LuxAI: Deep Reinforcement Learning pour un jeu de stratégie

Hadrien Crassous Pierre Jourdin
Aymeric Conti
Université Paris Saclay, CentraleSupélec, Gif-sur-Yvette, France
{hadrien.crassous, pierre.jourdin, aymeric.conti}@student-cs.fr

Abstract

La saison 3 du challenge Lux AI est une compétition présentée à NeurIPS 2024, dans laquelle les participants conçoivent des agents pour résoudre un problème d'optimisation, de contrôle d'unités et de collecte de ressources dans un scénario un-contre-un. En plus de l'exploitation des ressources, les agents doivent être capables d'analyser leurs adversaires et de les attaquer.

Notre approche est basée sur l'apprentissage par renforcement profond. Nous utilisons une architecture Pixel-To-Pixel pour le contrôle des agents et nous apprenons le modèle grâce à l'algorithme de Proximal Policy Optimization (PPO).

1. Introduction

La série LuxAI est une série de compétitions de jeux de stratégie multi-agents, hébergée par Kaggle et présentée à NeurIPS. Les jeux de stratégies multi-agents ont de nombreuses applications dans des cas réels en planification et en optimisation. Les deux grandes classes de solutions sont les solutions "rule-based", qui consiste à définir un agent basés sur des règles, et l'apprentissage par renforcement.

Nous avons participé à la troisième saison [9], qui a débutée en décembre 2024 et se clôt le 10 mars 2025. Nous présentons ici notre approche par apprentissage par renforcement profond, que nous avons confronté à des solutions 'rule-based'.

2. Présentation du jeu

Dans cette section, nous présentons d'abord les raisons pour lesquels ce jeu est difficile, puis nous présentons en détail les règles du jeu.

2.1. Principaux défis

Le jeu est un jeu de stratégie à grande échelle, complexe et dynamique. Maitriser ce jeu comporte plusieurs défis:

- **Exploration de la carte**: le joueur doit systématiquement trouver les ressources cachées de la carte.
- **Brouillard de guerre**: La vision d'un joueur est réduite a l'ensemble de petits voisinages autour de chacune des unités.
- **Collaboration des unités**: Pour découvrir des points de reliques, les unités doivent se coordonner.
- **Attaque**: Les meilleurs joueurs se départagent souvent par leurs qualités de déplacement et d'attaque.
- **Robustesse à l'aléatoire**: Chaque partie se déroule dans une carte différente et qui évolue dynamiquement pendant la partie, avec des paramètres différents.
- **Changement de la taille de l'équipe**: Le nombre de vaisseaux augmente au cours de la partie (plafonné à 16) Cette contrainte empêche de traiter le problème avec un agent qui aurait toujours 16 actions à prédire.

2.2. Regles du jeu

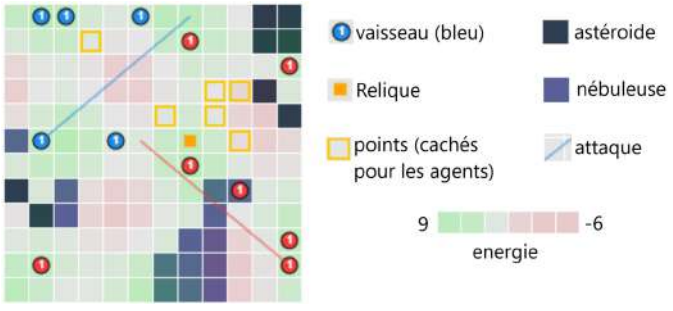

Figure 1. Extrait d'une partie. La vraie grille est de taille 24×24

Déroulement d'une partie. Dans Lux AI Season 3, deux équipes s'affrontent sur une carte en 2D de 24x24 cases. Les joueurs contrôlent des unités qui doivent explorer la carte et collecter des ressources L'objectif est de maximiser les points tout en empêchant l'adversaire

d'en obtenir. Les unités du joueur 1 apparaissent dans le coin Nord-Ouest et les unités du joueur 2 au coin opposé. Chaque joueur commence avec un vaisseau, et son équipe s'agrandit de un tous les 3 tours, plafonnée à 16 vaisseaux. Un aperçu d'une situation de jeu est décrite dans la Figure 1

Unités. Les unités sont des vaisseaux spatiaux pouvant à chaque tour se déplacer (haut, bas, gauche, droite, ou rester sur place) ou bien un tir pour drainer l'énergie des unités adverses situées sur une case cible et ses alentours. Chaque unité possède une réserve d'énergie utilisée pour les déplacements et les actions. Une unité qui tombe à 0 énergie est retirée du jeu.

Carte. Les 2 joueurs évoluent sur la même carte, symétrique et générée aléatoirement. On y trouve des ressources (reliques), un champ d'énergie et des obstacles. Les reliques sont fixes tandis que le champ d'énergie et les obstacles se déplacent au cours de la partie.

Ressources. Les reliques sonts les sources de points, mais seules certaines cases adjacentes (cachées) rapportent des points lorsqu'une unité alliée s'y trouve.

