UFR de mathématique et d'informatique
Université de Strasbourg

Projet de Master

Création de joueurs automatiques et nouvelles règles pour jeux de plateau par apprentissage par renforcement

---

Mémoire

Étudiants :
Charlotte Kruzic
Daniil Kudriashov
Zoé Marquis
Ekaterina Zaitceva

Septembre 2024 - Janvier 2025

Table des matières

1 Introduction

1.1 Présentation du projet

L'objectif de ce projet était de concevoir des agents intelligents capables de jouer de manière autonome à des jeux de plateau en utilisant les techniques d'apprentissage par renforcement (*Reinforcement Learning*, RL). Le projet visait également à enrichir les jeux en introduisant de nouvelles règles, afin d'optimiser les comportements des agents et d'améliorer l'équilibre des jeux.

Nous avons initialement décidé d'utiliser le jeu de plateau Labyrinthe en raison de sa complexité moyenne. Ce jeu offre un équilibre intéressant entre simplicité de compréhension, richesse des mécanismes stratégiques, et flexibilité des règles, ce qui permettait d'explorer différentes stratégies et variantes.

Cependant, au cours du projet, nous avons rencontré de nombreuses difficultés lors de la modélisation de l'environnement, notamment en raison de la complexité inhérente au jeu.

Face à ce défi, nous avons décidé de nous réorienter vers le **Ludo**, une variante du jeu des **Petits Chevaux**. Ce choix s'est avéré plus adapté à notre cadre de travail, offrant un environnement plus simple à modéliser tout en conservant un potentiel d'expérimentation intéressant pour le développement des agents RL. Cette réorientation nous a permis de progresser efficacement tout en maintenant un équilibre entre simplicité et richesse stratégique.

Le projet met en œuvre l'apprentissage par renforcement pour créer des agents capables de jouer aux Petits Chevaux de manière autonome. Nous avons développé plusieurs types agents, avec des comportements et des stratégies de jeu variés. Nous avons aussi développé plusieurs variantes des règles du jeu. Pour évaluer les performances des agents et mieux comprendre leurs comportements, nous avons réalisé des simulations massives, et analysé les données récoltées. Enfin, en complément, une interface graphique a été développée, permettant aux utilisateurs de jouer directement contre ces agents.

1.2 Fondements et cadre théorique du RL

1.2.1 Reinforcement Learning

L'apprentissage par renforcement est une branche de l'intelligence artificielle qui permet à un agent de développer une stratégie en interagissant avec un environnement. Plutôt que de s'appuyer sur des exemples préexistants comme dans l'apprentissage supervisé, le RL repose sur un processus d'exploration où l'agent reçoit des retours sous forme de récompenses. Ces récompenses guident l'agent pour identifier une politique optimale, c'est-à-dire un ensemble de règles définissant les meilleures actions à entreprendre dans chaque situation afin de maximiser son succès à long terme.

1.2.2 Concepts clés

Le RL repose sur plusieurs notions fondamentales :

- **Agent** : Entité décisionnaire qui interagit avec l'environnement en prenant des actions.
- **Environnement** : Monde simulé ou réel dans lequel l'agent évolue. Il répond aux actions de l'agent en modifiant son état, en fournissant des observations, et en renvoyant une récompense ou une pénalité.
- **Espaces d'actions** : Ensemble des actions que l'agent peut entreprendre à chaque étape.
- **Espaces d'observations** : Représentation des informations que l'agent perçoit sur l'état actuel de l'environnement.
- **Récompense** : Signal numérique fourni par l'environnement après chaque action, indiquant à l'agent dans quelle mesure cette action est bénéfique.

Ces interactions sont modélisées comme une boucle continue :

1. L'agent observe l'état actuel de l'environnement.
2. Il choisit une action à partir de cet état.
3. L'environnement évolue en conséquence, et fournit une nouvelle observation ainsi qu'une récompense.

1.2.3 Caractéristiques distinctives du RL

Contrairement à l'apprentissage supervisé, où des paires d'entrée-sortie sont disponibles pour guider le modèle, le RL fonctionne sans données étiquetées. Parmi ses caractéristiques essentielles, on peut citer :

- Source de supervision : Le RL dépend de signaux de récompense qui ne sont pas directement liés à une action unique, mais à une séquence d'actions et à leurs conséquences.
- Exploration vs exploitation : L'agent doit constamment équilibrer deux stratégies :
  - Exploitation : Utiliser les connaissances actuelles pour maximiser la récompense immédiate.
  - Exploration : Tester de nouvelles actions pour découvrir potentiellement de meilleures stratégies à long terme.
- Séquence temporelle : Dans le RL, les décisions prises à un instant donné influencent les états futurs, rendant les interactions dynamiques et corrélées.

1.3 Contexte existant

1.3.1 Revue de la littérature

L'apprentissage par renforcement a été largement étudié dans le contexte des jeux de plateau, avec des travaux pionniers et des avancées récentes qui ont démontré son efficacité dans des environnements complexes. Voici quelques études clés :

- **Samuel (1959) - Jeu de Dames** [1] : Arthur Samuel a développé l'un des premiers programmes d'apprentissage automatique pour jouer au Jeu de Dames. Son travail a introduit des concepts fondamentaux comme l'apprentissage supervisé et l'apprentissage par renforcement. Ce travail a jeté les bases de l'apprentissage automatique et a montré que les machines pouvaient apprendre à jouer à des jeux de manière autonome.
- **Silver et al. (2016) - AlphaGo** [2] : L'équipe de chercheurs de DeepMind ont développé AlphaGo, un programme utilisant des techniques de RL pour jouer au jeu de Go. AlphaGo a battu plusieurs champions humains, démontrant l'efficacité du RL dans des environnements extrêmement complexes. AlphaGo a marqué une avancée significative dans le domaine de l'IA, montrant que les agents combinants RL, réseaux de neurones et recherche sur arbre pouvaient surpasser les performances humaines dans des jeux de stratégie complexes.
- **Mnih et al. (2015) - Human-Level Control Through Deep Reinforcement Learning** [3] : Volodymyr Mnih et al. ont développé un algorithme de deep reinforcement learning (DRL) qui a permis à un agent de jouer à des jeux vidéos de l'entreprise Atari à un niveau comparable à celui des humains. Ce travail a introduit le Deep Q-Network (DQN), qui utilise des réseaux de neurones profonds pour estimer les valeurs des actions (appelées les valeurs Q, stockées dans la "table Q"). Cette étude a démontré que les techniques de DRL pouvaient être appliquées avec succès à des jeux complexes et variés, ouvrant la voie à de nombreuses applications futures.

1.3.2 Technologies et méthodologies actuelles

Les algorithmes couramment utilisés pour le RL incluent le Q-learning, le Deep Q-Network (DQN), et le Proximal Policy Optimization (PPO). Ces algorithmes permettent aux agents d'apprendre des politiques optimales en explorant différentes actions et en ajustant leurs comportements en fonction des récompenses reçues.

- **Q-Learning** [4] : C'est un algorithme simple qui apprend à choisir la meilleure action dans chaque situation en construisant une table (la "table Q") où sont stockées les valeurs des actions. Il est idéal pour les jeux avec peu d'états, comme certains jeux de plateau simples.
- **Deep Q-Network (DQN)** [3] : DQN est une extension de Q-learning qui remplace la table Q par un réseau de neurones. Cela permet de gérer des espaces d'état plus complexes. Il est utilisé pour des jeux de plateau plus complexes où l'espace d'état est trop grand pour être géré par une table Q.
- **Proximal Policy Optimization (PPO)** [5] : PPO est un algorithme de RL qui optimise la politique de l'agent en utilisant des mises à jour de politique proches de la politique actuelle. Il est connu pour sa stabilité et son efficacité, et est utilisé pour des jeux de plateau complexes où une optimisation stable de la politique est nécessaire.

1.3.3 Problématiques et défis

Complexité de la modélisation des environnements de jeux

Les jeux de plateau sont souvent utilisés comme bancs d'essai pour tester les algorithmes de RL, comme dans [2], en raison de leurs caractéristiques distinctives :

- **Complexité modérée à élevée :** Les jeux comme le Labyrinthe impliquent des mécanismes stratégiques nécessitant des choix tactiques et une anticipation des actions adverses.
- **Environnement structuré :** Ces jeux offrent un cadre délimité et des règles fixes, ce qui facilite la modélisation des états et des actions.
- **Variabilité :** La possibilité d'ajout ou de modification de règles permet de diversifier les scénarios, augmentant ainsi la richesse des expérimentations.

Équilibre entre simplicité et richesse des règles

Ce dilemme a été exploré par Sun et Spangler [6], qui ont utilisé l'apprentissage par renforcement pour affiner les règles du jeu de plateau Organism en cours de développement. Leur travail illustre les défis liés à l'équilibre entre les deux aspects suivants :

- **La simplicité des règles :** favorise l'accessibilité, mais des règles trop simples peuvent restreindre les possibilités d'apprentissage et limiter l'exploration de stratégies diversifiées par les agents.
- **La richesse stratégique :** est nécessaire pour maintenir l'intérêt stratégique des joueurs humains sur le long terme, mais des environnements trop complexes nécessitent une puissance de calcul importante et allongent les temps de convergence des algorithmes de RL.

Équilibre entre exploration et exploitation

Le défi de trouver un équilibre entre l'exploration et l'exploitation repose sur les décisions que l'agent doit prendre, un des dilemmes centraux dans l'apprentissage par renforcement. Comme l'expliquent Wang et al. (2019) [7], il s'agit de trouver un équilibre entre **l'exploration de territoires inconnus** pour acquérir de nouvelles informations et **l'exploitation des connaissances existantes** afin de maximiser les récompenses immédiates. Ce choix complexe doit également intégrer les coûts inhérents à l'exploration, notamment en termes de ressources et de temps.

1.3.4 Projets similaires

Pour développer notre projet, nous nous sommes appuyés sur divers travaux existants. Ils nous ont offert des méthodologies et des idées précieuses qui ont orienté notre approche. Voici un aperçu des projets similaires ayant inspiré notre travail :

- YARAL(Yet Another Reinforcement Learning Approach to Ludo) de NDurocher [8] : Ce projet est écrit en Python et implémente des agents RL à l'aide de algorithme Q-Learning.

- Blokus-AI de roger-creus [9] : Ce projet, dédié au jeu de Blokus, est également développé en Python et utilise un environnement Gymnasium, où les agents sont entraînés avec PPO.

2 Objectifs du projet

2.1 Notre compréhension du sujet et des objectifs du projet

Ce projet vise à développer différents agents autonomes pour des jeux de plateau en s'appuyant sur des techniques d'apprentissage par renforcement. L'objectif principal est double : d'une part, entraîner ces agents à adopter des stratégies optimales adaptées aux règles du jeu, et d'autre part, étudier les interactions entre différentes règles et différents types d'agents afin d'analyser leur influence sur la jouabilité et l'équilibre global du jeu.

1. Modélisation du jeu
   - Représenter le jeu de manière formelle, avec ses états, actions et espaces d'observation.
   - Intégrer dès le départ la possibilité d'ajouter des règles ou variantes futures, en prévoyant une architecture flexible.
2. Apprentissage par renforcement
   - Utiliser des techniques de RL pour entraîner des agents autonomes capables de jouer efficacement au jeu choisi.
3. Comportements et personnalisation des agents
   - Développer différents types d'agents, chacun ayant des stratégies distinctes.
   - Étudier leurs comportements face aux variantes des règles testées.
4. Ajout de nouvelles règles et variantes
   - Analyser comment ces variantes influencent les comportements des agents.
   - Tester l'impact de différentes règles sur la jouabilité et l'équilibre du jeu.
5. Analyse statistique des parties simulées
   - Collecter des statistiques détaillées sur les parties jouées par les agents pour évaluer leurs performances.
   - Définir des critères pour comparer les règles ou les variantes afin de déterminer ce qui améliore le *gameplay*.

2.2 Objectifs pédagogiques

Voici les principaux objectifs pédagogiques que nous avons identifiés et poursuivis :

1. Découverte et maîtrise de l'apprentissage par renforcement
   - L'un des principaux défis était d'acquérir une compréhension approfondie de l'apprentissage par renforcement, un domaine très peu exploré dans notre cursus. Bien qu'un cours théorique ait été proposé en Master 1, il n'y a jamais eu de Travaux Pratiques ni d'études détaillées des algorithmes. Nos connaissances initiales se limitaient donc à des concepts très généraux.
   - Nous avons exploré plusieurs algorithmes, tels que :
     - Proximal Policy Optimization
     - Q-Learning
     - Deep Q-Learning (DQN)

     Cela nous a permis de mieux appréhender leurs spécificités, leurs forces et leurs limites dans le contexte des jeux de plateau.
   - Nous avons mis en place et configuré les outils nécessaires à leur utilisation, tout en apprenant à interpréter les résultats fournis par ces modèles.
2. Gestion et analyse des données

- Une partie essentielle du projet consistait à collecter, organiser et analyser les données générées lors des simulations de parties entre agents.
- Mise en place d'une base de données
  - Cette étape nous a appris à structurer une base de données pour stocker les informations collectées lors des parties simulées.
  - Nous avons défini des indicateurs clés pour suivre les performances des agents (comme les taux de victoire, le nombre d'actions prises, ou encore les types de stratégies employées).
- Analyse des données
  - Les données collectées ont été utilisées pour visualiser les performances des agents et comparer l'impact des différentes variantes de règles.
  - Cette partie nous a permis de renforcer nos compétences en traitement et en visualisation des données.

3. Collaboration et travail en équipe
   Le projet nous a également permis de développer nos compétences de collaboration et de gestion d'équipe :
   - Travail régulier en groupe :
     - Nous avons appris à partager les tâches efficacement et à coordonner nos efforts pour avancer dans les différentes étapes du projet.
     - La régularité des réunions et la communication entre les membres ont été essentielles pour surmonter les difficultés techniques.
   - Documentation des avancées :
     - Nous avons produit des documents réguliers pour permettre au professeur encadrant de suivre nos progrès et nos décisions. Ces documents se sont également révélés essentiels pour notre propre organisation. Ils nous ont aidés à :
       - Suivre nos avancées de manière structurée.
       - Comprendre sur quoi chacun a travaillé, en assurant une meilleure coordination au sein de l'équipe.
       - Retrouver facilement des informations ou des choix réalisés plus tôt dans le projet.
       - Clarifier certains aspects complexes du code ou des implémentations, facilitant ainsi les phases ultérieures de développement et d'analyse.
     - Cela a renforcé nos compétences en rédaction technique et en synthèse, des aptitudes précieuses pour structurer et communiquer efficacement nos idées dans un cadre technique.

2.3 Objectifs techniques

1. Recherche et évaluation des outils nécessaires
   - Identifier et évaluer les outils les mieux adaptés au projet en fonction des besoins techniques et des objectifs pédagogiques.
   - Effectuer des recherches approfondies sur les frameworks d'apprentissage par renforcement, les bibliothèques pour la création d'environnements, et les outils de visualisation.
2. Apprentissage rapide et prise en main des outils
   - Maîtriser rapidement les bases des outils sélectionnés pour pouvoir les utiliser de manière efficace, malgré une expérience limitée en apprentissage par renforcement.
3. Développement d'un environnement de jeu
   - Créer un environnement personnalisé pour les jeux de plateau en utilisant des outils adaptés à l'apprentissage par renforcement et la simulation de comportements d'agents.
   - Adapter et modifier les programmes standards pour répondre aux exigences spécifiques des environnements simulés, tout en respectant les normes établies pour les jeux de plateau. Cela

inclut la refactorisation de code existant et la révision de la structure des boucles classiques de jeu afin qu'elles s'alignent avec les frameworks d'apprentissage par renforcement.
   - Permettre la variation des règles du jeu pour tester différentes configurations.
   - Concevoir des agents aux comportements différents, afin d'explorer diverses stratégies et interactions avec les règles modifiées.
4. Utilisation d'algorithmes de RL pour les agents
   - Utiliser des algorithmes d'apprentissage par renforcement adaptés pour créer des agents capables de jouer de manière stratégique.
5. Simulation et analyse des résultats
   - Réaliser des simulations massives pour analyser le comportement des agents et optimiser les stratégies utilisées.
   - Évaluer les résultats de ces simulations pour mieux comprendre l'impact des différentes stratégies sur les agents et le jeu.
6. Interface utilisateur pour l'interaction avec les agents
   - Développer une interface permettant aux utilisateurs de jouer contre les agents.

