MO50 - Rapport Final

Alexis Bouligand,Lucas Bruton,Nathan Damette,Oscar Dewasmes,Maxime Szymanski

June 11, 2023

Professeur référent :
Yazan Mualla

Contents

1 Introduction

1.1 Sujet

Le jeu Splendor se joue de 2 à 4 joueurs, le but est d'être le premier joueur à accumuler 15 points de victoires. Le plateau est composé d'une banque et d'un magasin où l'on peut acheter des cartes. À son tour, un joueur choisit une des 4 actions disponibles : Prendre 3 ressources différentes dans la banque. Prendre 2 ressources de la même couleur dans la banque. Réserver une carte à acheter plus tard. Et acheter une carte, du magasin ou de sa réserve.

Le plateau de jeu est facile à décrire informatiquement, et le nombre d'actions possible est faible. Cependant, le nombre de ramifications de l'arbre d'état d'une partie est complexe. Cela fait de Splendor un environnement propice au développement d'intelligence artificielle.

1.2 Contraintes

Ce projet doit respecter la contrainte légale du *copyright*. En effet, le texte des règles et les images du jeu Splendor sont sous copyright appartenant à SPACE Cowboys©. Cependant, une mécanique de jeu ne peux pas être contrainte au *copyright*, ainsi, un jeu de plateau peux toujours être adapté.

2 Spécifications

2.1 Réponse aux besoins

2.1.1 Besoins fonctionnels

- Nous avons créé une version de Splendor en application web
- Nous avons créer un joueur IA

2.1.2 Besoins non-fonctionnels

- L'IA joue en moins de 5 secondes (instantané)
- L'UI est agréable et facile à utiliser
- Nous avons créé une code base extensible

2.2 Limites du projet

- Il n'y aura qu'un joueur humain par partie
- Serveur web local, n'accepte qu'une seule connexion
- Jeu limité à 1 joueur contre 1 IA

2.3 Décisions de design

Au début du projet, nous avons pris des décisions de design que nous avons réussis à suivre :

- L'expérience utilisateur a primé sur la sécurité.
- L'extensibilité a primé sur la vitesse de jeu.
- Le temps de développement a primé sur la qualité.

2.4 Technologie utilisée

Nous avions prévu d'utiliser Django pour la partie back end et api du projet. Nous avons changé d'avis et utilisé flask à la place car c'est un framework moins lourd qui s'adapte mieux à nos besoins.

Front-end

- Angular (14.3.0) : framework web open-source basé sur le TypeScript

Back-end :

- Python (3.10.10)
- Flask (2.3.1) : microframework web open-source pour Python
- Open AI Gym (0.26.2) : librairie Python open-source permettant de développer des algorithmes d'apprentissage
- Numpy (1.24.0) : bibliothèque mathématique pour python

Organisation, communication, management :

- Git (2.40.0) : outil de gestion des versions et partage du code
- Trello (latest) : outil kaban pour l'organisation de projet
- Discord (latest) : outil de communication, organisation d'équipe
- PlantUML (1.2023.4) : outil de description UML

- draw.io (12.0.2) : outil de schématisation et dessin de diagrammes
- Overleaf (3.5.2) : Éditeur multi-utilisateur pour LaTeX

3 Organisation du projet

3.1 Méthode Agile

Pour gérer ce projet, nous avons utilisé la méthode Scrum. Les bonnes pratiques et rituels ont été ajoutés à la méthodologie au fil du développement et certains enlevés par soucis de praticité, comme nous allons le voir plus loin.

3.1.1 Backlog, product owner

Certaines parties de la méthode agiles n'ont été que partiellement mises en oeuvre.
Le backlog a été peu populé lors du début du sprint, et nous y avons rarement jeté notre regard à la fin d'un sprint. Le remplissage des user stories du sprint backlog s'est en réalité entièrement fait lors de nos sprint planning en créeant les user stories sur le moment.
Un product owner a été désigné au début du projet. Cependant, n'ayant pas de véritable user mais un livret de règles déjà défini, les tâches à réaliser étaient mineures.

3.1.2 Sprint

Nous avons décidé de faire des sprints de deux semaines. Cela nous a permis d'avoir un suivi plus serré du projet et de s'apercevoir des retards plus rapidement. La durée de ces sprints s'est resserrée en fin de projet. Ces objectifs ont été indiqués sur le Trello. En voici quelques exemples.

- 11/04 : Terminer model et intégrer à Django. Angular : créer l'interface, sans les call api. IA : Faire un agent. Masquer les actions invalides. Pipeline d'entrainement.
- 24/05 : Avoir un jeu terminé
- 30/05 : Terminer les derniers détails pour avoir un jeu qui fonctionne et commencer le rapport, la vidéo et l'oral

3.1.3 Stand up

Nous réalisions un stand up toutes les semaines afin d'avoir un point sur notre avancement respectif. Un stand up journalier aurait en effet été éxagéré.

3.1.4 Sprint retrospective

Avant chaque sprint planning, un feedback du sprint est donné et noté par écrit pour s'en souvenir. En voici les notes :

- Premier sprint : Ce qui n'a pas été fait : L'US(User Story) pour les references pas faites car trop de travail pour pas assez de reward. 2 US bloquées parce qu'on a pas le cour. Tuto view dans angular. Interface à corriger. Diagramme de classe à corriger. On a ajouté des choses à l'US risques en cours de dev donc US pas terminée Ajout d'une US pour le diagramme d'activité. Mettre une US sur l'IA pour Maxime.
- 11/04 : Mieux définir les taches. On a trop de gens sur une tache et lorsque certaines ne sont pas utiles pour tout de suite, on les remet dans le backlog.
- 24/04 : Oscar n'a pas regardé le Trello. Alexis s'est donné trop de boulot sur une US. Pour l'IA, mettre un seuil de validation. Éviter d'avoir des US sur plusieurs sprints et faire des US correctes.
- 08/05 : Mal évalué la charge de travail pour le front. Eviter de rerépartir la charge du front en milieu du sprint. Faire des git pull de master à chaque changement. Prévenir que l'arborescence des fichiers a changé. Un changement sur master a cassé les autres.

3.1.5 Sprint planning et sprint review

Notre sprint planning se faisait juste après la sprint retrospective. La sprint review se faisait à chacun de nos rendez vous avec M. Mualla.

3.1.6 User stories et epics

Nous avons utilisé Trello pour suivre nos user Stories et Epics. Chaque user story doit correspondre à une feature et etre décrite selon le schéma suivant :

```
As a ...
I want to ...
So that ...
```

La mise en place des user story a beaucoup évolué tout a long du développement. Au début, celles-ci décrivaient une solution technique, non pas une feature, et ne possédaient pas de descriptions. Au milieu du développement nous avons décidé d'être plus strict sur ces points et pallier progressivement à ces problèmes. Nous nous sommes alors aperçu que la plupart de nos user Story étant assez petite, la description n'était pas nécessaire.

