```

Observations :

On remarque que la stratégie est moins efficace à mesure que le nombre de pierres baisse, jusqu'à tendre vers une stratégie aléatoire quand le nombre de pierres baisse.

Test 6 : Stratégie prudente contre une stratégie aléatoire avec la configuration initiale (20,20,0) avec 7 cases

Nous allons vérifier si un nombre de pierres plus élevé va améliorer l'efficacité de la stratégie prudente.

```
Score :  Strategie 1 :  59  Strategie 2 :  38  Matchs nuls :  3
```

Observations :

En réalité, les proportions de victoire pour un grand nombre de pierres restent constantes. Cependant, les matchs nuls sont moins fréquents étant donné que plus il y a de pierres, moins la probabilité que les stratégies jouent la même chose est élevée.

Test 7 : Stratégie prudente par simplex et stratégie aléatoire avec la configuration initiale (13,15,0) avec 7 cases

Nous allons voir comment réagit la stratégie prudente en situation de désavantage en termes de nombre de pierres :

```
Score final :  Strategie 1 :  21  Strategie 2 :  20  Matchs nuls :  9
```

Observations :

Nous remarquons que malgré le désavantage initial, la stratégie prudente peut tout de même gagner dans la moitié des cas.

Test 8 : Stratégie prudente par simplex et stratégie prudente par SLSQP avec la configuration initiale (15,15,0) avec 7 cases

Au cours de ce projet, nous avons eu l'occasion de tester deux versions de la stratégie prudente, basées sur des algorithmes différents. En réalité, les deux stratégies donnent toujours la même valeur de gain. C'est dans les variations des probabilités qu'il y a quelques changements. On peut donc penser que les stratégies sont équivalentes, qu'il s'agit seulement de deux façons différentes de calculer la même chose.

```
Score final :  Strategie 1 :  0  Strategie 2 :  43  Matchs nuls :  7
```

Observations :

Finalement, on remarque que la stratégie du simplex est dominée par la stratégie basée sur le SLSQP. On peut en déduire que la stratégie SLSQP est meilleure que la stratégie par simplex. En réalité, cela dépend de la stratégie : contre notre stratégie agressive, la stratégie par Simplex gagne dans 100% des cas alors que l'autre stratégie dans 60% des cas. De plus, notre nouvelle stratégie prudente réagit mieux en situation critique que notre ancienne stratégie.

Autre idée de stratégie :

Une idée de stratégie serait d'avoir une stratégie basée sur l'apprentissage : On peut donner l'exemple d'un réseau de neurones, dont les différentes pondérations dépendent, initialement de la stratégie prudente, mais qui va ajuster les différents poids de ses neurones en fonction des victoires et des défaites qu'il subit. Il aurait été intéressant de confronter ce type de stratégie à la stratégie prudente pour voir si ce genre d'optimisation est pertinent ou si la stratégie prudente reste la meilleure alternative.

En tant que prolongement pour ce projet, cela aurait été la stratégie que nous implémenterions et testerions après avoir amélioré notre stratégie prudente.

TP noté théorie des jeux

Exercice 1 :

Les résultats suivants ont été trouvés à partir de la fonction Exercice1() du fichier main.py.

