haute école neuchâtel berne jura arc ingénierie www.he-arc.ch

POKER-FACE

PokerFace

Rapport Technique

Titre	PokerFace : Rapport technique
Superviseurs	Aïcha Rizzotti & Yassin Aziz Rekik
Étudiants	Diego Antognini, Alexandre Perez, Danick Fort
Début du projet	22 février 2013
Fin du projet	14 juin 2013
Version	1.0
Dernière modification	13 juin 2013

Table des matières

1 Abstract 2
2 Introduction 3
3 Conception 4
3.1 Interface graphique 4
3.2 Moteur de jeu 12
4 Division du pot 18
5 Intelligence Artificielle 19
5.1 Concept de base 19
5.2 Simulations 21
5.3 Suivre/checker ou se coucher ? 22
5.4 Miser/Relancer ? 30
5.5 Améliorations possibles 32
6 Planification 34
6.1 Planification initiale 34
6.2 Planification finale 35
6.3 Constatations 35
7 Conclusion 36
8 Bibliographie 37

1 Abstract

Ce projet implique la réalisation d'un jeu de *Poker* hors-ligne avec une représentation graphique, confrontant un joueur humain face à un ou plusieurs joueurs contrôlés par une intelligence artificielle. Un système de profil pour le joueur est aussi mis en place. Le cœur du projet réside en la réalisation de l'intelligence artificielle, bien que la réalisation du moteur soit tout aussi importante !

La problématique principale est d'avoir une intelligence artificielle qui effectuerait des choix cohérents tout en essayant de perdre le moins possible et donc d'avoir un profit maximal. Pour finir, elle est considérée comme efficace si elle gagne la plupart des parties !

Pour cela, nous avons réalisé dans un premier temps un système de simulations en temps réel, ce qui permet à nos intelligences artificielles de pouvoir effectuer des choix selon certaines probabilités. De plus, à partir de ces données, elles prennent en compte d'autres facteurs comme la mise à effectuer et la quantité déjà misée. Par la suite, nous avons réalisé des fonctions 4D pour chaque état, qui permettent selon tous ces facteurs de donner une probabilité de suivre l'enchère en cours. De plus, elle sait aussi relancer/miser si elle possède une bonne main ou qu'elle veut introduire la notion de bluffs. Pour finir, chaque intelligence artificielle possède un coefficient de risque, la rendant plus prudente ou plus agressive.

Une fois notre intelligence artificielle mise en place, nous avons constaté que la plupart du temps elle est agressive et qu'elle gagne souvent. L'ensemble des parties réalisées pour conclure sur ce résultat s'est passé à « jeu ouvert », c.-à-d. que nous connaissions les cartes de nos adversaires. Nous avons aussi pu constater que bien souvent le choix était judicieux, il pouvait y avoir certains résultats surprenants (mais pas forcément mauvais), introduit avec le facteur « chance », rendant notre intelligence artificielle imprévisible.

2 Introduction

Le *Poker* est un jeu populaire à travers le monde. Il est joué à tous les niveaux, de l'amateur au professionnel. Il existe beaucoup de tournois internationaux et amateurs et des dizaines de sites Internet proposent le *Poker* compétitif multi-joueurs. Le problème du *Poker* en ligne est qu'une connexion internet est nécessaire ; c'est pourquoi nous avons voulu proposer une version hors-ligne opposant le joueur à différents adversaires contrôlés par l'ordinateur.

Notre projet P2 Java consiste en la réalisation d'un jeu de *Poker*, dans sa variante *Texas Hold'em*,[1] dans lequel le joueur sera confronté à un ou plusieurs adversaires, contrôlés par une intelligence artificielle. La motivation principale du projet est la réalisation d'une intelligence artificielle qui serait capable de jouer correctement contre un joueur humain. Cela signifie qu'elle devra effectuer des choix cohérents selon ce qu'il se passe sur la table, ce qu'elle a en main, ainsi que la gestion de son argent. Au final, l'intelligence artificielle est considérée comme réussie dans le cas où elle devient imprévisible, qu'elle arrive à éliminer les autres joueurs et à réaliser des profits.

Un autre aspect intéressant dans ce projet fut la réalisation du moteur de jeu : le poker contient une certaine quantité de règles, qu'il faut tenir compte pour avoir la meilleure expérience de jeu possible. Outre les règles, il fallait aussi trouver un système efficace et léger qui permet de gérer autant bien 2 que 10 joueurs (joueurs maximum en général à une table).

En relation avec nos intelligences artificielles, nous proposons aussi des statistiques aux joueurs sur sa main, c.-à-d. les probabilités qu'il a de gagner/faire égalité/perdre et d'avoir chaque main particulière à chaque tour de jeu. Pour cela, nous avons réalisé un système de simulations en temps réel dont le joueur voit les résultats à chaque tour et les intelligences artificielles les utilisent.

Pour finir, nous avons dû aussi réaliser une interface graphique qui représente l'ensemble du système ainsi qu'un gestionnaire de profils pour le joueur.

Dans ce présent document, nous parlerons de la conception de notre système, de celle de l'interface graphique, puis de celle du moteur du jeu. Dans la conception, il y aura des descriptions de l'interface graphique ainsi que des diagrammes de classes, de séquences et des cas d'utilisations. Ensuite, nous parlerons de la division du pot, puis nous attaquerons le vif du sujet : notre intelligence artificielle et nous proposerons des pistes d'améliorations. Pour finir, nous comparerons notre planification initiale avec la planification finale afin de ressortir les différences qu'il y a eues, pour avoir de meilleures estimations lors de prochain projet.

Afin de pouvoir juger la qualité de notre travail et des efforts fournis, il est fortement conseillé de connaître un minimum les règles du *Poker Texas Hold'em No limit*. Certains mots utilisés se réfèrent directement au champ lexical du *Poker*.

Si vous ne connaissez pas ou que vous aimeriez un rafraîchissement sur les règles, nous vous conseillons, pour vous aider dans cette tâche, les liens suivants :

- Pour les règles des enchères : https://en.wikipedia.org/wiki/Betting_in_poker
- Pour les autres : https://en.wikipedia.org/wiki/Texas_hold%27em#Rules
- Pour les termes du poker : http://en.wikipedia.org/wiki/Glossary_of_poker_terms

[1] Vous trouverez une description plus approfondie du jeu sur https://en.wikipedia.org/wiki/Texas_hold_'em

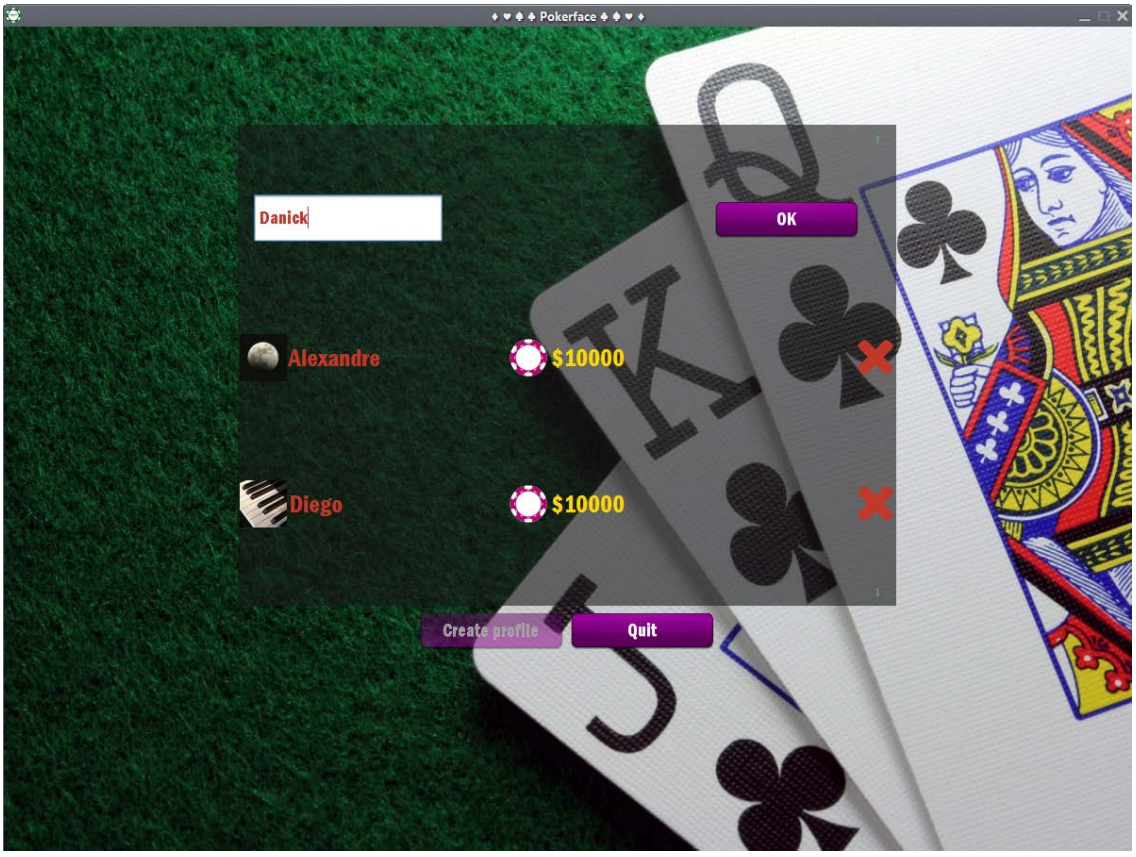
3 Conception

Dans cette partie se situe un ensemble de diagrammes UML considérés comme les plus utiles dans notre cas ainsi que des captures d'écran représentant notre interface graphique. Dans ces diagrammes, l'on retrouve des diagrammes de cas d'utilisation, de classes ainsi que de séquences. L'ensemble des diagrammes a été réalisé avec le logiciel *Astah Community*. Le diagramme de classe final est disponible en annexe 3.3.1.

3.1 Interface graphique

Dans les prochaines lignes, les différentes fenêtres de l'interface graphique y seront présentées. Nous les décrirons à l'aide d'une capture d'écran, puis de manière textuelle et les compléterons à l'aide d'un diagramme de cas d'utilisation. A la fin, nous vous présenterons le diagramme de classes ainsi que des commentaires appropriés.

- **Menu d'accueil**



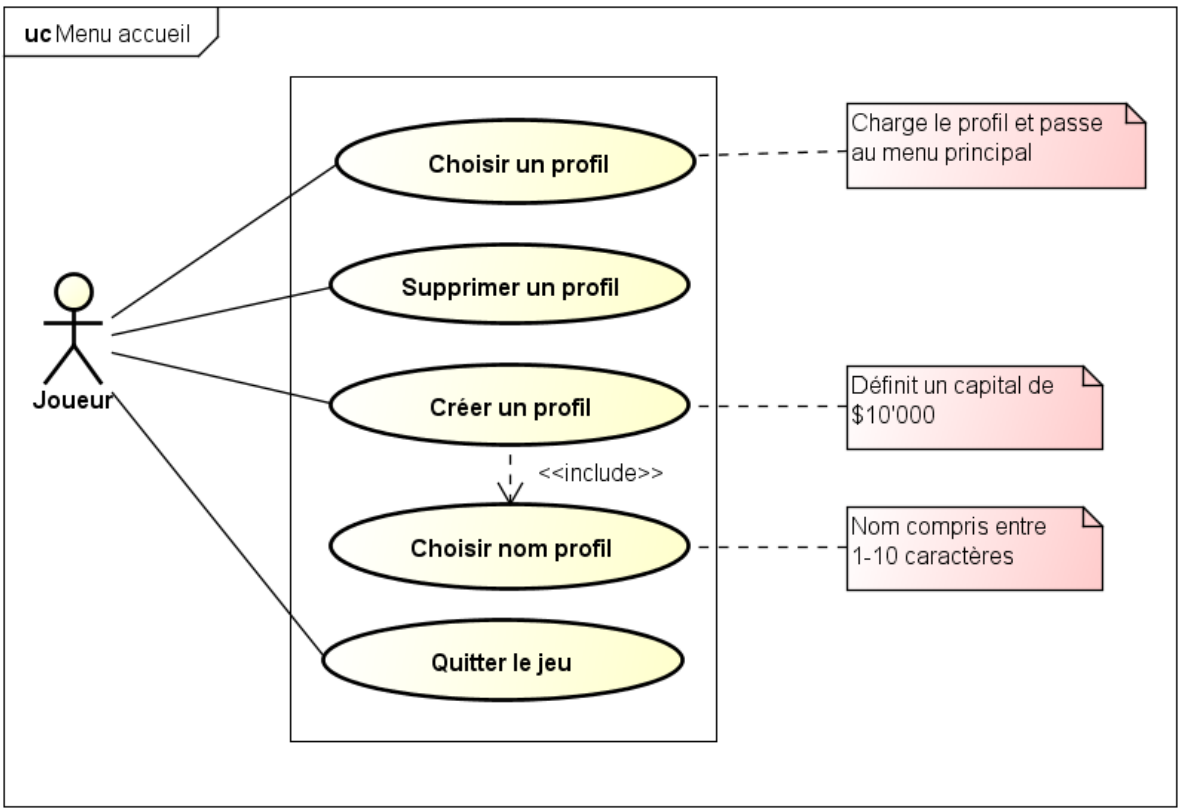
Rendu du menu accueil, Capture 1

Au lancement de l'application, il y a un menu pour choisir son profil. Dans ce menu, nous pouvons voir les différents profils avec leur nom, leur avatar ainsi que leur capital. L'utilisateur peut supprimer un profil à tout moment. Ce système permet de gérer plusieurs utilisateurs sur un même ordinateur ou plusieurs profils pour un même utilisateur. Il est possible de *scroller* dans la liste pour atteindre d'autres profils.

S'il le désire, il peut créer un nouveau profil à l'aide du bouton « Create profile », tout en pouvant choisir son nom. Un capital de départ est attribué. Quant à l'image, elle est définie de manière aléatoire. S'il le souhaite, il peut quitter l'application en cliquant sur le bouton « Quit ».

Pour choisir un profil, l'utilisateur doit effectuer un double-clic sur le profil désiré.

Dans la capture 1, on peut voir que l'utilisateur est en plein processus de création de profil.



Menu accueil, Diagramme de cas d'utilisation 1

Lorsque l'utilisateur choisit un profil, le système s'occupe de le charger puis de l'associer au joueur au cours de ses jeux. Supprimer un profil retirera toutes traces de celui-ci dans l'application.

Lorsqu'un nouveau profil est créé, le système lui attribue un capital de départ de 10'000.- ainsi qu'une image et lui associe le nom choisi (qui contiendra un nombre de caractères, compris entre 1 et 10).

Lorsque le joueur quitte l'application, celle-ci va sérialiser les profils dans un fichier, ce qui permettra de pouvoir les charger lors de la prochaine utilisation.