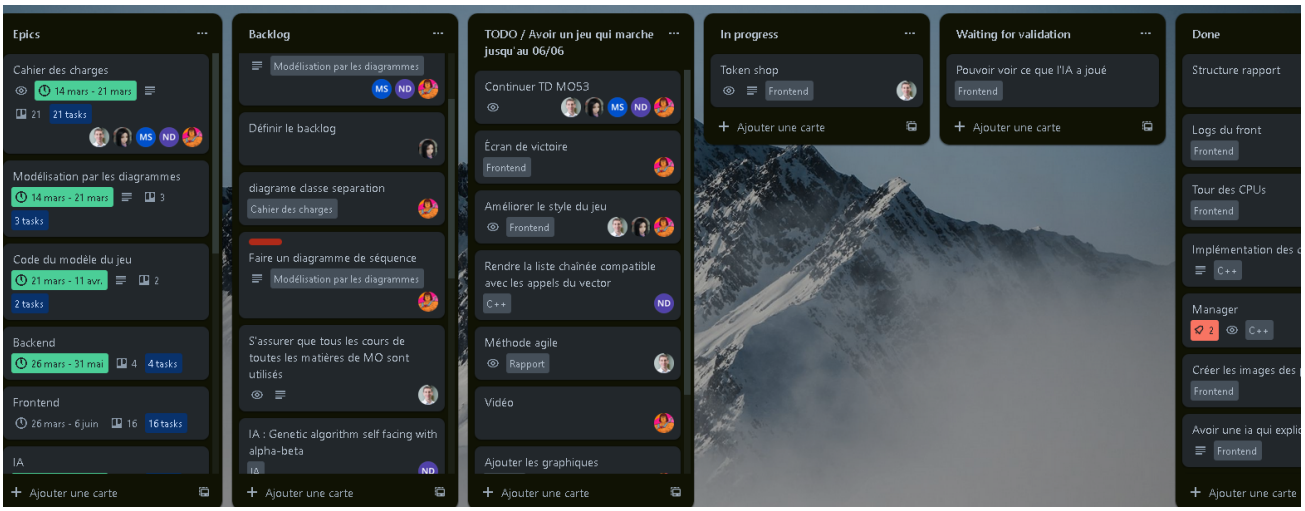

Figure 1 : Screenshot du Trello, les epics sont les cartes de la colonne à gauche

