UFR de mathématique et d'informatique
Université de Strasbourg

Projet de Master

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

---

Rapport intermédiaire

Étudiants :
Charlotte Kruzic
Daniil Kudriashov
Zoé Marquis
Ekaterina Zaitceva

3 2024 .

1 Introduction

1.1 Contexte et motivation

Les jeux de plateau offrent un environnement particulièrement adapté pour l'entraînement d'agents utilisant l'apprentissage par renforcement (*Reinforcement Learning*, RL). Grâce à la nature structurelle de ces jeux, incluant des règles clairement définies, des états discrets, et des interactions complexes, les jeux de plateau permettent d'explorer des stratégies d'optimisation et des comportements adaptatifs. Dans ce contexte, le RL peut être utilisé pour simuler des agents capables d'interagir avec des environnements variés, tout en prenant en compte des scénarios multiples et des choix stratégiques en fonction de différents types de joueurs. En plus de fournir une plateforme pour tester et adapter des algorithmes de RL, les jeux de plateau peuvent aussi être améliorés par des règles supplémentaires créant de nouvelles dynamiques de jeu, permettant d'étudier l'impact de telles modifications sur les stratégies des agents.

1.2 Objectif du projet

L'objectif principal de ce projet est de développer des agents autonomes capables de jouer à des jeux de plateau de manière performante et optimisée. Concrètement, ce projet vise à :

- Entraîner des agents utilisant le RL pour maîtriser des jeux de plateau et simuler des parties complexes ;
- Créer des agents avec des comportements variés, permettant de moduler la difficulté et de proposer divers styles de jeu ;
- Introduire de nouvelles règles et variantes, adaptées et testées par les agents, pour optimiser l'équilibre et la jouabilité des jeux de plateau ;
- Analyser les statistiques issues des parties simulées pour identifier des pistes d'amélioration dans les règles ou les mécaniques de jeu.

Ces agents serviront ainsi de base pour tester des stratégies variées, explorer des mécaniques de jeu alternatives, et générer des retours statistiques exploitables, tout en permettant aux humains de jouer contre ces agents, ces "IA".

1.3 Présentation du jeu sélectionné

1.3.1 Contexte et caractéristiques

Le jeu *Labyrinthe* est un classique des jeux de société conçu pour 2 à 4 joueurs, dans lequel les participants cherchent des trésors dissimulés dans un labyrinthe en constante transformation. Avec un plateau composé de plaques fixes et mobiles, les joueurs modifient continuellement la structure des couloirs pour atteindre leurs objectifs, tout en essayant de bloquer les autres ou de déjouer leurs plans. Les parties sont relativement courtes (environ 30 minutes), ce qui permet d'observer plusieurs stratégies dans un laps de temps limité.

1.3.2 Déroulement d'une partie

Chaque joueur a pour objectif de localiser des trésors spécifiques, révélés par des cartes qu'ils découvrent une par une. Le tour d'un joueur comporte deux étapes :

1. Modification des couloirs : Le joueur insère une plaque supplémentaire, avec l'orientation désirée, sur le bord du plateau, décalant ainsi une rangée ou une colonne pour ouvrir ou fermer des chemins.

2. Déplacement : Ensuite, le joueur peut se déplacer vers le trésor visible sur sa carte ou rester en position pour optimiser son mouvement au prochain tour.

Les trésors et la configuration changeante du labyrinthe obligent chaque joueur à s'adapter rapidement aux nouvelles configurations. Le vainqueur est le premier à avoir collecté tous ses trésors et à être retourné à sa case de départ.

1.3.3 Choix du labyrinthe comme environnement d'entraînement pour l'apprentissage par renforcement

Le *Labyrinthe* est particulièrement intéressant pour l'entraînement d'agents en apprentissage par renforcement en raison de plusieurs facteurs :

1. Complexité spatiale et variabilité : Les configurations changeantes du plateau offrent un environnement dynamique et imprévisible. Cette complexité spatiale exige des agents qu'ils planifient non seulement leurs mouvements, mais aussi qu'ils anticipent les modifications possibles du labyrinthe pour atteindre leurs objectifs de manière optimale.

2. Adaptabilité des stratégies : Les agents doivent adapter leurs stratégies en fonction des actions des autres joueurs, développant des comportements tels que l'anticipation, l'opportunisme ou le blocage, ce qui encourage un large éventail de comportements, de l'exploration à la collaboration temporaire ou à la compétition.