3 Organisation du travail

3.1 Méthodologie

3.1.1 Feuille de route initiale

Au début du projet, les objectifs que nous nous étions fixés étaient ambitieux et diversifiés, couvrant différents aspects du développement et de l'analyse d'un environnement d'apprentissage par renforcement. Voici les principaux objectifs que nous avions définis :

1. Sélectionner le jeu : Identifier un jeu de plateau adapté au cadre de l'apprentissage par renforcement.
2. Sélectionner les outils : Choisir les bibliothèques et frameworks les plus adaptés pour répondre aux besoins du projet.
3. Rechercher ou développer une implémentation : Trouver une implémentation existante du jeu ou en créer une.
4. Adapter l'implémentation à l'environnement : Modifier le jeu pour qu'il soit compatible avec les exigences d'un environnement standardisé, tel que Gymnasium, présenté dans la section 4.
5. Entraîner un agent : Concevoir et entraîner un agent capable de jouer au jeu de manière autonome.
6. Créer des variations de règles : Ajouter des règles alternatives pour enrichir et diversifier les scénarios de jeu.
7. Développer plusieurs agents : Expérimenter différents types d'agents et analyser leurs comportements respectifs.
8. Jouer via une interface utilisateur : Permettre des parties interactives entre agents et humains grâce à une interface conviviale.
9. Mettre en place des simulations massives : Effectuer des simulations à grande échelle et collecter les résultats dans une base de données pour une analyse ultérieure.
10. Analyser les performances : Effectuer des statistiques pour évaluer les performances des agents et des configurations de jeu.
11. Intégrer le multi-agent : Étudier les interactions et la dynamique entre plusieurs agents jouant simultanément.

3.1.2 Défi majeur rencontré : le choix du jeu et ses conséquences

Nous avons initialement opté pour le jeu Labyrinthe. Ce choix semblait prometteur en raison de sa complexité modérée, de son potentiel pour inclure des règles variées, et de l'interaction stratégique qu'il offre entre

les joueurs. Toutefois, comme nous détaillerons dans la section 7, ce choix a soulevé des défis techniques et conceptuels importants.

3.1.3 Repartir de zéro au milieu du semestre

En raison des limites imprévues du jeu Labyrinthe et des défis techniques rencontrés, nous avons dû repartir de zéro à mi-parcours du projet. Cette décision, bien que difficile, a également été une opportunité d'appliquer les connaissances acquises et de capitaliser sur l'expérience déjà développée en apprentissage par renforcement.

Avec une meilleure compréhension des outils comme Gymnasium et Stable-Baselines, ainsi que des algorithmes sous-jacents, nous avons pu reconstruire un environnement de jeu de manière beaucoup plus rapide et efficace. Ces acquis techniques et conceptuels nous ont permis de surmonter les contraintes de temps — il ne restait alors que deux mois avant la fin du projet — tout en optimisant les performances d'apprentissage des agents.

3.1.4 Révision de la feuille de route

Après avoir réévalué la faisabilité de nos objectifs initiaux, nous avons ajusté nos priorités :

1. Créer un agent capable d'apprendre à jouer correctement à une version de base du jeu, en réutilisant les outils que nous avions déjà découverts et appris à utiliser pour le jeu Labyrinthe.
2. Créer des variantes simples du jeu pour tester différents comportements, tout en adaptant notre propre implémentation afin de pouvoir faire varier les agents et les règles du jeu.
3. Entraîner plusieurs agents différents et analyser leurs performances.
4. Mettre en place des simulations massives pour tester de nombreuses configurations et collecter des données dans une base de données pour une analyse approfondie.
5. Analyser l'impact des règles et agents : Effectuer des statistiques limitées pour comprendre quelles combinaisons fonctionnent bien ensemble.
6. Développer une interface permettant des interactions utilisateur simples.

3.2 Répartition des tâches

Au début du projet, de nombreuses tâches étaient communes à l'ensemble de l'équipe. Cela incluait le choix du jeu, la sélection des outils, et de nombreuses discussions sur les approches possibles. Cette phase initiale a été marquée par une collaboration intense, et bien que les tâches aient été partiellement réparties à la suite de ces échanges, il n'a pas été possible de définir clairement quatre rôles distincts, ce qui à nécessiter une forte coordination entre les membres.

Une fois le choix du jeu Labyrinthe défini, nous avons pu structurer les tâches en deux groupes : Zoé et Ekaterina ont pris en charge l'amélioration de l'interface, avec Zoé également responsable du nettoyage du code du jeu existant et de la mise en place de la base de données, tandis que Daniil et Charlotte se sont concentrés sur la création de l'environnement, son intégration avec le jeu existant, et l'entraînement des agents. L'ensemble de l'équipe a ensuite collaboré pour essayer d'instaurer un apprentissage fonctionnel.

Cependant, lors du revirement vers le jeu Petits Chevaux, fin novembre, une nouvelle organisation s'est mise en place. Zoé a, en l'espace d'une semaine, jeté les bases de ce nouveau projet en mettant en place :

- une première version du jeu simplifiée,
- une connexion fonctionnelle à l'environnement Gymnasium,
- une interaction permettant de simuler des parties dans le terminal

À partir de ce socle, chaque membre de l'équipe a pu se reconnecter et apporter sa contribution :

- Charlotte a modifié le travail initial sur la base de données du Labyrinthe, afin de l'adapter au nouveau jeu, a mis en place des simulations massives, et a réalisé les analyses sur l'entrainement des agents.
- Ekaterina a remplacé l'interaction dans le terminal par une interface graphique qu'elle a développée pour rendre le jeu plus intuitif et accessible.
- Daniil a réalisé les analyses statistiques finales, permettant de comprendre comment les différents agents et variantes de règles interagissent.
- Zoé a poursuivi l'implémentation des règles du jeu en veillant à garantir sa robustesse, a développé un véritable environnement compatible avec l'apprentissage par renforcement, a créé plusieurs variantes du jeu, et a mis en place des tests automatiques pour s'assurer de la fiabilité et de la stabilité de l'ensemble.

Chaque membre de l'équipe a rédigé sa sous-partie dans la suite de ce rapport, détaillant les tâches qu'il considère comme importantes dans sa contribution tout au long du semestre.

Zoé Marquis

Études préliminaires et reformulation du sujet (en collaboration avec le groupe)

- Analyse et sélection du jeu à utiliser pour le projet, ainsi que l'exploration de différents comportements d'agents et règles susceptibles de varier.
- Étude et sélection des outils nécessaires : Gym/Gymnasium, Stable Baselines, bases de données, etc.

Mise en place de l'environnement de travail

- Création d'outils et de canaux de communication pour l'équipe (Discord pour faciliter les échanges et les réunions, Notion pour le partage de ressources et la rédaction de documentation, Overleaf pour la rédaction du mémoire, etc.)

Exploration du premier jeu : Labyrinthe

- Exploration initiale :
  - Jouer au jeu en conditions réelles pour comprendre les règles et identifier des stratégies.
  - Étude du code Python existant pour le jeu.
- Améliorations techniques :
  - Restructuration du code Python pour corriger de nombreux défauts et garantir sa conformité avec les règles officielles.
  - Ajout de fonctionnalités : support pour 4 joueurs, design initial pour l'UI (*User Interface*) et l'UX (*User Experience*).
- Responsabilités et réalisations :
  - Modification du code Python Labyrinthe des IUT [10] pour le rendre compatible avec un environnement Gymnasium.
  - Mise en place d'une base de données PostgreSQL et des fonctions fondamentales, à l'aide de SQLAlchemy, pour la collecte initiale de données à des fins statistiques.
  - Collaboration avec Charlotte sur la mise en place de l'environnement et l'entraînement des agents.
  - Rédaction d'un rapport intermédiaire pour formaliser les idées, documenter les progrès, et exposer les limites rencontrées.

Transition vers un nouveau jeu : Ludo / Petits Chevaux

- Exploration initiale (en une semaine, fin novembre) :
  - Recherche personnelle et étude approfondie de plusieurs dépôts de code liés au jeu :
  - Recherche personnelle et étude approfondie de plusieurs dépôts de code liés au jeu, notamment [9], [8], [11], [12], [13], [14], [15], [16].
  - Décision, prise de manière autonome, de ne pas utiliser de code existant pour garantir une meilleure adaptation aux besoins du projet.

    - Développement d'une première version simplifiée du jeu, incluant les mécanismes fondamentaux.
    - Mise en œuvre avec Gym/Gymnasium et Stable Baselines PPO, validée par une confirmation que l'agent apprend à jouer.
- Partage des avancées :
    - Présentation et explication détaillée aux autres membres de l'équipe des premières réussites, pour les mettre à jour et intégrer leurs retours.

Perfectionnement du jeu : Ludo / Petits Chevaux

- Développement du jeu et de l'environnement Gymnasium :
    - Implémentation de toutes les variantes des règles :
        - Gestion des règles variables (exemple : possibilité de tuer un pion ou non).
        - Modification des espaces d'action et d'observation en fonction des paramètres choisis.
    - Création de tests avec Pytest pour valider la logique et la robustesse des différentes versions du jeu.
- Organisation du projet :
    - Développement d'un environnement structuré permettant :
        - À Ekaterina d'intégrer une interface utilisateur (fichier `render.py` lié à la fonction `render()` de Gymnasium).
        - À Daniil d'implémenter différents agents en ajustant les récompenses.
        - À Charlotte d'enregistrer et analyser les statistiques des parties.

Collaboration et assistance régulières

- Participation active à la coordination et à l'assistance des membres de l'équipe dans leurs tâches respectives pendant les dernières semaines du projet, notamment par l'organisation de réunions de groupe et de sessions de travail en binôme pour les aider à avancer ou finaliser leurs tâches, ainsi qu'à clarifier les aspects techniques ou méthodologiques.

Préparation du rendu final

- Responsable de la rédaction des sections suivantes du rapport : Introduction (Présentation du projet 1.1, Fondements et cadre théorique du RL 1.2), Objectifs (2), Organisation du travail (3, à l'exception des sous-parties rédigées par chaque membre de l'équipe à propos de sont travail personnel), et Présentation des outils développés (5, sous-sections 5.1 à 5.5), à l'exception des parties concernant les différents agents.
- Rédaction d'une partie de la documentation technique via le fichier `README.md`, détaillant la structure du dossier, les différentes règles du jeu et les réalisations effectuées dans le cadre du projet.
- Contribution active à l'amélioration et à la réécriture de sections du rapport final, afin d'assurer la clarté et la qualité rédactionnelle du projet.
- La mise en place d'un environnement virtuel (venv) pour permettre les tests des diverses fonctionnalités du projet.

Charlotte Kruzic

Contribution aux études préliminaires et exploration :

- Analyse des jeux sélectionnés pour le projet, avec recherche de variantes et de types d'agents possibles : Labyrinthe et Petits Chevaux.
- Recherche de versions existantes et de codes sources pour le jeu Labyrinthe, notamment Labyrinthe des IUT, avec la rédaction d'une revue de code pour documenter l'analyse et identifier les optimisations nécessaires.

Contribution sur le jeu du Labyrinthe :

- Mise en place de l'environnement initial Gymnasium, accompagné d'un document de présentation.
- Mise en place initiale de notebooks d'entraînement des agents.
- Mise en place de TensorBoard et de callbacks pour observer les statistiques sur l'entraînement des agents et suivre leur évolution en temps réel.
- Intégration des agents dans le jeu, permettant de jouer contre eux.
- Participation aux modifications de la logique du jeu pour corriger ou adapter les règles.
- Recherche approfondie sur l'optimisation de l'entraînement des agents dans des environnements complexes : adaptation des récompenses, utilisation des masques d'actions, modification des espaces d'action et d'observation...
- Rédaction d'un rapport d'expérimentation pour relater les tests d'améliorations effectués n'ayant pas abouti, et les raisons des blocages.

Contributions sur le jeu Petits Chevaux :

- Participation au développement des actions : tuer un adversaire, rebondir contre un adversaire (empêcher de doubler et d'être sur la même case), et réalisation de fichiers de tests associés.
- Mise en place d'un système d'automatisation des parties entre agents, avec enregistrement des données pertinentes sur les parties et les joueurs dans notre base de données.
- Uniformisation et simplification des accès aux agents et aux appels à l'environnement de jeu, avec la centralisation des configurations, des règles de jeu, du stockage des agents, des fichiers d'entraînement...
- Modification et adaptation de la base de données PostgreSQL, initialement créée par Zoé, pour l'adapter aux besoins spécifiques du projet et du jeu des Petits Chevaux : ajout et modification régulière de tables et de colonnes pour répondre aux besoins des autres membres du projet.
- Rédaction d'une documentation sur la configuration et l'utilisation de la base de données, et d'une présentation des tables et attributs.
- Création d'un fichier permettant d'exporter les données de la base de données au format CSV, afin de faciliter leur accès pour les analyses.
- Analyse de l'entraînement des agents : création d'un notebook avec des fonctions et des visualisations adaptées, et analyse des résultats obtenus.

Rédaction et documentation :

- Rédaction de différentes documentations (citées précédemment) liées à mes tâches pour faciliter l'accès aux autres, et suivre ce qui a été réalisé.
- Rédaction des parties que j'ai réalisées dans le ReadMe et le rapport final.
- Participation à la relecture, à l'amélioration, et à la réécriture de plusieurs parties du rapport final.

Ekaterina Zaitceva

Études préliminaires et reformulation du sujet (en collaboration avec le groupe) :

- Étude et choix du jeu à utiliser pour le projet, accompagnés d'une exploration des différents comportements des agents et des règles pouvant être modifiés.
- Pratiquer le jeu en conditions réelles afin de comprendre les règles et d'identifier des stratégies.

Contribution sur le jeu du Labyrinthe :

- Participation à l'adaptation et à la modification de la logique du jeu afin de corriger ou ajuster les règles.
- Developpement de theme "ocean" pour l'UI/UX
- Amélioration de l'interface existante pour rendre le jeu plus fluide et compréhensible.
- Recherche sur l'optimisation de l'entraînement des agents dans des environnements complexes.

Contributions sur le jeu Petits Chevaux :

- Développement d'une interface utilisateur prenant en compte l'environnement structuré développé par Zoé.
- Mise en place d'un système permettant de lancer des parties entre humains et/ou agents, à partir de configurations choisies par l'utilisateur en ligne de commande (en s'appuyant sur le fichier `launcher.py` écrit par Daniil et Zoé).
- Enregistrement de GIFs et d'images illustrant le déroulement du jeu.

Préparation du rendu final :

- Responsable de la rédaction des sections suivantes du rapport : Introduction (Contexte existant 1.3), Présentation des outils développés (Interface 5.6).
- Participation à la relecture et à l'amélioration de différentes parties du rapport final.