3.2 Diagramme WBS

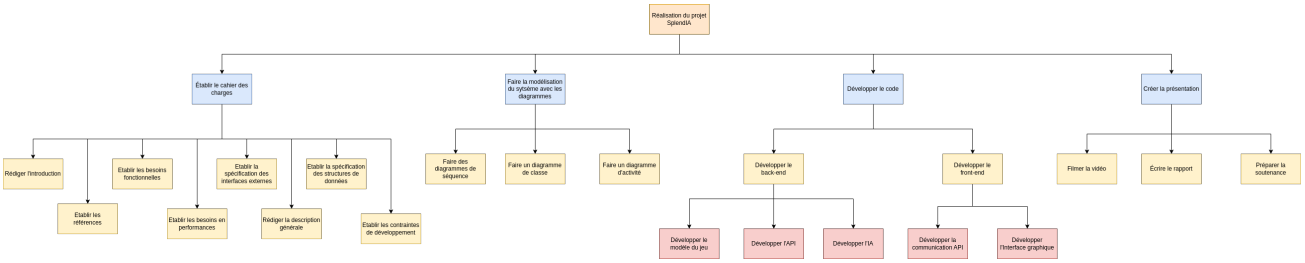

Figure 2 : Diagramme WBS

3.3 Diagramme Pert

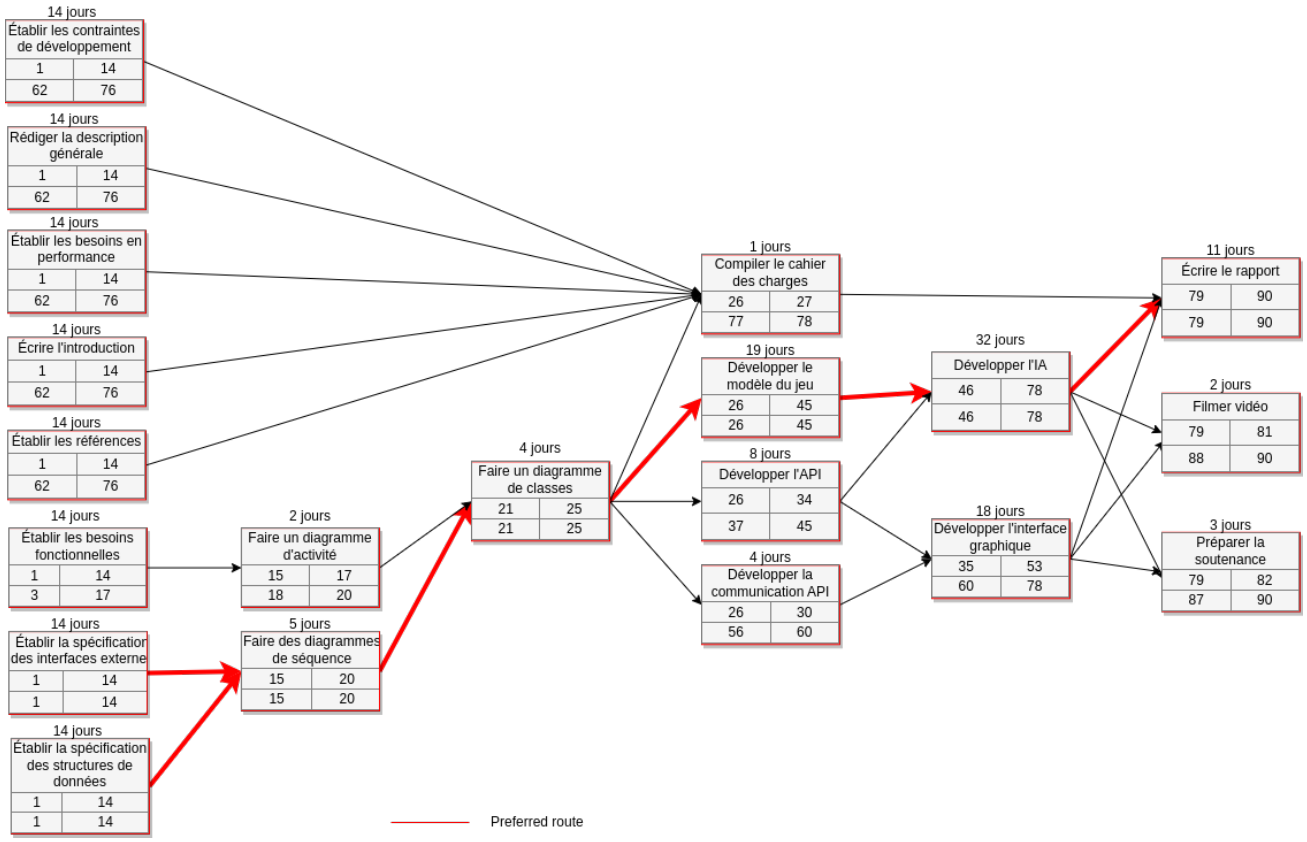

Figure 3 : Diagramme Pert

3.4 Diagramme de Gantt

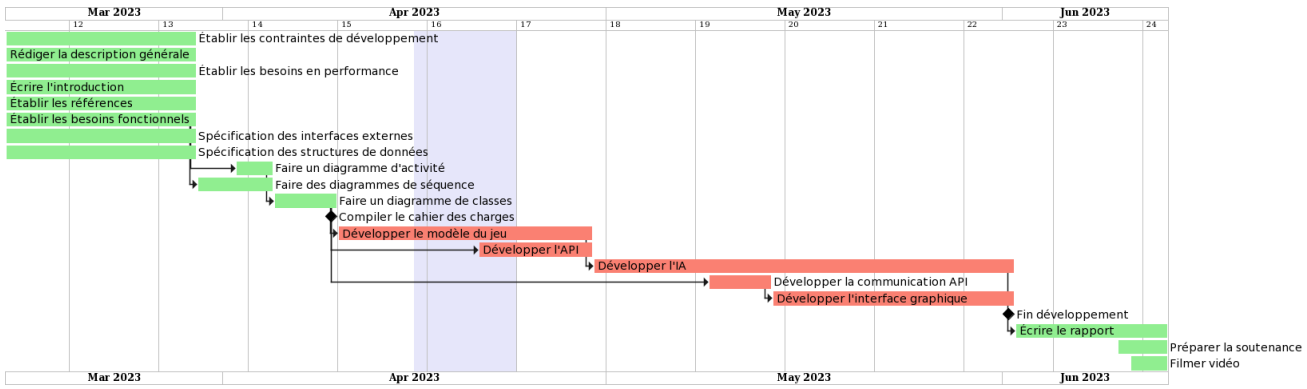

Figure 4 : Diagramme de Gantt

3.5 Github

4 Modélisation

Une important phase de modélisation nous a apporté beaucoup d'avantages, d'ordre organisationnel mais aussi qualitatif. En effet, modéliser nous a permis de mieux estimer la quantité de travail à fournir pour chaque parties du projet et la modélisation nous a permis de créer une code base extensible, logique et facile à modifier.

4.1 Diagramme de cas d'utilisation

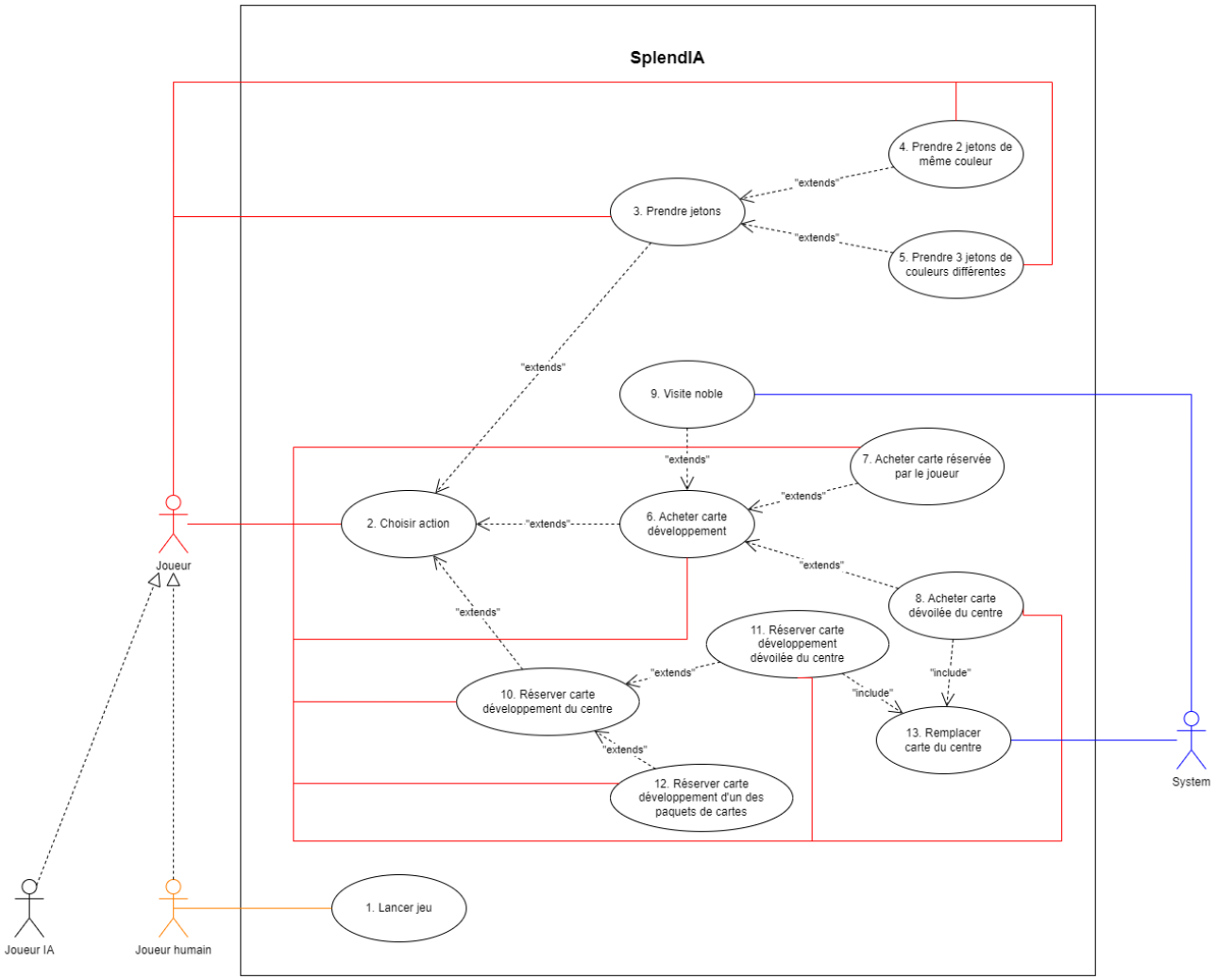

Figure 5 : Diagramme des cas d'utilisation

4.2 CRC cards

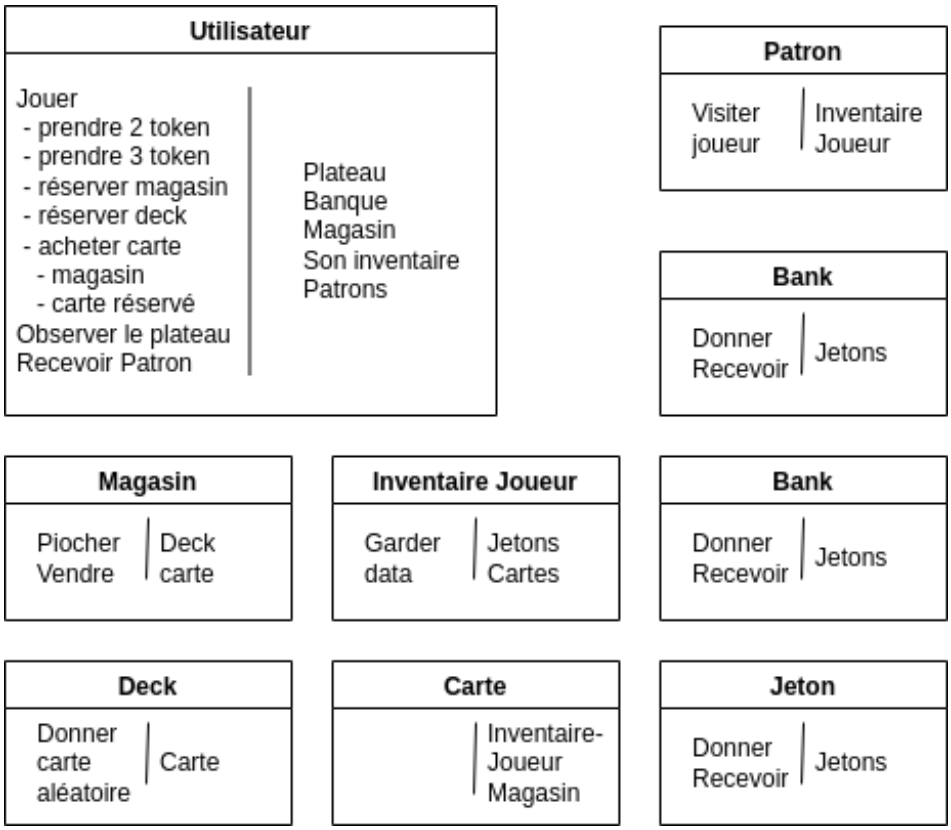

Figure 6 : Diagramme de classes responsabilité et collaboration

4.3 Diagramme de classes

Un diagramme de classes complet du modèle est disponible en annexe.