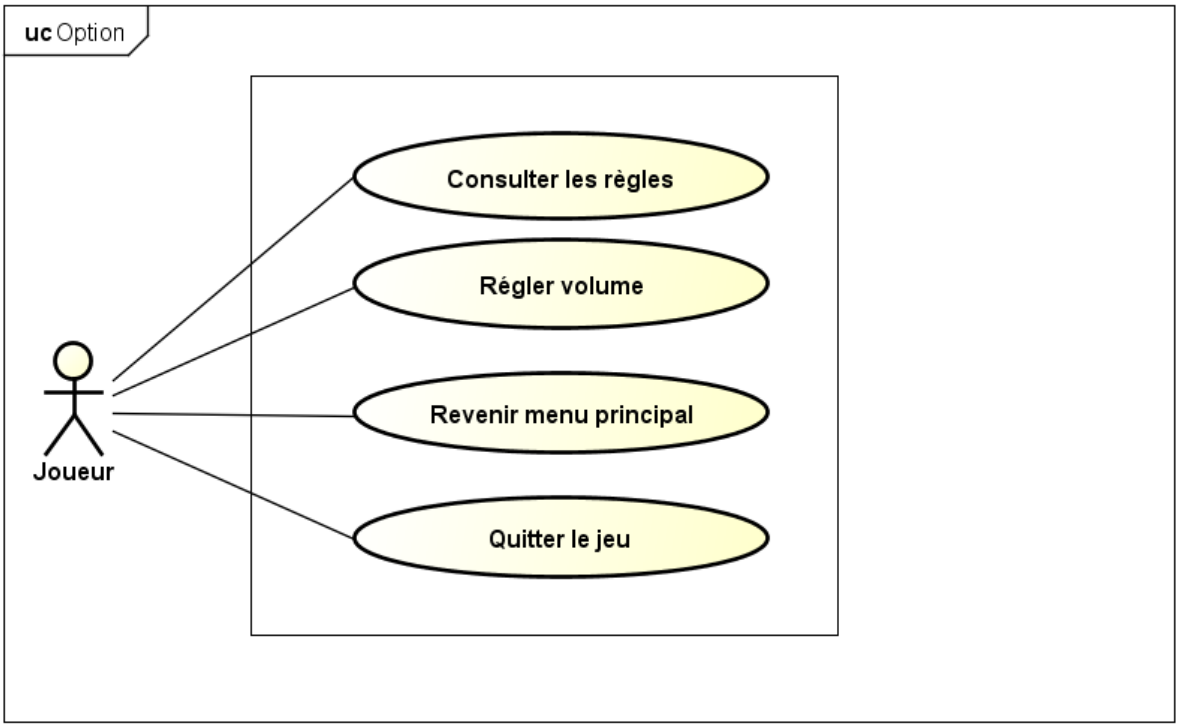
- **Options**

Barre d'options, Capture 2

Une fois le profil sélectionné, on peut voir qu'il apparaît dans le bandeau à option, placé en haut de l'écran. Cette barre présente les différentes informations du profil, tels que son nom, son avatar ainsi que son capital. Un bouton « ? » affiche une boîte de dialogue contenant les règles du *Poker* permettant aux utilisateurs néophytes de s'informer. De l'autre côté de cette barre, se situe la barre du volume et l'heure actuelle de l'ordinateur.

Un *slider* permet de régler le niveau de volume à tout moment. Il est important pour le joueur d'avoir un contrôle sur l'environnement sonore pour jouer le plus confortablement possible.

Pour finir, l'utilisateur peut revenir au menu d'accueil (Capture 1) afin de pouvoir changer de profil. La dernière icône permet de quitter l'application.

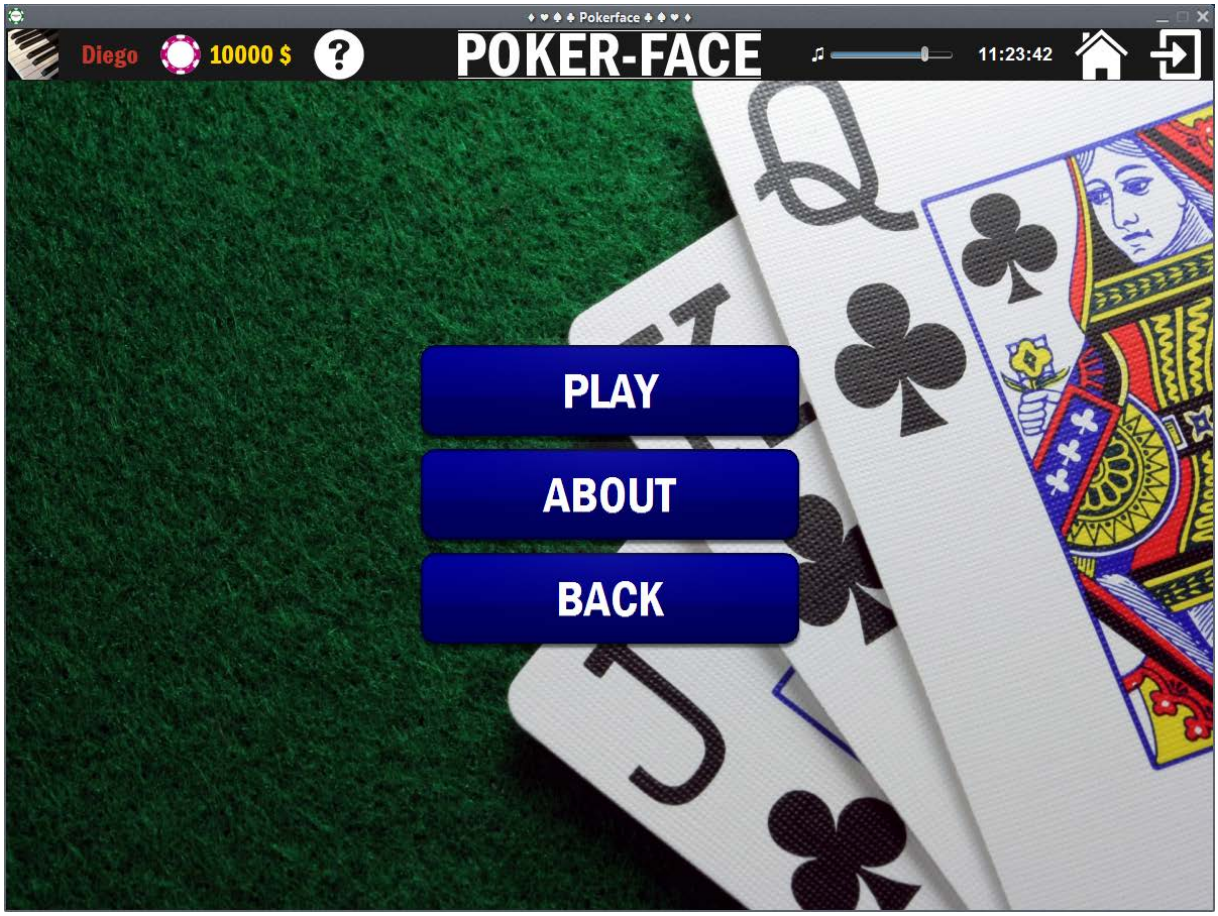


Barre d'options, Diagramme de cas d'utilisation 2

Concernant le volume, la position du rond représente une valeur sur une échelle logarithmique. Toute la musique du jeu y est adaptée. Plus le rond se situe à gauche, plus le son est faible et inversement, plus le rond est à droite, plus il est fort.

A tout moment, l'utilisateur a la possibilité de revenir au menu d'accueil (Capture 1), afin de pouvoir choisir un autre profil. Dans le cas où le joueur serait en pleine partie, seul son capital restant sera conservé et sa mise sera considérée comme perdue.

- **Menu principal**



Rendu du menu d'accueil, Capture 3

Le joueur peut consulter les crédits de l'application (bouton « About ») et revenir dans le choix de profil avec le bouton « Back » (en plus du bouton dans la barre d'options).

Quand le joueur se sent prêt, il peut lancer une partie en cliquant sur le bouton « Play ». Celui-ci amène un *pop-up* (Capture 4) permettant de configurer les différentes options de la table.

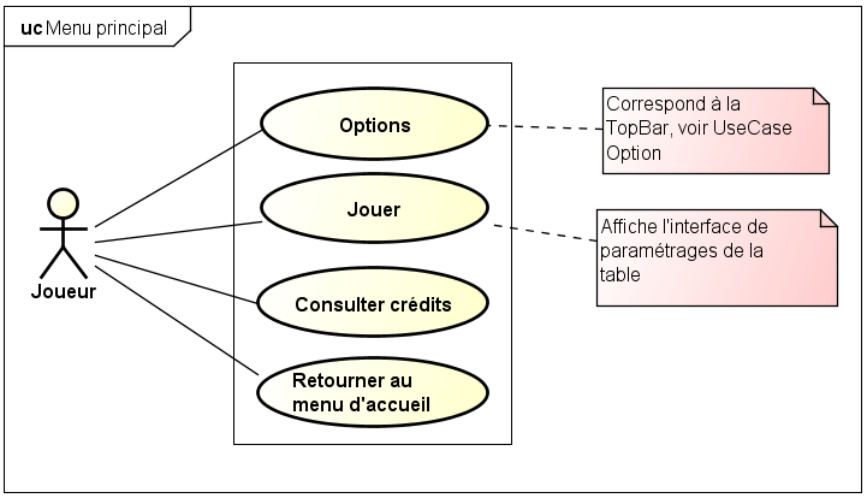
Rendu du menu d'accueil, Capture 4

Ces options sont notamment les *blinds*[2], le nombre de joueurs total (minimum 2 et maximum 10), ainsi que la bourse de départ pour pouvoir entrer sur la table.

En cliquant sur le bouton « Cancel », l'utilisateur revient au menu principal.

[2] Les *blinds* ont pour but de forcer des joueurs à miser une certaine somme lors du *preflop*. Généralement il y a une *small blind* et une *big blind*. A chaque tour, les *blinds* changent de joueur.

Une fois les options entrées et validées, le joueur arrive sur la table de jeu.

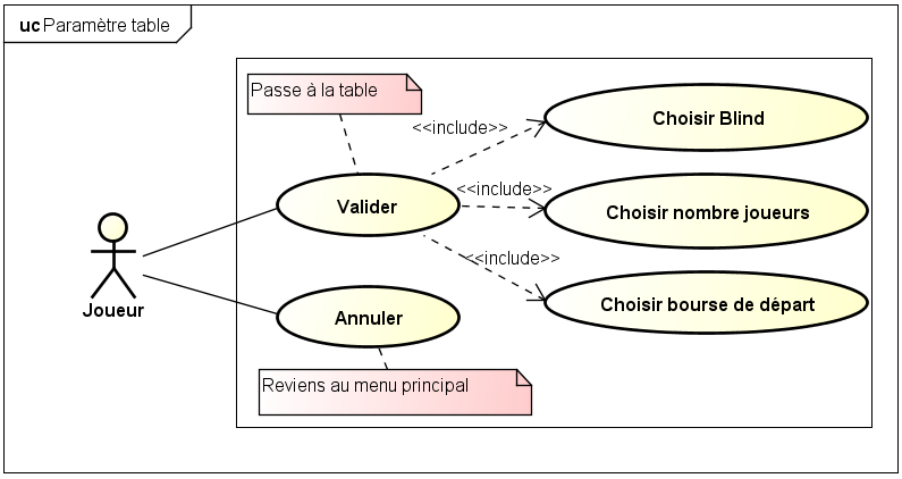


Menu principal, Diagramme de cas d'utilisation 3

Le joueur peut à tout moment changer les options (voir Diagramme de cas d'utilisation 2).

Si l'utilisateur le souhaite, il a la possibilité de consulter les crédits ou de changer son profil en revenant sur la page d'accueil (voir Capture 1).

Pour finir, il peut lancer une partie, et donc arriver sur l'interface de configuration de la table (voir Diagramme de cas d'utilisation 4).



Paramètre de table, Diagramme de cas d'utilisation 4

Pour pouvoir lancer une partie, le joueur doit choisir au préalable :

- la taille de la *small blind* (la *big blind* est calculée en conséquence), qui doit être inférieur au quart de la bourse de départ,
- le nombre de joueur, de 2 à 10 joueurs,
- la bourse de départ, pour pouvoir entrer sur la table.

Si l'utilisateur le souhaite, il a la possibilité de revenir au menu principal (Capture 2), en annulant le paramétrage.

- **Table**

Rendu de la table, Capture 5

Une fois les options de la table réglées, nous arrivons sur la table de jeu. Dans celle-ci, se retrouve à nouveau la barre d'option. Dans la partie du haut, les joueurs sont répartis de manière uniforme, le long d'une ellipse. L'utilisateur ne changera jamais de place, et donc aura toujours ses cartes dans la partie du bas.

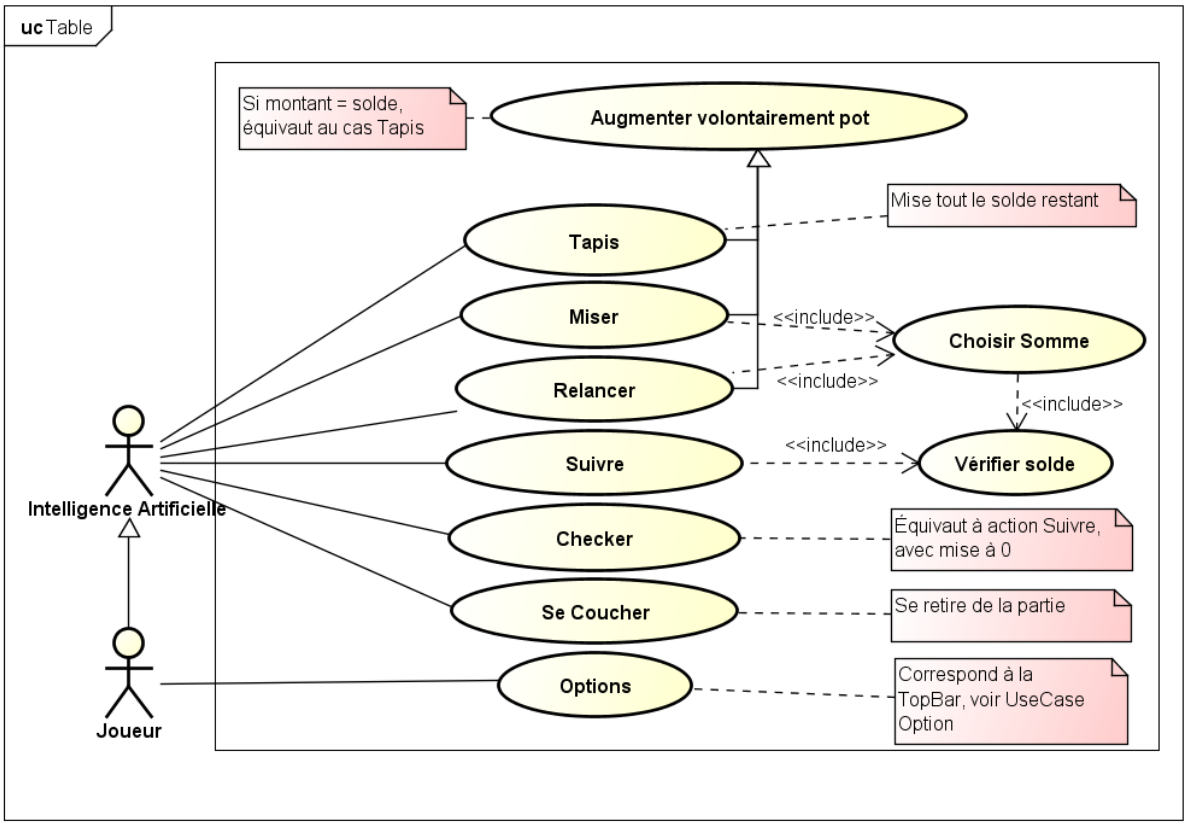
Au centre de cette ellipse se situe le *board*[3], dans lequel les cartes apparaissent. A droite de celle-ci, l'on peut distinguer la mise à payer pour suivre (*current bet*), le montant de l'état (*state total*) ainsi que celui de la partie (*turn total*).

Pour chaque opposant, leurs cartes sont cachées, puis éventuellement retournées à la fin du jeu. Au-dessus des cartes des joueurs, on peut y distinguer l'argent restant (en jaune), l'argent investi dans la partie actuelle (en bleu) ainsi que celui investi dans l'enchère actuelle (en vert).

Chaque joueur a, à un certain moment de la partie, un jeton. Ce jeton peut être celui du *dealer*, *small blind* ou *big blind*. A chaque fin de tour (c.-à-d. quand une personne a remporté le pot), ces jetons changent de place, en allant au voisin de gauche, dans le sens des aiguilles d'une montre.