3. Interactions sociales et multi-agents : Ce jeu permet l'implémentation d'agents au comportement collaboratif, ce qui est idéal pour tester et développer des stratégies dans un contexte multi-agents, simulant des interactions proches de celles observées dans des environnements complexes réels.

4. Diversité des actions et des récompenses : Avec des actions variées telles que la manipulation du plateau et les déplacements, chaque agent peut être optimisé pour différents objectifs et types de récompense (comme le blocage, la découverte, l'adaptation rapide, etc.). Cela permet une analyse approfondie des choix d'actions et de leurs impacts.

2 Description des méthodes et des techniques utilisées

2.1 Environnement et Bibliothèques

Gymnasium

Pourquoi Gymnasium ? Gymnasium, une version améliorée et maintenue d'OpenAI Gym, est un cadre couramment utilisé pour la création et l'interaction avec des environnements d'apprentissage par renforcement. Le choix de Gymnasium se justifie pour ce projet par plusieurs facteurs :

1. Simplicité et flexibilité : Gymnasium propose une interface standardisée pour les environnements RL, facilitant ainsi l'intégration avec des algorithmes d'apprentissage par renforcement. Son utilisation permettra de créer rapidement un environnement de *Labyrinthe* simple et de tester les algorithmes sans configurations excessives.

2. Compatibilité avec des algorithmes RL bien établis : Les bibliothèques telles que Stable-Baselines3 s'intègrent naturellement avec Gymnasium, simplifiant ainsi la mise en œuvre des algorithmes et la gestion des sessions d'entraînement. L'objectif de commencer avec un environnement plus simple peut être facilement atteint avec cette bibliothèque.

3. Support de communauté et documentation : Gymnasium bénéficie d'une grande communauté et de documentations exhaustives, ce qui accélère le développement et la résolution de problèmes. Pour un projet académique ou d'exploration, cet aspect est essentiel pour faire évoluer le code sans investir trop de temps en débogage.

4. Extension future vers des environnements multi-agents (PettingZoo) : Si le projet progresse vers des configurations multi-agents, Gymnasium peut être étendu avec PettingZoo, un cadre spécifiquement conçu pour les environnements multi-agents. Cela permet de maintenir une certaine continuité et de faciliter la transition entre un environnement mono-agent et multi-agents sans reconstruire tout le système.

En somme, Gymnasium est un excellent point de départ pour tester rapidement le jeu *Labyrinthe* en mode mono-agent, avec la possibilité d'intégrer des complexités multi-agents plus tard grâce à PettingZoo. Ce choix initial simplifie le prototypage et ouvre la voie à une extension future.

2.2 Code du labyrinthe utilisé

Le projet *Labyrinthe des IUT* est une version numérique du jeu de plateau classique *Labyrinthe*, entièrement conçu en Python et accessible sur GitHub. Dans cette version interactive, les utilisateurs ont la possibilité d'explorer le labyrinthe à la recherche de trésors, tout en se confrontant à des "IA"(terme utilisé par le développeur du projet). Ces "intelligences artificielles"adoptent des comportements simples et non adaptatifs : en mode offensif, elles choisissent le chemin le plus direct vers leur objectif, tandis qu'en mode défensif, elles s'efforcent de bloquer leurs adversaires en s'appuyant sur une heuristique de jeu, sans recourir à des techniques d'apprentissage par renforcement.

Le jeu offre trois écrans principaux permettant de gérer les différentes étapes et options de la partie :

1. Écran de Menu : Le menu d'accueil propose trois choix : Jouer, Options, ou Quitter 1. Cette interface simplifiée permet une prise en main rapide du jeu.

2. Écran d'Options : L'écran des options 2 offre des réglages de partie, tels que :

   - Le nombre de joueurs, de 1 à 3
   - Le nombre d'intelligences artificielles
   - Le nombre de trésors total, configurable entre 12 et 34
   - Le nombre de trésors par joueur

   Ces paramètres permettent de personnaliser le niveau de difficulté et l'expérience de jeu.