Daniil Kudriashov

- Études préliminaires et exploration (en collaboration avec le groupe)
  - Étude comparative des différents jeux de plateau pour le projet
  - Analyse des outils d'apprentissage par renforcement disponibles
  - Participation à l'évaluation des frameworks et bibliothèques
- Développement sur le jeu Labyrinthe
  - Exploration initiale :
    - Participation aux séances d'analyse du jeu en équipe
    - Étude des mécaniques et stratégies du jeu
  - Contributions techniques :
    - Création et amélioration de l'environnement d'apprentissage
    - Mise en place du processus d'entraînement des agents
    - Développement et optimisation des agents d'apprentissage
- Développement sur le jeu Ludo / Petits Chevaux
  - Conception des agents :
    - Collaboration avec Zoé sur la structure de l'environnement
    - Développement d'un système flexible de récompenses
    - Création de différents profils d'agents (agressif, défensif, etc.)
    - Fine-tuning des paramètres de récompense pour chaque type d'agent
  - Analyse des performances :
    - Mise en place d'un système de matchups entre agents
    - Conception des protocoles de test pour différentes configurations
    - Collecte et analyse des statistiques de performance
- Analyse et évaluation
  - Évaluation des performances :
    - Analyse des taux de victoire et des stratégies
    - Étude de l'impact des différentes règles sur le gameplay
    - Mesure de l'équilibre du jeu dans diverses configurations
  - Statistiques de jeu :
    - Analyse de la durée moyenne des parties
    - Évaluation de la proximité des scores
    - Mesure de l'efficacité des différentes stratégies d'agents

4 Outils utilisés et analyse des choix techniques

Python : Langage principal de notre projet, choisi en raison de sa large adoption dans la communauté scientifique et de la richesse de ses bibliothèques pour l'apprentissage par renforcement et la science des données.

- Avantages : Bibliothèques bien établies (Gymnasium, Stable-Baselines3, PyTorch), simplicité d'utilisation, forte communauté de support.
- Inconvénients : Moins performant pour certains calculs intensifs par rapport à d'autres langages comme C++.

Gym / Gymnasium [17] : Bibliothèque utilisée pour créer et gérer l'environnement de jeu des agents RL.

- Avantages : Base standardisée, large adoption, intégration facile avec d'autres outils RL, mise à jour régulière avec Gymnasium.
- Inconvénients : Besoin d'adaptations pour des environnements plus complexes, limite pour des interactions multi-agents avancées.

Stable-Baselines3 [18] : Framework basé sur PyTorch utilisé pour implémenter les algorithmes RL.

- Avantages : Documentation claire, nombreux exemples pratiques, bonne intégration avec Gymnasium, facilité d'utilisation.
- Inconvénients : Ne propose pas d'options très avancées pour personnaliser les algorithmes.

Pygame : Utilisé pour la visualisation en temps réel du jeu.

- Avantages : Interface simple pour créer des visualisations interactives, bien adapté pour les jeux en 2D.
- Inconvénients : Moins puissant pour des visualisations complexes en 3D ou pour des jeux avec des besoins graphiques élevés.

Ray RLlib : Bien que cette bibliothèque offre une grande variété d'algorithmes d'apprentissage par renforcement, sa complexité l'a rendue inadaptée à notre projet, et nous avons donc décidé de ne pas l'utiliser.

- Avantages : Bonne évolutivité, gestion des environnements distribués.
- Inconvénients : Complexité d'utilisation et de configuration, nécessite du temps pour la prise en main.

PettingZoo : Bibliothèque initialement envisagé pour des scénarios multi-agents mais n'a pas été intégré en raison des contraintes de temps.

- Avantages : Spécialisé dans les environnements multi-agents, facile à utiliser pour des jeux impliquant plusieurs agents en interaction.s

TensorFlow : Bien que robuste, ce framework a été écarté en faveur de PyTorch, qui est plus simple pour le prototypage rapide.

- Avantages : Robuste, largement utilisé dans les projets d'IA à grande échelle.
- Inconvénients : Plus complexe à utiliser par rapport à PyTorch pour des applications spécifiques à RL.

Matplotlib : Cette bibliothèque a été utilisée pour générer des graphiques statistiques à partir des données collectées lors des simulations.

- Avantages : Très efficace pour créer des visualisations statiques et des graphiques analytiques.
- Inconvénients : Pas adaptée pour des interfaces graphiques nécessitant de l'interactivité.

PostgreSQL : Utilisé comme base de données relationnelle pour stocker et gérer les données du projet.

- Avantages : Système robuste et performant, adapté aux applications de taille moyenne à grande ; facilite la gestion de relations complexes grâce au support des clés étrangères et des jointures ; compatible avec SQLAlchemy pour interagir avec la base en Python.
- Inconvénients : Installation, configuration et gestion du serveur peuvent être complexes.

SQLAlchemy : Utilisé comme ORM (Object-Relational Mapping) pour interagir avec PostgreSQL, facilitant la gestion des schémas et des opérations CRUD.

- Avantages : Supporte les relations complexes (One-to-Many, Many-to-Many) ; simplifie l'ajout de tables et la modification des schémas ; compatible avec Pandas pour l'exportation vers des formats comme CSV.
- Inconvénients : Configuration initiale complexe pour des schémas avec dépendances croisées ; moins performant que les requêtes SQL directes sur de grandes bases.

Fichiers CSV : Choisis comme format d'export des données depuis PostgreSQL pour faciliter leur analyse.

- Avantages : Format simple et lisible, supporté par de nombreux outils ; léger, facile à transférer et stocker.
- Inconvénients : Absence de métadonnées ou contraintes, pouvant causer des erreurs d'interprétation ; peu adapté pour de grandes quantités de données.

Jupyter Notebooks : Utilisés pour l'analyse des données exportées de PostgreSQL (au format CSV).

- Avantages : Outil interactif idéal pour l'exploration, la visualisation et le traitement des données ; permet d'ajouter textes, graphiques et code dans un même document ; compatible avec Python et ses bibliothèques.
- Inconvénients : Peu adapté au suivi des versions dans les systèmes comme Git ; limites en mémoire et interactivité pour de grands volumes de données ; nécessite une configuration cohérente des dépendances.

Pandas : Utilisé dans les Jupyter Notebooks pour manipuler, analyser et visualiser les données.

- Avantages : Idéal pour traiter des données tabulaires ; chargement direct des fichiers CSV ; nombreuses fonctions intégrées pour le filtrage, le tri, les statistiques descriptives, etc.
- Inconvénients : Peut devenir lent ou consommer beaucoup de mémoire avec des datasets volumineux.

5 Présentation des outils développés

5.1 Jeu Petits Chevaux / Ludo

Le jeu **Petits Chevaux**, également connu sous le nom de Ludo dans ses variantes modernes, est un jeu de plateau traditionnel qui se joue généralement entre 2 et 4 joueurs. Chaque joueur contrôle un ensemble de 4 pions, souvent appelés chevaux, qu'il doit déplacer autour du plateau en fonction des résultats obtenus avec un dé.

L'objectif du jeu est de faire parcourir à ses chevaux un circuit complet pour les ramener dans la zone d'arrivée spécifique à chaque joueur. Le premier joueur à réussir à placer tous ses chevaux dans la zone d'arrivée remporte la partie. Les déplacements sont régis par des lancers de dé, et un pion ne peut être déplacé qu'après avoir obtenu un 6 au lancer, ce qui lui permet de quitter sa case de départ.

Un pion peut être capturé si un pion adverse atterrit sur la même case qui lui. Et lorsqu'un pion est capturé, il doit retourner à sa position de départ, ralentissant ainsi la progression du joueur. Ce mécanisme ajoute une dimension stratégique au jeu, car il faut choisir entre avancer rapidement ou bloquer les adversaires.

La variante **Ludo** introduit quelques ajustements aux règles classiques. Par exemple, les pions peuvent être protégés sur des cases spéciales, souvent marquées par des étoiles, qui agissent comme des zones de sécurité. De plus, le jeu peut inclure des mécanismes optionnels, comme des raccourcis permettant de réduire la distance à parcourir ou des règles supplémentaires pour les doubles au lancer de dé, qui offrent des tours bonus ou des avantages spécifiques. Ces modifications rendent la version plus stratégique et dynamique, tout en conservant l'accessibilité des règles de base.

Ces nuances rendent le jeu à la fois accessible pour des joueurs débutants et intéressant à modéliser pour des simulations d'apprentissage par renforcement, grâce à ses multiples règles modifiables, ses interactions complexes entre joueurs et son mélange de hasard et de stratégie.

5.2 Notre version du jeu

Règles de Base :

- Chaque joueur commence avec tous ses pions dans son écurie.
- Un 6 au dé est requis pour sortir un pion de l'écurie.
- Une fois sur le plateau, les pions avancent sur un chemin commun de 56 cases menant à un escalier de 6 cases, propre à chaque joueur, qui aboutit à une case objectif.
- Règles pour les déplacements dans le chemin commun :
  - Tuer un pion adverse : Un pion peut éliminer un pion adverse uniquement s'il tombe exactement sur la même case.
  - Rejoindre un pion allié : Un pion d'un joueur peut rejoindre un autre pion du même joueur uniquement si le résultat du lancer de dé correspond exactement à la distance qui les sépare.
  - Rester bloqué derrière un pion : Si la valeur du dé est supérieure au nombre de cases jusqu'au pion suivant sur le plateau (qu'il appartienne au même joueur ou à un adversaire), le pion avancera jusqu'à la case précédant l'obstacle. Les dépassements de pions sont interdits. Ces règles ne s'appliquent pas à l'escalier.
- Disposition des écuries selon le nombre de joueurs :
  - 2 joueurs : Les écuries sont placées à l'opposé l'une de l'autre sur le plateau. Ainsi, la case 1 du chemin pour un joueur correspond à la case 29 pour l'autre.
  - 3 ou 4 joueurs : Les écuries sont réparties de manière équidistante toutes les 14 cases. Une même case peut être perçue différemment selon le point de vue du joueur : par exemple, la case 1 pour un joueur sera la case 15, case 29 ou case 43 pour les autres joueurs, en fonction de leur position de départ.

  Cela garantit une répartition équilibrée des positions de départ sur le plateau.

Variations des règles :

- **Nombre de joueurs** :
  - Le jeu peut être joué à **2, 3 ou 4** joueurs.
- **Nombre de pions par joueur** :
  - Chaque joueur peut avoir **entre 2 et 6** petits chevaux en jeu.
- Conditions de **victoire** :
  - Victoire **rapide** : Le premier joueur à atteindre l'objectif avec un seul pion gagne.
  - Victoire **complète** : Tous les pions d'un joueur doivent atteindre l'objectif pour déclarer sa victoire.
- Règles pour atteindre le **pied de l'escalier** :

    - **Exactitude** nécessaire : Un pion doit atteindre exactement la case située au pied de l'escalier pour commencer à le gravir.
        - Si le lancé de dé dépasse la distance requise pour atteindre cette case, le pion peut avancer puis reculer, à condition que ce mouvement réduise la distance qui le sépare de la case au pied de l'escalier.
        - Si, en avançant puis reculant selon la valeur du dé, le pion finit par s'éloigner davantage de la case pied de l'escalier, alors il ne peut pas être déplacé.
    - Progression **simplifiée** : Si la valeur du dé dépasse le pied de l'escalier, le pion grimpe directement comme si l'escalier faisait partie du chemin.
- Ordre de progression sur l'**escalier** :
    - Ordre **simplifié** : Un pion avance de la distance indiquée par le dé. Cela lui permet de monter plusieurs marches de l'escalier en un seul lancer, et d'atteindre directement la case objectif si la valeur du dé le permet ou la dépasse.
    - Ordre **strict** (autorisé uniquement si l'**exactitude** est requise pour atteindre le **pied de l'escalier**) :
        - Chaque marche de l'escalier nécessite un lancer spécifique : 1 pour la première marche, 2 pour la deuxième, ..., et 6 pour atteindre la case objectif.
        - Une règle optionnelle peut être ajoutée pour permettre ou non au pion de **rejouer après chaque montée de marche**.
- **Rejouer en cas de lancer d'un 6** : Cette règle peut être activée ou désactivée.
- **Protection des pions** : Si deux pions d'un même joueur se trouvent sur la même case, ils deviennent invulnérables et ne peuvent pas être éliminés. Cette règle peut être activée ou désactivée.

Cela est illustré par un schéma récapitulatif en 10.

Agents

Dans notre projet, nous avons développé six types d'agents distincts, chacun avec sa propre stratégie et son système de récompenses :

- **Agent Balanced (Équilibré)** : Agent utilisant une stratégie équilibrée entre attaque et progression, servant de référence pour comparer les autres agents.
- **Agent Aggressive (Agressif)** : Agent privilégiant les actions offensives avec des récompenses maximales pour l'élimination des pions adverses.
- **Agent Rusher (Pressé)** : Agent focalisé sur la progression rapide vers l'objectif avec un système de récompenses favorisant les mouvements vers l'avant et l'atteinte du but.
- **Agent Defensive (Défensif)** : Agent priorisant la sécurité des pions avec des récompenses importantes pour l'atteinte des zones sûres et la protection mutuelle des pions.
- **Agent Spawner (Sortie rapide)** : Agent concentré sur la sortie rapide des pions de l'écurie avec des récompenses maximales pour les actions de sortie.
- **Agent Suboptimal (Sous-optimal)** : Agent simulant un joueur inexpérimenté avec des récompenses plus élevées pour les actions généralement moins efficaces.

Chaque agent dispose de sa propre table de récompenses, adaptée selon sa stratégie. Cette diversité permet d'explorer différentes approches du jeu et d'étudier leur efficacité relative dans différentes configurations de règles.

Les valeurs de récompense ont été calibrées pour encourager des comportements spécifiques tout en maintenant une cohérence globale dans l'apprentissage. Par exemple, les actions interdites sont systématiquement pénalisées pour tous les agents, afin de décourager les mouvements invalides.

5.3 Notre implémentation du jeu et définition de l'environnement

5.3.1 Description générale

Notre projet s'appuie sur une architecture modulaire, inspirée de la structure du dépôt GitHub blokus-ai [9], pour séparer clairement les interactions de l'apprentissage par renforcement des règles spécifiques du jeu. Cette approche facilite les modifications ou extensions futures, telles que la variation des règles ou l'intégration de différents agents.

Modules principaux :

- Logique du jeu (`game_logic.py`) :
  - Gestion des règles spécifiques et des interactions du jeu, notamment :
    - Validation des actions (par exemple, déplacement autorisé ou non).
    - Gestion des événements, comme l'élimination d'un pion adverse et son retour à la position de départ.
    - Calcul de l'état du jeu, y compris la détection des conditions de victoire.
- Environnement Gymnasium (`env.py`) : L'interface principale permet les interactions entre l'agent et le jeu, et l'environnement est configurable à l'aide de paramètres prédéfinis, incluant le nombre de joueurs, le nombre de pions par joueur, la variante des règles appliquées, ainsi que le type d'agent utilisé, le cas échéant. De plus, trois modes peuvent être sélectionnés :
  - Mode entraînement : destiné à l'entraînement des agents dans différentes configurations.
  - Mode jeu : permet de lancer une partie avec une interface graphique pour une expérience utilisateur interactive.
  - Mode statistique : permet de collecter des statistiques détaillées sur les performances des agents, ce qui influence les ajustements de stratégie et l'évaluation des comportements.

  L'environnement est structuré autour de deux espaces clés, qui établissent la communication entre l'agent et le jeu :
  - Espace des actions : définit les mouvements ou décisions possibles pour l'agent, permettant ainsi de contrôler l'évolution du jeu.
  - Espace des observations : contient les informations nécessaires pour évaluer l'état actuel du jeu et guider les choix de l'agent.

5.3.2 Espace des actions

L'espace des actions est défini comme un espace discret (`gym.spaces.Discrete`), où chaque action est représentée par un entier unique. La taille de cet espace varie en fonction des configurations du jeu. Dans un espace discret, chaque entier correspond à une opération bien définie dans le jeu, ce qui permet à l'agent de naviguer efficacement parmi les options disponibles.