Dans la partie du bas se situe des boutons représentants les différentes actions que peut effectuer le joueur. Leur valeur sont mises à jour à chaque fois que le joueur doit jouer. Le *slider* à gauche permet de modifier la valeur à miser/relancer. A côté de ces boutons, on peut distinguer, à gauche une aide permettant au joueur d'évaluer sa main Et à sa droite, l'on peut observer un journal qui écrit chaque action effectuée sur la table, indiquant le gagnant de la partie avec les mains ainsi que les gains de chaque joueur. Pour finir, tout à droite, on peut voir des statistiques à l'état actuel, servant à informer le joueur quant à la position de sa main et ce qu'il peut espérer.

[3] Le *board* représente l'endroit où les cartes du *flop*, *turn* et *river* seront disposées.



Table, Diagramme de cas d'utilisation 5

A ce moment de l'application, un deuxième acteur entre en compte, il s'agit de l'intelligence artificielle (qui peut être au nombre de 1 à 9). Le joueur et elle ont les mêmes possibilités de jeu, la seule différence se situe au niveau des options qu'elle n'a pas, car comme son nom l'indique, cet acteur n'est pas humain. Pour simplifier les explications, nous considérons que l'intelligence artificielle est un joueur.

Le joueur peut à tout moment se coucher, ce qui le retire du tour en cours. Il peut aussi effectuer un tapis, ce qui misera l'ensemble de son solde.

Lorsqu'aucune mise n'a encore eu lieu, il peut miser ou checker. Checker n'inclut aucune mise et est considéré comme « suivre » avec une mise nulle. Concernant « miser », si celui-ci mise tout son solde, alors l'action sera considérée comme un tapis.

Dans le cas contraire, le joueur peut suivre ou relancer. La relance est toujours disponible dans ce cas, et inclut que le joueur paye la mise et en ajoute une nouvelle.

Toute action incluant une mise supérieure à 0, si le joueur n'a pas assez d'argent pour suivre, sera considérée comme tapis et il jouera dans un pot à part.

- **Diagramme de classes**

Les commentaires se réfèrent tous à l'annexe 3.1.1

L'interface graphique est basée sur la technologie Swing de Java. Chaque écran de jeu est encapsulée dans une classe JFrame appelée « JFrameMain ».

- Menu Screens

Dans le package « Menu Screens » seront présents les différents menus accessibles. Chaque écran est une classe dérivée de « JPanel ».

Le premier écran qui est présenté à l'utilisateur est « JPanelProfile ». Dans celui-ci, l'utilisateur pourra sélectionner un profil déjà créé, en supprimer un ou encore en créer un nouveau.

« JPanelMainMenu » est le panel du menu principal du jeu. Il apparait après avoir sélectionné un profil. Deux options sont disponibles : jouer ou quitter le jeu.

Si l'utilisateur décide de jouer, une boite de dialogue dérivé de « JOptionPane », appelée « JOPTableConfiguration » apparait, qui contiendra les paramètres de la table de jeu.

- Profile

Le menu de sélection de profils est en fait composé d'éléments graphiques « ProfileComponent » et « NewProfileComponent » qui sont des « JComponents ». Chacun de ces éléments permet de présenter visuellement les informations du profil choisi, ainsi que l'avatar qui lui est attaché.

- Options

Ce package contient les éléments graphiques concernant le réglage de différentes options. On y retrouve la barre d'option, « JPanelTopBar ».

- Game Screen

On retrouve dans « Game Screen » le panel de jeu principal appelé « JPanelGameBoard ». Celui-ci contient notamment les deux parties principales de la zone de jeu.

« JPanelGameArea » est la zone où se situent les cartes des joueurs et la « *board* ». Chaque joueur est graphiquement représenté par un « PlayerComponent » qui contient l'argent actuel du joueur, ses cartes (révélées ou non), ainsi que quelques informations de son profil. Dans ce « JComponent » y figurera aussi le « *token* » du joueur (*Dealer, small blind, big blind)*. Le centre de la « game area » sera rempli par les cartes de la board « BoardCardsPanel » ainsi que des informations sur le tour courant et sur la partie actuelle.

« JPanelGameControl » est la zone où le joueur décidera de sa prochaine action. Les actions disponibles seront sélectionnables via des boutons « JButton » ainsi qu'un slider « JSlider » pour le montant de la mise.

A gauche du game control se trouve une image rappelant les mains possibles au poker. A droite se situe un logger qui est en fait un « JScrollPane », ainsi qu'un panel « JPanelStatistics contenant des informations de probabilités de gains selon les cartes actuellement en jeu. A chaque changement d'état, ce panel est actualisé.

- Tools

Dans le package « Tools » se situe des classes servant de boîte à outils pour d'autres classes. « ImageShop », « ColorShop » servent à instancier une seule fois chaque image et couleur du projet, sous forme de liste d'objets « static final ». « ImageTools » est une classe qui permet de charger les images. « ButtonTools » donne un style spécifique aux boutons.

3.2 Moteur de jeu

Ci-dessous, la conception de notre moteur de jeu y est présentée. Nous y aborderons la logique que nous avons utilisée pour le réaliser, à l'aide d'un diagramme de classes et de plusieurs diagrammes de séquences.

• Diagramme de classes

Les commentaires se réfèrent tous à l'annexe 3.2.1

- Cards

Dans ce *package*, l'on retrouve la représentation d'une carte. Nous avons défini des énumérations, la première représentant la valeur de la carte, et la deuxième sa couleur.

- Subsets

Chacune des classes représente un ensemble de cartes. La classe parent « CardSubset » contient toutes les opérations intéressantes à effectuer sur les ensembles de cartes. Pour les classes enfants, elles ont une particularité en commun, le nombre de cartes qu'elles peuvent contenir mais cette valeur diffère :

- *Deck*, en contient 52 puis ce nombre diminue au cours du temps, car les cartes y sont piochées
- *Pocket*, contient les 2 cartes du joueur
- *Board*, ne contient rien au début, puis 3,4 et enfin 5
- *Hand*, représente une combinaison, contient donc 5 cartes

- Statistics

On retrouve à l'intérieur, une classe ne possédant que des méthodes statiques, qui permet de récupérer des statistiques durant les différentes phases du jeu ainsi que de lancer les simulations pour les obtenir si nécessaires. Parmi ces méthodes, il y a la méthode « getOuts » qui récupère le nombre d'outs[4] de la main courante. Cette méthode a été utilisée au début de l'intelligence artificielle mais ne l'est plus actuellement. Elle a toujours une utilité dans le cas où l'on souhaiterait l'améliorer. Nous avons aussi une classe « Odds », représentant les cotes au poker, elle aussi utilisée pour notre premier jet de l'intelligence artificielle.

- Simulation

On peut observer la présence de 2 classes : « Data » et « Simulation ». La première permet de représenter les différentes valeurs obtenues au cours d'une simulation. La deuxième permet de lancer des simulations qui enregistreront leurs données dans la classe « Data ». Il est intéressant de noter que

[4] Les outs représentent l'ensemble des cartes pour améliorer une main. Cependant, pendant le calcul, il faut prendre en compte les nombreux cas spéciaux, dans lesquels certaines cartes « logiques » n'améliorent pas une main. Pour plus d'information, voir http://www.tightpoker.com/poker_odds.html.

la classe « Simulation » utilise un double *pattern Observer*. Cela permet de notifier le joueur une fois qu'elle a fini, et inversement d'être notifiée quand arrêter une simulation, dans le cas où le joueur doit jouer et que les simulations ne seraient pas terminées.

- Compute

On y retrouve la classe principale « HandsPokerMap », qui s'occupe d'aller lire dans un dictionnaire (rempli au préalable à partir d'un fichier), pour donner certaines informations pour une main donnée, notamment sa puissance. Notons que cette classe utilise le *pattern Singleton*, car il est inutile d'avoir différentes instances pour lire dans le même tableau.

La classe « HandsPokerValue » représente les différentes informations, présentées par la classe « HandsPokerMap ».

La classe « ComputeBestHand » calcule la meilleure main à partir d'un ensemble de cartes. Une main est composée de 5 cartes, et nous l'utiliserons généralement dans des ensembles de 5,6 et 7 cartes, même si elle pourrait calculer pour des ensembles plus grands.

Pour finir, nous avons aussi une énumération représentant la combinaison générale de la main.

- Player

Représente le joueur. On y retrouve dans le package *profil*, le profil d'un joueur. S'il s'agit de l'utilisateur, le profil correspondra à celui qu'il a choisi, dans le cas contraire, un profil aléatoire destiné aux intelligences artificielles sera choisi. Pour finir, le joueur a un rôle, et possède un jeton correspondant au rôle de *dealer, small blind, big blind* ou aucun rôle.

- GameCore

C'est ici que le cœur du moteur de jeu est représenté. C'est dans la classe « GameEngine » que le déroulement des tours se déclenche. Pour cela, il utilise des classes spécialisant *State* afin d'effectuer les différentes actions aux différentes phases. Notons que nous utilisons le *pattern State* pour cela. Il y a aussi une énumération « StateType », pour indiquer l'ancien état et ainsi choisir le bon prochain état, ce qui nous permet d'économiser quelques classes. La classe « SoundEngine » est utilisée pour gérer le volume et les différents sons. Afin de savoir quel son jouer, elle utilise la classe « Action » (aussi utilisée pour le journal). Pour finir, il y a la classe « Pot » qui représente le pot du jeu.

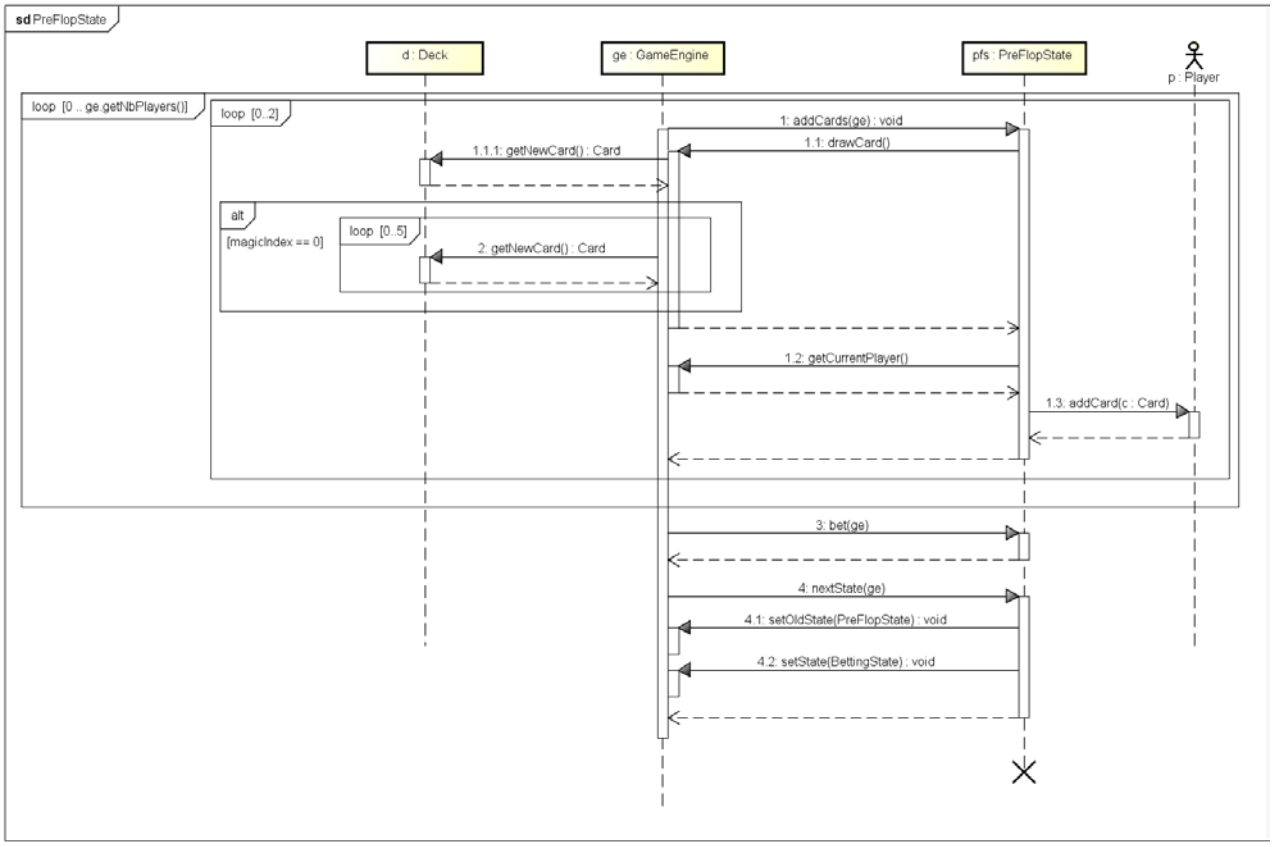
- ## Diagramme de séquences

Dans les diagrammes qui sont présentés ci-dessous, les différents objets, instances et méthodes utilisées se réfèrent au diagramme de classe du moteur de jeu (voir annexe 3.2.1).

Nous avons décidé de réaliser ceux représentant les différents états du jeu, afin de voir le déroulement de ceux-ci, tout en utilisant le *pattern State*.

Dans les diagrammes représentants les états, le *GameEngine* effectuera toujours les actions dans le même ordre : *addCards*, *bet* et *nextState*. Pour la méthode *bet*, celle-ci n'est utilisée que dans l'état *Betting*, sinon elle ne fait rien. Quant à *nextState*, elle fait toujours passer à l'état *Betting*, sauf à l'état *Betting* où elle fait passer au prochain état correspondant.

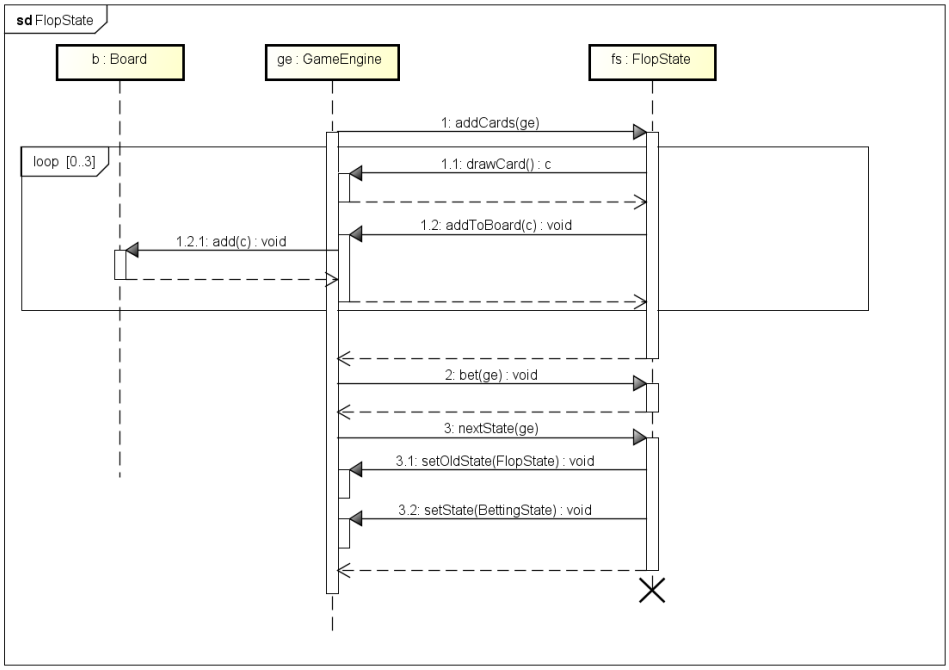
- État *preflop*



État *Preflop*, Diagramme de séquences 1

La méthode *addCard* pioche 2 cartes du *Deck* pour chaque joueur.