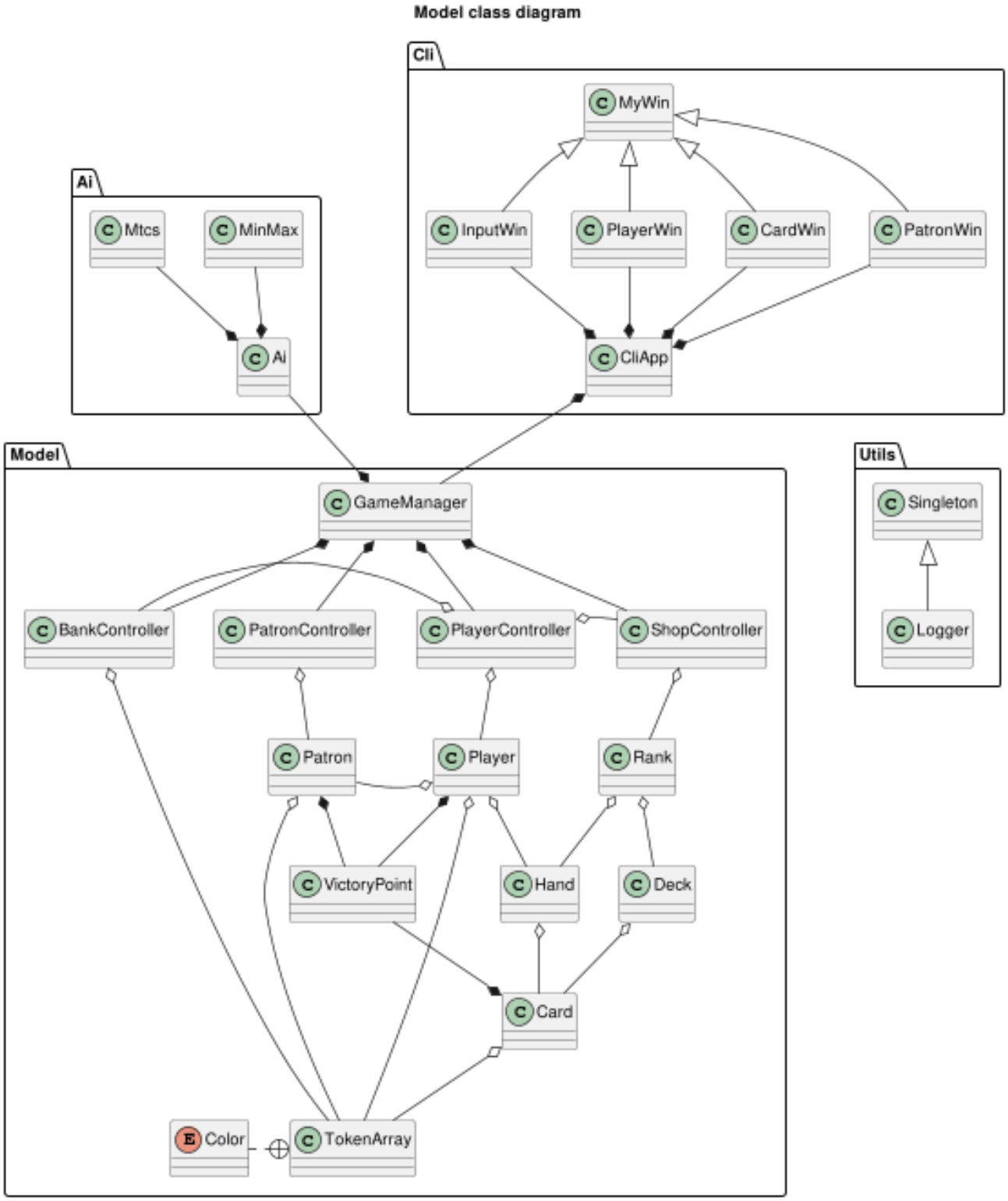

Figure 7 : Diagramme de classes simplifié

4.4 Diagramme de séquence

Plus de diagrammes de séquences sont disponibles en annexe.

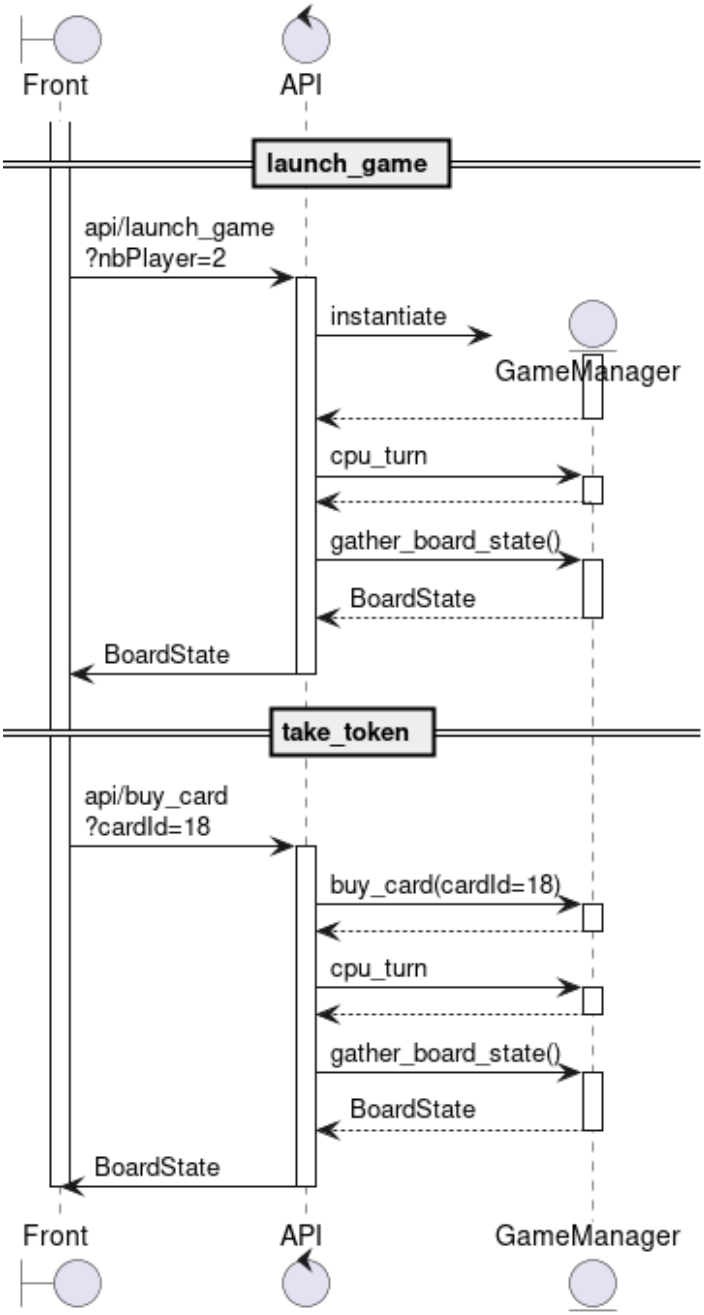

Figure 8 : Diagramme de séquence

5 Intelligence artificielle

5.1 Algorithmes utilises

5.1.1 PPO

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize policy network parameters θ
Initialize value network parameters ϕ
Set hyperparameters: ϵ, γ, K, T
Initialize empty experience buffer D
for iteration $= 1$ **to** K **do**
  Collect trajectories using the current policy: $s_t, a_t, r_t{}_{t=1}^{T}$
  Compute advantages A_t using the value function estimates
  Normalize advantages: $\hat{A}_t \leftarrow \frac{A_t - \mu(A_t)}{\sigma(A_t)}$
  for $t = 1$ **to** T **do**
    Compute old policy probability: $P^{\text{old}}(a_t|s_t)$
    Update policy parameters via gradient ascent:

$$\theta \leftarrow \theta + \alpha \min\left(\frac{P(a_t|s_t)}{P^{\text{old}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{P(a_t|s_t)}{P^{\text{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)$$