L'espace des actions a été encodé de manière à tenir compte du nombre de chevaux (ou pions) par joueur. Chaque action possible est associée à un numéro unique, représentant un comportement spécifique du jeu. Par exemple, une action pourrait correspondre à sortir un pion de l'écurie, atteindre un objectif avec un pion, ou encore éliminer un pion adverse. L'action choisie par l'agent dépend du pion qu'il souhaite utiliser et de l'action qu'il veut lui faire accomplir. Ce système garantit que chaque action correspond à une opération valide et définie dans les règles du jeu.

Lors de l'appel à la fonction `step`, l'agent sélectionne une action qui sera ensuite décodée. Cette approche permet une gestion cohérente des actions tout en facilitant l'interaction avec l'environnement.

5.3.3 Espace des observations

L'espace d'observation est créé avec la classe `gym.spaces.Dict`, qui regroupe plusieurs informations nécessaires à l'évaluation de l'état du jeu, à la manière d'un dictionnaire. Cet espace permet à l'agent d'obtenir un aperçu complet de l'état du jeu à chaque étape, incluant :

- `my_ecurie` : Nombre de pions restant dans l'écurie.
- `my_chemin` : Position des pions du joueur actuel (valeurs positives) ainsi que ceux des adversaires (représentées par des valeurs négatives) sur le plateau.
- `my_escalier` : Progression des pions du joueur actuel dans la zone finale, l'escalier.
- `my_goal` : Nombre de pions du joueur actuel ayant atteint l'objectif.
- `dice_roll` : Résultat du lancer de dé.

Lorsqu'une observation est requise, généralement à chaque `step`, la méthode `_get_observation` de `env.py` est appelée. Elle compile ces données et les retourne à l'agent, offrant ainsi une vue d'ensemble détaillée de l'état du jeu avant et après une action.

5.3.4 Fonctions essentielles

Gymnasium impose la définition de plusieurs fonctions pour garantir la compatibilité de l'environnement avec les algorithmes de RL.

1. `reset()`
   - Cette fonction initialise ou réinitialise l'environnement au début d'un nouvel épisode.
   - Elle renvoie l'état initial, sous forme d'observation, comme point de départ pour l'agent.
2. `step(action)`
   - Prend une action en paramètre et effectue les opérations correspondantes dans l'environnement.
   - Renvoie :
     - `observation` : Une nouvelle observation de l'état du jeu après exécution de l'action.
     - `reward` : Une récompense numérique associée à l'action effectuée, reflétant son impact dans l'environnement.
     - `done` : Une valeur booléenne indiquant si l'épisode est terminé (`True`) ou non (`False`).
     - `info` : Un dictionnaire contenant des informations supplémentaires sur l'état ou les performances, utiles pour le débogage ou l'analyse. Ce dictionnaire a été utilisé pour extraire des données spécifiques, permettant de collecter des informations pour la base de données et les statistiques liées aux performances des agents.
3. `render()`
   - Permet de visualiser l'état actuel du jeu.

5.3.5 Gestion des actions interdites

Afin de gérer les actions interdites dans notre environnement, nous avons implémenté trois modes de traitement.

- En mode entraînement, si l'agent propose une action impossible, une récompense de -10 est retourné pour pénaliser cette erreur, guidant ainsi l'agent vers des actions permises et optimisant son apprentissage.
- En mode jeu, étant donné que l'agent peut commettre des erreurs, l'action demandée est alors transformée en une action valide selon un ordre prédéfini en fonction du type d'agent.
- En mode statistique, nous collectons à la fois l'action demandée et l'action réellement exécutée, afin d'analyser les erreurs de l'agent.

5.4 Stable Baselines, PPO et notre implémentation

5.4.1 Intégration avec l'environnement Gymnasium

En intégrant notre implémentation du jeu avec Gymnasium, créant ainsi un environnement standardisé, nous avons pu utiliser l'algorithme PPO directement disponible dans Stable Baselines, simplifiant ainsi l'implémentation des mécanismes d'entraînement et de mise à jour des politiques.

5.4.2 Théorie : Fonctionnement de PPO

L'algorithme PPO optimise une politique en utilisant deux principes clés :

- **Clipping de la politique** : limite les changements trop brusques dans la politique en utilisant un ratio entre l'ancien et le nouveau modèle.
- **Avantage estimé** : combine récompenses futures et erreur pour ajuster la politique.

Ces mécanismes permettent à PPO de gérer exploration et exploitation tout en maintenant la stabilité du processus d'apprentissage.

5.4.3 Implémentation : Adaptation de PPO à notre environnement

Nous avons adapté l'algorithme PPO en utilisant une politique multi-input, conforme à la structure définie pour l'espace d'observation. Cela permet de gérer efficacement les multiples sources d'information nécessaires à la prise de décision. Une autre amélioration clé est l'introduction d'un masque d'actions valides dans la classe `MaskedPPO`. Cette approche empêche l'agent de sélectionner des actions invalides en modifiant les logits correspondants. Cela a significativement amélioré la convergence et la stabilité de l'apprentissage.

1. **Calcul du masque** : Génère une liste des actions valides en fonction de l'état actuel du jeu et construit un tableau de masque adapté.
2. **Modification des logits** : Les logits associés aux actions invalides sont pénalisés, ce qui réduit considérablement leur probabilité d'être sélectionnées.
3. **Choix de l'action** : Les logits ajustés sont normalisés pour former une distribution de probabilités, à partir de laquelle une action est choisie aléatoirement en respectant les probabilités résultantes.

5.4.4 Ajustement des hyperparamètres

Le paramètre `total_timesteps` détermine la durée de l'entraînement et influence directement la qualité de l'agent. Des valeurs plus élevées permettent aux agents de mieux explorer l'environnement, mais augmentent aussi le risque de sur-apprentissage ou de stagnation.

Nous avons utilisé ce paramètre pour évaluer l'entraînement des agents en testant différentes valeurs : 50 000, 100 000, 200 000, et 400 000 pas. Nous avons alors pu observer l'évolution des performances des agents en fonction de leur durée d'entraînement. Les résultats de cette analyse sont détaillés dans la section 6.1.

L'ensemble des agents utilisés, aussi bien pour les parties contre des joueurs humains que pour l'analyse des agents (cf. 6.2), ont été entraînés sur 200 000 pas. Ce choix représente un compromis entre le temps d'entraînement et les capacités des agents.

5.4.5 Expérimentation avec d'autres algorithmes

Avant de nous concentrer sur PPO, nous avons testé Q-Learning, mais nous avons rapidement constaté qu'il n'était pas aussi performant pour notre cas d'usage. Chaque algorithme demande une adaptation de l'environnement, ce qui rend difficile la possibilité de tester rapidement différentes approches. Finalement, ces contraintes nous ont conduits à privilégier notre variante de PPO, plus adapté à notre environnement et facilement intégré avec Gymnasium et notre modélisation du jeu.

5.5 Entraînement

Les métriques clés issues de Stable Baselines ont été utilisées pour suivre la progression de l'apprentissage et identifier d'éventuels problèmes. Une attention particulière a été portée à trois indicateurs principaux : `ep_len_mean` (longueur moyenne des épisodes), `ep_rew_mean` (récompense moyenne par épisode), et `loss` (perte). Ces métriques, observées à chaque itération, permettent d'évaluer la progression de l'entraînement et de déterminer si l'agent améliore ses performances au fil du temps.

Résultats et observations sur les performances de PPO

Exemple d'un affichage d'output de StableBaselines lors de l'entraînement :

```
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 66.2        |
|    ep_rew_mean          | -174        |
| time/                   |             |
|    fps                  | 2451        |
|    iterations           | 98          |
|    time_elapsed         | 81          |
|    total_timesteps      | 200704      |
| train/                  |             |
|    approx_kl            | 0.0028495255 |
|    clip_fraction        | 0.0269      |
|    clip_range           | 0.2         |
|    entropy_loss         | -0.577      |
|    explained_variance   | 0.28        |
|    learning_rate        | 0.0003      |
|    loss                 | 1.31e+03    |
|    n_updates            | 970         |
|    policy_gradient_loss | -0.00691    |
|    value_loss           | 2.07e+03    |
-----------------------------------------
```

Analyse des principales métriques :

- Récompense moyenne par épisode (`ep_rew_mean`) : Récompense moyenne par épisode, indiquant si l'agent apprend à maximiser les récompenses. Une augmentation de cette métrique est le principal indicateur de la progression de l'agent. Dans les premières étapes de l'entraînement, des valeurs négatives ou faibles montrent que l'agent explore sans encore optimiser sa stratégie.
- Longueur moyenne des épisodes (`ep_len_mean`) : Cette métrique représente la durée moyenne des épisodes en nombre de pas. Une diminution progressive peut refléter une convergence vers des stratégies plus efficaces. En particulier, une baisse peut indiquer que l'agent effectue de moins en moins d'actions interdites ou inefficaces, réduisant ainsi le nombre de pas nécessaires pour atteindre ses objectifs.
- Perte totale (`loss`) : La perte totale suit l'ajustement de l'agent lors de l'entraînement. Une diminution régulière suggère une amélioration dans l'estimation des valeurs d'état et la sélection d'actions optimales, témoignant d'un apprentissage efficace.

Analyse des outputs de l'entraînement

Les résultats montrent une progression marquée dans l'apprentissage :

1. **Exploration initiale** : L'agent teste différentes actions de manière aléatoire, avec des performances faibles.
2. **Optimisation progressive** : Au fil des épisodes, l'agent identifie et privilégie des actions plus efficaces, ce qui se traduit par une augmentation des récompenses moyennes.

3. **Adaptation continue** : Les variations observées dans la longueur des épisodes et les performances montrent que l'agent ajuste sa stratégie en fonction des situations rencontrées.

5.6 Interface

5.6.1 La classe Renderer

L'interface est implémentée dans le fichier `renderer.py`. Ce fichier implémente une classe `Renderer` qui gère l'affichage graphique du jeu "Petits Chevaux" en utilisant la bibliothèque Pygame.

Cette classe permet de représenter visuellement l'état actuel du jeu, incluant :

- Les positions des pions sur le plateau.
- Les zones spécifiques (écurie, chemin, escalier, objectif).
- Le joueur actuel
- Les actions disponibles pour le joueur humain.
- Le résultat d'un lancer de dé.
- Le message indiquant le gagnant.

L'interface du jeu est illustrée dans la figure 12, disponible en annexe.

5.6.2 Configuration de la partie

Le fichier `launcher.py` permet de paramétrer une partie.

Fonctionnalités principales :

- Collecte des paramètres via la ligne de commande, résumé des choix effectués et retour d'un dictionnaire de configuration.
- Traduction des règles définies en une configuration technique adaptée aux agents et à l'environnement.
- Génération et vérification des chemins des modèles pré-entraînés selon les paramètres définis.

5.6.3 Lancement de la partie

La fonction `play_game` gère la logique principale via une boucle adaptable :

- Tour humain : Gestion des interactions via l'interface grâce aux boutons définissant les différentes actions possibles.
- Tour agent : Récupération et traitement de l'output généré par l'environnement.

5.6.4 Fin de partie

La partie se termine lorsque la condition `done` (retournée par `step(action)` de l'environnement) est atteinte. Un message de fin est affiché, et la fenêtre Pygame se ferme après un délai défini.

5.7 Bases de données

Pour analyser les performances de nos agents, il a été essentiel de collecter des données sur les simulations de parties, et de bien les structurer afin de pouvoir les analyser par la suite.

Pour cela, nous avons créé une base de données relationnelle PostgreSQL permettant de centraliser et d'organiser ces informations. Ce système de gestion de bases de données a été choisi notamment car il est capable de gérer des relations complexes entre les tables, et est compatible avec SQLAlchemy permettant une intégration plus facile avec Python.

La base de données que nous avons créé comporte sept tables, le schéma détaillé est diponible en annexe à la figure 11. Chaque table a été réfléchie et structurée pour capturer des informations pertinentes sur les agents et leurs performances. Au cours de nos analyses, nous avons identifié de nouveaux besoins et donc ajusté et enrichi sa structure.

Les tables finales sont les suivantes :

- Player : Stocke les informations sur les joueurs/agents.
- Game : Stocke les détails des parties.
- SetOfRules : Stocke les configurations (ensembles de règles).
- IsRuleOf : Lien entre les tables SetOfRules et GameRule.
- GameRule : Stocke les détails de chaque règle.
- Participant : Stocke les informations sur les participants de chaque partie.
- ActionStats : Stocke les actions réalisées par les participants à chaque partie.

Une fois les données enregistrées dans la base de données, nous les avons exportées en csv, un format plus simple d'utilisation pour nos analyses.

5.8 Simulations massives

Pour analyser les performances de nos agents dans différentes configurations, nous avons lancé un grand nombre de parties de façon automatique. Ces simulations ont permis de collecter un nombre suffisant de données sur les parties jouées, les actions des agents et leurs résultats.

Les données collectées incluent les performances des agents, telles que le nombre de mouvements effectués, le nombre d'erreurs commises et les scores obtenus. Nous avons également recueilli des statistiques sur les actions demandées et celles réellement réalisées par les agents, afin d'étudier leur comportement. Enfin, les résultats des parties, comme le joueur gagnant, le nombre de pions ayant atteint leur objectif et les récompenses attribuées, ont été enregistrés.

Les simulations ont été configurées et exécutées à l'aide du fichier `ludo_stats_play.py`, qui contient plusieurs fonctions adaptées à nos besoins.

L'une de ces fonctions permet de lancer des parties en définissant, directement depuis le terminal, le nombre de joueurs, de pions, la configuration des règles, les agents à utiliser, ainsi que le nombre de parties à jouer.

Une autre fonction automatise les parties entre agents identiques en précisant les mêmes paramètres. Cette fonction est appelée plusieurs fois dans une autre qui exécute des 100 simulations avec diverses configurations, afin de générer les données nécessaires à l'analyse de l'entraînement des agents que nous détaillerons dans la section 6.1.

Enfin, pour analyser les performances des différents types d'agents les uns contre les autres, nous avons mis en place un système de simulation automatique de matchups (confrontations de différents types d'agents). Ces simulations ont été configurées pour tester les agents dans des conditions similaires, avec un nombre fixe de pas d'entraînement (200 000 steps) afin d'assurer une comparaison équitable.

Les simulations de matchups ont été organisées en deux catégories principales :

- **Parties à 2 joueurs**
  - Organisation de matchups entre toutes les paires possibles d'agents, y compris les confrontations entre agents de même type
  - 100 parties simulées pour chaque combinaison de types d'agents
  - Collecte des statistiques sur les taux de victoire, les actions effectuées et les scores moyens
- **Parties à 4 joueurs**
  - Simulation de deux formats de jeu :

    - Format "chacun pour soi" (1v1v1v1) avec quatre agents différents
    - Format "équipe contre équipe" (2v2) avec des paires d'agents identiques s'affrontant
  - Toutes les combinaisons possibles d'agents ont été testées pour le format 1v1v1v1
  - Pour le format 2v2, focus sur les confrontations entre différents types d'agents en équipe

Cette approche systématique des simulations a permis d'obtenir un large échantillon de données pour analyser les forces et faiblesses de chaque type d'agent dans différentes configurations de jeu. Les résultats de ces simulations seront analysés en détail dans la section 6.2.

6 Analyse des résultats

6.1 Analyse de l'entraînement des agents

Afin de vérifier si l'entraînement de nos agents fonctionnait bien, nous avons analysé l'évolution de leurs performances en fonction de leur nombre de pas d'entraînement, dans des configurations de jeu présentant une complexité croissante. L'objectif était de montrer que les agents s'améliorent au cours de l'entraînement.