Champ d'énergie. Chaque case est plus ou moins hostile, selon son niveau d'énergie: un case d'énergie positive régénère l'énergie d'un vaisseau, tandis qu'une case d'énergie négative la fait baisser.

Obstacles. La carte contient des obstacles: les astéroïdes sont des obstacles infranchissables, tandis que les nébuleuses sont des cases traversables qui réduisent la vision des unités et leur faire perdre de l'énergie.

Conditions de victoire. Les joueurs s'affrontent au cours d'une série de 5 manches de 100 tours. Une manche est remportée en ayant plus de points de reliques que l'adversaire. L'équipe gagnante est celle qui remporte le plus de manches parmi les 5.

2.3. Pourquoi du Deep Learning ?

Le jeu est complexe, rendant les stratégies basées sur des règles rapidement inefficaces. Nous utilisons donc le deep learning pour exploiter au mieux les informations pertinentes de l'état de la partie à chaque tour. De plus, les approches à base de deep learning semblent être les plus performantes pour ce type de jeux, comme le suggère [2], qui utilise du deep learning pour battre le vainqueur de Lux AI S1 (jeu assez similaire au nôtre).

3. Etat de l'art

3.1. Proximal Policy Optimization

L'algorithme de Proximal Policy Optimization [8] (PPO) est un algorithme de type Actor-Critic pour l'apprentissage profond.

Les algorithmes Actor-Critic utilisent une policy paramétré $\pi_\theta : S \to \Delta(\mathcal{A})$ (qui associe à un état s une distribution de probabilité d'actions) et un critique paramétrée $V_\theta : S \to \mathbb{R}$ (qui estime la valeur d'un état s). De tels algorithmes mettent à jour les paramètres en maximisant la récompense espérée $J(\theta) = \mathbb{E}_{s,a}[R]$ à l'aide du policy gradient :

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a}[\nabla_\theta \log \pi_\theta(a|s) A^\theta(s_t, a_t)]$$

où $A^\theta(s, a)$ est l'avantage de prendre l'action a à l'état s. L'avantage est estimé avec la trajectoire $\{(s_r, a_r)\}_{r \geq t}$. Il existe plusieurs manières d'estimer $A(s, a)$ et de collecter des trajectoires, ce qui donne lieu à différentes variantes des algorithmes Actor-Critic, les plus influentes sont PPO, IMPALA et A3C. La spécificité de PPO est l'introduction d'un clipping dans le calcul de l'avantage pour éviter une update trop grande. PPO constitue l'état de l'art du "model-free RL".

PPO est classiquement associé à la technique de General Advantage Estimation [7] (y compris dans l'article original présentant PPO):

$$\hat{A}_t^{\theta,GAE}(\gamma, \lambda) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^\theta \tag{1}$$

avec $\delta_t^\theta = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$ et r_t la récompense (reward).

L'introduction de GAE permet de mieux traiter le problème du "temporal credit assignment", sans lequel l'algorithme comprendrait moins bien les dépendences à long terme entre actions et récompenses.

3.2. Architecture Pixel-To-Pixel pour le MARL

Plusieurs approches sont dispoinbles pour le MARL (Multi-Agent Reinforcement Learning).

Le MARL décentralisé repose sur une approche où chaque agent de l'équipe prend ses décisions de manière indépendante, en fonction de sa propre observation et sans accès direct aux informations des autres agents. Cette approche est notamment utilisée par [6] pour le Multi-Agent PathFinding. Ceci permet d'avoir une équipe de taille flexible, mais également de traiter rapidement des cartes très grandes (jusqu'à $160\times$ dans l'article). D'après [3], le MARL décentralisé pose souvent problème pour le "credit assignment", c'est-à-dire que les agents indépendants apprenent difficilement avec le signal de reward commun à toute l'équipe.

L'article [3] propose une architecture Pixel-To-Pixel centralisée comme alternative au MARL décentralisé. Les auteurs utilisent un réseau qui prédit une action par pixel de la carte, représentée en une grille. Cette approche s'inspire de la segmentation d'image, où chaque pixel reçoit une étiquette spécifique. Chaque cellule de la grille prédit une action, et si un agent occupe une cellule, il est contrôlé selon l'action correspondante. Cette approche permet de traiter

de façon très naturelle la contrainte d'avoir une équipe de taille flexible. De plus, l'utilisation de réseaux convolutifs permet également d'incorporer un partage des paramètres entre les agents.

L'article [3] met en pratique cette approche sur les benchmark de jeux de stratégies StarCraft et Mixed army battle (MAB), tandis que l'article [2] l'applique pour la saison 1 de LuxAI. Les auteurs utilisent en entrée la grille $N \times N$, décomposée en plusieurs channels, pour prédire les actions sous la forme d'une grille $N \times N$, en utilisant des ResNet. Nous nous plaçons dans ce même cas, avec une grille de taille 24×24, et notre architecture est fortement inspirée de cet article.

3.3. Curriculum Learning

Le jeu LuxAI est complexe et la récompense que l'on donne à l'agent doit le guider pour qu'ils développent des compétences à bas niveau (exploration, tir, exploitation des ressources), et la récompense épisodique (+1 en cas de victoire -1 en cas de défaite) ne permet pas de comprendre facilement ce genre de patterns.