    Update value function parameters by minimizing the mean squared error:

$$\phi \leftarrow \phi - \beta\frac{\partial V(s_t;\phi)}{\partial \phi}(V(s_t;\phi) - R_t)$$

  end for
end for

L'algorithme Proximal Policy Optimization (PPO) est une méthode d'apprentissage par renforcement utilisée pour entraîner des agents à prendre des décisions dans des environnements dynamiques. PPO est une approche basée sur la politique qui vise à optimiser directement la politique de prise de décision de l'agent.

L'algorithme PPO est itératif et consiste en plusieurs étapes clés. Tout d'abord, les paramètres du réseau de politique θ et du réseau de valeur ϕ sont initialisés. Ensuite, des hyperparamètres tels que ϵ, γ, K et T sont définis pour régler le comportement de l'algorithme.

L'étape principale de l'algorithme PPO est la collecte de trajectoires à l'aide de la politique actuelle. Ces trajectoires sont générées en interagissant avec l'environnement et en enregistrant les états s_t, les actions a_t et les récompenses r_t pour chaque pas de temps jusqu'à l'horizon T.

Une fois les trajectoires collectées, les avantages A_t sont calculés en utilisant les estimations de la fonction de valeur. Les avantages mesurent à quel point une action était meilleure que prévu dans un état donné. Ensuite, les avantages sont normalisés pour faciliter l'optimisation.

L'étape suivante de PPO consiste à mettre à jour la politique et la fonction de valeur. Pour mettre à jour la politique, une montée de gradient est effectuée pour maximiser la probabilité des actions prises par la politique actuelle. Cependant, une contrainte de mise à jour est introduite pour éviter des modifications drastiques de la politique. Cette contrainte est réalisée en utilisant une fonction de pénalité qui clippe la montée de gradient à un intervalle donné autour du ratio des probabilités d'actions entre la nouvelle et l'ancienne politique.

En parallèle, la fonction de valeur est mise à jour en minimisant l'erreur quadratique moyenne entre les estimations de la valeur de l'état et les récompenses réelles obtenues. Cela permet d'obtenir une estimation plus précise de la valeur des états, ce qui facilite l'estimation des avantages.

Ces étapes de collecte de trajectoires, de calcul des avantages et de mise à jour de la politique et de la fonction de valeur sont répétées pour un nombre fixe d'itérations défini par le paramètre K. Cela

permet à l'agent d'apprendre progressivement à prendre de meilleures décisions dans l'environnement.

5.1.2 Algorithme Génétique

1. Générer une population initiale de solutions aléatoires
2. Évaluer chaque individu de la population en utilisant une fonction d'évaluation
3. Initialiser une génération courante à 0
4. Tant que le critère d'arrêt n'est pas satisfait:
   (a) Sélectionner les individus parents pour la reproduction en utilisant des méthodes de sélection (ex. : tournoi, roulette)
   (b) Appliquer des opérateurs de recombinaison (ex. : croisement, crossover) pour créer une nouvelle population d'individus
   (c) Appliquer des opérateurs de mutation pour introduire de la diversité dans la population
   (d) Évaluer chaque nouvel individu de la population en utilisant la fonction d'évaluation
   (e) Sélectionner les meilleurs individus pour former la prochaine génération
   (f) Incrémenter la génération courante
5. Retourner la meilleure solution trouvée dans la population finale

1. Générer une population initiale de solutions aléatoires : Au début de l'algorithme, une population de solutions candidates est créée de manière aléatoire. Chaque solution représente un individu potentiel.

2. Évaluer chaque individu de la population en utilisant une fonction d'évaluation : Chaque individu est évalué en utilisant une fonction d'évaluation qui mesure à quel point il est performant par rapport à l'objectif du problème. Cette évaluation permet de quantifier la qualité de chaque solution.

3. Initialiser une génération courante à 0 : Une variable de comptage est initialisée pour suivre les générations d'individus créées.

4. Tant que le critère d'arrêt n'est pas satisfait : L'algorithme génétique continue à évoluer la population jusqu'à ce qu'un critère d'arrêt prédéfini soit satisfait. Ce critère peut être un nombre maximal de générations, une performance minimale atteinte, ou toute autre condition spécifique.

4.1. Sélectionner les individus parents pour la reproduction : Les individus les plus performants de la génération courante sont sélectionnés comme parents pour la reproduction. Différentes méthodes de sélection peuvent être utilisées, telles que le tournoi où les individus sont comparés et les meilleurs sont choisis, ou la roulette où la probabilité de sélection est proportionnelle à la performance.

4.2. Appliquer des opérateurs de recombinaison : Les opérateurs de recombinaison, tels que le croisement (crossover), sont appliqués aux parents sélectionnés pour créer de nouveaux individus. Le croisement combine les caractéristiques des parents pour générer des enfants qui peuvent potentiellement avoir des performances améliorées.

4.3. Appliquer des opérateurs de mutation : Pour introduire de la diversité dans la population, des opérateurs de mutation sont appliqués aux nouveaux individus. La mutation modifie aléatoirement certaines parties de la solution pour explorer de nouvelles possibilités et éviter de rester bloqué dans des optima locaux.

4.4. Évaluer chaque nouvel individu de la population : Les nouveaux individus, issus de la recombinaison et de la mutation, sont évalués en utilisant la même fonction d'évaluation que précédemment.

4.5. Sélectionner les meilleurs individus pour former la prochaine génération : Les individus les plus performants de la nouvelle population, parmi les parents et les enfants, sont sélectionnés pour former la génération suivante. Cela garantit une évolution vers de meilleures solutions au fil des générations.

4.6. Incrémenter la génération courante : La variable de comptage des générations est incrémentée pour suivre la progression de l'algorithme.

Retourner la meilleure solution trouvée dans la population finale : Une fois que le critère d'arrêt est satisfait, l'algorithme génétique se termine et la meilleure solution trouvée au sein de la population finale est renvoyée comme résultat.

5.1.3 Monte-Carlo

L'algorithme de Monte Carlo est une méthode statistique utilisée pour estimer des quantités inconnues en utilisant des échantillons aléatoires.

Étape 1 : Générer des échantillons

Dans un premier temps, des échantillons aléatoires sont générés à partir de la distribution appropriée. Cela peut être fait en utilisant des méthodes telles que l'échantillonnage aléatoire simple ou l'échantillonnage selon une distribution spécifique.

Étape 2 : Évaluer les échantillons

Ensuite, chaque échantillon est évalué à l'aide de la fonction ou du modèle approprié. Par exemple, si nous voulons estimer l'espérance d'une variable aléatoire, chaque échantillon est évalué en utilisant la fonction d'intérêt.