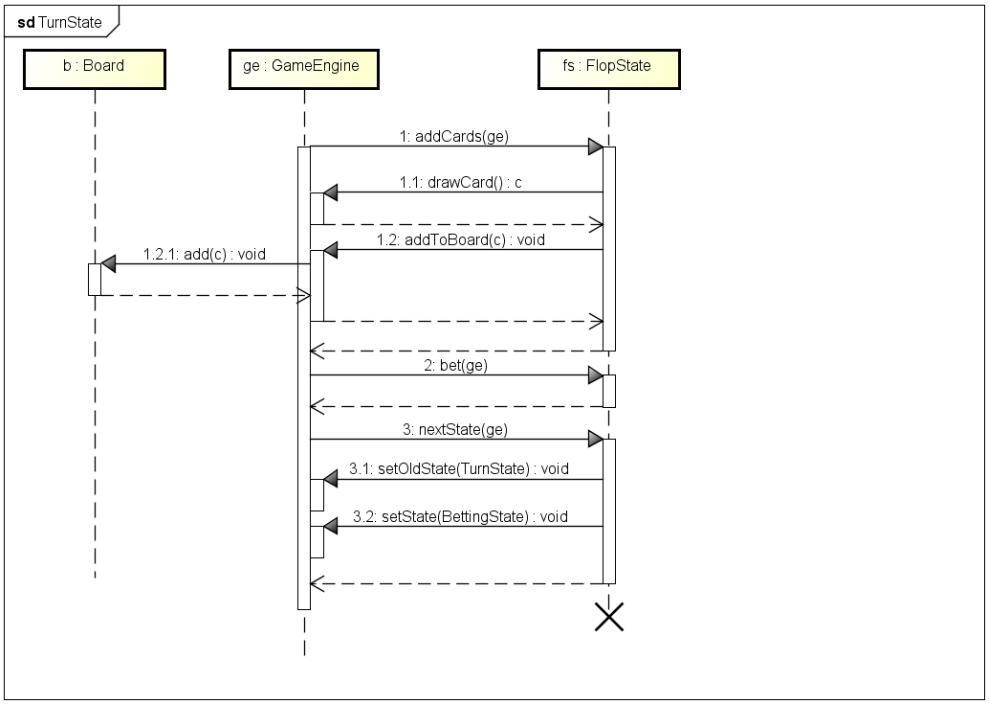
- État *Flop*



État *Flop*, Diagramme de séquences 2

La méthode *addCard* pioche 3 cartes du *Deck* et ajoutera celles-ci au *Board*.

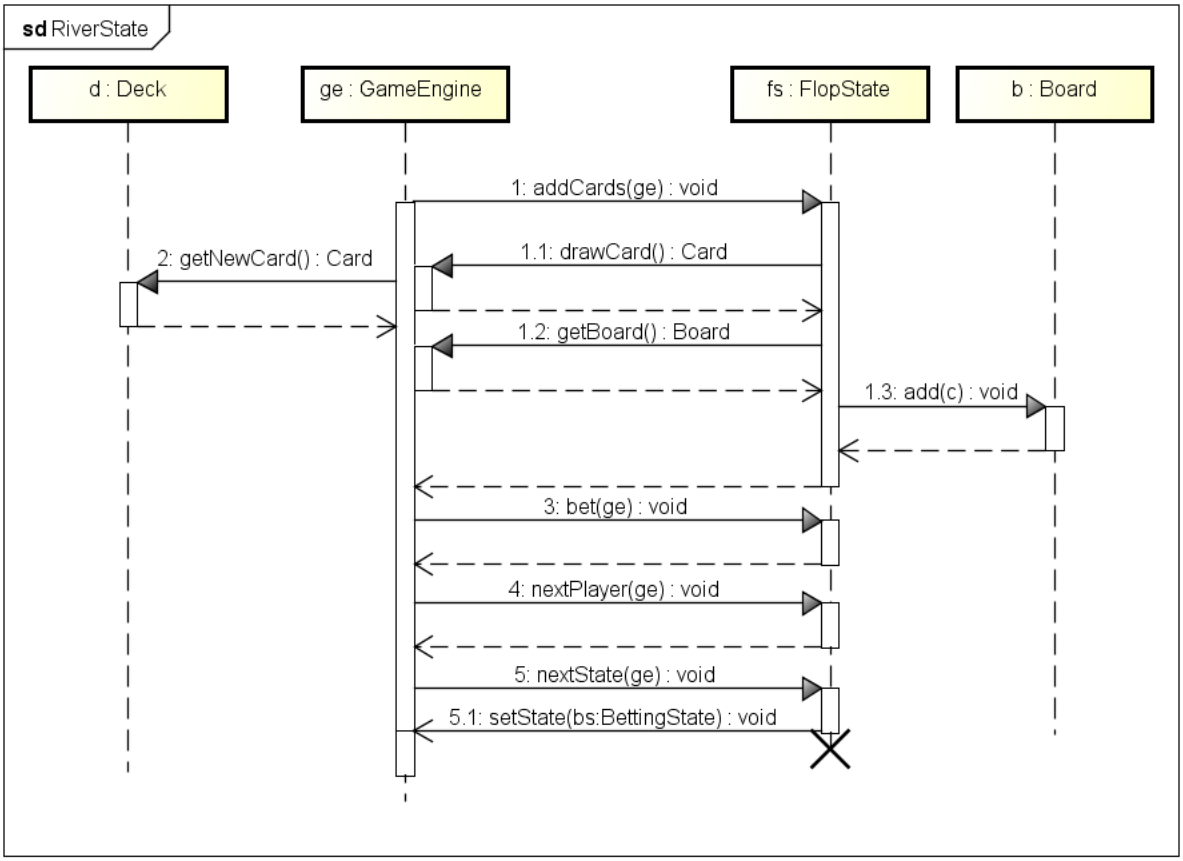
- État *Turn*



État *Turn*, Diagramme de séquences 3

La méthode *addCard* pioche 1 carte du *Deck* et ajoutera celle-ci au *Board*.

- État *River*



État *River*, Diagramme de séquences 4

La méthode *addCard* piochera 1 carte du *Deck* et ajoutera celle-ci au *Board*.

- État *Betting*

Les commentaires se réfèrent à l'annexe 3.2.2 et 3.2.3

La méthode *addCard* ne fait rien.

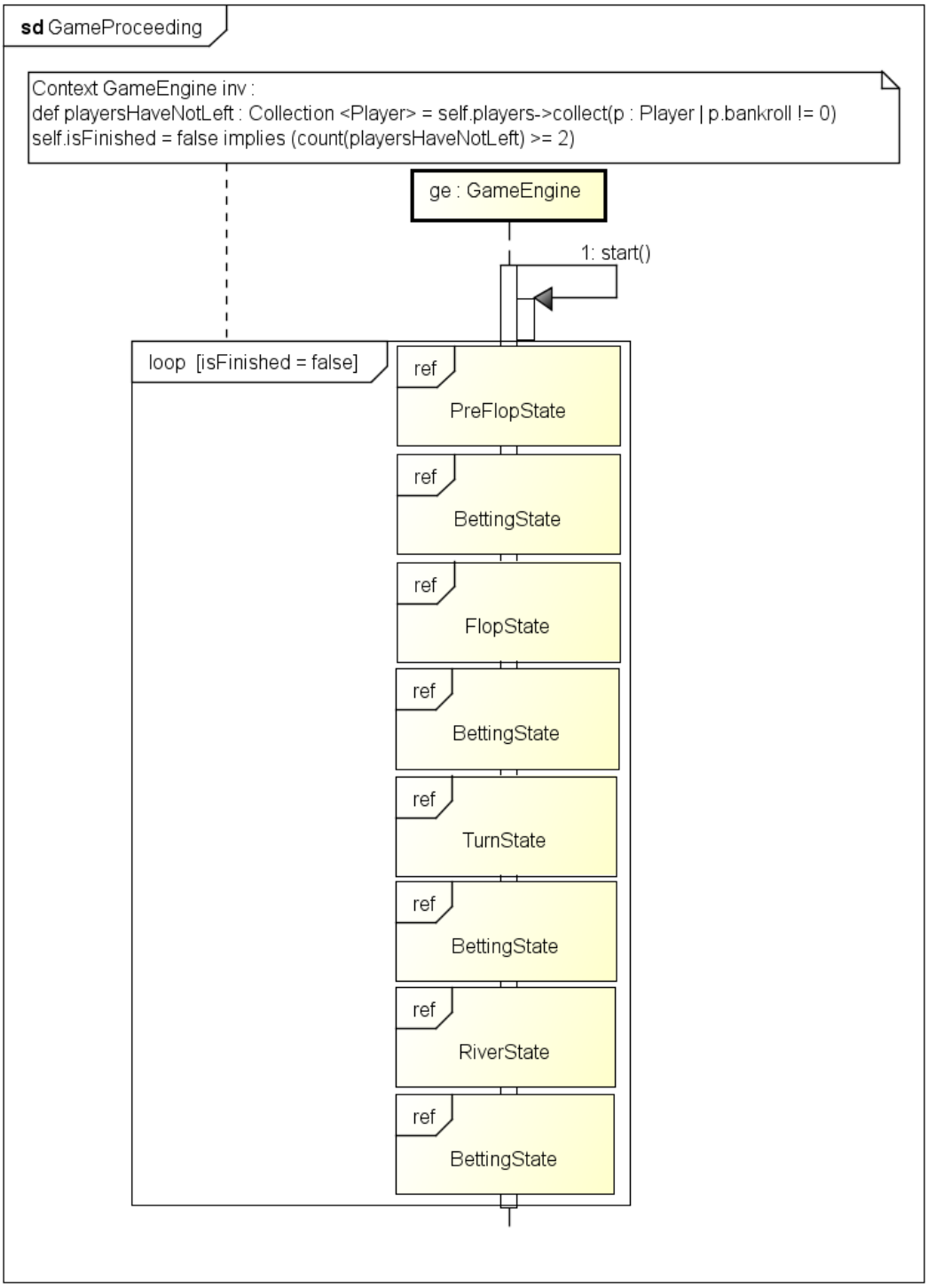
La méthode *bet* attend une action du joueur (s'il est humain, attend la notification du *GUI*, si c'est une intelligence artificielle, attend le retour de *whatToDo*).

Ensuite, le joueur effectue une des différentes actions, et son solde se voit modifié si nécessaire. Si le joueur a effectué une mise, relance ou a simplement suivi, le pot se retrouve enchéri et la valeur de la dernière mise/relance se voit modifiée. Si aucun joueur n'a misé ou relancé, les enchères s'arrêtent.

Pour finir, *nextState* fait passer à l'état suivant, selon l'état précédent.

Lorsque le joueur effectue une action (annexe 3.2.3) on peut voir comment interagit le *GameEngine* avec l'argent et les mises.

- Déroulement d'une partie

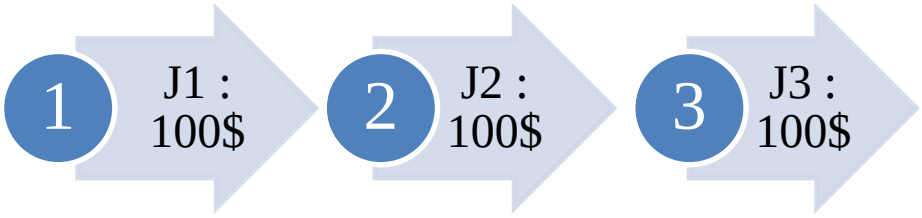


Déroulement d'une partie, Diagramme de séquences 5

Tant qu'il y a encore 2 joueurs avec un solde supérieur à 0, le moteur de jeu appellera les différents états dans cet ordre.

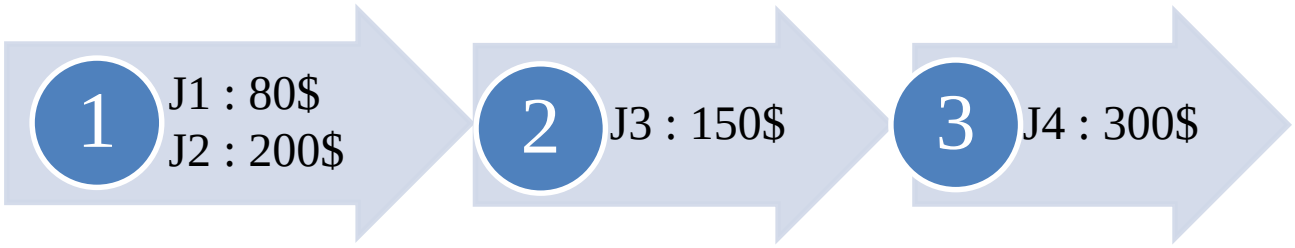
4 Division du pot

La division du pot en général ne représente pas un problème. Imaginons la situation suivante :



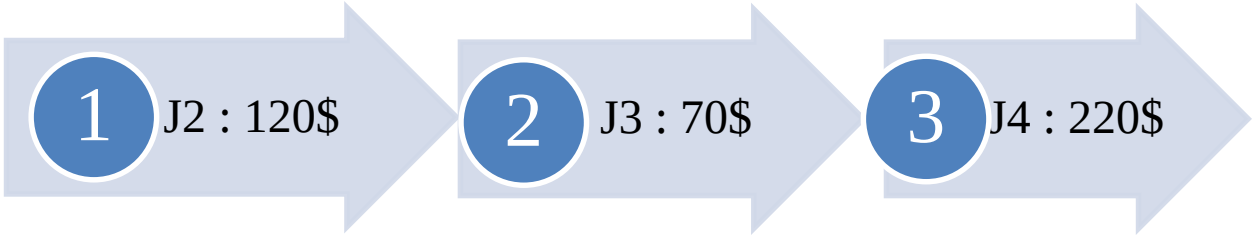
Dans ce cas, J1 remportera 300$ car il a misé la même quantité d'argent que les autres joueurs.

Maintenant, prenons un cas où les joueurs sont en fin de partie et ne misent plus la même quantité, car leur *bankroll* se retrouve limitée :

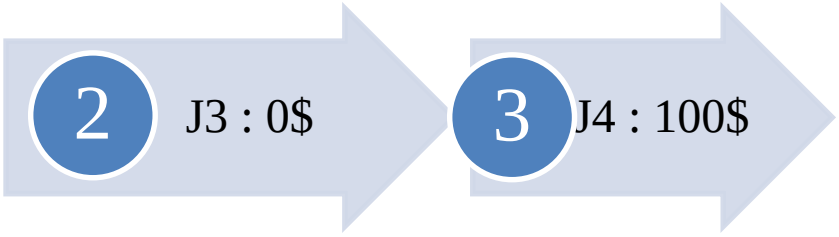


Dans ce cas, nous avons les deux premiers joueurs qui se retrouvent à égalité avec des mises différentes. Le troisième joueur a misé moins que le deuxième, mais plus que le premier. Pour finir, le dernier joueur a misé plus que tous les autres. Comment répartissons-nous la somme ?

Premièrement, il faut se positionner dans le premier groupe (les gagnants) et prendre la mise minimale, donc 80$. On retire 80$ à tout le monde (ou si un joueur à moins de 80$, on prend tout le reste). Avec cette somme, on le divise par le nombre de joueurs du groupe, donc 160$ pour joueur 1 et joueur 2 et on retire le joueur ayant eu la mise minimale. On obtient le cas suivant :



On a pour le moment comme gain : J1 160$, J2 160$, J3 0$, J4 0$. Refaisons-le même pour le joueur 2 :



L'on obtient : J1 : 160$, J2 470$, J3 0$ et J4 0$. Nous refaisons le même processus pour le joueur 3, mais il lui reste rien donc on le saute. Joueur 4 est le dernier joueur qui reste donc il repart avec ses 100$. Au final nous avons donc : J1 160$, J2 470$, J3 0$ et J4 100$.

5 Intelligence Artificielle

5.1 Concept de base

L'idée de base pour notre intelligence artificielle, est qu'elle va d'abord décider si elle va suivre l'enchère ou non. Dans le cas où elle ne suit pas, elle se couchera. Cependant si l'enchère est nulle, elle préféra checker, quitte à se coucher plus tard. Si elle a suivi l'enchère, elle va regarder si son jeu est suffisamment bon pour miser ou relancer. Dans le cas contraire, elle peut essayer d'introduire une notion de bluff. Si maintenant elle a décidé de miser ou de relancer, elle va utiliser une fonction de mise qui va indiquer le montant qu'elle va miser. Dans le cas où le montant est inférieur à la dernière mise effectuée, celle-ci se retrouve équivalente à la dernière mise (comme le veulent les règles). Chaque intelligence artificielle se voit également attribuer un coefficient de risque, $c \in [0.8;1.2]$ qui influencera le résultat des fonctions utilisées. Ce coefficient permettra de rendre plus prudente ou plus agressive une intelligence artificielle, ceci afin de rendre les adversaires différents des uns des autres.