L'article [2] propose d'utiliser le curriculum learning [1] pour séparer l'entraînement en plusieurs phases : La première phase est bas niveau, avec une reward précise (exploration, récolte d'énergie...), donnée après chaque action (reward dite dense), pour que l'agent apprenne les bases. La dernière phase correspond à la reward "officielle" : +1 en cas de victoire -1 en cas de défaite à la fin de la manche.

4. Methodes mises en oeuvre

Dans cette section, nous décrivons les méthodes que nous avons mis en place pour définir et apprendre un réseau profond qui joue au jeu LuxAI.

4.1. Traitement des features

Features visuelles On fournit au réseau une représentation de la grille en plusieurs channels. Ces features sont sélectionnées en décomposant la grille pour extraire les caractéristiques de la carte (astéroïdes, reliques, positions des alliés et ennemis, etc). La liste est détaillé en annexe A.

Features non visuelles On fournit au réseau des données scalaires globales, qui ne sont pas relatives à la grille. Les paramètres de la partie (voir Annexe A), les points du joueur et de son adversaire, le nombre de reliques trouvées, et le temps (la manche actuelle et l'étape actuelle).

Pour le temps, on arrondit par 5 l'étape actuelle (e.g. l'étape 7 devient l'étape 5, l'étape 2 devient l'étape 0, ...), et on one-hot encode l'étape actuelle et la manche actuelle, comme proposé par l'article [2].

Ces features non-visuelles sont concaténées avec les features visuelles grâce à un expansion (on répète la valeur scalaire sur toute la grille 24×24).

4.2. Autres traitements

Mémoire Puisque la vision du joueur est limitée, il se peut qu'il oublie les reliques qu'il a rencontré. Nous avons donc ajouté un mécanisme indépendant du réseau de neurone qui prend en charge la mémoire de la position des reliques. Nous avons également implémenté une mémoire des points cachés, basés sur le fait que on peut deviner quelles cases donnent des points en fonction des points que l'on vient de gagner.

Action Space Le réseau observe l'état de la partie s et renvoie une action par vaisseau actif. Une action est alors un choix parmi 6: se déplacer vers le haut, le bas, la droite, la gauche, ne pas bouger, et tirer vers l'ennemi. Nous avons fait une hypothèse simplificatrice consistant à choisir la cible de l'attaque parmi un ennemi proche arbitraire, au lieu de prédire la cible.

Environment Sampling Nous avons décidé de modifier la génération de la carte pour s'entraîner sur des cartes plus souvent sur des cartes difficiles. Pour cela, nous avons légèrement biaisé le placement des ressources pour qu'elles apparaissent plus proche de la frontière (la diagonale qui est l'axe de symétrie de la carte) au lieu de les placer uniformément. Cette idée nous vient de l'article [6], dans lequel la densité de la grille générée favorise les grilles les plus difficiles.

4.3. Architecture

L'architecture du réseau Actor-Critic est un Residual Net [4]. Le réseau Actor-Critic prend en entrée les features visuelles et scalaires de l'état s de la partie, et renvoie

- un scalaire $V_\theta(s)$, représentant la valeur de l'état
- $\pi_\theta(s)$ les probabilités d'action des 16 vaisseaux.
- $I_\theta(s)$, une prédiction d'où se trouve les cases de points sue la carte, sous la forme d'une matrice de coefficients réels entre 0 et 1.

Le bloc "Pos-Gather" désigne le fait que l'on récupère les valeurs des 'grid-wise logits' aux différentes positions des 16 vaisseaux. On applique ensuite un soft-max pour obtenir les probabilités.

Le Residual Net est composé de 6 blocs résiduels (resblock). Chaque resblock est consistué de 2 couches convolutives de noyau 5×5, une Squeze-And-Excite Layer et un shortcut (voir le diagramme en annexe B). Le réseau contient environ 1 million de paramètres apprenables.

Nous n'avons pas utilisé de BatchNorm dans ces blocks, mais une Spectral Normalization [5] comme proposé par [2]. Il s'agit d'une technique de régularisation pour les représentations visuelles qui a été proposé pour les GANs.

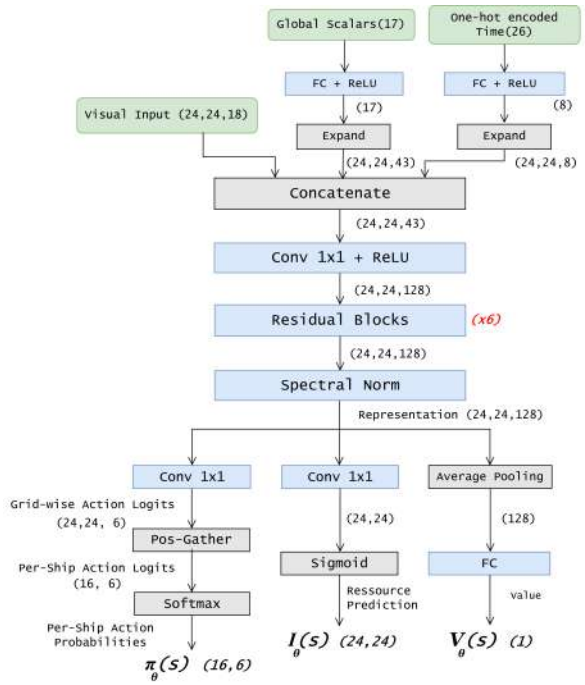

Figure 2. Notre architecture Pixel-To-Pixel, inspirée de [2]. En vert: les entrées. En bleu: blocs apprennables. ResBlock: voir annexe B