Étape 3 : Calculer l'estimation

Une fois que tous les échantillons ont été évalués, une estimation de la quantité inconnue est calculée à partir des échantillons. Cela peut être fait en prenant la moyenne des évaluations ou en utilisant d'autres techniques statistiques appropriées.

Étape 4 : Répéter le processus

Pour améliorer la précision de l'estimation, les étapes précédentes sont répétées en générant un plus grand nombre d'échantillons et en recalculant l'estimation. Cela permet de réduire l'erreur d'estimation et d'obtenir une meilleure approximation de la quantité inconnue.

Étape 5 : Analyser les résultats

Une fois que suffisamment d'itérations ont été effectuées, les résultats obtenus sont analysés. Cela peut inclure le calcul de l'erreur d'estimation, l'intervalle de confiance ou toute autre mesure pertinente pour évaluer la qualité de l'estimation.

En résumé, l'algorithme de Monte Carlo est une méthode statistique puissante qui utilise des échantillons aléatoires pour estimer des quantités inconnues. En générant des échantillons, en les évaluant et en répétant le processus, il permet d'obtenir des estimations de plus en plus précises. Cependant, le succès de l'algorithme dépend de la qualité des échantillons générés et de la modélisation adéquate de la quantité inconnue.

5.1.4 Alpha Beta

L'algorithme Alpha-Beta est une technique utilisée dans les jeux à deux joueurs avec une stratégie d'exploration de l'arbre de recherche pour améliorer l'efficacité de la recherche en élaguant certaines branches non pertinentes.

Étape 1 : Recherche récursive

L'algorithme Alpha-Beta effectue une recherche récursive dans l'arbre de recherche du jeu. Il explore les différents coups possibles en alternant entre les joueurs jusqu'à une certaine profondeur ou jusqu'à ce qu'une condition de terminaison soit atteinte.

Étape 2 : Valeur d'évaluation

À chaque nœud de l'arbre, une fonction d'évaluation est utilisée pour estimer la valeur de la position du jeu. Cette fonction attribue une valeur numérique à chaque position du jeu en fonction de certaines heuristiques spécifiques au jeu.

Étape 3 : Alphabeta

L'algorithme Alpha-Beta utilise deux paramètres, α (alpha) et β (beta), pour effectuer l'élagage. α représente la meilleure valeur maximale trouvée par le joueur maximisant jusqu'à présent, tandis que β représente la meilleure valeur minimale trouvée par le joueur minimisant.

Étape 4 : Élagage Alpha-Beta

Pendant la recherche, l'algorithme effectue un élagage des branches inutiles en comparant les valeurs α et β avec les valeurs d'évaluation des nœuds. L'algorithme met à jour les valeurs α et β en fonction des valeurs d'évaluation rencontrées, en gardant seulement les branches qui ont une chance d'améliorer la valeur finale.

Étape 5 : Coupe Alpha-Beta

Lorsque l'algorithme rencontre une situation où β est inférieur ou égal à α, il sait que la branche actuelle peut être coupée, car le joueur minimisant ne la choisira pas. Ainsi, l'algorithme s'arrête d'explorer cette branche et remonte dans l'arbre de recherche.

Étape 6 : Retourner le meilleur coup

Une fois la recherche terminée, l'algorithme Alpha-Beta retourne le meilleur coup trouvé, qui est le coup qui mène à la position avec la meilleure valeur d'évaluation possible, en tenant compte des élagages effectués.

5.2 Resultats

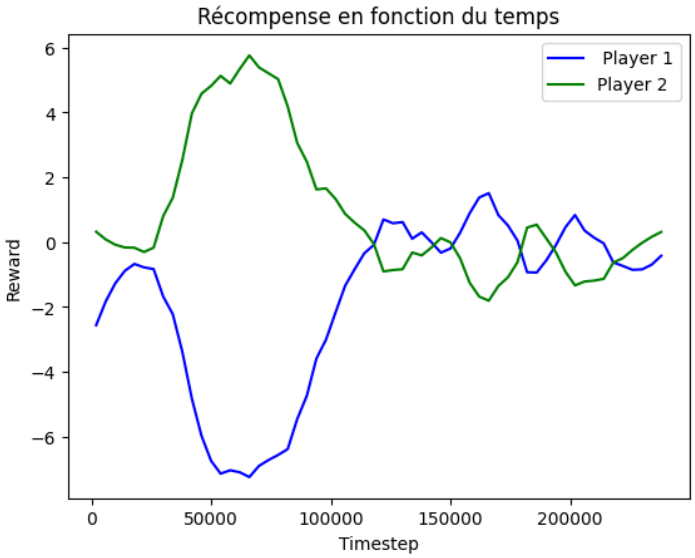

Figure 9 : Reward per timestep

Au cours de notre étude approfondie du jeu Splendor, nous avons comparé plusieurs algorithmes d'intelligence artificielle pour déterminer leur efficacité dans la résolution de ce jeu complexe. Nous avons implémenté les algorithmes Monte Carlo, Alpha-Beta, les algorithmes génétiques et l'algorithme PPO (Proximal Policy Optimization) pour évaluer leurs performances respectives.

Nos résultats ont révélé que les approches traditionnelles telles que Monte Carlo, Alpha-Beta et les algorithmes génétiques, bien qu'elles aient montré certaines capacités à jouer à Splendor, étaient limitées par des temps de calcul considérables pour des résultats souvent peu encourageants. Les méthodes basées sur Monte Carlo ont souffert d'une convergence lente et d'une précision limitée, ce qui a rendu difficile leur utilisation pour prendre des décisions stratégiques efficaces. De plus, bien qu'Alpha-Beta ait pu réduire l'espace de recherche, il était toujours coûteux en termes de temps de calcul et a eu du mal à rivaliser avec des joueurs humains expérimentés. Les algorithmes génétiques ont montré des résultats prometteurs, mais ils nécessitaient souvent un grand nombre d'itérations pour converger vers des stratégies optimales, ce qui entraînait des temps de calcul prolongés.

En revanche, l'algorithme PPO a démontré des performances exceptionnelles dans le jeu Splendor.

PPO utilise une combinaison de méthodes de Monte Carlo et d'optimisation basée sur des politiques pour améliorer les performances des agents. Grâce à son architecture spécifique et à son processus d'optimisation, PPO a réussi à surpasser les autres algorithmes et à rivaliser directement avec les joueurs humains. Il a démontré une capacité à prendre des décisions stratégiques précises, à exploiter les opportunités offertes par le jeu et à s'adapter aux différentes situations de jeu.

Ces résultats mettent en évidence l'efficacité de l'algorithme PPO dans la résolution de jeux complexes tels que Splendor. PPO a le potentiel de révolutionner la façon dont les jeux sont joués, en offrant des adversaires virtuels compétitifs et stimulants pour les joueurs humains. De plus, l'efficacité de PPO en termes de temps de calcul le rend attrayant pour des applications dans d'autres domaines où des décisions stratégiques doivent être prises rapidement et avec précision.

5.3 Pistes d'amélioration

Dans le contexte spécifique du jeu Splendor, l'algorithme PPO présente de solides performances et peut être une solution prometteuse pour rivaliser avec les joueurs humains. Contrairement aux approches traditionnelles telles que Monte Carlo, Alpha-Beta et les algorithmes génétiques, qui peuvent nécessiter beaucoup de temps de calcul pour des résultats mitigés, PPO a démontré une capacité à prendre des décisions stratégiques précises et à s'adapter aux différentes situations de jeu.