Présentation de l'analyse

Les données utilisées pour cette analyse ont été collectées à partir de simulations massives de parties entre agents identiques, c'est-à-dire de même type et de même nombre de pas d'entraînement.

Pour évaluer l'apprentissage de nos agents, nous avons utilisé les métriques suivantes :

- La répartition des types d'actions demandées par l'agent
- Le pourcentage moyen d'erreurs
- Le score moyen

Ces métriques nous ont permis d'évaluer si les agents s'adaptaient correctement à l'environnement de jeu et aux règles.

Étant donné le nombre élevé de configurations de règles possibles, nous en avons sélectionné un sous ensemble. Les 3 configurations de jeu utilisées ont été :

- Configuration 16 : Règles de base sans contraintes
- Configuration 12 : Règles intermédiaires avec contraintes
- Configuration 17 : Règles complètes avec interactions avancées

Le choix de ces configurations ayant un nombre de contraintes croissantes, nous a permis d'observer si l'ajout de règles/contraintes a un impact sur l'apprentissage.

Tous les types d'agents ont été analysés (Balanced, Aggressive, Rusher, Defensive, Spawner, et Suboptimal), et ont été évalués dans des parties comprenant :

- 2 joueurs avec 2 pions
- 2 joueurs avec 4 pions
- 4 joueurs avec 4 pions

Pour chaque configuration, nous avons entraîné des agents sur des périodes de 50 000, 100 000, 200 000 et 400 000 pas d'entraînement. Ces agents ont ensuite été utilisés lors des parties, et les données issues de leurs performances ont été collectées. Ce sont ces données que nous utilisons pour l'analyse.

Résultats

Les résultats de nos analyses montrent que les agents RL apprennent efficacement dans des environnements simples (Configuration 16), avec une diminution progressive du pourcentage des erreurs et une amélioration

des scores, quel que soit le nombre de joueurs. Cependant, l'évolution des performances ralentit lorsque le nombre de pions sur le plateau augmente, avec des scores atteints pour un même nombre de pas d'entraînement moins élevés et un pourcentage d'erreurs plus important (Voir Figures 1, et 2).

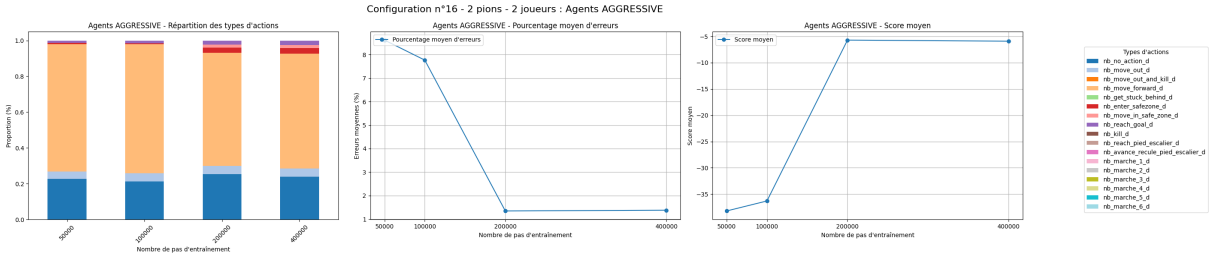

Figure 1 – Performance des agents AGGRESSIVE dans la configuration 16 (2 pions - 2 joueurs)

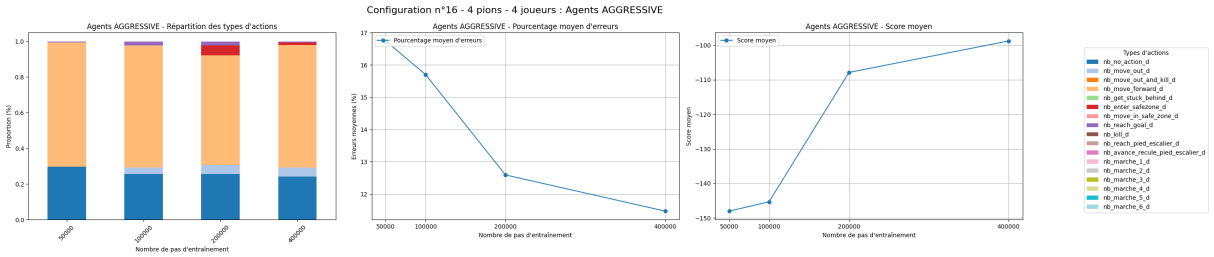

Figure 2 – Performance des agents AGGRESSIVE dans la configuration 16 (4 pions - 4 joueurs)

Lorsque la complexité des règles augmente (Configurations 12 et 17), les performances des agents deviennent plus variables et limitées. Les agents montrent une certaine capacité d'apprentissage pour des actions spécifiques, mais leurs performances globales stagnent ou régressent, comme illustré par la Figure 3. Dans ces configurations, le pourcentage d'erreurs reste élevé, les scores moyens sont très faibles, et l'apprentissage de nouvelles actions est plus lent.

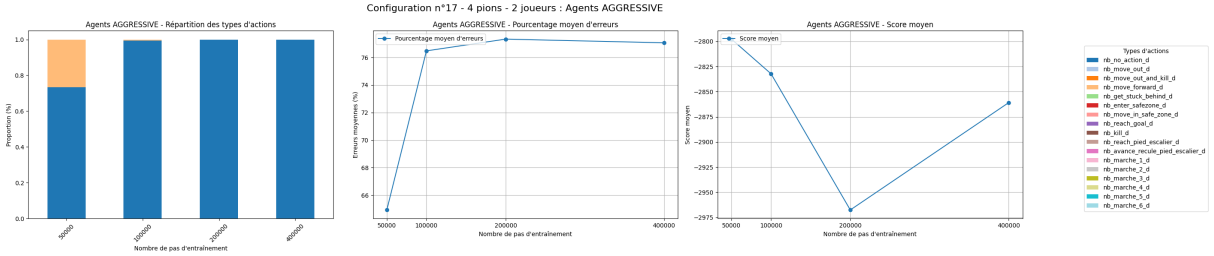

Figure 3 – Performance des agents AGGRESSIVE dans la configuration 17 (4 pions - 2 joueurs)

Ces résultats montrent que les agents RL s'entraînent efficacement dans des environnements simples, mais mettent en évidence des limites dans leur capacité à s'adapter à des configurations de règles plus complexes.

Pistes d'améliorations futures

Il serait intéressant d'entraîner plus longtemps les agents, en évitant tout de même le sur-apprentissage, afin de voir si des tendances se dessinent et s'ils arrivent à atteindre de meilleures performances, cependant, le temps d'entraînement actuel étant déjà très élevé, nous n'avons pas pu les entraîner plus longtemps. De plus, tester des configurations supplémentaires, avec des variations intermédiaires, pourrait nous permettre de mieux comprendre ce qui ralentit l'entraînement.

6.2 Analyse des parties avec des agents entrainés

Pour mettre en contexte nos analyses de résultats, nous avons réalisé trois types d'études distinctes afin d'évaluer différents aspects des agents et des configurations de jeu.

6.2.1 Performance des agents au sein d'une configuration fixe

Dans un premier temps, nous nous sommes intéressés à la performance relative des différents types d'agents (*balanced*, *aggressive*, *rusher*, *defensive*, *spawner*, *suboptimal*) au sein d'une configuration fixe. Dans les parties à deux joueurs, nous avons mesuré les taux de victoire de chaque type d'agent contre tous les autres types d'agents possibles, ce qui nous a permis d'identifier les stratégies les plus efficaces dans un contexte donné. Pour les parties à quatre joueurs, nous avons analysé la distribution des positions finales (de la première à la quatrième place) en fonction du nombre de pions ayant atteint leur objectif, offrant ainsi une vision plus nuancée de l'efficacité des différentes stratégies dans un environnement plus complexe.

6.2.2 Satisfaction des agents selon les règles du jeu

Notre deuxième axe d'analyse s'est concentré sur la satisfaction des agents face aux différents ensembles de règles. Pour cette étude, nous avons fixé le nombre de joueurs et de pions, ne faisant varier que les règles du jeu. Cette approche nous a permis d'évaluer quelles variations des règles étaient les plus appréciées par chaque type d'agent. Par exemple, les agents agressifs pourraient montrer une satisfaction accrue dans des configurations permettant plus d'interactions entre joueurs, tandis que les agents défensifs pourraient préférer des règles favorisant la sécurisation des pions. Pour quantifier cette satisfaction, nous avons utilisé plusieurs métriques adaptées à chaque type d'agent :

- Pour les agents *aggresive* : le ratio d'actions offensives réussies et le nombre de pions adverses renvoyés à leur écurie
- Pour les agents *defensive* : le taux de pions sécurisés et le nombre d'actions défensives réussies
- Pour les agents *rusher* : la vitesse moyenne d'atteinte des objectifs
- Pour les agents *spawner* : le nombre de pions qu'il a mis sur le plateau
- Pour les agents *balanced* et *suboptimal* : une combinaison pondérée des différentes métriques

6.2.3 Métriques de jouabilité

Enfin, notre troisième perspective d'analyse s'est focalisée sur les métriques de jouabilité, qui évaluent la qualité et l'équilibre des parties. Un aspect crucial de cette analyse concerne la longueur des parties en fonction des types d'agents qui s'affrontent. Pour les parties à quatre joueurs en configuration 2v2 (2 joueurs de même type contre 2 autres), nous avons étudié la distribution du nombre de pions atteignant leur objectif pour chaque équipe. Une distribution similaire entre les équipes indique des parties équilibrées et donc potentiellement plus intéressantes pour les joueurs. Cette analyse a été complétée par l'étude de l'impact du nombre de pions sur la satisfaction des agents, permettant de comprendre comment la complexité du jeu influence l'expérience de jeu pour chaque type de stratégie.

Cette approche à trois niveaux nous permet d'avoir une vue complète de l'efficacité des différentes stratégies, de l'impact des règles sur le comportement des agents, et de la qualité générale de l'expérience de jeu qui en résulte.

6.3 Résultats

L'étude des performances des agents dans notre jeu présente un défi analytique significatif en raison du nombre important de configurations possibles. En effet, avec 32 configurations de règles différentes, des parties à 2 ou 4 joueurs, les 6 types d'agents différents, et des possibilités de 2 ou 4 pions par joueur, le

nombre total de combinaisons à analyser est considérable. Il n'est donc pas réaliste de présenter une analyse exhaustive de tous les scénarios dans ce rapport. Nous avons choisi de nous concentrer sur certaines configurations représentatives afin d'illustrer les types d'analyses et d'insights qu'il est possible d'obtenir avec nos outils. Pour une analyse plus complète, le lecteur est invité à consulter les notebooks d'analyse disponibles dans le dépôt du projet.

6.3.1 Performance des agents au sein d'une configuration fixe

Pour illustrer l'analyse des performances relatives des agents, nous nous concentrons sur la configuration 8, qui offre un équilibre intéressant entre les différentes stratégies. La figure 4 présente la matrice des taux de victoire entre les différents types d'agents dans les parties à deux joueurs.

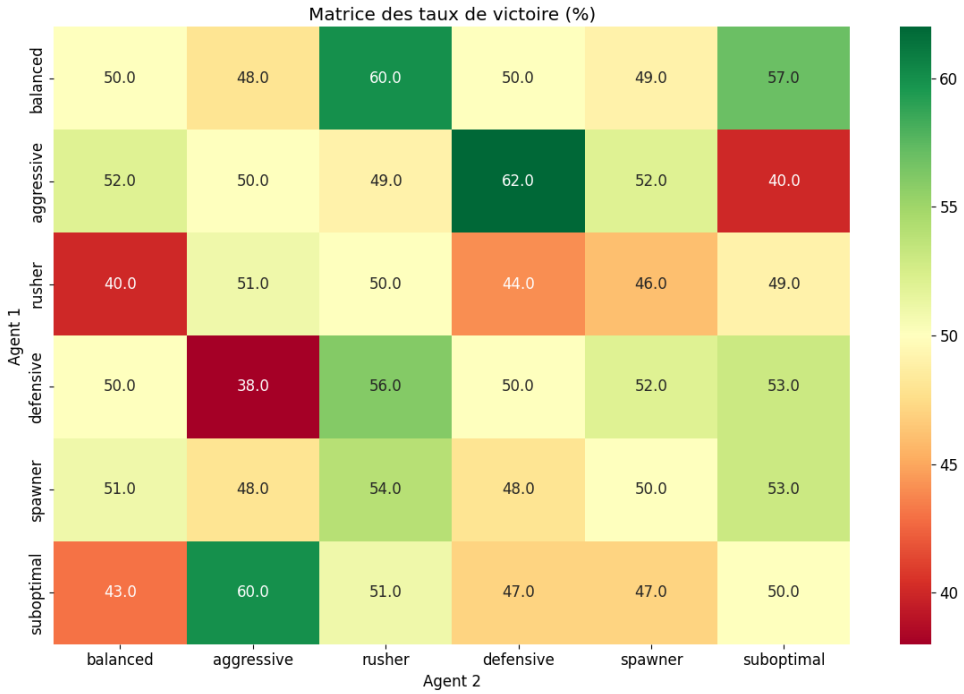

Figure 4 – Matrice des taux de victoire entre agents pour la configuration 8

Cette matrice révèle des différences significatives dans l'efficacité des stratégies. Par exemple, l'agent *aggressive* démontre une forte dominance contre l'agent *defensive*, remportant 62% des confrontations directes. Cependant, cette domination n'est pas uniforme : face à l'agent *rusher*, l'agent *aggressive* voit son taux de victoire baisser à 49%, suggérant une plus grande difficulté à contrer cette stratégie.

La dynamique change significativement dans les parties à quatre joueurs, comme le montre la figure 5.

Dans ce contexte plus complexe, où chaque agent fait face à des adversaires de différents types simultanément, les écarts de performance tendent à se réduire. L'agent *defensive*, par exemple, parvient à atteindre la première place aussi fréquemment que l'agent *aggressive*. La différence principale se situe au niveau de la distribution des deuxièmes places, où l'agent *defensive* montre des performances inférieures à l'agent *aggressive*. Ces observations fournissent des informations précieuses pour les créateurs de jeux cherchant à équilibrer les différentes stratégies et à créer une expérience de jeu engageante pour tous les styles de jeu.

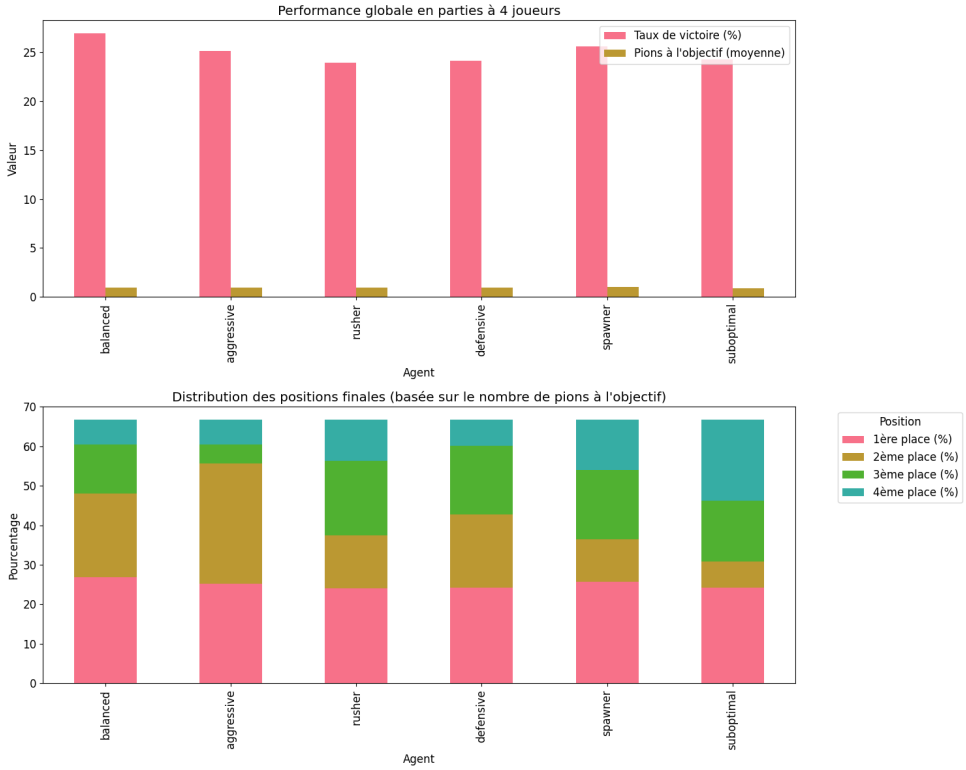

Figure 5 – Distribution des positions finales par type d'agent en parties à 4 joueurs

6.3.2 Satisfaction des agents selon les règles du jeu

Pour illustrer l'analyse de la satisfaction des agents selon les différentes configurations de règles, concentrons nous sur l'agent *aggressive* dans un contexte de parties à deux joueurs avec deux pions. La figure 6 montre le nombre moyen d'éliminations de pions adverses (kills) réalisées par cet agent à travers différentes configurations de règles.