Pour utiliser nos différentes fonctions, l'intelligence artificielle a besoin de savoir 3 paramètres :

1. La probabilité de gagner
2. La valeur à mettre pour continuer, exprimée en pourcentage par rapport à sa *bankroll* actuelle
3. L'argent déjà investi pendant le tour, exprimé en pourcentage par rapport à sa *bankroll* au début de la partie (donc depuis le *preflop*).

Les points 2 et 3 ne posent pas de problème. Ce qui est plus dur à calculer c'est le point 1. Pour cela, nous avons mis en place un système de simulations décrit au chapitre 5.2.

Selon l'état actuel, la fonction mathématique 4D appelée ne sera pas forcément la même. Nous avons défini des comportements différents selon les états.

Dans le cas du *preflop*, nous considérons qu'il faut être assez prudent car à ce stade nous ne connaissons que 2 cartes et que nous ne pouvons pas savoir assez vite si nous sommes favoris ou non. Mais si l'on est favori maintenant, il se peut que l'on ne le soit plus à la fin, d'où la nécessité de jouer la prudence. Dans le cas où elle aimerait relancer, nous avons défini que la mise minimale sera équivalente à la dernière relance effectuée (comme le stipule les règles) et la maximale le double.

Le cas du *flop*, *turn* est assez similaire : nous connaissons 3 puis 4 cartes sur la table et avons déjà une idée approximative de ce que l'on peut espérer. Elle peut commencer à miser/relancer sérieusement car elle sait déjà ce qu'elle a actuellement et ce qu'elle peut espérer avoir. Dans le cas où elle n'est pas favorite, elle se couchera ou suivra simplement selon sa main.

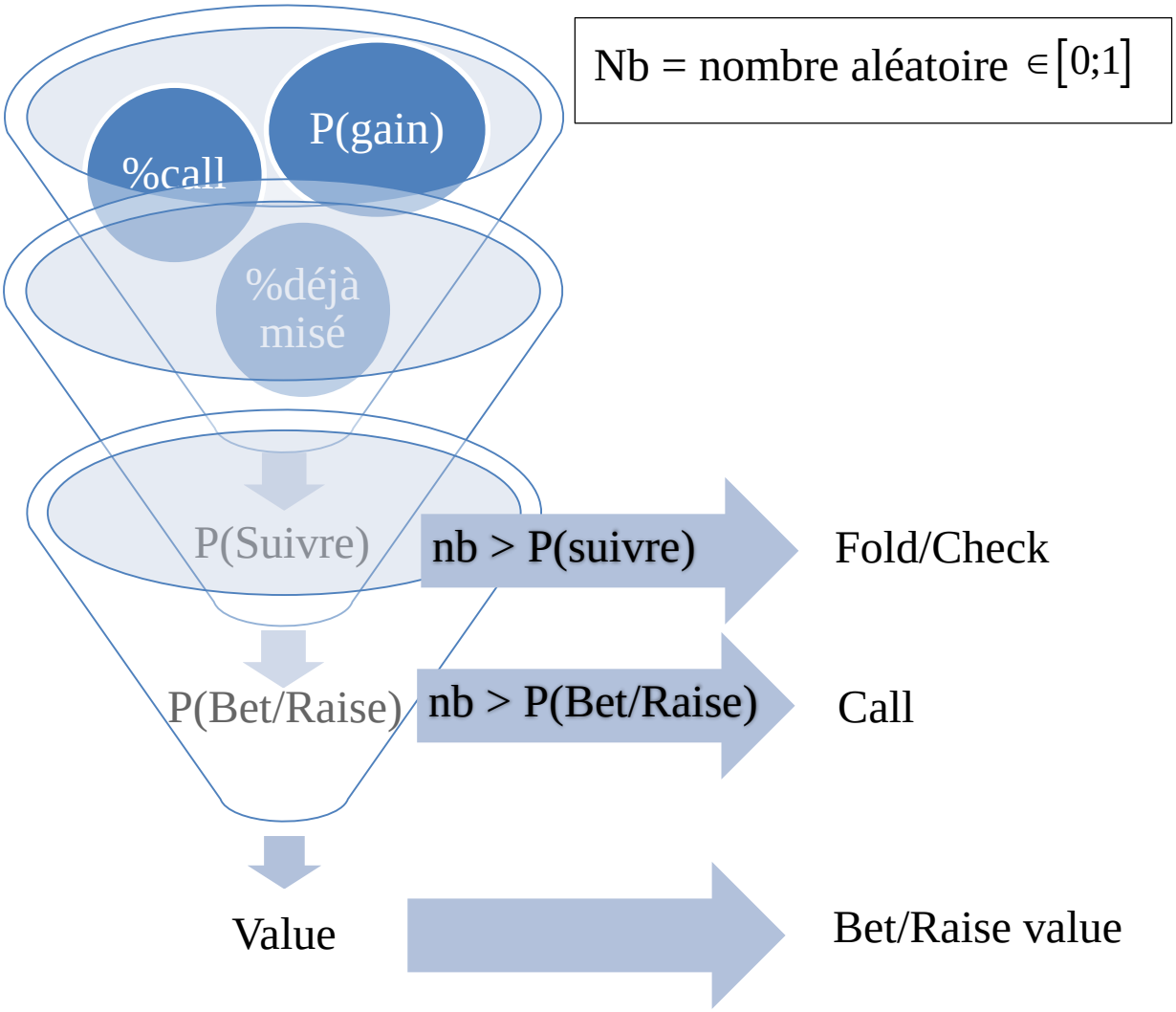
Pour le dernier cas, *River*, toutes les cartes sont sur la table et tous les joueurs savent exactement ce qu'ils ont. Le défi est d'estimer si les mains adverses sont meilleures que la sienne et si c'est le cas contraire, elle peut quelques fois faire des relances pour tenter de dissuader l'adversaire, mais malheureusement pour elle, notre intelligence artificielle réagit bien à ceci mais peut-être pas le joueur humain ! Le profil à ce stade est assez clair, on tente de gagner le plus d'argent, donc souvent de grandes relances sont effectuées. L'intérêt est double :

1. Si les adversaires se couchent, leur argent déjà investi est quand même à prendre
2. Moins d'adversaires, plus de chance de gagner !

Dans les parties suivantes, chaque état en détail sera abordé avec leur fonction, les paramètres et la logique réalisée derrière.

Pour résumer notre algorithme, une représentation visuelle est disponible à la figure 1.

Représentation graphique de l'algorithme décisionnelle de l'IA, Figure 1



5.2 Simulations

Commençons par définir une simulation au poker telle que nous l'avons implémentée :

"Une simulation au poker est une simulation qui, pour une main de départ donnée et à un certain stade de départ (preflop, flop, turn ou river), va indiquer si le joueur a gagné, perdu ou fait une égalité. Les actions du joueur et des autres joueurs sont constituées de check et de call, afin d'avoir tous les adversaires possibles, qui se coucheraient alors qu'ils ont une bonne main."

A ne pas confondre avec une simulation d'état, qui elle est un ensemble de simulations pour l'état actuel.

Il est important de comprendre pourquoi il y a 4 états de simulations possibles ! Dans le cas du *preflop*, seules nos 2 cartes sont connues et donc tout peut arriver ($C_5^{50-2\cdot na}, na = nomberAdvesaires$) ! Pour le *flop*, nous connaissons nos 2 cartes et les 3 cartes sur la table, donc l'espace représentant les différentes combinaisons restantes est restreint ($C_2^{47-2\cdot na}, na = nomberAdversaires$). Dans le cas du *turn*, cet espace se voit encore plus restreint car il ne reste plus qu'une carte à découvrir ($C_1^{46-2\cdot na}, na = nomberAdversaires$). Pour le dernier cas, le *river*, toutes les cartes sont dévoilées mais maintenant il reste à déterminer si notre main finale est mieux que celles des adversaires.

Il faut maintenant déterminer le nombre de simulations par personne. Logiquement, il y aurait un total de $4 \cdot nj, nj = nombreJoueur$ simulations. Mais, nous avons pu constater qu'il y a seulement 169 mains de départ possible au poker, ce qui fait que nous pouvons pré-simuler celles-ci et donc gagner du temps pour les autres restantes. Donc au final, il n'y a plus que $3 \cdot nj, nj = nombreJoueur$ simulations en temps réel à chaque partie et $169 \cdot nj, nj = nombreJoueur$ simulations pré-simulées.

La dernière chose à définir est le nombre de simulations par état, car il est logique que simuler sur une partie n'est pas suffisant. Une grande méditation a été effectuée sur le sujet. Différentes options ont été envisagées :

- Lancer des simulations jusqu'à que ça soit au joueur de jouer et donc d'arrêter la simulation du joueur à l'état actuel. Le problème est qu'il y a constamment des simulations en cours, ce qui pourrait commencer à faire chauffer l'ordinateur de l'utilisateur et donc augmenter la vitesse des ventilateurs. Ceci pourrait déranger l'utilisateur de notre programme.
- Effectuer plusieurs tests avec un n augmentant au fur et à mesure, et constater à partir de quel moment le pas de n à effectuer pour gagner une précision correcte devient trop grand. Bien sûr, il faut que ce n ne soit pas trop grand afin d'éviter le problème de la première solution.
- Utiliser le théorème des centrales limites et définir la valeur des différentes variables nécessaires et calculer le nombre de simulations que l'on souhaite pour avoir une erreur maîtrisée (c.-à-d. savoir l'erreur exacte selon le nombre de simulations). Pour cela, il faudrait utiliser l'équation suivante :

$$\sup_{x\in\mathbb{R}} \left| P\left(\frac{1}{\sqrt{n\cdot\sigma^2}}\sum_{k=1}^{n}(X_k - E(X_K)) \le x\right) - \phi(x)\right| \le \frac{E(|X_1 - E(X_1)|^3)}{\sigma^3\cdot\sqrt{n}}$$

Equation permettant de trouver le nombre de simulation, Équation 1

On a une affirmation qui dit que si $n \to \infty$ on obtient une erreur nulle comme ci-dessous :

$$\lim_{n\to\infty} \sup_{x\in\mathbb{R}} \left| P\left(\frac{1}{\sqrt{n\cdot\sigma^2}}\sum_{k=1}^{n}(X_k - E(X_K)) \le x\right) - \phi(x)\right| = 0$$

Affirmation montrant que si $x \to \infty$, l'erreur est nulle, Affirmation 2

Nous avons opté pour la deuxième solution. Bien que la troisième solution soit fort intéressante, notre niveau au début du projet en probabilités n'était pas suffisant et actuellement, nous aurions toujours de la peine à résoudre l'équation. Nous avons donc opté pour la deuxième solution avec un $n = 20'000$. On a pu constater que sur les machines des différents étudiants et professeurs, l'ordinateur avait le temps d'effectuer nos $3 \cdot 20000 \cdot nj, nj = nombreJoueur$ simulations.

De toute façon, il faut savoir que lorsque le joueur doit jouer et que la simulation relative à l'état actuel ne s'est pas déroulée entièrement, le joueur l'arrête automatiquement. Ceci permet d'éviter d'attendre que les simulations se terminent et donc au joueur d'attendre, quitte à avoir un nombre de simulations légèrement inférieur au nombre de base.

Puisque l'on parle de temps, actuellement nous laissons à nos intelligences artificielles 1 seconde et demie de « réflexion ». Ce temps est nécessaire afin que le joueur humain puisse suivre la partie, et aussi laisser un peu plus de temps pour les simulations.

Quant à l'implémentation de la communication entre les simulations et le joueur, nous avons implémenté un système en utilisant un double *pattern Observer*. Quand une simulation se termine, le joueur se voit notifier et quand un joueur doit jouer et que la simulation est en cours, il va la faire s'arrêter.

5.3 Suivre/checker ou se coucher ?

Nous allons aborder les différentes fonctions réalisées afin de définir un modèle à suivre pour chaque état. Toutes les fonctions ont un point commun : elles sont toutes des fonctions à 4 dimensions. Pour chaque état, le schéma sera le même : préambule, formule, représentation visuelle et description.

Chaque fonction est définie sous la forme de :

$$f : [0;1]^3 \rightarrow [0;1] \text{ ou sous une autre forme } f(x,y,z) \in [0;1] \; et \; \{x,y,z\} \in [0;1]$$

x représente la probabilité de gagner, y le pourcentage de notre *bankroll* actuelle qu'il faut miser pour continuer le jeu et z le pourcentage de notre *bankroll* au début du tour que nous avons déjà investie.

- ***Preflop***

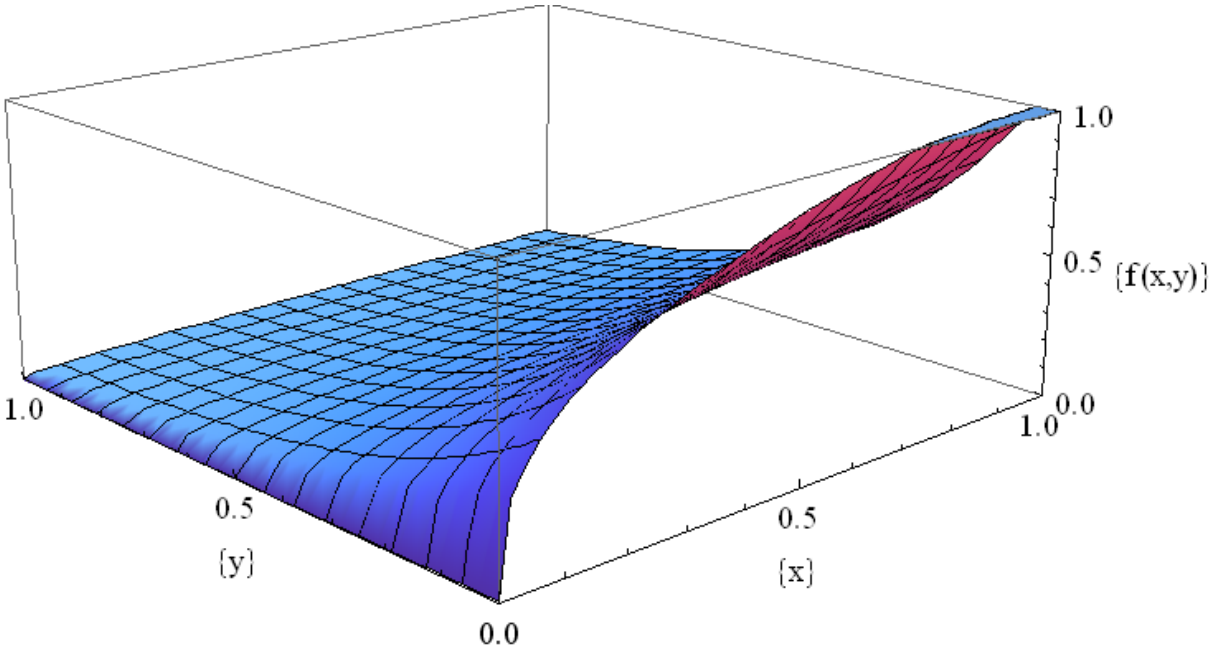
Il faut savoir qu'au *preflop*, selon la main de départ que l'on a, si parmi les 169 mains de départs possibles, on a une chance de gain inférieur à 20%, nous décidons de ne même pas jouer. Le nombre de mains concernées est relativement faible, quel que soit le nombre de joueurs. Au contraire, si nous avons une chance de gain supérieur à 70%, nous suivons à coup sûr.