Pour la configuration (25,21,-1), la distribution de probabilités est la suivante, dans l'ordre décroissant du nombre de pierres lancées :

[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0.]

Remarque : notre stratégie prudente n'étant pas correctement implémentée, la stratégie du joueur 1 n'est pas rationnelle lorsque le joueur 1 est trop en difficulté. Cette distribution de probabilités n'est pas cohérente.

Pour le gain : -1

Pour la configuration (25, x,-1), le jeu est favorable au joueur 2 à partir de x = 15.

Cela est cohérent car la position du troll est plus du coté du joueur 1, donc le nombre de pierres minimal du joueur 2 doit être inférieur à 25

Exercice 2 :

La stratégie prudente est une stratégie mixte. Elle a, pour principal intérêt, de maximiser les gains du joueur qui la joue, quelque soit la stratégie jouée par le joueur adverse . Cependant, nous pouvons mesurer l'efficacité de la stratégie prudente seulement sur un grand nombre d'itérations. En tant que stratégie optimale, la stratégie prudente est supposée être efficace face à une stratégie qui intègre

de l'IA. Théoriquement, les meilleures stratégies issues de l'IA tendront vers la stratégie prudente. Il s'agit du principe du théorème MinMax : le jeu du troll étant un jeu à somme nulle (les gains du joueur 1 correspondent aux pertes du joueur 2), la stratégie prudente est optimale.

Exercice 3 :

Une stratégie a été ajoutée dans le module Strategies.py : Il s'agit de la fonction StrategieAleatoireExercice3(n).

Cette dernière va lancer une pierre aléatoirement entre 1 et n2/3.

Nous allons l'incorporer au jeu afin que le joueur 2 l'utilise.

Les résultats décrits ci-après peuvent être retrouvés dans la fonction Exercice3() du module main.py.

a)

Les résultats suivants ont été trouvés à partir de la fonction Exercice3() du fichier main.py :

Passer par un simplex nous donne un gain général du joueur 1 si le joueur 2 joue optimalement. Dans le cas ou le joueur 1 connait la distribution de probabilités du joueur 2, le gain du joueur 1 peut être meilleur :

Pour trouver le gain du joueur 1 sachant que le joueur 2 joue aléatoirement, nous avons fait comme ceci :

- Pi la probabilite de lancer n1-i pierres pour le joueur1
- Pj la probabilite de lancer n2-j pierres pour le joueur2
- Gij le gain quand il reste i pierres au J1 et j pierres au J2
- G = somme des i (Xi*(somme des j (Xj*Gij)))

On trouve un gain de 2/3.

b) En jouant (n2/3) + 1 pierres pour le joueur 1, le troll va constamment avancer vers le joueur 2.

On peut donc construire une stratégie qui s'approche du gain de 1 en permettant au joueur 1 d'envoyer un nombre constant de pierres. L'algorithme du simplex n'est donc pas nécessaire, et la stratégie est déterministe : elle va toujours jouer la même chose. Nous avons implémenté cette stratégie dans une fonction du module Strategies.py : il s'agit de la fonction `StrategieContreExercice3()`. Après avoir modifié la stratégie de chaque joueur dans le fichier JeuDuTroll.py, nous avons lancé notre main avec Strategie 1 la stratégie que nous venons de décrire et Strategie 2 la stratégie du joueur 2 qui consiste à envoyer un nombre de pierres aléatoire entre 1 et n2/3.