3. Écran de Jeu 3 :

   - Le plateau de jeu avec les positions des trésors et des joueurs
   - La prochaine pièce à insérer
   - Le joueur en cours

- Le nombre de trésors restants à atteindre pour chaque joueur

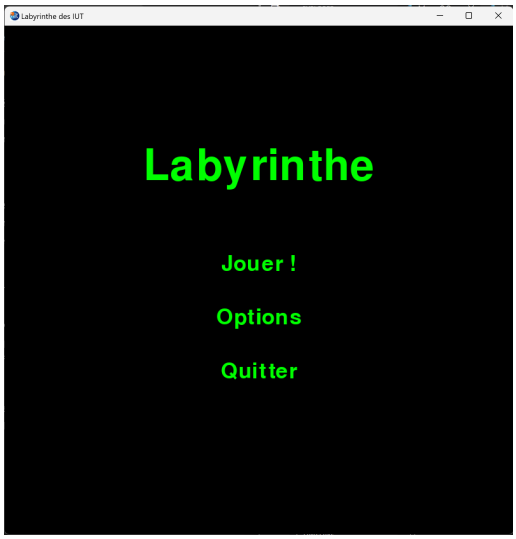
En plus de la version graphique pour une interface plus immersive, une version console est également disponible pour les utilisateurs préférant une interface en ligne de commande 4.

Le jeu est développé en Python et utilise la bibliothèque Pygame pour gérer l'affichage en 2D ainsi que les interactions avec les utilisateurs. Pygame permet une gestion efficace des images et des éléments visuels. Nous avons intégré des thèmes personnalisables, permettant aux utilisateurs de sélectionner les styles visuels des tuiles et des trésors.

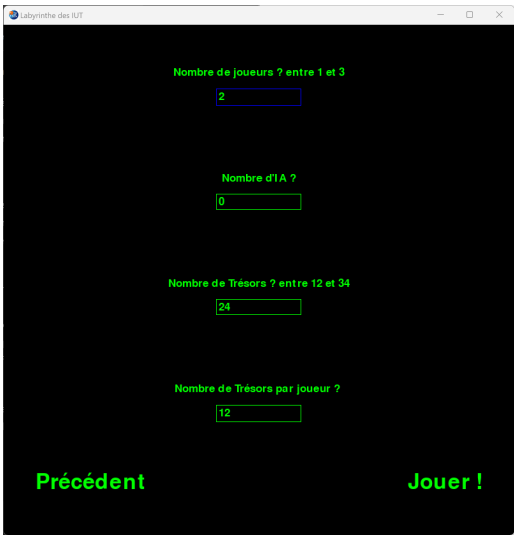
Nos améliorations

En plus d'un nettoyage global du code pour éliminer les mauvaises pratiques et instaurer des standards de codage, plusieurs modifications fonctionnelles et esthétiques ont été réalisées :

- L'interface a été simplifiée avec la suppression du menu, permettant un accès plus direct aux options de jeu en ligne de commande.
- Le nombre de trésors est désormais fixé à 24, répartis uniformément avec 6 trésors par joueur, quelle que soit la configuration du jeu. Par exemple, même avec seulement deux joueurs, les 24 trésors restent visibles sur le plateau. Contrairement à la version initiale du Labyrinthe des IUT, les trésors ne disparaissent pas lorsqu'ils sont trouvés, permettant aux joueurs (humains comme IA) de se baser sur leur mémoire pour optimiser leur parcours.
- Les pièces fixes sur le plateau respectent fidèlement l'original, tant dans leur forme que dans leur orientation et leurs symboles. Cela assure une cohérence visuelle et un sentiment de réalisme pour les habitués du jeu physique. Les autres pièces mobiles respectent également la distribution des formes : coudes, lignes droites et intersections en "T". Toutefois, les trésors y sont placés aléatoirement, ce qui apporte une touche d'aléatoire, s'éloignant légèrement de la disposition originale.
- Le *Labyrinthe des IUT* ne comportait pas la règle de retour à la base pour gagner après avoir récupéré tous les trésors.
- L'interface propose deux thèmes : une version fidèle à l'esthétique du jeu d'origine 5, et un thème "océan" 6 avec la mascotte "Кит"(qui signifie "baleine"en russe). Les messages de contrôle et les informations de jeu sont redessinés et alignés sur le côté droit de l'écran, dans un style de cartes rappelant les éléments du plateau, offrant ainsi une expérience plus immersive.