Nous avons créé la fonction suivante :

$$f(x,y,z) = \frac{\sqrt[3]{x} \cdot (e^{(1-y)})^3}{\sqrt[3]{0.7} \cdot e^3} + z$$

Fonction au *preflop*, Fonction 1

Pour le moment, faisons abstraction de z . Nous pouvons visualiser l'interprétation au Graphique 1.

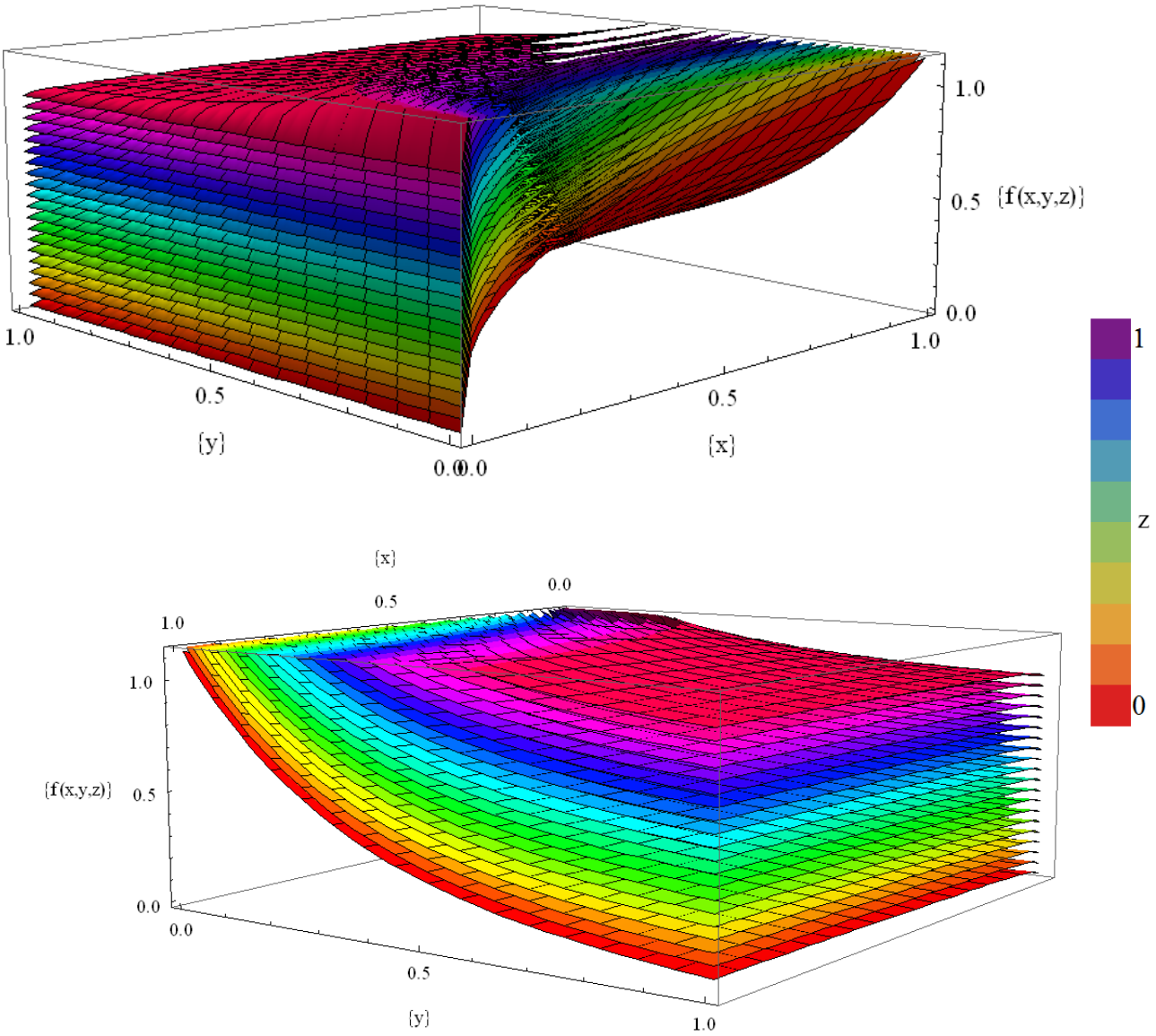


Représentation de la Fonction 1, Graphique 1

Nous voulions que si nous avions une des plus mauvaises mains au départ, se coucher, peu importe l'argent mis en jeu. Au contraire, si nous avons une très bonne main, nous voulions suivre un maximum de fois possible. Selon l'argent à investir, plus celui-ci est faible, plus nous allons suivre. On peut remarquer qu'il sera rare de suivre un *allin* à cet état, comportement qui est tout à fait réaliste.

Pour le reste, il fallait trouver un compromis à faire, entre la valeur de notre main et l'argent à investir. Naturellement, si nous devons investir un grand montant, les chances de continuer se retrouvent réduites.

Si nous prenons en compte maintenant du paramètre z, nous pouvons constater que celui-ci va augmenter les chances de suivre proportionnellement au montant déjà investi. Si nous faisons une représentation visuelle, nous obtenons le Graphique 2.



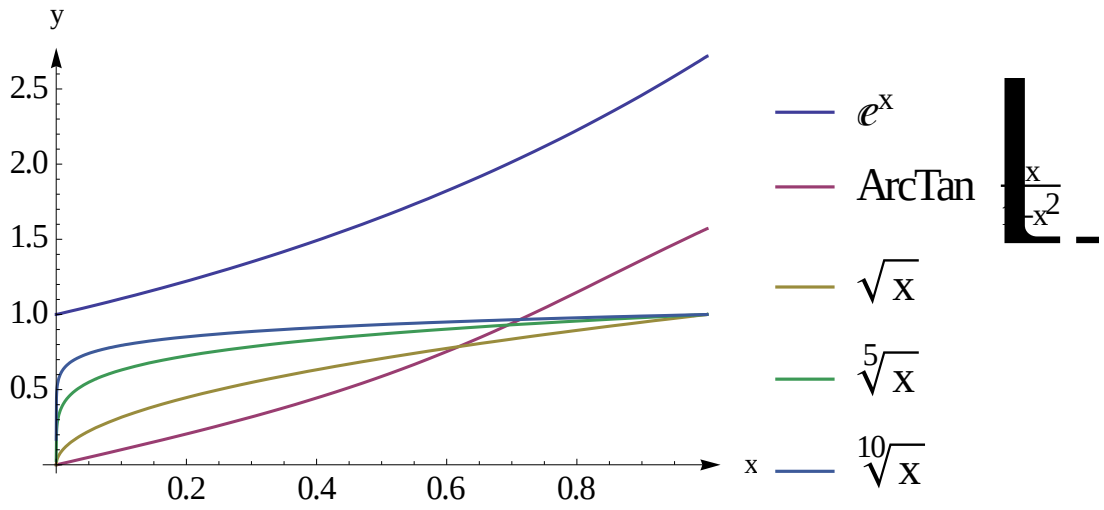
Représentations 4D de la Fonction 1, Graphique 2

- ***Flop/Turn***

Nous avons considéré que le profil du joueur lors du *flop* ou *turn* est le même. Dans les 2 cas, nous avons une idée approximative de la main que nous avons et savons qu'est-ce que nous pouvons espérer. Dans le cas où l'on a déjà une très bonne main, au *flop* ou *turn*, on suivra très souvent et relancera très sûrement.

Contrairement à avant, nous considérons tous les cas. Car on peut très bien avoir une très bonne main de départ et par la suite la voir devenir très mauvaise. Et inversement, on peut très bien avoir une main moyenne ou faible et la voir se transformer en une excellente main qui nous ferait être le favori.

Pour la fonction, nous avons pensé que la fonction la plus adaptée à notre cas est l'inverse de la tangente, à combiner avec une fonction racine. Contrairement à l'exponentielle, celle-ci possède une courbe moins brutale. Nous pouvons voir les différents cas dans le graphique 3. Dans le cas précédent nous voulions quelque chose avec une courbe plus raide et avons donc combiné la fonction exponentielle avec celle de la racine.



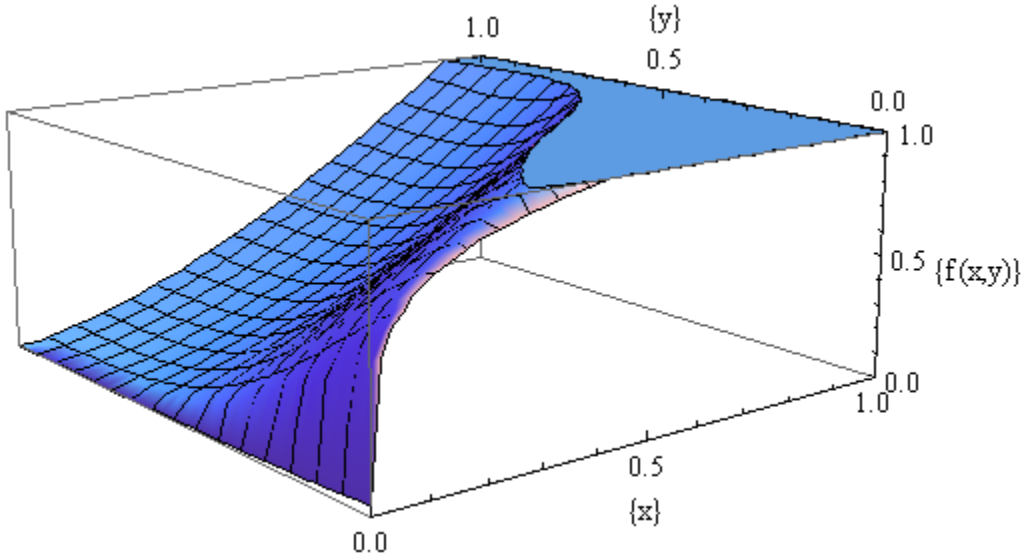
Différentes fonctions intéressantes, Graphique 3

Nous avons imaginé la fonction suivante :

$$f(x,y,z) = \left[ArcTan\left[\frac{1.3 \cdot \sqrt[5]{x} \cdot (1-y)^3}{1.1 - \sqrt[5]{x}\sqrt{1-y}} \right] \cdot \frac{2}{\pi} + \sqrt{x^3 \cdot \sqrt[3]{y}} \right] \cdot 1.15 + z$$

Fonction au *flop/turn*, Fonction 2

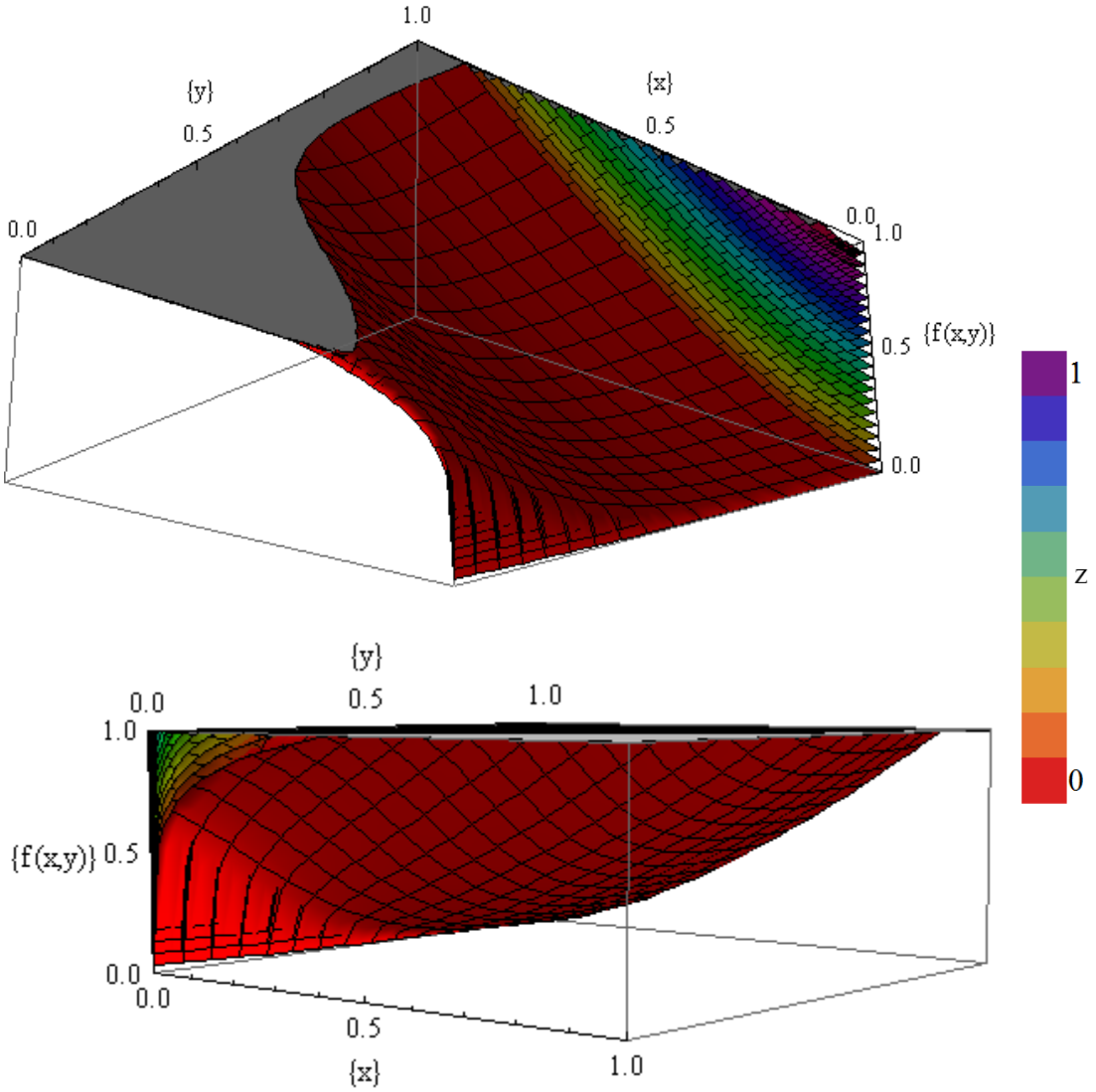
Pour le moment, faisons abstraction de z . Nous pouvons visualiser l'interprétation au Graphique 4.



Représentation de la Fonction 2, Graphique 4

Nous désirions, que si on ne devait rien miser, l'on devrait suivre la plupart du temps. On peut observer les cas où $x \to 0$, on se couchera systématiquement. Ceci est un cas extrêmement rare, car il faudrait partir avec une main moyenne ou bonne et arriver à ce stade où notre main ne vaudrait plus rien. On voulait que si nous avions une bonne main, suivre très souvent, même s'il fallait investir un certain nombre d'argent. Pour les cas où l'on a une main moyenne/potable, plus notre main est forte, plus on aura des chances de suivre selon la quantité d'argent à investir.

Si nous prenons en compte maintenant le paramètre z , nous pouvons constater que celui-ci va augmenter les chances de suivre proportionnellement au montant déjà investi. Le même phénomène qu'au *preflop* se produit. Si nous faisons une représentation visuelle, nous obtenons le Graphique 5. On peut aussi constater, que cette fonction a un « ventre » pour représenter que plus il y a d'argent en jeu, moins on va suivre.