4.4. Algorithme d'apprentissage

Pour chaque état s, le réseau prédit une action jointe, composée 16 actions, une par vaisseau. On tire les 16 actions de façon indépendante en suivant $\pi_\theta(a_j|s)$. Pour calculer la probabilité de l'action jointe $\pi_\theta(a|s)$, on utilise le produit des 16 actions, en excluant les vaisseaux qui sont inactifs car n'ayant pas encore apparu ou à court d'énergie.

$$\pi_\theta(a|s) = \Pi_{\text{si vaisseau j actif}}\, \pi_\theta(a_j|s) \tag{2}$$

Actor Loss L'actor loss traduit l'objectif de maximiser la probabilité des actions les plus avantageuses. L'équation provient de PPO [8].

$$\mathcal{L}^{\theta_{\text{old}}}_{actor}(\theta) \doteq \sum_{s_t,a_t} \min\{\eta_\theta(a_t|s_t)A^{\theta_{\text{old}}}(s_t,a_t), \text{clip}\left(\eta_\theta(a_t|s_t), 1-\epsilon, 1+\epsilon\right) A^{\theta_{\text{old}}}(s_t,a_t)\} \tag{3}$$

avec $\eta_\theta(a|s) \doteq \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$

Value Loss La value loss traduit l'objectif de bien prédire la valeur de l'état actuel. L'équation provient de PPO [8].

$$\mathcal{L}_{\text{value}}(\theta) \doteq \sum_{s_t,a_t} (V_\theta(s_t) - V^{\text{target}}(s_t,a_t))^2 \tag{4}$$

avec $V^{\text{target}}(s_t,a_t) = V_{\theta_{\text{old}}}(s_t) + A_t^{\theta_{\text{old}}}$

Ressource Prediction Loss Nous avons introduit une Ressource Prediction Loss, qui traduit l'objectif de bien prédire la localisation des ressources sur la carte. Pour cela, on compare la prédiction $I_\theta(s_t)[x,y]$ avec la vraie répartition des ressources $I^{\text{true}}[x,y]$ qui vaut 1 si il y a une ressource en (x,y) sinon 0.

$$\mathcal{L}_{\text{pred}}(\theta) \doteq \sum_{s_t} \sum_{(x,y)} \text{BCE}(I_\theta(s_t)[x,y], I^{\text{true}}[x,y]) \tag{5}$$

avec BCE la binary cross-entropy loss:

$$\text{BCE}(p,y) = y\log(p) + (1-y)\log(1-p)$$

L'introduction de cet loss est inspirée de l'article [6], dans lequel les auteurs introduisent une prédiction de si l'agent bloque une de ses équipiers ou non. Cette prédiction est entraînée par la binary cross-entropy loss.

Combined Loss La loss que l'on utilise est une combinaison des 3 précédentes, avec l'entropie.

$$\mathcal{L}(\theta) \doteq \mathcal{L}^{\theta_{\text{old}}}_{actor}(\theta) + 0.5 \times \mathcal{L}_{\text{value}}(\theta) + 0.05 \times \mathcal{L}_{\text{pred}}(\theta) - 0.05 \times \mathcal{H}(\pi_\theta)$$

avec $\mathcal{H}(\pi_\theta)$ l'entropie de la distribution $\pi_\theta(s)$.

Pendant l'entraînement, on veut maximiser l'entropie pour que l'agent ne garde pas toujours le même comportement.

Algorithm 1 Algorithme PPO, avec N parties parallèles

Initialiser des poids θ_{old}
Créer N parties $G^{(1)}, G^{(2)}, \ldots, G^{(N)}$ à l'état initial
for iteration = 1, 2, … **do**
  for $t = k, k+1, \ldots, k+T$ **do**
    Observer les états des N parties $s_t^{(1)}, \ldots, s_t^{(N)}$
    En groupant les N états dans un forward, calculer pour chaque partie: $\pi_\theta(s_t^{(i)})$, $V_\theta(s_t^{(i)})$ et $I_\theta(s_t^{(i)})$
    Tirer l'action $a_t^{(i)} \sim \pi_\theta(s_t^{(i)})$
    Jouer les actions $a_t^{(i)}$
    Observer la reward sur chaque partie $r_t^{(i)}$.
    Si partie finie, la redémarrer et changer de seed.
  end for
  Pour chaque i, calculer les avantages grâce à l'équation 1, et normaliser les avantages.
  Optimiser $\mathcal{L}$ par rapport à θ, avec K époques et une taille de minibach $M \leq NT$.
  $\theta_{\text{old}} \leftarrow \theta$
end for