PPO bénéficie de certaines améliorations clés qui le distinguent des autres algorithmes. Il utilise une combinaison de méthodes de Monte Carlo et d'optimisation basée sur des politiques pour améliorer les performances des agents. Cela lui permet de surpasser les approches traditionnelles en termes d'efficacité et de capacité à rivaliser avec les joueurs humains expérimentés.

Dans le jeu Splendor, PPO peut être optimisé en utilisant des techniques spécifiques à ce jeu. Cela comprend l'encodage efficace des informations du jeu, l'exploration stratégique adaptée à Splendor, l'optimisation des hyperparamètres spécifiques à Splendor, l'exploitation des stratégies adverses et l'entraînement sur des scénarios spécifiques pour se spécialiser dans des situations particulières.

En explorant ces pistes d'amélioration spécifiques à Splendor, il est possible de renforcer davantage les performances de PPO dans ce jeu. L'amélioration de l'algorithme PPO permettrait de créer des adversaires virtuels compétitifs et stimulants pour les joueurs humains, ouvrant ainsi de nouvelles perspectives passionnantes pour l'utilisation de l'intelligence artificielle dans le domaine des jeux de société et des jeux stratégiques en général.

5.4 Hybrid Intelligence

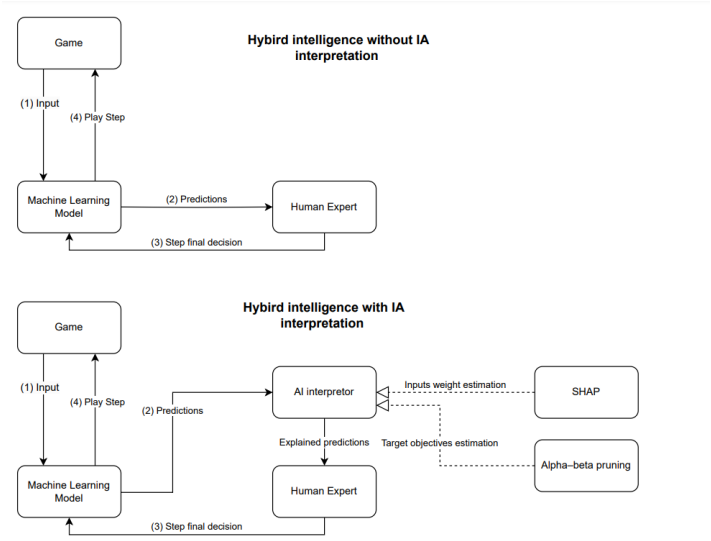

Figure 10 : Hybrid intelligence diagrams

L'intelligence hybride utilisée dans ce système de jeu repose sur une collaboration entre un modèle de machine learning et un expert humain. Lorsqu'une situation de jeu se présente, l'état du jeu est transmis au modèle de machine learning qui analyse cet état et calcule les meilleurs coups possibles en fonction de ses connaissances et de son apprentissage. Le modèle sélectionne ensuite les coups les plus prometteurs.

Ces coups sont ensuite soumis à l'expert humain qui a la décision finale quant à quel coup jouer. L'expert examine les propositions du modèle et, en utilisant son expertise et sa compréhension du jeu, prend une décision éclairée sur le coup à jouer parmi les choix proposés. Cette décision de l'expert est renvoyée au modèle de machine learning qui exécute alors le coup dans le jeu.

Pour faciliter la collaboration entre l'expert humain et le modèle de machine learning, un interpréteur d'IA peut être intégré. Cet interpréteur agit comme un pont entre les prédictions du modèle de machine learning et les retours fournis à l'expert. Il peut utiliser des algorithmes tels que SHAP (Shapley Additive Explanations) ou l'Alpha-Beta Pruning pour fournir à l'expert une meilleure compréhension des propositions du modèle. Cela permet à l'expert de prendre des décisions plus éclairées et de mieux comprendre les raisons derrière les choix proposés par le modèle de machine learning.

5.5 Résumé

Name	Pros	Cons	Results	Commentary
PPO vs PPO: self play with one neural network	Can play alone without human hard coded data. Can simulate thousands of games very quickly. Can strengthen its own weakness by beating himself.	Difficulty to benchmark. Can struggle in local minima. Lack of gameplay variation.	Weak results. Difficulty to converge. Lot of blocked games (more than 30%).	Good starting point, had to be tried, but other methods are really better.
PPO vs PPO: self-play with two neural networks	Can learn a lot of policies by simulating multiple opponents. Allow to generalize by playing against various policies. Can benchmark against the best agent called "The Champion".	Difficulty to converge, and to maintain a good winrate against all those policies. Risk to be stuck and never converge.	First results seem good, but humans have to play manually against the champion to evaluate it.	To be continued...
MTCS: Monte Carlo Tree Search	No need of heuristic values (i.e., no need to understand the game). Can generalize the policies and explore all possible actions at a given turn.	Huge computational resources are needed to evaluate MTCS. Lack of accuracy due to the depth of the Splendor's tree and the random factor of the game.	Weak results for naive MTCS. Takes too much time to compute (2 min for one decision for 1000 games for each child node).	Naive MTCS is a bad idea due to the depth of Splendor's tree. Yet, we are thinking about an advanced MTCS algorithm that explores only 'good children', determined by a neural network.
PPO vs MTCS	Same as PPO vs PPO and MTCS	Same as PPO vs PPO and MTCS	Takes too much time, MTCS is not well-suited to train the PPO because MTCS itself is too weak.	Very quickly avoided.
Alpha-Beta algorithm	Can define a specific policy using heuristic value. Used pruning to only compute relevant nodes. Have a 'mid-short term' policy by taking into account the 3-4 next turns.	Heuristic value is hard to find, especially in Splendor where the optimal policy depends on the game's state. Computationally intensive due to the depth of the tree. No long-term policy.	Not yet tested.	Seems to be a good idea if the heuristic is good.

Genetic Algorithm	Totally avoids local minima. Can explore policies very quickly. Allows finding the general optimal policy.	Computationally expensive. Can never converge.	Not yet tested.	Seems to be a good idea, but we are not sure if there is really a good global policy on Splendor. This algorithm will help us to find the best heuristic value, which will be used for the Alpha-Beta algorithm.
Hybrid Training: PPO and Human collaborating vs opponents	The human learns from the machine, and the machine learns from the human. May allow PPO to learn faster with the help of the human, compared to a self-play PPO. Bring new policies to the PPO.	Need a human to play with AI (takes time). Still need a heuristic value or a value network, at least to return the short-term impact of the human decision. Need an expert human.	Not yet tested.	We will add explainability to this part to improve the understanding of the AI's behavior.