Représentations 4D de la Fonction 2, Graphique 5

- ***River***

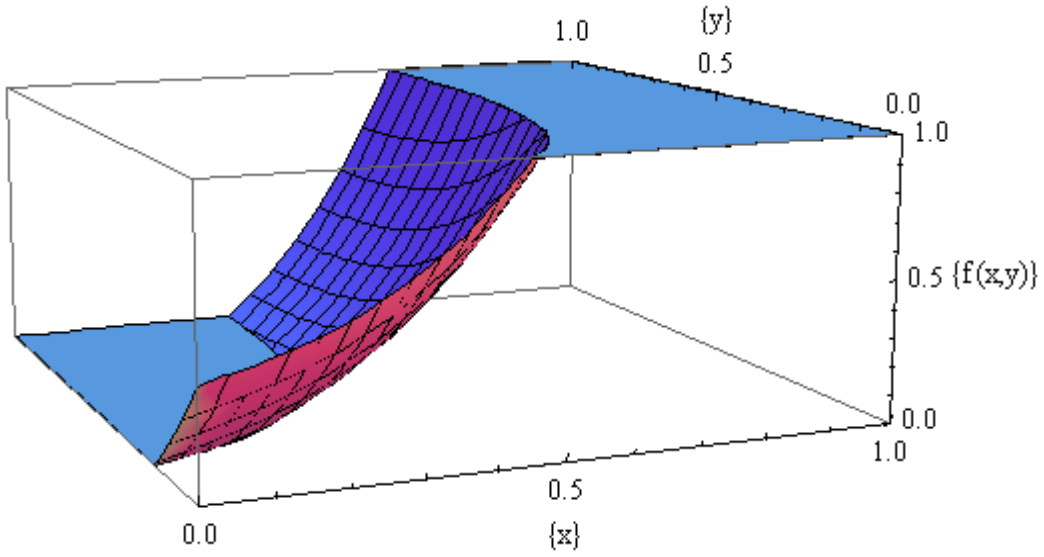
Pour ce dernier cas, on sait si on a une bonne main ou pas et on a une bonne idée si on est favori. Donc pour notre fonction, nous avons utilisé l'exponentielle car cette fois-ci il est important que la courbe du côté de la probabilité de gagner et celle de l'argent à miser aient une pente qui augmente rapidement.

Nous avons trouvé la fonction suivante :

$$f(x,y,z) = \frac{(e^{\frac{\sqrt{x}}{0.17}} + (1-y)\cdot \mathrm{e}^{\frac{\sqrt{1-mise}}{0.24}})^{1.04}}{100} - 0.4 + 1.5\cdot z$$

Fonction au *river*, Fonction 3

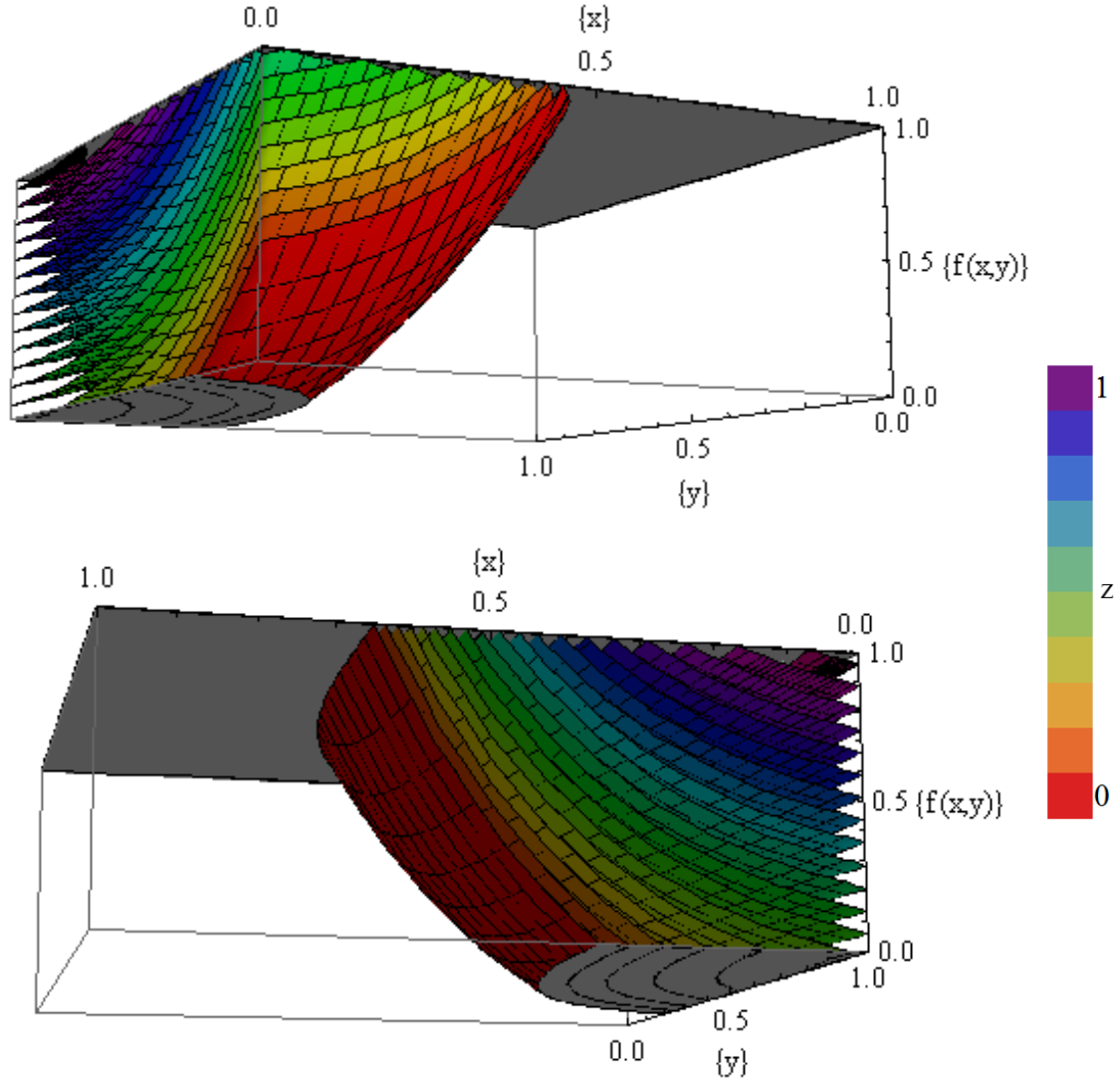
Pour le moment, faisons abstraction de z . Nous pouvons visualiser l'interprétation au Graphique 6.



Représentation de la Fonction 3, Graphique 6

Cette fois-ci, nous voulons que si l'on a une mauvaise main, on se couche presque systématiquement mais plus notre main est forte plus on est prêt à suivre. Pour finir, dès que l'on dépasse les 60% de chances de gagner, on suit de toute façon. Il faut savoir que 60% de chance de gagner, que ça soit à 2 ou 10 joueurs, cela est énorme !

Si nous prenons en compte maintenant du paramètre z, nous pouvons constater que celui-ci va augmenter les chances de suivre de manière proportionnelle au montant déjà investi. Mais cette fois-ci avec un facteur. Si nous faisons une représentation visuelle, nous obtenons le Graphique 7.



Représentations 4D de la Fonction 3, Graphique 7

On est à la fin des fonctions qui représentaient le profil selon les différents cas, et l'on peut constater qu'au *preflop*, le modèle est assez prudent, au *flop*, *turn* il commence à suivre souvent et au *river*, s'il vous suit, c'est que vous être probablement en mauvaise posture !

5.4 Miser/Relancer ?

Que serait une intelligence artificielle qui ne ferait que de se coucher, suivre ou checker ? Cela représenterait un jeu bien ennuyeux. Dans cette partie, on vous fera part du modèle que l'on a établi dans le cas où l'intelligence artificielle souhaiterait relancer.

- **Principe**

Peu importe à quel état on se situe, nous avons notre fonction qui nous donne la probabilité que l'on va continuer la partie ou non. Dans plusieurs cas, cette probabilité de suivre équivaut à 100%. Il serait bien que dans ces cas-là (et d'autres où l'on a une bonne main) que l'on mise ou relance. Si on pouvait faire que de temps en temps, des mains un peu moins fortes, voire faible relance, on pourrait introduire cette légère notion de bluff qui rendrait le jeu imprévisible. Le modèle reste le même, peu importe l'état auquel on se situe.

L'idée est qu'avec la probabilité de suivre que l'on a reçu grâce à notre fonction, il serait intéressant de l'utiliser pour savoir si on veut relancer. Nous avons donc défini une nouvelle probabilité, la probabilité de miser/relancer. Elle se définit comme suit :

$$P(bet / raise) = \begin{cases} \dfrac{P(suivre)}{ntb^2} & P(gain) >= 0.7 \\ ntb^2 + 1 + nbr & Sinon \end{cases} \qquad \begin{array}{l} ntb = numero\ du\ tour\ d'enchère \\ nbr = nombre\ de\ relance\ du\ tour \end{array}$$

Fonction de relance, Fonction 4

D'une idée générale, on prend le résultat sortie par la fonction 4D de l'état, qui nous donne la probabilité de suivre. On divise ce résultat par le numéro du tour d'enchère au carré. Si on a une probabilité de gain inférieur à 70%, on ajoute au diviseur le nombre de relance du tour. On fait tout ce système pour éviter d'avoir des mises/relances à l'infini et mettre en place une barrière pour freiner ces relances. Dans les cas où l'on a une très bonne main, cette barrière est moins élevée que pour les autres.

- ## Combien miser/relancer

La quantité à miser au *preflop* ou dans les autres états n'est pas le même. Au *preflop*, il serait dangereux de faire des grosses mises/relances alors que même si l'on a une très bonne main, elle peut devenir rapidement obsolète.

Pour le *preflop*, si l'intelligence artificielle décide de miser/relancer, elle va générer un nombre entre 1 et 2, puis le multiplier par la dernière mise/relancer effectuée.

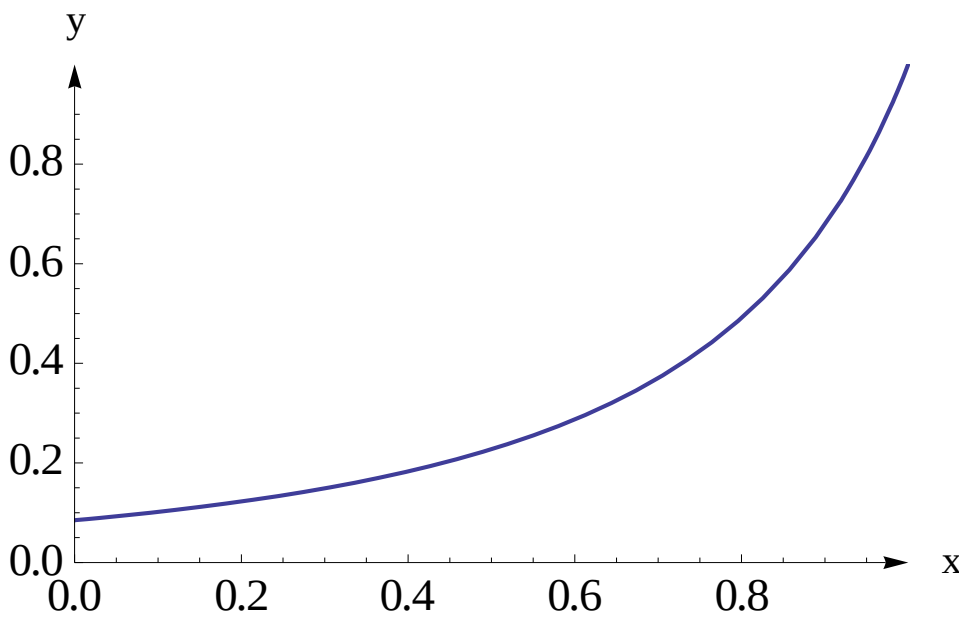
Pour les autres états, nous avons défini une fonction qui augmente de manière exponentielle selon notre probabilité de gagner. Finalement, nous avons décidé que si nous obtenons un montant de relance supérieur à 90% de notre *bankroll*, nous effectuons un *Allin.*

Nous avons donc la fonction suivante :

$$f(x) = \frac{e^{x}}{\frac{1.3 - x}{\frac{e}{0.3}}}$$

Fonction de mise/relancer, Fonction 5

Nous pouvons voir sa représentation graphique au Graphique 8.



Représentation de la Fonction 5, Graphique 8

5.5 Améliorations possibles

Actuellement, notre intelligence artificielle semble bien se débrouiller et à l'heure actuelle, nous n'avons trouvé aucune faille qui permettrait de la battre systématiquement.

Nous pensons évidement que notre intelligence artificielle n'est pas parfaite et qu'il y aurait moyen de l'améliorer. Nous allons émettre les différentes pistes qui pourraient éventuellement améliorer notre intelligence artificielle.

Avant de parler d'améliorations, il serait judicieux de préciser que les chercheurs actuels dans le domaine, n'ont toujours pas trouvé de modèle capable de battre les joueurs professionnels à 10 joueurs. Pour 2 joueurs, de nombreuses recherches ont été réalisées arrivant à battre un joueur professionnel.

Nous vous invitons aussi à lire la thèse de master, disponible en à référence 12, qui a été réalisée dans une des universités participants activement dans la réalisation de *Pokerbots*[5].

- **Les outs**

Au début de la réalisation de l'intelligence artificielle, nous étions partis sur un système qui compterait les *outs* de la main courante à partir de l'état *flop*. Pour finir, nous avons abandonné ce système car indirectement les simulations les prennent en compte. On va expliquer le principe ainsi que l'algorithme qui permet de les compter.

Définissons tout d'abord ce qu'est un *out* :

« Un out est une carte non connue qui peut améliorer une main courante. On compte les outs seulement lors de l'état flop et turn. »

L'idée est de trouver ces *outs* puis en déduire la probabilité d'améliorer notre main d'après le nombre de *outs* et de cartes inconnues. Cependant lors de la recherche d'*outs* il faut faire attention de ne pas compter des faux *outs*, cartes qui améliorerait plus une main d'un adversaire que la nôtre.

L'algorithme développé teste pour chaque carte inconnue, si elle améliore notre main et qu'elle n'améliore pas la *board*. Si c'est le cas alors on a un *out*. Puis nous enlevons les faux *outs* comme mentionné ci-dessus.

Exemple de calcul d'outs simple :

Nous avons dans notre main 4♠, J♠ et après le tirage du flop on a 6♠, A♥, T♠ sur la *board*. On va alors chercher d'améliorer notre main en visant la couleur. Nous avons 4♠, alors il en reste 7 (13 - 4) qui ne sont pas connues. Ainsi nous avons donc 7 *outs*.

Exemple de calcul d'outs avancé:

Nous avons dans notre main J♠, 8♣ et après le tirage du *flop* on a 9♠, T♥, J♣ sur la *board*. On va alors chercher d'améliorer notre main en visant la suite. Nous avons donc 8 *outs* qui sont les quatre Q et les quatre 7. Cependant s'il tombe une Q au *turn*, on aura bien une suite 8♣, 9♠, T♥, J♣, Q♦. Mais il y a trop de chance qu'un adversaire possède un K et donc qu'il ait aussi une suite, mais au Roi. 9♠, T♥, J♣, Q♦, K[♠|♥|♣|♦]. On a donc 4 faux *outs* les quatre Q, et les seules cartes qu'on espère vraiment sont les 7. Ainsi nous avons donc 4 *outs*.

[5] Plus d'informations sur https://en.wikipedia.org/wiki/Pokerbots

- ## Processus stochastique avancé

Dans un premier temps, on pourrait aussi introduire une fonctionnalité calculant l'espérance de gain d'un modèle pour une intelligence artificielle. Par exemple, il serait intéressant de pouvoir calculer l'espérance de gain afin de voir si au final notre intelligence artificielle est efficace ou non.

On pourrait réaliser un système qui évaluerait la main des joueurs adverses à l'aide d'une inférence bayésienne. On prend en compte ce que mise l'adversaire et on suppose la valeur du jeu de celui-ci. Si cela ne correspond pas aux attentes, alors on réajuste les calculs pour tenir compte de son caractère agressif. Ce qui amène à une nouvelle idée qui serait que chaque intelligence artificielle effectue un profil pour chaque adversaire (en utilisant l'équilibre de *Nash*[6] par exemple) et qu'elle en prend compte pour ses futurs choix.

Une dernière idée serait de remplacer nos simulations par des simulations utilisant la méthode de Monte Carlo[7].