Lors des phases 2 et 3, on utilise $\gamma = 1$ car la récompense est épisodique. Le réseau est entrainé avec l'optimiseur AdamW, avec un taux d'apprentissage de 10^{-4}, et avec du gradient clipping de norme maximale 1.0. Nous avons fixé

Paramètre	Valeur
Nombre d'environnements N	64
Sequence Length T	64
Taille de minibatch M	512
Nombre d'époques K	4
Clipping Factor ϵ	0.2
γ	0.98
GAE λ	0.95

Table 1. Hyperparamètres utilisés

le temps d'entraînement à 10^8 timesteps, soit 2×10^5 parties joués, et l'entraînement a duré environ 40 heures avec un GPU NVIDIA RTX 3070 Ti et un processeur 12 coeurs. En comparaison, les auteurs de l'article [2] ont entrainé pendant 2×10^7 parties, soit 100 fois plus que nous.

4.5. Self-Play

Nous avons décidé d'entraîner le réseau par self-play, c'est-à-dire que le réseau joue contre des anciennes copies de ses poids. Cette technique est notamment employée par AlphaGo, et permetpermet d'apprendre à gagner contre des adversaires de plus en plus forts. Nous nous sommes inspirés de l'implémentation du self-play présente dans la librairie Unity ML-Agents unity: Nous enregistrons les poids par intervalle de 100 itérations. L'adversaire contre qui le réseauest tiré au sort parmi les 10 copies précédemment enregistrées. Cette fenêtre glissante de taille 10 permet d'assurer plus de continuité sur le niveau de l'adversaire.

4.6. Phases d'apprentissage

L'objectif des agents dans Lux est de collecter plus de points que l'adversaire, mais le résultat final ne fournit qu'une récompense épisodique, c'est à dire qu'il n'y a qu'un signal de reward à la fin de la partie ($\pm$ selon le résultat de la manche), ce qui donne lieu à un problème d'exploration difficile. Le reward shaping est une méthode courante pour guider l'agent en lui donnant des objectifs plus bas niveau (gagner des points, de l'énergie, attaquer, etc). On divise l'entrainement en 3 phases.

- Phase 1: Nous utilisons des récompenses attribués après chaque action. Elles encourangent les compétences de base: exploration, collecte de points, et attaque des ennemis. Différentes statistiques (ressources découvertes, points obtenus, nombre d'attaques) sont combinées, les poids choisis sont détaillés dans l'annexe A.

- Phase 2: Dans la phase 2, une récompense n'est donnée qu'à la fin d'une manche. Cependant, notre politique a encore besoin d'être guidée par un raisonnement à long terme. Nous modifions la récompense avec un léger signal sur la condition de victoire, c'est-à-dire $\pm\sqrt{|N_{self} - N_{op}|}$, où N_{self} et N_{op} dénotent le nombre de points de l'agent et de son adversaire. Cette reward incite à maximiser l'avantage en points.

- Phase 3: A la fin de la manche, l'agent recoit +1 pour une victoire et -1 pour une défaite.

4.7. Monitoring

Pour améliorer les hyperparamètres, nous avons examiné plusieurs métriques pendant l'entrainement.

Six métriques sont relatives aux loss: Actor Loss, Value Loss, Points Prediction Loss, Clip Fraction (fraction de clipping dans l'actor loss), et la variance expliquée.

Huit métriques sont relatives aux faits de jeux: Nombre de points gagnés, nombre de reliques découvertes, nombre de points découverts, nombre de cases visitées, nombre d'attaque tentées, nombre de morts, énergie accumulée, et nombre de collisions.

Un résumé de notre monitoring est disponible en annexe C. Les métriques de loss (Figure 7) convergent et les statistiques (Figure 8) illustrent un apprentissage clair des compétences de base.

5. Expériences

Dans cette section, nous présentons notre méthode pour estimer la progression de notre agent au cours du temps.

5.1. Protocole

Pour évaluer la progression de l'agent, nous enregistrons des copies des poids à intervalle réguliers durant l'entraînement, et on procède à un tournoi rassemblant ces différents agents, pour estimer leurs classement ELOs. On initialise leurs ELO à 1000, on joue au total 2000 matchs, et on met à jour les scores ELO après chaque match avec la règle suivante:

$$P_A = \frac{1}{1 + 10^{(E_B - E_A)/400}}, P_B = \frac{1}{1 + 10^{(E_A - E_B)/400}}$$

$$E'_A = E_A + K(S_A - P_A), E'_B = E_B + K(S_B - P_B)$$

où E_A et E_B sont les ELO des joueurs avant le match, E'_A et E'_B sont les ELO mis à jour après le match, S_A et S_B sont les scores obtenus (1 pour victoire, 0 pour défaite), $K = 50$ le facteur d'ajustement P_A et P_B représentent les probabilités de victoire estimées avant le match.

5.2. Baselines basées sur des regles

Nous utilisons trois baselines basée sur des règles, pour compléter le tournoi.

- **Agent Random**: L'agent Random joue aléatoirement.