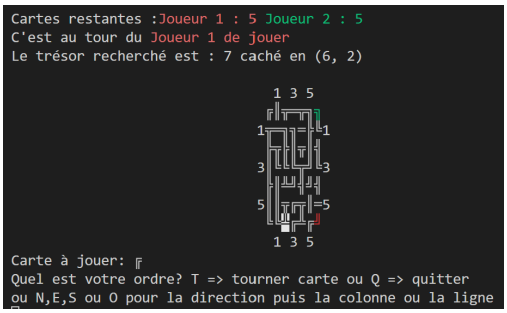


. 1: Capture d'écran du menu du jeu



. 2: Capture d'écran de la fenêtre d'options

. 3: Capture d'écran de la fenêtre de jeu



. 4: Capture d'écran du jeu (version console)

. 5: Capture d'écran de la fenêtre de jeu thème original

. 6: Capture d'écran de la fenêtre de jeu thème kity

3 Résumé de la progression du développement

1. **Reformulation et appropriation du sujet** :
    - Une première phase où nous avons pris le temps de reformuler le sujet et de nous l'approprier pleinement, en comprenant les objectifs spécifiques et les attentes du projet.
2. **Étude et sélection du jeu** :
    - Nous avons exploré divers jeux tels que *Blokus*, *Carcassonne*, et *Petits Chevaux* pour en évaluer les potentialités en termes d'interactions, de règles et de développement d'agents intelligents. Finalement, le Labyrinthe a été retenu pour sa complexité stratégique et ses règles adaptables.
3. **Mise en place de l'environnement de travail** :
    - L'environnement de travail a été structuré en intégrant toutes les applications nécessaires, les dépôts de code, et les supports de communication. Notion a été adopté pour stocker et organiser les documents, facilitant ainsi leur exportation vers des formats compatibles avec GitLab. Cette organisation permet un suivi clair et une centralisation des informations pour l'équipe, favorisant la productivité et la collaboration.
4. **Étude des algorithmes et outils de renforcement** :
    - Analyse des algorithmes d'apprentissage par renforcement adaptés à notre jeu, ainsi que des librairies pertinentes pour implémenter les agents intelligents.
5. **Exploration du jeu** :
    - Nous avons joué à des parties du *Labyrinthe*, tant en groupe qu'individuellement, pour analyser les stratégies, les règles et les interactions possibles entre les agents. Cette expérience nous a aidés à identifier les défis et les comportements attendus. Nous avons également joué à *Labyrinthe des IUT* afin de repérer les problèmes potentiels.
6. **Définition des agents et règles de jeu** :
    - Identification des différents types d'agents possibles et des récompenses (*rewards*) qui pourraient les motiver dans le jeu.
    - Création de nouvelles règles avec l'introduction d'éléments aléatoires et de variations de règles.
7. **Recherche de code existant** :
    - Nous avons exploré des implémentations existantes du jeu en Python afin de gagner du temps dans le développement. Bien que peu de versions en Python aient été trouvées, nous avons identifié et évalué la version développée *Labyrinthe des IUT*. Nous avons également découvert une version mobile en Java, mais notre choix s'est porté sur Python pour pouvoir utiliser des bibliothèques telles que Gymnasium et Petting Zoo.
8. **Documentation des avantages et limites du jeu** :

- Nous avons réalisé une analyse approfondie des fonctionnalités de la version actuelle du jeu, mettant en évidence ses atouts et ses faiblesses. Cette étude a permis d'établir une liste de tâches nécessaires pour améliorer et déboguer la version initiale, tout en planifiant les actions à venir.

9. **Étude de jeux similaires avec plusieurs joueurs en RL** :
    - Exploration d'autres jeux avec plusieurs joueurs et des comportements variés, afin de nous inspirer des mécanismes d'interaction et de compétition.
10. **Amélioration de l'interface utilisateur (UI/UX)** :
    - Travail sur le design des tuiles, cartes, et pions pour une meilleure lisibilité et attractivité visuelle.
    - Modifications de l'interface graphique pour une expérience utilisateur plus fluide.
11. **Nettoyage et amélioration du code** :
    - Adaptation et débogage de la version du jeu trouvée afin de corriger les erreurs, d'optimiser les performances, et de préparer le jeu pour l'intégration d'agents intelligents.
    - Correction d'oublis et ajustement concernant certaines règles.
12. **Mise en place de l'environnement Gymnasium** :
    - Intégration de l'environnement Gymnasium pour structurer le cadre d'entraînement des agents. Plusieurs configurations sont actuellement en test, impliquant une ou deux phases par tour pour chaque joueur. L'environnement est encore en évolution et des ajustements peuvent être apportés.
13. **Développement d'un premier notebook d'entraînement** :
    - Création d'un notebook pour initier l'entraînement des agents, où nous pouvons monitorer les performances et ajuster les paramètres d'apprentissage par la suite.