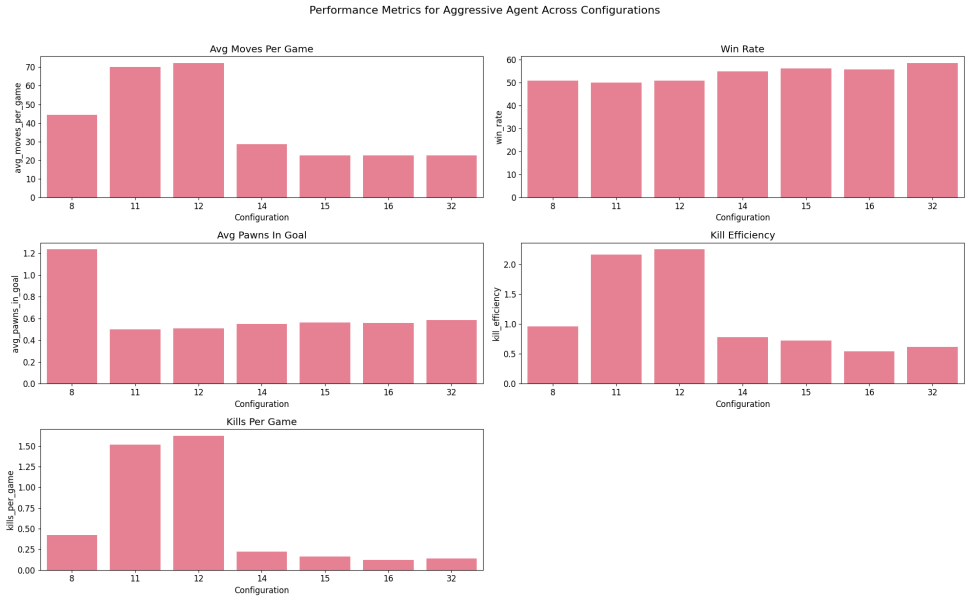

Figure 6 – Nombre moyen de kills par partie pour l'agent aggressive selon les configurations

On observe que les configurations 11 et 12 permettent à l'agent *aggressive* d'obtenir le plus grand nombre d'éliminations par partie. Cette performance s'explique par la nature plus stricte de ces configurations : elles imposent des contraintes plus importantes pour atteindre l'objectif, notamment en exigeant des valeurs exactes de dé pour progresser dans l'escalier. Par conséquent, les pions restent plus longtemps sur le plateau de jeu, offrant davantage d'opportunités d'élimination.

Il est intéressant de noter que ces configurations, bien qu'optimales pour la satisfaction de l'agent *aggressive* en termes de kills, ne correspondent pas nécessairement à celles où il obtient ses meilleurs taux de victoire. Cette distinction souligne la différence entre l'efficacité objective (mesurée par les victoires) et la satisfaction subjective (mesurée par les opportunités d'élimination) d'un type de stratégie.

Cette analyse met en évidence l'importance de considérer plusieurs métriques lors de l'évaluation de l'expérience de jeu. En cherchant un équilibre entre le taux de satisfaction (ici, le nombre de kills) et le taux de victoire, les concepteurs de jeux peuvent identifier les configurations de règles qui offriront l'expérience la plus satisfaisante pour chaque type de joueur, même si ces configurations ne maximisent pas nécessairement leurs chances de victoire.

6.3.3 Métriques de jouabilité

Pour évaluer l'équilibre et la jouabilité des différentes configurations, examinons en détail la configuration 7. La figure 7 présente la distribution des longueurs de parties pour les différentes combinaisons d'agents dans les parties à deux joueurs avec deux pions.

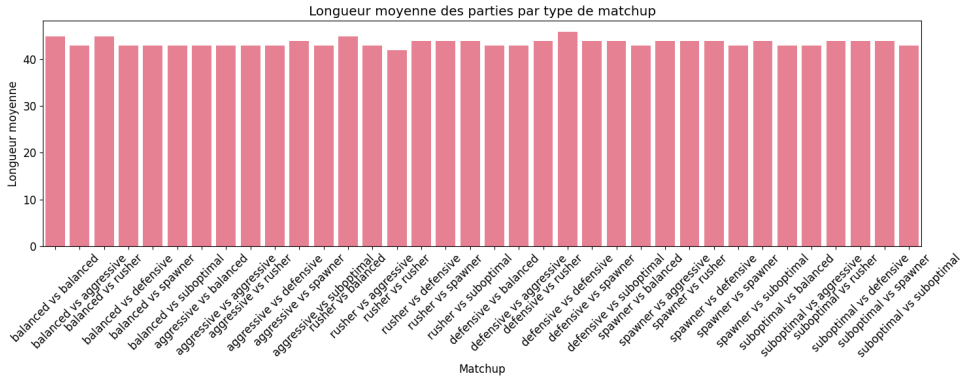

Figure 7 – Distribution des longueurs de parties selon les types d'agents en confrontation

À première vue, cette configuration semble bien équilibrée : les variations de durée des parties sont minimales quelle que soit la combinaison d'agents qui s'affrontent. Cette uniformité suggère que la configuration offre un cadre de jeu stable où aucune stratégie ne permet de raccourcir ou de rallonger significativement la durée des parties.

Cependant, l'analyse des parties à quatre joueurs en configuration 2v2 (deux agents de même type contre deux autres agents de même type) révèle une autre dimension de l'équilibre du jeu, comme le montre la figure 8.

Ce graphique de dispersion montre le nombre moyen de pions atteignant l'objectif pour chaque équipe. Les points proches de la diagonale représentent des parties équilibrées où les deux équipes parviennent en moyenne à placer un nombre similaire de pions à l'objectif. La dispersion importante des points observée suggère que, contrairement à ce qu'indiquait l'analyse des durées de parties, cette configuration présente des déséquilibres significatifs dans les confrontations d'équipes. Certaines combinaisons de stratégies semblent nettement plus efficaces que d'autres pour amener les pions à l'objectif, remettant en question l'équilibre global de cette configuration.

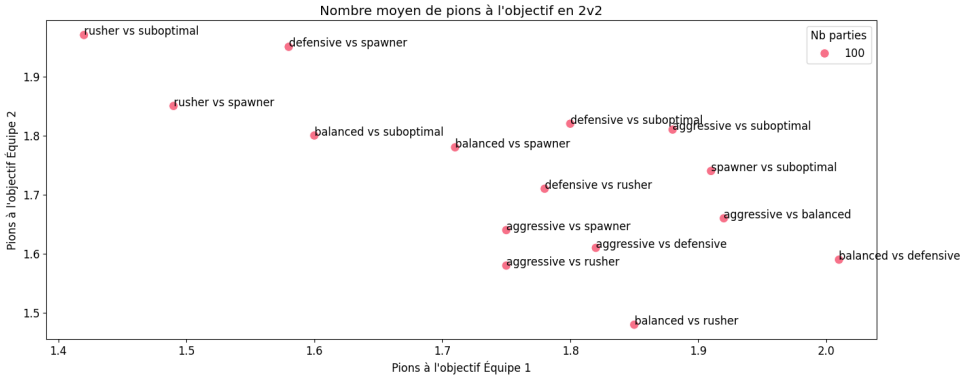

Figure 8 – Distribution du nombre moyen de pions à l'objectif par équipe en configuration 2v2

6.3.4 Cas de sous-performance

Nos analyses ont révélé des cas de sous-performance significative pour certains agents dans des configurations spécifiques. Un exemple particulièrement marquant concerne l'agent *balanced* dans la configuration 15, comme illustré dans la figure 9.

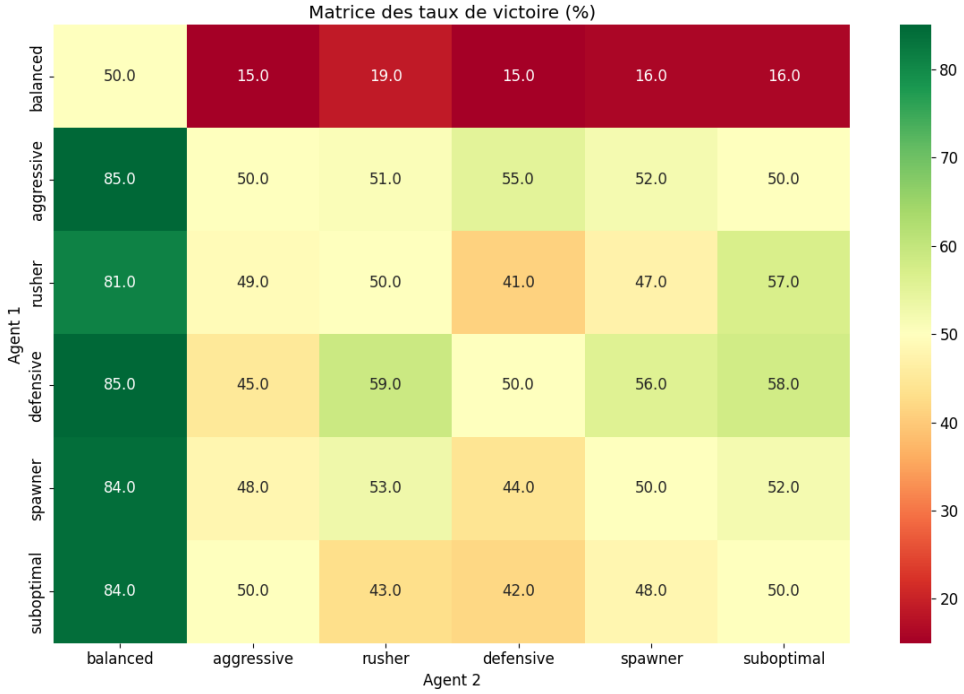

Figure 9 – Taux de victoire de l'agent balanced en configuration 15 (2 joueurs, 2 pions)

Dans cette configuration, l'agent *balanced* présente des taux de victoire remarquablement bas, inférieurs à 20%. Cette sous-performance peut s'expliquer par deux facteurs principaux : la nature spécifique de la configuration 15, qui favorise des stratégies plus spécialisées, et la distribution des récompenses associées aux différentes actions. En effet, la nature généraliste de l'agent *balanced* semble particulièrement désavantagée dans ce contexte où une approche plus ciblée serait nécessaire.

6.4 Prise de recul sur l'implémentation

L'analyse des résultats nous permet de porter un regard critique sur nos choix d'implémentation et sur les performances des agents. Plusieurs points méritent une attention particulière.

6.4.1 Limites de l'optimisation

Une limitation importante de notre approche concerne l'optimisation des valeurs de récompense. L'entraînement des agents et la phase de jeu étant des processus chronophages, nous n'avons pas pu optimiser finement les valeurs de récompense pour chaque configuration. Cette contrainte temporelle a potentiellement impacté les performances de certains agents dans des configurations spécifiques. Cependant, il est important de noter que notre objectif principal n'était pas de créer des agents parfaits, mais plutôt de simuler des comportements de joueurs réels pour évaluer différentes configurations de jeu.

6.4.2 Complexité des interactions

L'analyse des parties à quatre joueurs a mis en évidence la complexité des interactions entre les différentes stratégies. Les performances observées en confrontation directe (2 joueurs) ne se traduisent pas toujours de manière linéaire dans les parties à quatre joueurs, suggérant que notre modèle de récompense pourrait être enrichi pour mieux prendre en compte ces dynamiques complexes.

6.4.3 Équilibre entre exploration et exploitation

La variabilité des performances selon les configurations soulève également des questions sur l'équilibre entre exploration et exploitation durant la phase d'apprentissage. Certaines configurations, particulièrement celles impliquant des règles plus complexes, pourraient nécessiter une phase d'exploration plus longue pour permettre aux agents de découvrir des stratégies optimales.

6.4.4 Synthèse des observations

Ces observations nous permettent d'identifier plusieurs pistes d'amélioration potentielles, notamment :

- L'ajustement des valeurs de récompense en fonction des spécificités de chaque configuration
- L'introduction de mécanismes d'apprentissage plus adaptés aux situations multi-agents
- L'optimisation des hyperparamètres d'entraînement selon la complexité des configurations

Néanmoins, malgré ces limitations, notre approche a permis d'obtenir des résultats significatifs pour l'analyse comparative des configurations de jeu et des performances des agents, remplissant ainsi notre objectif principal d'évaluation des différentes variantes des règles et leurs adaptations par différents types de joueurs.

7 Retour d'expérience du développement du jeu Labyrinthe

7.1 Introduction au jeu Labyrinthe

Le jeu Labyrinthe est un jeu de plateau stratégique où les joueurs doivent retrouver des trésors dans un labyrinthe en constante évolution. Le plateau est composé de plaques fixes et mobiles, que les joueurs peuvent déplacer pour modifier les couloirs. Chaque joueur doit atteindre les trésors indiqués sur ses cartes et revenir à son point de départ pour remporter la partie.

Une partie se déroule en plusieurs tours. Lors de son tour, un joueur dispose d'une tuile mobile qu'il peut faire pivoter dans le sens souhaité avant de l'insérer dans le plateau (en respectant la règle qui interdit d'insérer une tuile à l'endroit où la dernière insertion a été faite). Cette action se compose généralement de deux ou

trois phases : rotation de la tuile, insertion dans le plateau, puis déplacement de son pion en fonction des nouveaux chemins créés.

Le choix de ce jeu pour notre projet était motivé par sa complexité spatiale et stratégique. Il offre un environnement dynamique qui oblige les agents à anticiper les changements et à s'adapter aux actions des autres joueurs. La richesse des interactions possibles et des stratégies nous avait poussé à le choisir comme environnement de l'apprentissage par renforcement.

7.2 Travail réalisé

Pour démarrer ce projet, nous avons utilisé comme base le projet "Labyrinthe des IUT", disponible sur GitHub [10], une version du jeu Labyrinthe développée en Python. Cette approche nous a permis de gagner du temps en réutilisant les mécanismes et les règles de jeu déjà implémentés. Nous avons ensuite amélioré le code en corrigeant les bugs existants et en ajustant les fonctionnalités pour qu'elles respectent les règles du jeu. De plus, nous avons développé une interface utilisateur enrichie, et commencé à mettre en place une base de données. Une illustration de l'interface améliorée est disponible en annexe 13.