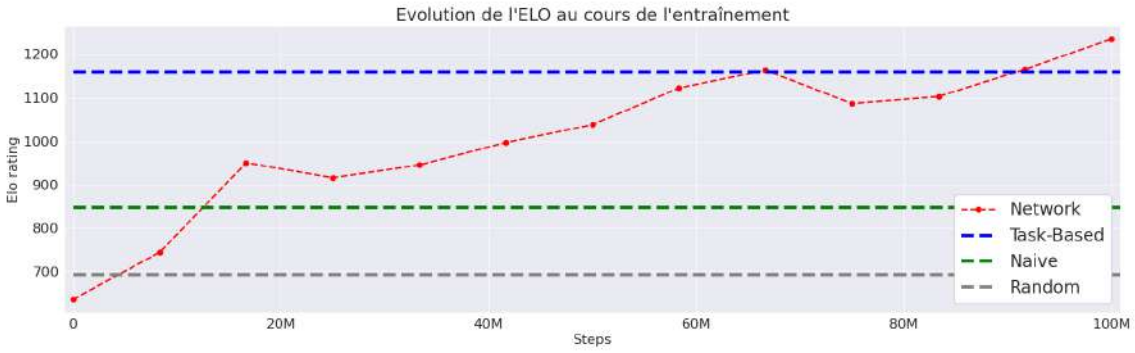

Figure 3. Evolution de l'ELO du modèle en fonction du temps. En Rouge: Notre agent.

- **Agent Naïf**: L'agent Naïf se dirige vers une case aléatoire pour ses 30 premieres actions, puis se dirige vers la plus proche relique, et choisit aléatoirement une direction s'il est à 2 cases d'une relique.

- **Agent Task-Based**: L'agent Task-Based attribue dynamiquement des tâches aux différents vaisseaux parmi 4 tâches: explorer pour découvrir des reliques, découvrir des cases de points, se diriger vers une case de point connue, et accumuler de l'énergie. Les vaisseaux utilisent l'algorithme de pathfinding A* pour atteindre les cases d'intérêt. Cet agent dispose aussi de la même mémoire que notre agent.

5.3. Résultats

On observe bien sur la Figure 3 la progression en ELO attendues au cours de l'entraînement. La progression n'apparaît pas monotone, mais cela peut être dû au faible nombre de parties dans le tournoi. On constate aussi que notre méthode surpasse les trois baselines proposées.

5.4. Comparaison des frameworks PyTorch / JAX

Pour l'implémentation, nous avons privilégié le framework JAX à PyTorch, pour des questions de vitesse. JAX est une librairie d'algèbre linéaire haute performance développée par Google, disposant d'une librairie de deep learning nommée FLAX. La force de JAX repose dans la "just-in time compilation" qui permet d'accélérer toutes les opérations sur GPU, optimisées avec le backend XLA. L'entièreté de notre boucle d'entraînement a ainsi pu être compilé, ce qui donne une accélération considérable.

On compare ici la vitesse de notre implémentation en PyTorch avec notre implémentation en JAX. On voit sur la figure 4 qu'en utilisant JAX, le bottleneck réside dans le calcul du réseau de neurones, et non dans la simulation de l'environnement de jeu.

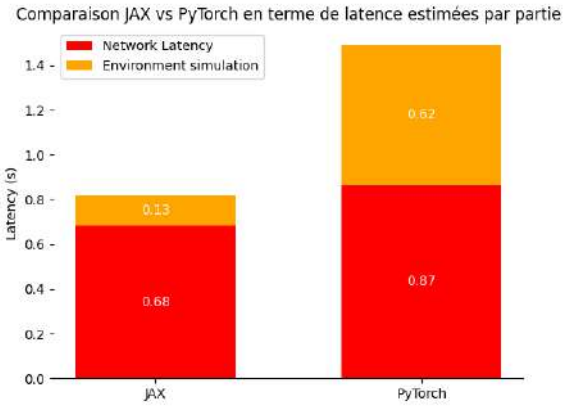

Figure 4. Comparaison des temps de latence par partie

6. Discussion

6.1. Manque d'études d'ablations

Nous sommes conscients que des études d'ablation compléteraient très bien ce projet, en particulier sur les aspects suivants:

- **Environment Sampling**
- **Ressource Prediction Loss**
- **Spectral Normalization**
- **Curriculum Learning**

Nous n'avons pas pu documenté ces études d'ablation, puisqu'une phase d'entraînement dure 15 heures, mais nous avons fait deux observations informelles:

- Il y a un clair plateau dans l'entraînement en l'absence de curriculum learning

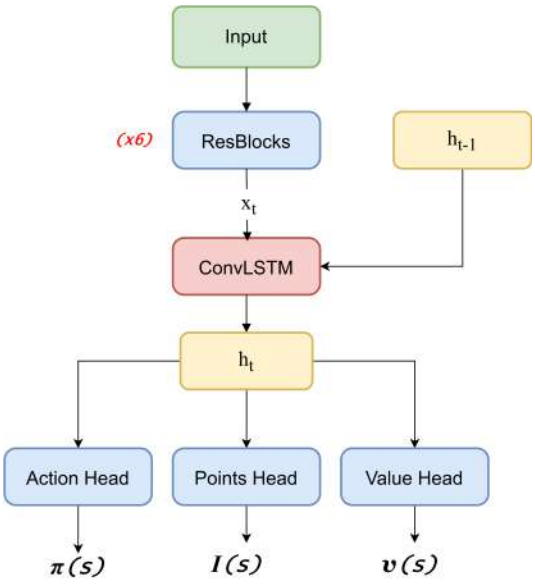

Figure 5. Architecture de policy récurrente envisagée

- La Ressource Prediction Loss a été introduite assez tard dans notre projet et ne semble pas avoir fait une grande différence dans la première phase, nous l'avons implémentée principalement par curiosité.