4 État actuel du projet

Le projet a atteint des points d'avancement importants :

- Le jeu est désormais pleinement fonctionnel avec une interface visuelle, permettant à des joueurs humains de participer.
- Une base de données a été créée pour le stockage des informations de jeu, des états des agents et des statistiques de performance.
- L'environnement Gymnasium a été configuré, offrant ainsi un cadre pour entraîner et faire jouer des agents.

Limitations rencontrées La gestion de la grande variété d'actions disponibles pour les agents dans l'environnement a présenté un défi : 4 directions de rotation, 12 options d'insertion et 49 cases potentielles de déplacement. Ce vaste éventail d'options a généré des bugs, ce qui a rendu nécessaire une simplification par la séparation des actions en deux phases (MultiDiscrete).

- Phase 1 : insertion de la tuile
- Phase 2 : déplacement de l'agent

Par ailleurs, toutes les cases et directions ne sont pas pertinentes : seules deux rotations sont possibles pour la tuile de forme « tout droit », 11 options d'insertion (annuler le dernier coup étant interdit) et moins de 49 cases sont réellement atteignables.
À présent, l'environnement est fonctionnel et peut être testé via le script `./main_env.py`, qui lance une partie avec deux IA s'affrontant avec des actions aléatoires. Une interface graphique permet de visualiser les mouvements des agents en temps réel. Les agents gèrent désormais l'exploration et l'exploitation de manière progressive, en augmentant l'exploitation en fin de partie.

Agents et entraînement Les agents ont été entraînés sur 150 000 étapes avec enregistrement du modèle tous les 2 500 pas. Utilisant l'algorithme PPO (*Proximal Policy Optimization*), cet entraînement a démontré de bons résultats, exploitant des actions MultiDiscretes. Les performances des agents sont analysées grâce à TensorBoard, qui permet de suivre l'évolution des pertes d'entropie, des récompenses moyennes et des longueurs moyennes des épisodes, bien que certaines métriques restent perfectibles.

État actuel du jeu Les agents peuvent désormais jouer contre des joueurs humains, les tours sont bien synchronisés, et le labyrinthe est partagé et généré à partir de l'environnement. Les classes du jeu ont été modifiées pour permettre cette interaction IA-joueurs.

5 Prochaines étapes et développement futur

1. Amélioration de l'environnement
    - Optimiser les récompenses et pénalités pour guider les agents plus efficacement, en ajustant les sanctions pour des comportements comme la répétition excessive d'insertion de tuiles au même endroit ou des déplacements redondants.
    - Affiner le format des actions pour des mouvements plus naturels en passant de cellule en cellule, comme les joueurs humains.
    - Adapter l'environnement pour prendre en charge divers types d'agents et envisager des scénarios multi-agents à terme.
    - Revoir la logique d'exploration/exploitation pour obtenir de meilleures performances.
2. Extensions du code et du suivi
    - Intégrer des agents avec des modèles différents pour comparer leurs stratégies et performances.
    - Ajouter des fonctions de suivi dans TensorBoard pour des métriques comme la position, la validité des actions et les performances générales de chaque agent.
3. Base de données et compatibilité

- Synchroniser le schéma de la base de données avec les besoins de l'environnement et des statistiques du jeu pour s'assurer que les données nécessaires sont bien collectées. Cela permettra des analyses plus complètes, comme l'évaluation des stratégies et l'identification des meilleures règles.

4. Recherche approfondie et stratégie des agents

    - Étoffer l'état de l'art pour situer notre travail dans les recherches actuelles et orienter les choix stratégiques d'entraînement des agents, ainsi que les règles et statistiques envisagées pour leur évaluation finale.

Point bloquant principal Actuellement, les actions des agents diffèrent de celles attendues par le jeu dans leur format, ce qui entraîne des bugs. La conversion est fonctionnelle pour l'insertion des tuiles mais reste à ajuster pour les déplacements.