Nous avons développé un environnement Gymnasium, afin d'avoir une interface standardisée pour les algorithmes RL. L'espace d'actions a été séparé en deux phases distinctes : la phase d'insertion/rotation de la tuile, et la phase de déplacement du pion. Cet environnement incluait également la gestion des états du jeu, tels que la position des joueurs, la structure du labyrinthe, et les trésors restants. Des mécanismes de récompense ont été définis pour guider les agents, attribuant des récompenses pour l'atteinte des trésors ou la progression vers un objectif, et imposant des pénalités pour les actions inutiles ou impossibles.

Pour entraîner nos agents, nous avons étudié différents algorithmes tels que DQL, Q-Learning et PPO avec Stable-Baselines3, mais notre choix final s'est porté sur l'algorithme PPO. Afin de traiter la complexité du jeu et de réduire les erreurs liées aux actions impossibles, nous avons intégré SB3 Contrib, une extension de Stable-Baselines3 qui permet le masquage des actions interdites. Les performances des agents lors de l'entraînement ont été suivies à l'aide de TensorBoard afin de visualiser des métriques comme les récompenses moyennes et la longueur des épisodes.

Enfin, nous avons intégré nos agents entraînés dans le jeu du Labyrinthe afin de leur permettre d'affronter les joueurs humains. Nous nous sommes alors rendu compte de plusieurs problèmes. En effet, les agents avaient du mal à jouer de manière cohérente, répétant sans cesse des mouvements impossibles, restant bloqués sur une même pièce ou se déplaçant de manière incorrecte, comme en traversant des murs.

Malgré de nombreuses tentatives pour simplifier les règles, modifier les paramètres ou tester différents algorithmes, nous n'avons pas réussi à obtenir des résultats satisfaisants. Cela a mis en évidence la difficulté pour les agents de planifier efficacement leurs actions dans un environnement aussi dynamique et complexe. Cette situation nous a conduit à analyser en détail les blocages rencontrés, afin de mieux comprendre les causes de ces échecs et d'identifier des pistes d'amélioration, nous le détaillons dans la section suivante.

7.3 Analyse des blocages

Problèmes rencontrés

L'un des principaux obstacles provenait de la complexité intrinsèque du jeu. Le Labyrinthe offre un espace d'actions très large, avec des phases distinctes pour insérer et faire pivoter les plaques, puis pour déplacer les pions vers une des 49 cases possibles. Cette multiplicité des choix rendait l'apprentissage difficile pour les agents, qui peinaient à identifier des stratégies efficaces. De plus, l'environnement dynamique du Labyrinthe, où les configurations changent constamment, exigeait une planification à court et long terme, un défi que les agents n'ont pas su relever efficacement.

Les limites techniques des algorithmes utilisés ont également joué un rôle important. L'approche PPO n'a pas pu gérer efficacement la séparation des actions en deux phases distinctes. Bien que des masques d'actions aient été introduits pour éviter les choix impossibles, leur intégration a généré des problèmes de compatibilité, rendant l'apprentissage incohérent. Malgré ces masques, les agents exécutaient encore beaucoup d'actions invalides, indiquant une compréhension imparfaite voire nulle des règles et des contraintes de l'environnement. En complément, les algorithmes DQN et Asynchronous Advantage Actor-Critic (A3C) ont également été testés. Cependant, leurs performances se sont avérées encore moins satisfaisantes que celles de PPO, échouant à converger vers des comportements adaptés ou à tirer parti des mécanismes de l'environnement.

Principales tentatives pour surmonter les obstacles

Une stratégie *epsilon-greedy* a été testée pour encourager l'exploration de nouvelles actions et éviter des choix répétitifs. Bien que ces efforts aient introduit une certaine diversité dans le comportement des agents, ils ont aussi contribué à renforcer des stratégies inefficaces en interprétant parfois des actions aléatoires comme bénéfiques.

Différents espaces d'actions ont également été explorés pour répondre aux spécificités des phases du jeu. Par exemple, un espace `gym.spaces.MultiDiscrete([4, 12])` a été utilisé pour représenter les phases d'insertion et de rotation, tandis qu'un espace `gym.spaces.Discrete(49)` a été testé pour les déplacements sur la grille. Bien que cette séparation visait à structurer le processus décisionnel de l'agent, elle s'est heurtée à l'incapacité de l'algorithme PPO standard de gérer efficacement des espaces d'actions dynamiques, entraînant des incompatibilités et rendant l'apprentissage incohérent.

Certaines actions inutiles ou non valides ont été masquées afin de réduire la taille de l'espace d'actions et de simplifier l'apprentissage des agents. Pour cela, l'extension SB3 Contrib et l'algorithme MaskablePPO ont été intégrés, permettant une gestion plus efficace des actions impossibles. Ce masquage visait à éliminer les choix non valides tout en guidant les agents vers des décisions plus pertinentes. Cependant, cette approche a révélé des incompatibilités lors des tests où les agents affrontaient des humains. Ces limitations ont empêché une validation complète des comportements des agents et restreint l'évaluation de leur progression dans des conditions réalistes.

Une autre tentative a consisté à appliquer une pénalisation lorsque les agents sélectionnaient des actions impossibles, comme essayer de se déplacer vers une cellule inaccessible. Bien que cette méthode visait à décourager ces comportements indésirables, les agents continuaient malgré tout à exécuter ces actions une fois entraînés, indiquant une compréhension limitée des contraintes de l'environnement.

Nous avons également testé de limiter chaque partie à un nombre prédéfini d'étapes si aucune victoire n'était obtenue. Bien que cette approche ait permis de réduire la durée de chaque session d'entraînement, elle n'a pas aidé les agents à converger vers des stratégies gagnantes, car le nombre de séquences d'apprentissage était trop restreint.

Par ailleurs, des algorithmes alternatifs tels que DQN et A3C ont été testés pour tenter de surmonter les limitations de PPO. Malheureusement, ces méthodes ont produit des résultats encore moins satisfaisants, échouant à converger ou à exploiter efficacement les mécanismes de l'environnement.

Enfin, une approche basée sur le retour des masques d'actions après chaque étape (`step()`) a été envisagée pour fournir à l'agent une information supplémentaire sur les choix possibles. Cependant, l'algorithme PPO de Stable-Baselines3 n'a pas pu exploiter ces informations, ce qui a mené à un apprentissage inefficace et incohérent.

7.4 Synthèse des enseignements

Le développement d'agents en apprentissage par renforcement pour le jeu du Labyrinthe nous a permis de tirer plusieurs enseignements.

Tout d'abord, nous avons constaté que la complexité de l'environnement a un impact direct sur la capacité des agents à apprendre. Un espace d'actions trop vaste et des règles dynamiques ont rendu l'apprentissage inefficace, mettant en évidence la nécessité de simplifier les environnements pour permettre une progression plus rapide et stable des agents.

Ensuite, l'importance des outils et algorithmes adaptés est devenue évidente. Les solutions standard comme PPO, bien qu'efficaces dans des environnements plus simples, n'étaient pas suffisamment robustes pour gérer la complexité du Labyrinthe. L'introduction de SB3 Contrib a apporté des améliorations, mais des problèmes d'intégration subsistaient, limitant son efficacité.

Enfin, l'utilisation d'outils de suivi tels que TensorBoard a mis en évidence l'importance de visualiser et d'analyser les performances des agents pour identifier des comportements problématiques et ajuster les paramètres d'entraînement. Cependant, bien que cet apprentissage ait été essentiel, nous avons opté pour d'autres méthodes de suivi et d'analyse dans la suite du projet, adaptées à nos besoins spécifiques.

Ces expériences ont renforcé notre compréhension des défis liés au développement d'agents RL pour des environnements complexes et nous ont permis de mieux cerner les étapes nécessaires pour obtenir des résultats concluants.

7.5 Pistes d'améliorations

Pour surmonter certaines des limitations rencontrées lors du développement des agents pour le jeu du Labyrinthe, plusieurs pistes d'amélioration peuvent être envisagées.

La première étape consisterait à simplifier l'environnement. En réduisant la taille du plateau ou en limitant le nombre d'actions possibles, on pourrait diminuer la complexité pour les agents, leur permettant de se concentrer sur des tâches essentielles et d'acquérir des stratégies de base avant de s'attaquer à un environnement plus complexe.

Une autre approche consisterait à explorer des algorithmes plus avancés. Les méthodes hiérarchiques, qui décomposent les tâches complexes en sous-tâches plus simples, pourraient offrir une meilleure structure d'apprentissage. De plus, les algorithmes multi-agents, conçus pour gérer les interactions entre plusieurs entités, pourraient être particulièrement adaptés à ce jeu où les actions des agents influencent directement celles des autres joueurs.

Pour mieux comprendre et analyser les comportements des agents, l'introduction d'outils de visualisation plus détaillés serait également bénéfique. Ces outils permettraient de suivre en temps réel les décisions des agents, d'identifier les points bloquants et de tester plus efficacement différentes configurations ou paramètres d'apprentissage.

7.6 Conclusion à propos du Labyrinthe

Après être restés bloqués pendant un certain temps sur ce problème complexe sans trouver de solution ou d'exemple existant pour le surmonter, nous avons décidé, en concertation avec notre professeur, de repartir de zéro afin de produire un résultat concret dans le délai limité de moins de deux mois.

Même si nous avons dû revoir nos objectifs initiaux à la baisse, cette décision stratégique a permis de surmonter les blocages et de relancer les autres tâches en suspens.

Cette fois-ci, nous avons choisi de ne pas partir d'un code existant, mais de concevoir notre propre implémentation du jeu, de l'interface, de l'environnement, ainsi que de leur intégration avec des outils tels que Stable-Baselines, des bases de données et des systèmes de suivi statistique. Ce choix stratégique nous a permis de développer un projet entièrement fonctionnel tout en mettant en valeur notre contribution originale à chaque composante.

8 Conclusion

8.1 Objectifs atteints

Nous avons développé plusieurs agents capables de jouer au jeu des Petits Chevaux en tenant compte de différentes variantes de règles et de comportements. Une interface a également été conçue pour permettre aux utilisateurs de jouer contre ces agents. Par ailleurs, des simulations à grande échelle ont été réalisées afin de mener des analyses statistiques approfondies.

Ce projet nous a permis de développer un environnement complet d'analyse pour le jeu des Petits Chevaux, combinant apprentissage par renforcement et analyse statistique approfondie. Nos résultats mettent en évidence la complexité inhérente à l'équilibrage d'un jeu, même apparemment simple, lorsqu'on cherche à satisfaire différents types de joueurs.

Nos analyses selon trois axes principaux - performance pure, satisfaction subjective et métriques de jouabilité - ont révélé des tensions intéressantes dans la conception du jeu. Par exemple, nous avons observé qu'une configuration optimale pour le taux de victoire d'un type d'agent ne correspond pas nécessairement à celle qui maximise sa satisfaction subjective. De même, une configuration qui semble équilibrée en termes de durée de partie peut présenter des déséquilibres significatifs dans d'autres aspects, comme la distribution des pions à l'objectif en mode équipe. Nous avons aussi constaté que certains agents sont plus performants pour jouer avec certaines règles que d'autres.

Cette multiplicité des critères d'évaluation souligne l'importance d'une approche holistique dans la conception des règles du jeu. Un équilibre doit être trouvé entre :

- L'efficacité pure des stratégies, mesurée par les taux de victoire
- La satisfaction subjective des différents types de joueurs
- L'équité globale du jeu, notamment dans les configurations multi-joueurs
- La durée et le rythme des parties
- La richesse des interactions entre joueurs

8.2 Utilité et applications de ce projet

Notre travail démontre l'utilité des techniques d'apprentissage par renforcement comme outil d'analyse pour les concepteurs de jeux. Bien que nos agents ne visent pas à être parfaits, ils permettent de simuler efficacement différents styles de jeu et d'évaluer rapidement de nombreuses configurations de règles.

Cette approche pourrait être étendue à d'autres jeux de plateau, offrant aux créateurs un outil précieux pour tester et affiner leurs mécaniques de jeu avant même les phases de test avec des joueurs humains.

Nous avons développé un jeu avec plusieurs variantes de règles et d'agents, permettant aux joueurs d'affronter des agents entraînés par nos soins. Techniquement, ce jeu est prêt à être publié et peut offrir une expérience interactive et dynamique pour les joueurs.

8.3 Perspectives

Les limitations identifiées dans notre étude, notamment en termes d'optimisation des récompenses et d'équilibrage fin des configurations, ouvrent des perspectives intéressantes pour des travaux futurs. Une analyse plus approfondie des interactions entre les différents paramètres du jeu, potentiellement assistée par des techniques d'optimisation automatique, pourrait permettre d'identifier des configurations encore plus équilibrées et satisfaisantes pour tous les types de joueurs.

Bien que l'implémentation d'une approche multi-agents n'ait pas été possible dans le cadre actuel du projet, il serait intéressant d'explorer cette voie à l'avenir. L'adaptation de l'environnement Gymnasium vers PettingZoo permettrait de simuler des interactions naturelles entre agents, qu'elles soient compétitives ou coopératives, et d'introduire de nouvelles dynamiques. Cette extension offrirait des perspectives enrichissantes pour le projet, en permettant d'approfondir l'étude des comportements complexes dans des environnements multi-agents.

8.4 Ce que nous avons appris

Ce projet nous a offert une occasion unique d'explorer des concepts et des outils rarement abordés dans notre formation, tout en renforçant des compétences essentielles :

- Apprentissage par renforcement : Nous avons approfondi nos connaissances sur des algorithmes tels que PPO et Q-Learning, en les appliquant à des jeux de plateau. Cela nous a permis de mieux comprendre leurs limites et leurs applications pratiques.
- Développement technique : La création d'un environnement de jeu personnalisé et d'une interface utilisateur interactive nous a permis de combiner apprentissage par renforcement, simulation, et expérience utilisateur.
- Analyse et gestion des données : Nous avons mis en place une base de données PostgreSQL pour collecter et organiser les métriques des simulations, tout en développant des compétences en visualisation et en interprétation de données statistiques.
- Collaboration et gestion de projet : Ce travail d'équipe a renforcé nos capacités de communication, de coordination et de documentation technique pour structurer efficacement le projet.

En somme, ce projet nous a non seulement permis d'acquérir des compétences techniques et analytiques, mais aussi d'améliorer notre capacité à collaborer sur un projet complexe, tout en explorant des concepts innovants en science des données et en intelligence artificielle.

9 Bibliographie

Références

[1] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[4] C. J. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[6] Gwanggyu Sun and Ryan Spangler. Using reinforcement learning to optimize the rules of a board game.

[7] Haoran Wang, Thaleia Zariphopoulou, and Xunyu Zhou. Exploration versus exploitation in reinforcement learning : a stochastic control approach, 2019.

[8] NDurocher. Yaral. https://github.com/NDurocher/YARAL, 2022.

[9] roger creus. blokus_ai. https://github.com/roger-creus/blokus-ai, 2024.

[10] cookiehacker29. Labyrinthe_python. https://github.com/cookiehacker29/Labyrinthe_Python, 2018.

[11] SimonLBSoerensen. Ludopy. https://github.com/SimonLBSoerensen/LUDOpy, 2023.

[12] Seldre99. Ludo-rl. https://github.com/Seldre99/Ludo-RL, 2024.

[13] BAW2501. Multiagentludoenv. https://github.com/BAW2501/MultiAgentLudoEnv, 2024.

[14] andresc889. Ailudoplayer. https://github.com/andresc889/AILudoPlayer, 2017.

[15] mirjanic. Rludo. https://github.com/mirjanic/RLudo/tree/master, 2020.

[16] rajtilakls2510. solving_ludo1. https://github.com/rajtilakls2510/solving_ludo1, 2024.

[17] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium : A standard interface for reinforcement learning environments, 2024.

[18] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3 : Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

10 Annexes

Schéma de configuration du jeu

La mindmap ci-dessous illustre les différentes options disponibles dans le launcher pour la configuration initiale du jeu, notamment la sélection du nombre de joueurs, le type de joueurs (agents ou humains), et d'autres paramètres.