6.2. Autre approches envisagées

MARL Décentralisé Comme expliqué dans la section état de l'art, le MARL décentralisé est une approche intéressante que l'on aurait aimé expérimenter. Nous avons privilégié l"approche Pixel-to-Pixel pour sa simplicité d'implémentation dans le cas de ce jeu.

Policy récurrente Le jeu LuxAI est un jeu à observation partielle et certaines informations nécessitent d'avoir un mécanisme de mémoire, notamment pour prédire les déplacements des obstacles et pour anticiper les mouvements de l'adversaire. Ce mécanisme de mémoire est parfois implémenté par une policy récurrente, une technique où l'on fournit la représentation de l'état précédent en entrée au réseau pour prendre sa décision. L'article [6] implémente une policy récurrente pour le Multi-agent Path-Finding avec un LSTM. Dans notre cas, on aurait par exemple pu appliquer cette idée et ajoutant une cellule de ConvLSTM après les blocs résiduels (voir figure 5)

Autres architecture Pixel-To-Pixel. Les U-Net auraient pu être une alternative aux ResNets que nous avons utilisés.

References

[1] Y. Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. volume 60, page 6, 06 2009. 3

[2] Hanmo Chen, Stone Tao, Jiaxin Chen, Weihan Shen, Xihui Li, Chenghui Yu, Sikai Cheng, Xiaolong Zhu, and Xiu Li. Emergent collective intelligence from massive-agent cooperation and competition, 2023. 2, 3, 4, 5

[3] Lei Han, Peng Sun, Yali Du, Jiechao Xiong, Qing Wang, Xinghai Sun, Han Liu, and Tong Zhang. Grid-wise control for multi-agent reinforcement learning in video game AI. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2576–2585. PMLR, 09–15 Jun 2019. 2, 3

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 3

[5] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018. 3

[6] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, July 2019. 2, 3, 4, 7

[7] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. 2

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. 2, 4

[9] Stone Tao, Akarsh Kumar, Bovard Doerschuk-Tiberi, Isabelle Pan, Addison Howard, and Hao Su. Neurips 2024 - lux ai season 3. https://kaggle.com/competitions/lux-ai-season-3, 2024. Kaggle. 1

A. Détails supplémentaires

A.1. Features Visuelles

Table 2. Features visuelles utilisées. (*): la channel est normalisée en divisant par le maximum possible

Nom	Type
La case est visible	Bool
La case est un astéroïde	Bool
La case est une nébula	Bool
La case est une relique	Bool
La case est à moins de 2 cases d'une relique	Bool
La case est une case de points	Bool
La distance à la plus proche relique (*)	Float
Valeur du champ d'énergie	Float
Coordonnées X de la case (*)	Float
Coordonnées X de la case (*)	Float
Distance de la case à la frontière (*)	Float
Distance de la case à la diagonale (*)	Float
Distance de la case au centre (normalisée)	Float
La date de dernière visite de la case (*)	Float
Nombre d'allies sur la case	Entier
Nombre d'ennemis sur la case	Entier
Energie des ennemis sur la case (*)	Float
Energie des alliés sur la case (*)	Float

A.2. Features non visuelles.

Table 3. Features non visuelles utilisées (P): Paramètres de la partie, changeant à chaque partie, normalisé en centrant autour du min/max.

Nom	Type
Global Scalars	
(P) Coût en énergie d'un déplacement	Entier
(P) Coût en énergie d'une attaque	Entier
(P) Range de Vision d'un vaisseau	Entier
(P) Vision réduite par une nébula	Entier
(P) Energie réduite par une nébula	Entier
(P) Dégât principal d'une attaque	Entier
(P) Dégât colatéral d'une attaque	Entier
(P) Vitesse de déplacemt. du champ d'énergie	Entier
(P) Magnitude de déplacemt. du champ d'énergie	Entier
(P) Fréquence de déplacemt. des astéroides	Entier
Points du joueur équipe	Entier
Points de l'adversaire	Entier
La première relique a été trouvée	Bool
La deuxieme relique été trouvée	Bool
La troisième relique été trouvée	Bool
Positions X,Y des reliques trouvées	
Manche actuelle	One-Hot
Etape actuelle	One-Hot

A.3. Rewards de la phase 1.

L'agent reçoit une récompense après chaque action, calculés avec les points suivants.