6 Ressources

6.1 Références

6.1.1 Splendor

- Rulebook
- Splendor sur Steam
- Splendor sur Board Game Arena
- Splendee
- Ephod

6.1.2 Technologies employées

- Angular
- Flask
- Open AI Gym

6.1.3 Intelligence Artificielle

- Rinascimento: Optimising Statistical Forward Planning Agents for Playing Splendor
- Wikipedia Montecarlo Tree Search

7 Annexe

7.1 Diagrammes complémentaires

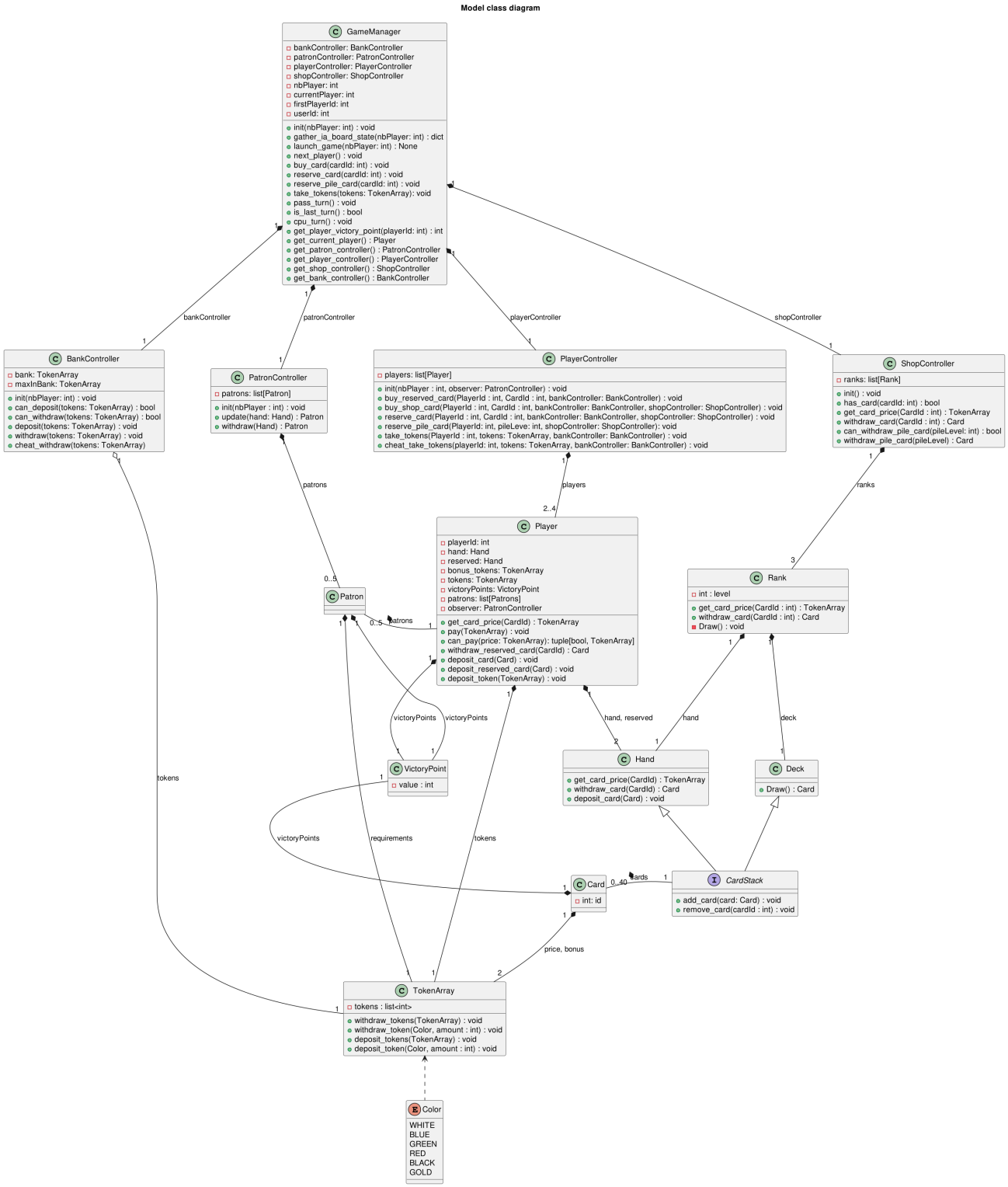

Figure 11 : Diagramme de classe du modèle

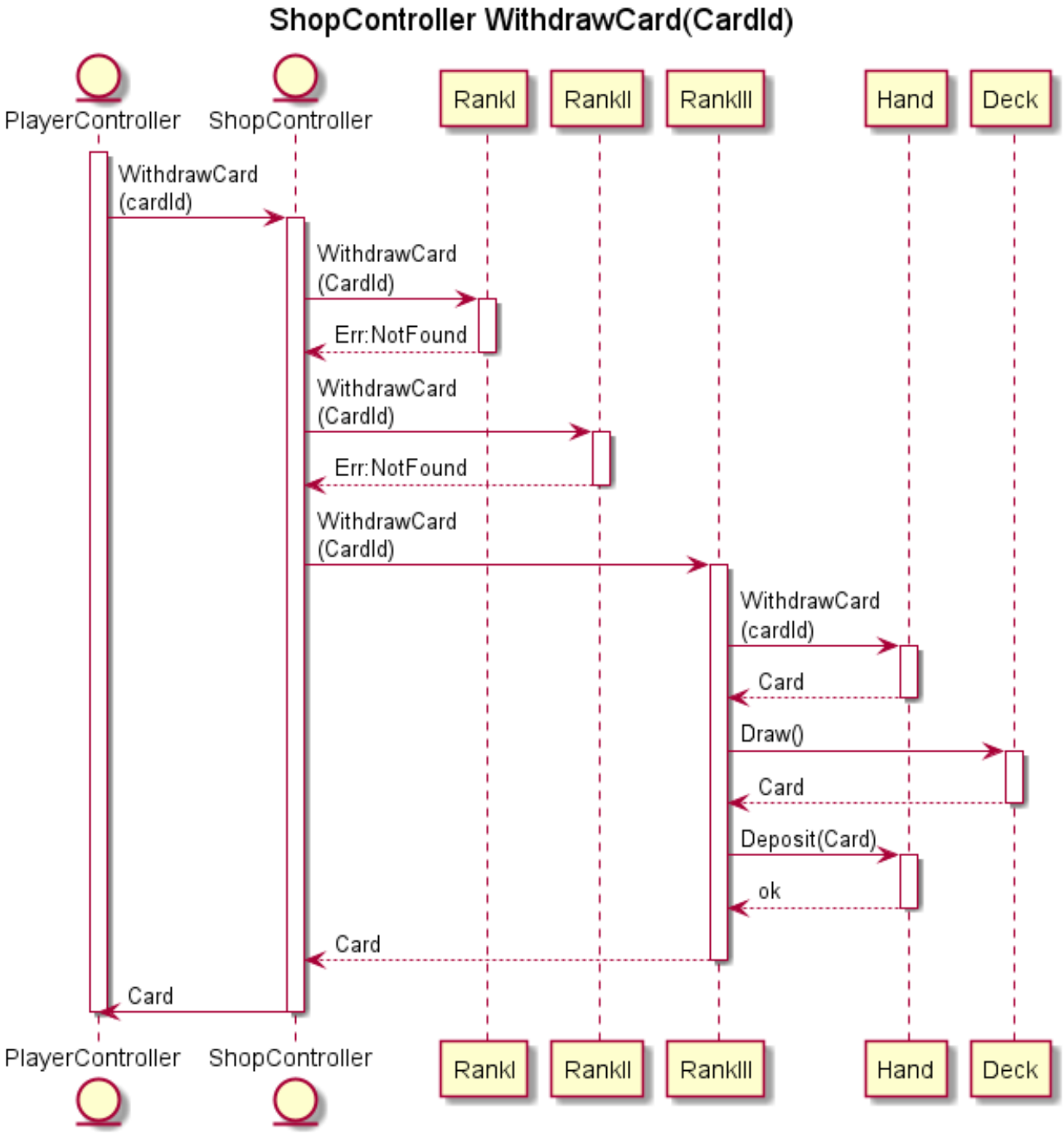

Figure 12 : Diagramme de séquence de l'action WithdrawCard