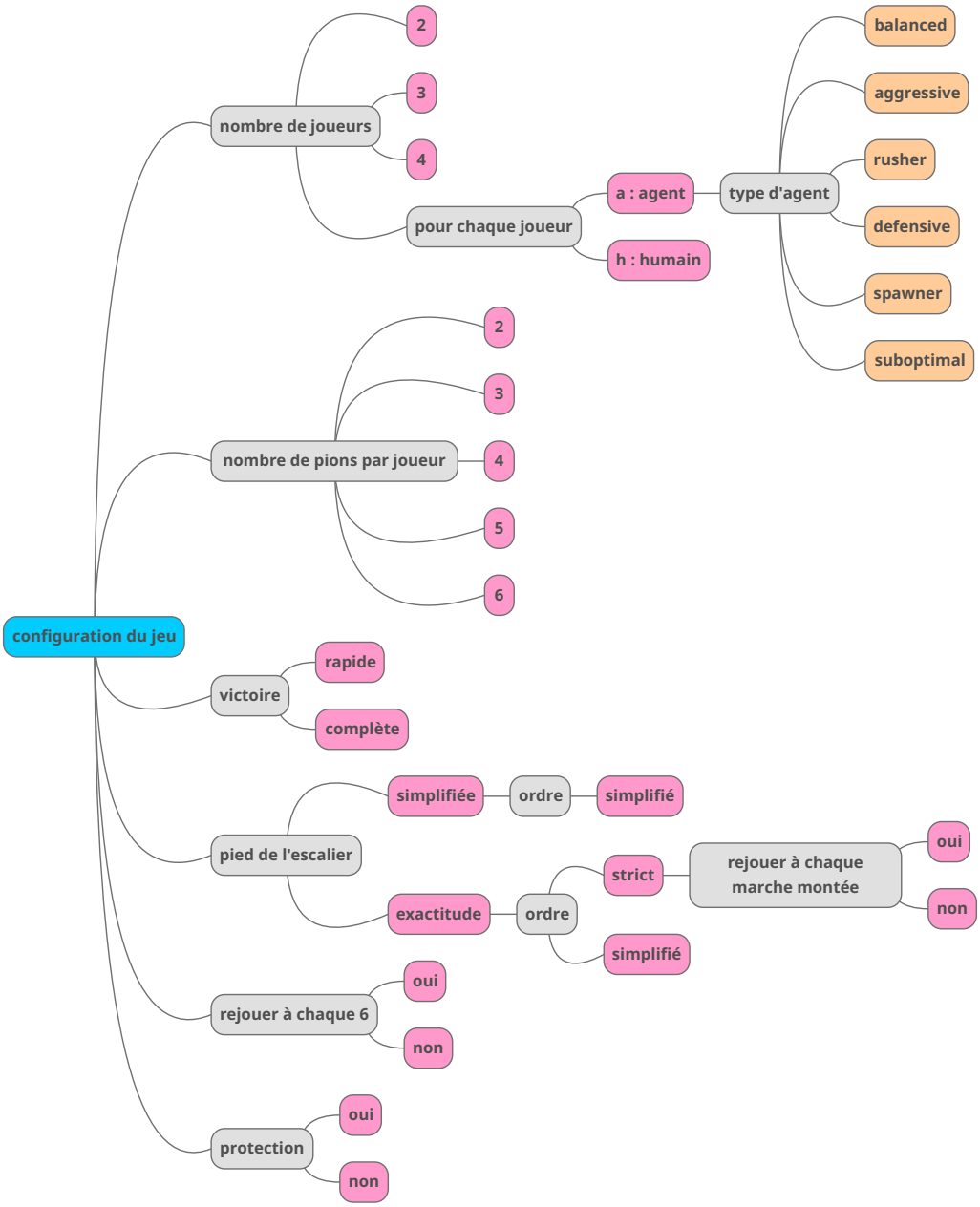

Figure 10 – Mindmap des options de configuration du jeu

Schéma de la base de données

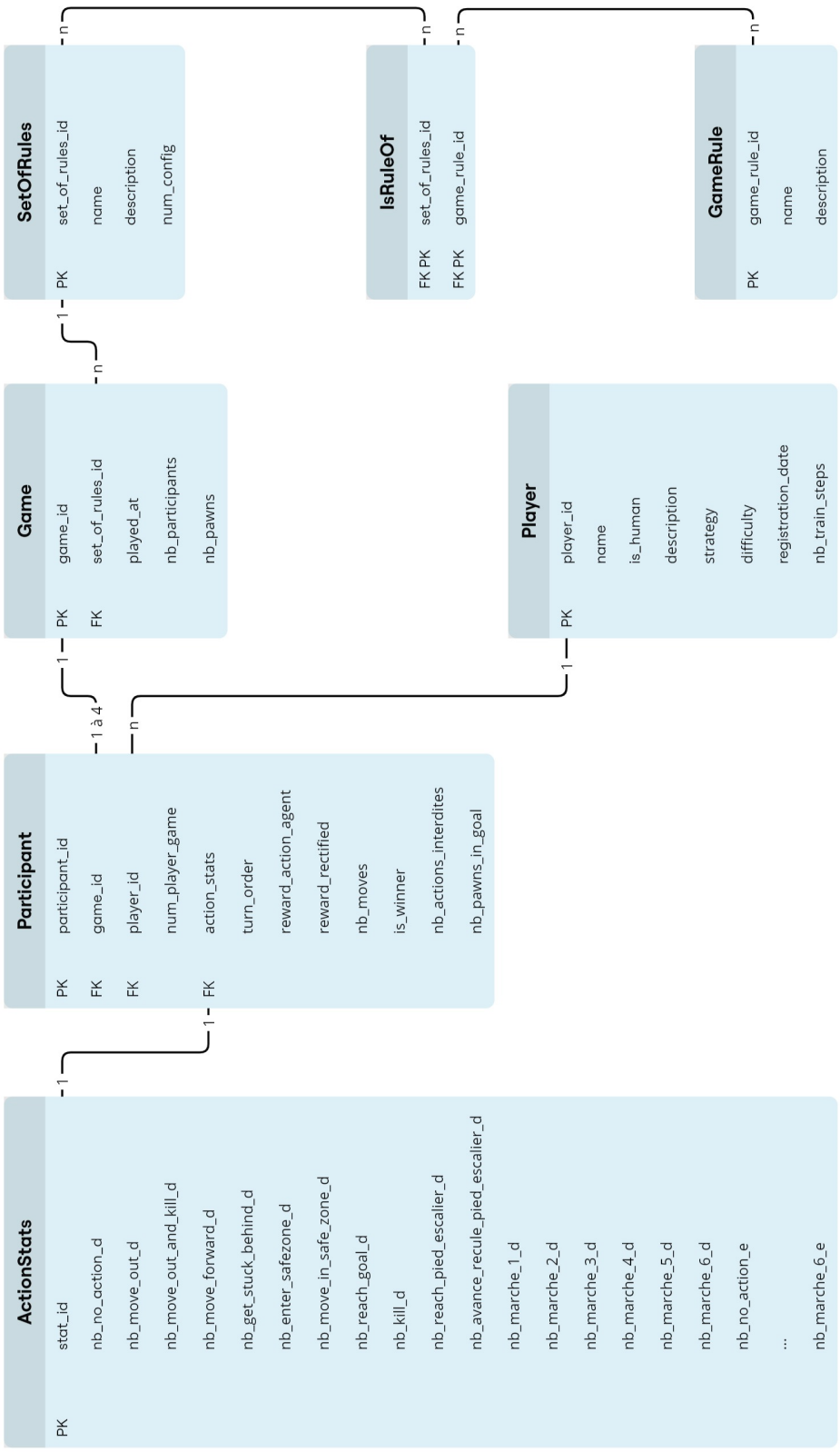

Figure 11 – Schéma de la base de donnée ludo_stats

Interface du jeu "Petits Chevaux"

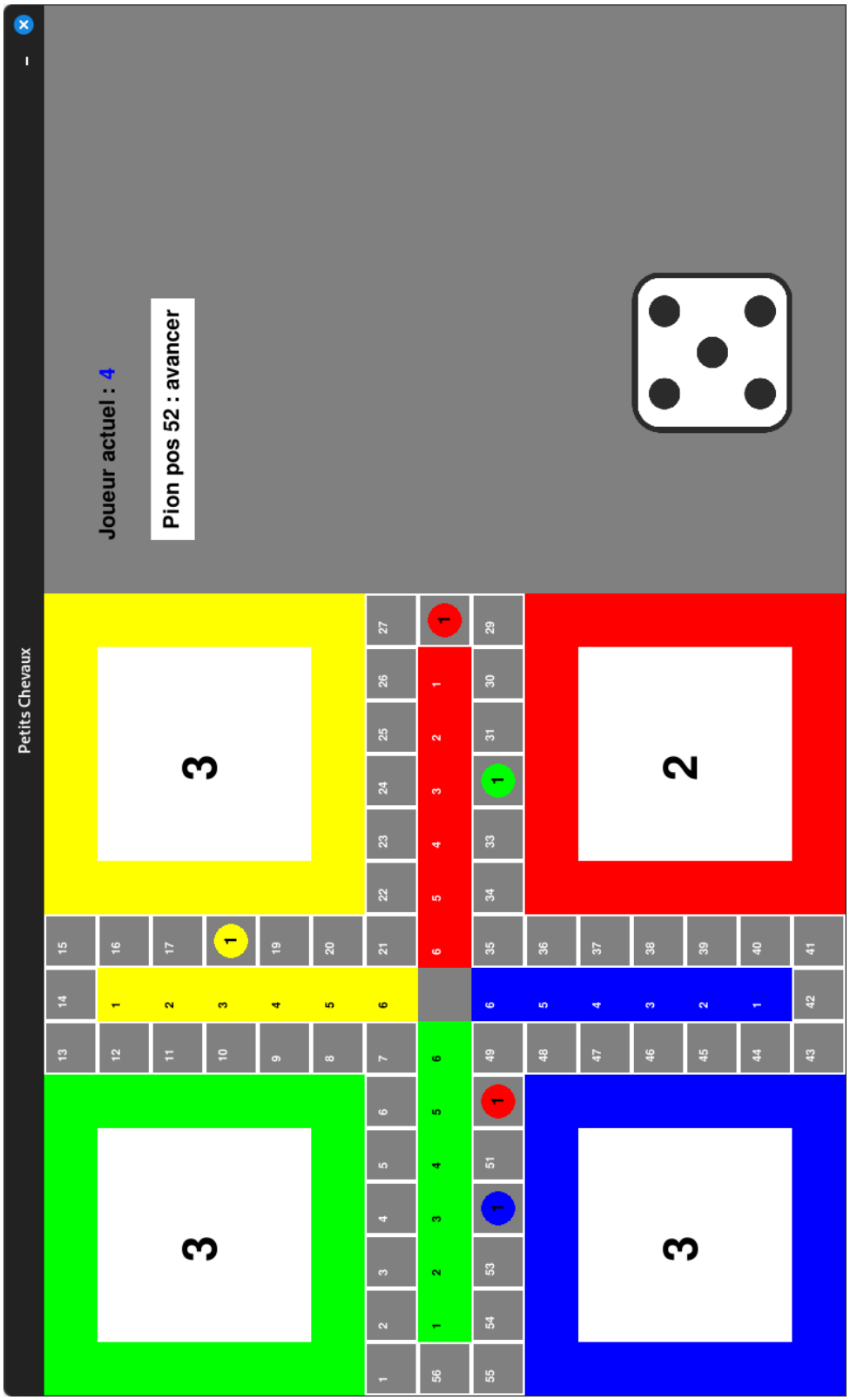

Figure 12 – Interface du jeu "Petits Chevaux".

Interface du jeu "Labyrinthe"

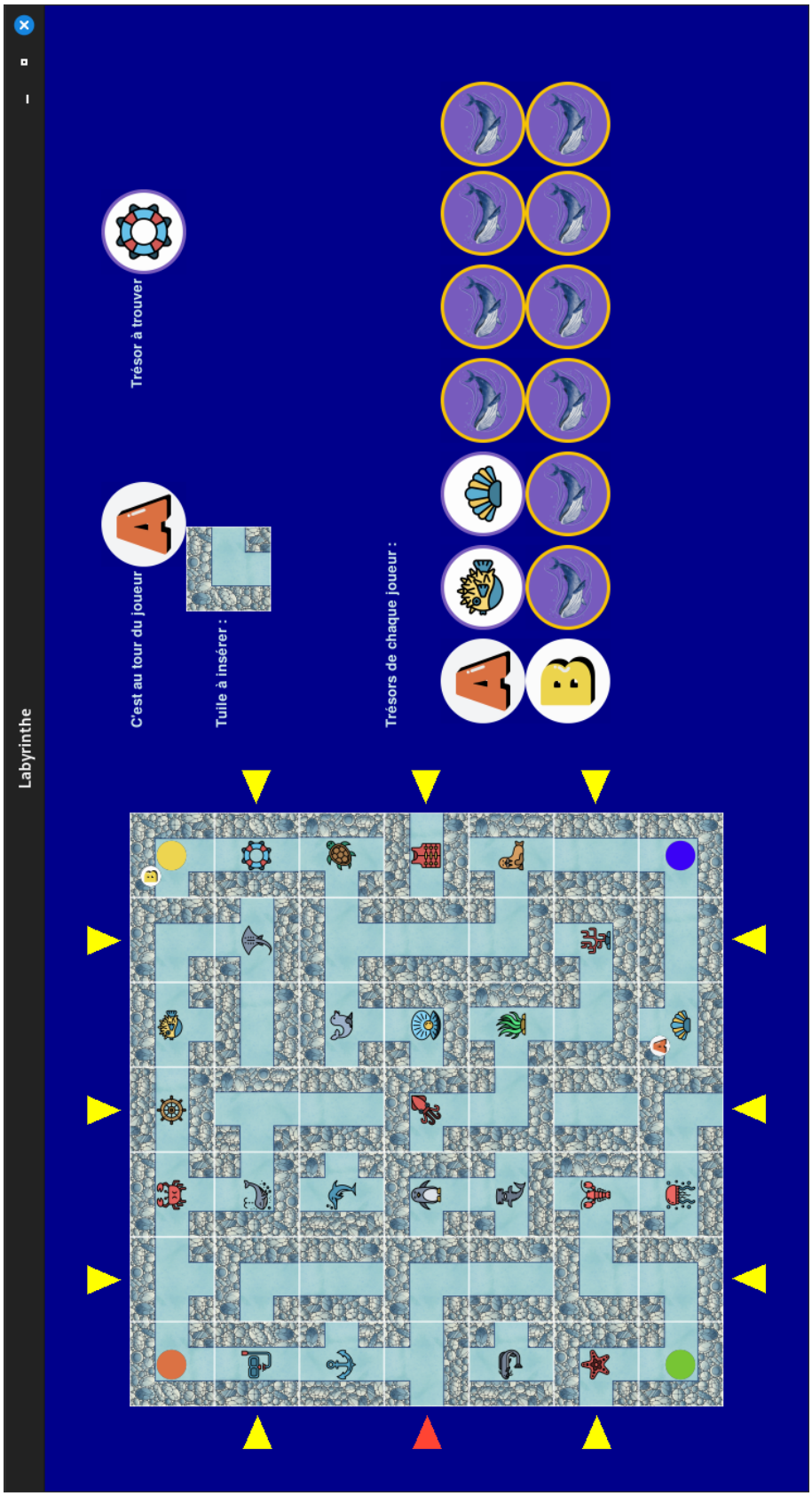

Figure 13 – Interface du jeu "Labyrinthe" avec le thème "kity".