Table 4. Poids de chaque statistique dans le calcul de la reward dense

Nom	Récompense
Point gagné	0.2
Relique découverte	2.0
Case de point découverte	2.0
Attaque	0.01
Energie gagnée	0.0001

B. Schéma du resblock

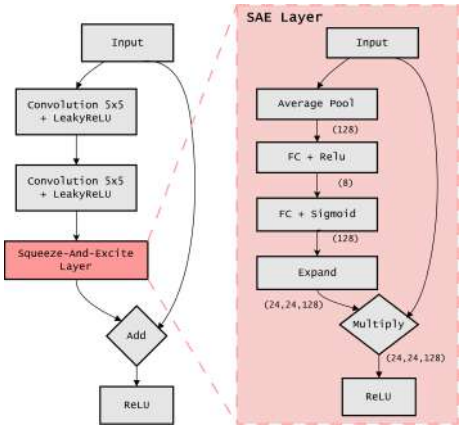

Figure 6. Diagramme notre resblock.

C. Résumé du monitoring du resblock

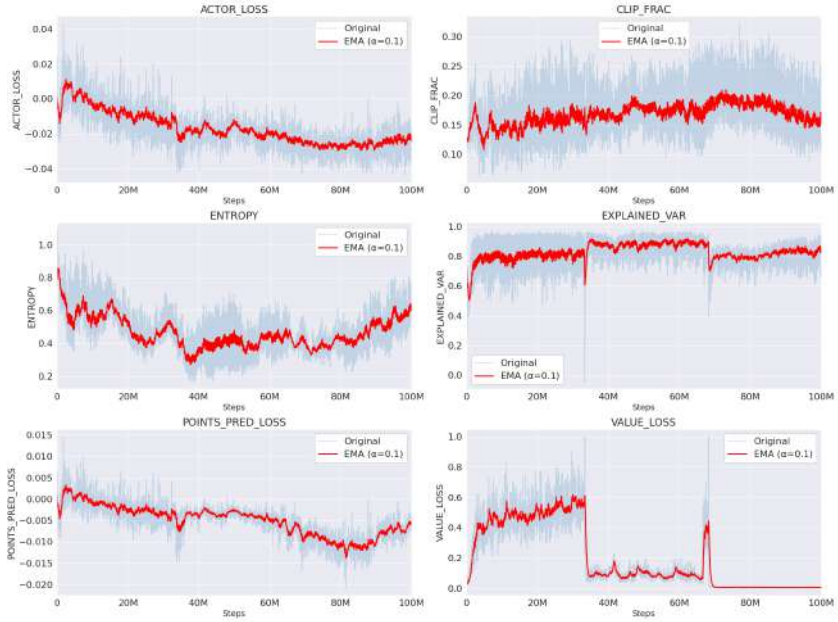

Figure 7. Evolution des métriques de loss pendant l'entrainement

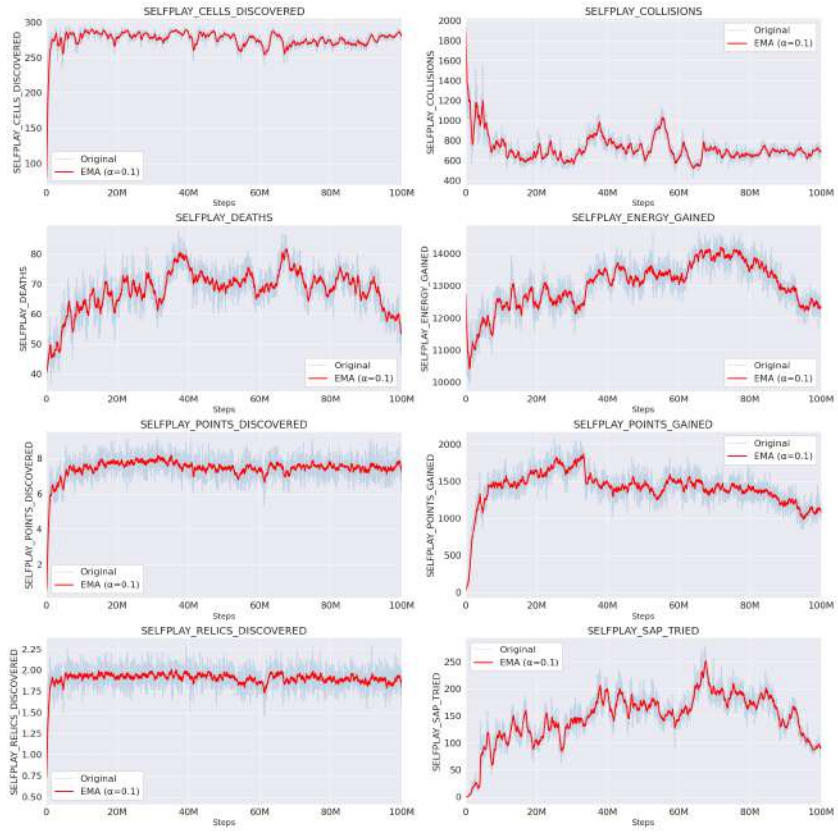

Figure 8. Evolution des statistiques de jeux pendant l'entrainement