```
Score final :  Strategie 1 :  995  Strategie 2 :  5  Matchs nuls :  0
```

On remarque que le joueur 1 gagne dans 99,5% des cas, sur 1000 itérations. Cela est meilleur que la stratégie prudente, qui est de 2/3.

Il est à noter que cette stratégie n'est pas rationnelle et n'est pertinente que si le joueur adverse n'est pas rationnel non plus (comme c'est le cas ici). Cependant, contre un joueur rationnel, la stratégie prudente aura de meilleurs résultats :

En confrontant notre stratégie prudente face à cette stratégie, nous avons le résultat suivant :

```
Score final :  Strategie 1 :  10  Strategie 2 :  0  Matchs nuls :  0
```

Le temps d'exécution étant plus long, nous avons arrêté le nombre d'itérations assez tôt dans les tests. Ce test ne présente donc pas de valeur de gains de la stratégie prudente sur la stratégie que nous venons d'implémenter qui est fiable. Cela nous permet, cependant, d'avoir un aperçu de l'efficacité de la stratégie prudente sur notre nouvelle stratégie.

Exercice 4 :

Toutes les modifications effectuées dans le code pour cette partie seront regroupées dans un nouveau module « Exercice4.py ». Il va s'agir de fonctions similaires aux fonctions déjà existantes dans le TP, mais avec des modifications qui seront explicitées ci-après, ainsi qu'en commentaire dans le code. Il est à noter qu'étant donné notre stratégie prudente non optimale, les résultats obtenus ne seront pas nécessairement exacts. Les exécutions de tests qui donnent les valeurs suivantes sont disponibles dans la fonction Exercice4 de main.py (les temps d'exécutions peuvent être très longs) :

Pour le cas où le joueur 2 ne peut lancer qu'un nombre impair de pierres :

- Ajout de la fonction PartieImpair, analogue à la fonction Partie de JeuDuTroll.py . Cette dernière va s'assurer que le joueur 2 lance un nombre impair de pierres, par une vérification de l'input, ou alors par une boucle qui va vérifier que la stratégie du joueur 2 renvoie bien un coup impair (limite de 500 pour éviter les boucles infinies).
- De même pour la fonction MatriceGainsImpair, analogue à la fonction MatriceGains du module Solveur.py : certains coups ne sont pas disponibles pour le joueur 2. Donc, il faut modifier la taille de la matrice de gains pour correspondre aux coups disponibles par le joueur2. (des valeurs infinies resteront dans la matrice si la configuration est impossible). Cela aura pour effet de réduire le nombre de contraintes du simplexe.
- Enfin, pour la fonction StrategiePrudenteImpair, analogue à la fonction StrategiePrudente du module Strategie.py, nous avons appelé la fonction MatriceGainsImpairs pour calculer la matrice de gains en fonction des nouvelles règles et enlevé les colonnes de valeurs infinies.

Pour un jeu (20,20,0) avec cette modification :

La distribution de probabilites est la suivante :

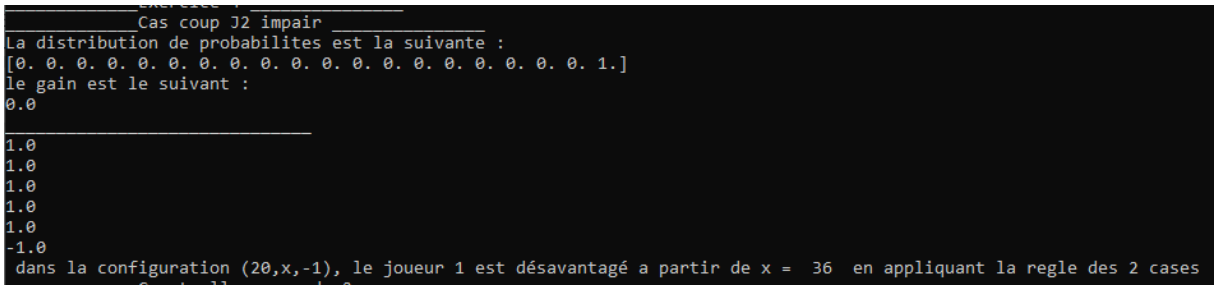
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]

le gain est le suivant :

0.0

Remarque : Le gain du joueur 1 devrait être supérieur à 0 en théorie, cependant, étant donné notre stratégie prudente, nous trouvons cette valeur.

G(20,x,0) < 0 pour x = 36



Pour le cas où le troll avance de 2 cases vers le joueur 2 si le joueur 1 lance 2 pierres de plus que le joueur 2 :

- Ajout de la fonction Partie2Cases, qui va faire avancer le troll de 2 cases si le joueur 1 a lancé au moins 2 pierres de plus que le joueur 2, lors de la modification de la position du troll.
- Ajout de la fonction MatriceGains2Cases qui prend en compte les modifications afin la vérification de la position du troll
- Ajout de la stratégie associée StrategiePrudente2Cases.

Pour un jeu (20,20,0) avec uniquement cette modification par rapport à la version de base, on a (avec nombre de pierres lancees dans l'ordre décroissant):

La distribution de probabilites est la suivante :

[0.01580428 0.03160857 0.03371972 0. 0. 0.07796585

0.04268479 0.00725107 0.05894699 0.06244104 0.04440357 0.05720574

0.0652094 0.06424103 0.06689071 0.07033293 0.07233913 0.07413038

0.07586136 0.07896344].

le gain est le suivant :

0.9030631374009039

G(20,x,0) < 0 pour x = 36.

Pour le cas où le joueur 2 vise mal avec une certaine probabilité :

- Nous avons créé une fonction PartieViseMal qui va vérifier, pour chaque pierre lancée par le joueur2, si la pierre qu'il a lancé a toucher sa cible, selon un nouveau paramètre.
- Faute de temps, nous ne sommes pas parvenus à implémenter la stratégie prudente dans ce cas-là. Nous pensons, cependant, qu'il s'agit de la même matrice que le jeu de base, qui présente, finalement, les gains du joueur 1 dans le pire cas (le cas où le joueur 2 touche toujours sa cible). Nous pensons donc, à cet effet, que la stratégie prudente ne serait pas changée non plus.