- ## Réseau de neurones ou algorithme génétique

Imaginons que nous réalisons qu'en modifiant certains paramètres de nos fonctions, nous arrivons à obtenir une meilleure espérance de gain. Ça pourrait être intéressant d'implémenter un réseau de neurones ou un algorithme génétique qui évoluerait en fonction du temps. Par exemple l'intelligence artificielle jouerait d'une certaine manière et au bout d'un certain temps, elle modifierait certains paramètres et vérifierait si ces changements sont mieux. Si ça les améliore, elle repart sur ces nouvelles bases et ré-effectue de nouveaux ajustements. Dans le cas contraire, elle reviendrait sur ses pas et modifierait différemment les paramètres.

- ## Simulation, efficace ?

Actuellement nous avons des simulations en temps réel. Est-ce vraiment « bien » ? N'y aurait-il pas un meilleur moyen d'avoir autant d'efficacité voire une meilleure, tout en enlevant ces simulations et ainsi pouvoir utiliser ces ressources pour autres choses ? Y-aurait-il possibilité d'implémenter les simulations tout en utilisant la carte graphique ? Nous avons essayé vaguement cette dernière, mais nous aurions perdu beaucoup de temps si nous avions continué sur cette voie.

Dans le cas où les simulations sont bien, il serait intéressant de trouver le nombre de simulations nécessaires pour avoir une erreur maîtrisable, que nous définirons par exemple à 1%. Pour rappel :

$$\sup_{x\in\mathbb{R}} \left| P\left(\frac{1}{\sqrt{n\cdot\sigma^2}}\sum_{k=1}^{n}(X_k - E(X_K)) \le \mathrm{x}\right) - \phi(x)\right| \;\le\; \frac{E(|X_1 - E(X_1)|^3)}{\sigma^3\cdot\sqrt{n}}$$

Equation permettant de trouver le nombre de simulation, Équation 3

$$\lim_{n\to\infty}\sup_{x\in\mathbb{R}} \left| P\left(\frac{1}{\sqrt{n\cdot\sigma^2}}\sum_{k=1}^{n}(X_k - E(X_K)) \le \mathrm{x}\right) - \phi(x)\right| = 0$$

Affirmation montrant que si $n \to \infty$, l'erreur est nulle, Affirmation 4

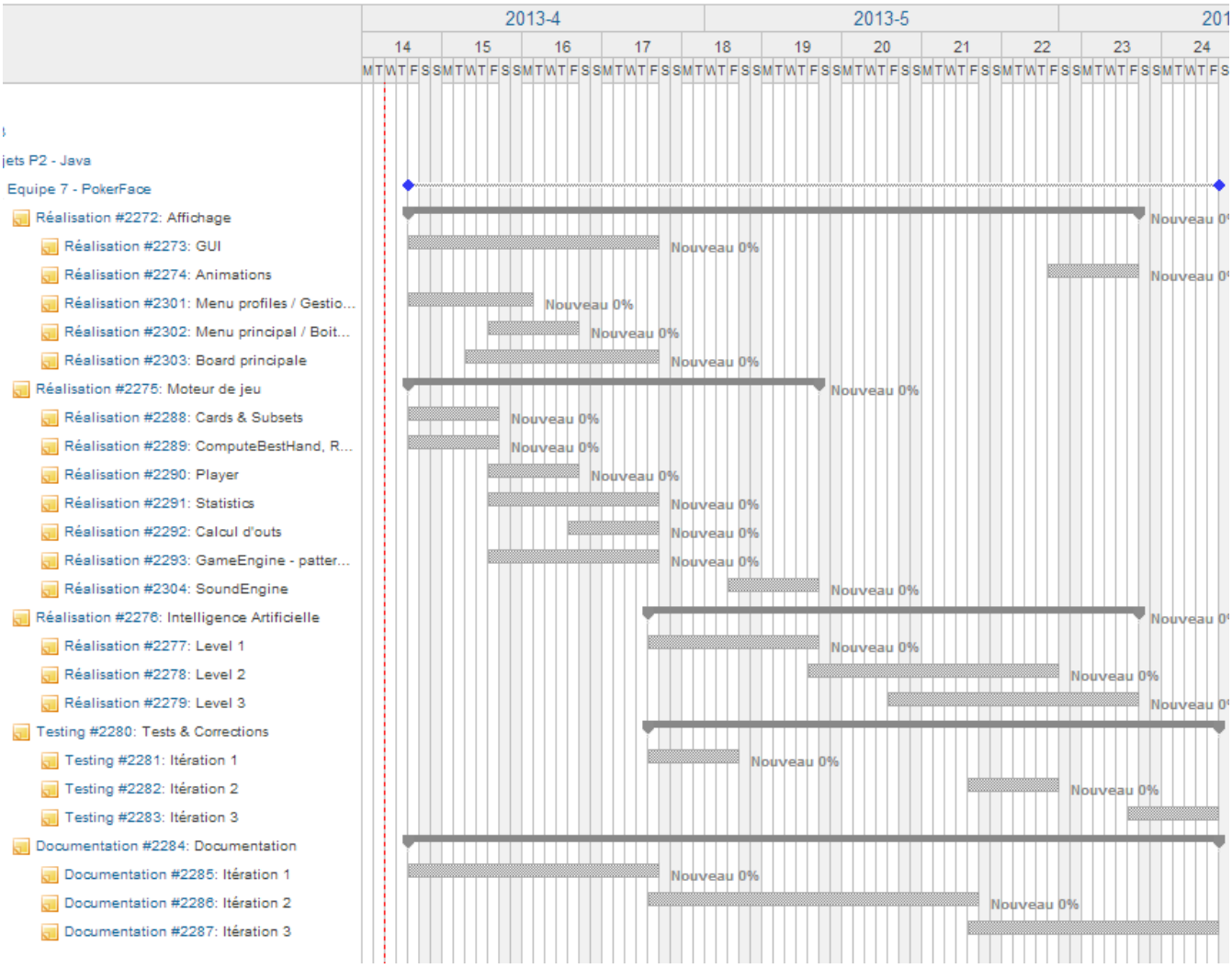
[6] Pour plus de détails, http://en.wikipedia.org/wiki/Nash_equilibrium
[7] Pour plus de détails, http://en.wikipedia.org/wiki/Monte_Carlo_simulation

6 Planification

Dans cette partie, nous allons décrire la planification initiale que nous avions établie puis la planification finale. Dans un deuxième temps nous allons comparer les différentes entre les deux et effectuer des constations que nous prendrons en compte pour nos futurs projets.

6.1 Planification initiale

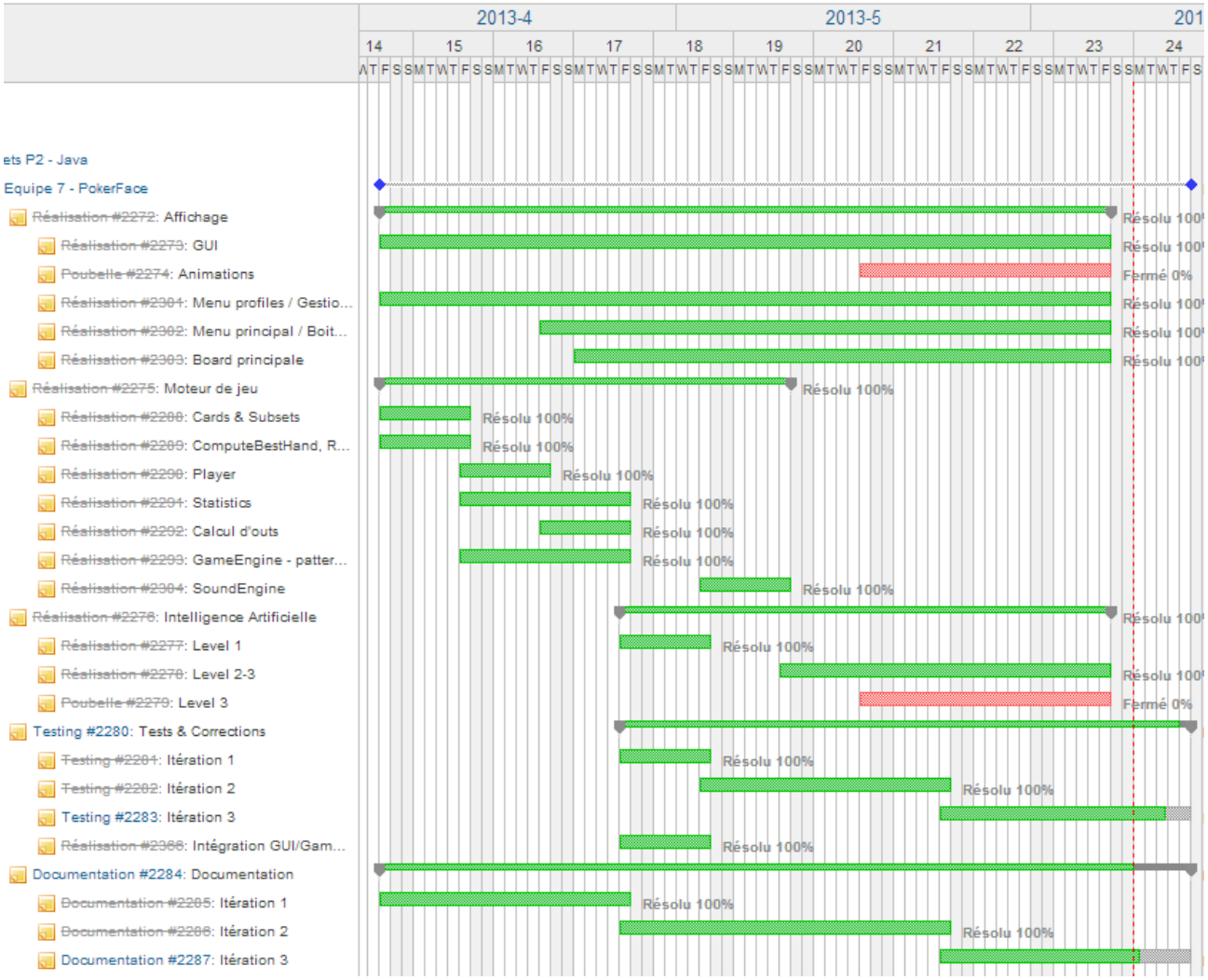
Voici la planification initiale que nous avons effectuée le 02 avril 2013 :



La planification a été découpée en 3 itérations de 3 semaines. Pour chacun des itérations nous avions une phase de tests et corrections ainsi que de documentation. Nous avions pensé que la partie graphique ne prendrait qu'une itération avec 1 personne et que l'on serait 3 à réaliser l'intelligence artificielle pendant 2 itérations. Tout ce qui concerne le moteur de jeu était prévu pour être fait en 1 itération par 2 personnes.

Pour les rôles, nous avions défini que Danick Fort s'occuperait en partie de l'interface graphique puis participerait au moteur de jeu avec Diego Antognini & Alexandre Perez.

6.2 Planification finale



Voici la planification actuelle (quasi finale), il ne reste plus qu'à finir le présent document. L'on peut voir que l'interface graphique prend les 3 itérations et avons supprimé les animations, pour deux raisons : temps et utilité. Pour finir, l'intelligence artificielle de niveau 1 a représenté peu de travail et nous avons regroupé celle de niveau 2-3 pour obtenir le résultat que nous avons.

Concernant les rôles, finalement Danick Forst s'est occupé essentiellement de l'interface graphique, sans participer à la réalisation du moteur de jeu.

6.3 Constatations

Premièrement, nous pouvons voir que nous avions mal estimé le temps nécessaire pour l'interface graphique et avons même dû supprimer la partie animation. L'intelligence artificielle de niveau 1 a été réalisée qu'en une semaine (elle ne représentait de suivre selon la cote du pot et la cote de la main) et nous avons regroupé celle de niveaux 2 et 3. A notre grand étonnement, nous n'avons jamais eu de retard dans notre application et pouvons constater que la différence entre la planification initiale et finale est moindre.

Quant aux rôles, ils n'ont pas été changés, mais nous avons estimé qu'il était préférable de laisser une personne à temps plein sur l'interface graphique, car de nombreuses étaient à réaliser.

7 Conclusion

A travers ce projet, nous avons eu l'occasion de nous initier à la réalisation d'une intelligence artificielle dans le domaine du *Poker*, partie principale de notre projet. Pour cela, nous avons basé notre système sur des simulations, des fonctions à 4 dimensions, un facteur risque ainsi qu'un facteur chance. Nous avons pu constater que dans l'ensemble l'intelligence artificielle semble effectuer des choix cohérents, tout en réalisant des bluffs ou des actions surprenantes, ce qui la rend imprévisible.

Actuellement, notre intelligence artificielle utilise, parmi un des facteurs, une probabilité de victoire selon la main qu'elle a, à l'aide de simulations qui peuvent être lourdes selon les ordinateurs. Si nous utilisons notre programme sur une vieille machine, la qualité de notre intelligence artificielle se verrait réduite car le nombre de simulations qu'elle pourrait effectuer se retrouvera diminué. Peut-être qu'elle semblera efficace mais seulement avec un nombre de joueurs peu élevé, nombre dépendant selon la puissance de la machine.

Sur plusieurs parties effectuées à jeu ouvert (cartes de tous les joueurs visibles) nous avons constaté que l'intelligence artificielle bat souvent le joueur humain, que ça soit à 2 ou 10 joueurs. Nous ne pouvons pas garantir qu'elle est efficace, tout ce que l'on peut dire est qu'avec des observations, il nous a semblé qu'elle jouait d'une manière cohérente et agressive. Pour cela il faudrait calculer l'espérance de gain ou alors utiliser un autre référentiel. Dans le monde du *Poker*, il y a une ligne idéale à suivre si l'on souhaite gagner. Les joueurs professionnels sont toujours très proches de cette ligne, quant aux joueurs débutants, ils ont tendance à en être éloignés et donc à perdre. Il serait intéressant de pouvoir estimer à quelle distance se situe notre intelligence artificielle. Pour finir, nous proposons diverses voies améliorations au chapitre 5.5.

L'autre aspect intéressant de ce projet, même si on a tendance à l'oublier, fût la réalisation du moteur de jeu, notamment la division du pot, les enchères ainsi que les différentes règles à appliquer. Nous avons constaté qu'il y avait plusieurs exceptions compromettantes que nous avons dû tenir compte lors de la réalisation du moteur de jeu. Mais finalement, nous pouvons constater que le moteur semble fonctionner. Dans sa version finale, nous ne constatons aucune erreur.

Neuchâtel, 14 juin 2013

Diego Antognini Danick Fort Alexandre Perez

8 Bibliographie

1. Probabilités au poker, http://www.chip-leaders.com/les-probabilites-damelioration-dune-main/
2. Probabilités au poker, http://www.poker-academie.com/apprendre-poker/strategie-poker/no-limit-holdem/
3. Cotes au poker, http://www.conseilpoker.com/strategies/odds-outs-les-probabilites-du-poker
4. Cotes au poker, http://www.howstuffworks.com/how-to-calculate-poker-odds.htm
5. Système statistique en live, http://fr.pokernews.com/poker-tools/poker-odds-calculator.htm
6. Les 7462 mains uniques, http://www.suffecool.net/poker/7462.html
7. Les *outs* au poker, http://fr.pokernews.com/strategie/poker-calcul-outs-probabilite-amelioration.htm
8. Les *outs* au poker, http://www.tightpoker.com/poker_odds.html
9. Calcul des *outs* au poker, http://www.codeproject.com/Articles/19091/More-Texas-Holdem-Analysis-in-C-Part-1
10. Article intéressant sur l'évaluation de main, http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup/
11. *Probabilities of Poker Hands with Variations*, de Jeff Duda, http://www.meteor.iastate.edu/~jdduda/portfolio/492.pdf
12. Thèse de master "*Opponent Modeling in Poker : Learning and Acting in a Hostile and Uncertain Environment*", par Aaron Davidson, de *University of Alberta*, http://webdocs.cs.ualberta.ca/~darse/Papers/davidson.msc.pdf