Apprentissage par renforcement multi-agents: De SlimeVolley à la RoboCup

Client : Ludovic Hofer

Pelagie Alves, Elias Debeyssac, Alexis Hoffmann, Alexis Lheritier, Nicolas Majorel, Yves-Sebastian Pages

Master 1 Informatique
Université de Bordeaux
2020-2021

Chargé de TD : Adrien Boussicault

Objectifs et contexte

- **Implémenter un simulateur de la Robocup permettant d'entraîner des agents**
  - *Apprentissage par renforcement : Agent et Environnement*
  - *Multi-agents*
- **Paramétrer un environnement**
  - *Intérêt pour la Robocup*
  - *Interface Gym,*
- **Récolter des résultats des matchs**

logo Robocup 2020

Objectifs et contexte

- **Implémenter un simulateur de la Robocup permettant d'entraîner des agents**
  - *Apprentissage par renforcement : Agent et Environnement*
  - *Multi-agents*
- **Paramétrer un environnement**
  - *Intérêt pour la Robocup*
  - *Interface Gym*
- **Récolter des résultats des matchs**

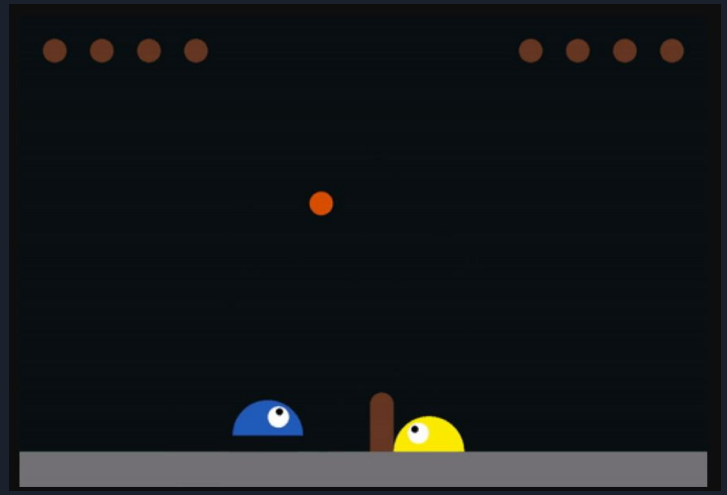
logo Gym

Objectifs et contexte

- **Implémenter un simulateur de la Robocup permettant d'entraîner des agents**
  - *Apprentissage par renforcement : Agent et Environnement*
  - *Multi-agents*
- **Paramétrer un environnement**
  - *Intérêt pour la Robocup*
  - *Interface Gym*
- **Récolter des résultats des matchs**

Positionnement par rapport à l'existant

- SlimeVolleyGym : projet d’apprentissage par renforcement
  - *Utilise Gym*
  - *Ressemblance avec Robocup*



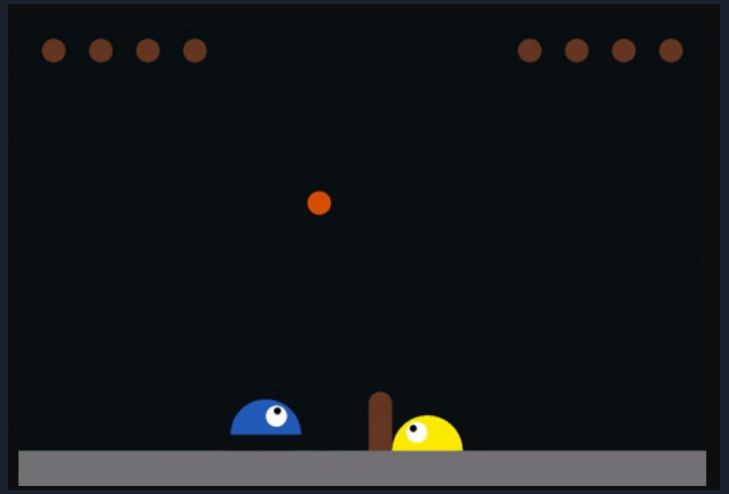
https://github.com/hardmaru/slimevolleygym

Slime Volley Gym

projet d'apprentissage par renforcement

Fonctionnalités :

- Réaliser l'apprentissage par renforcement single-agent (contre une IA random, Baseline Policy = Agent expert, ...) avec les scripts d'apprentissage
- Tester les politiques entrainées avec les scripts d'évaluation
- Jouer manuellement contre une IA et se rendre compte de notre faible performance

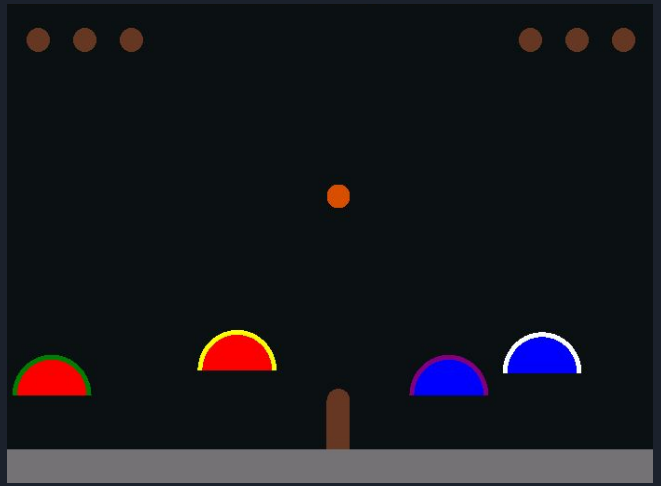


https://github.com/hardmaru/slimevolleygym

Extension de Slime Volley Gym
mode 2vs2 et l'apprentissage multi-agents

Fonctionnalités :

- Mode 2vs2 dans Slime Volley Gym
- Entraînement d'IA en 2vs2 (multi-agents)
- Scripts de training et d'évaluation pour le mode 2vs2
- Collision entre alliés
- Matchs de slime volley paramétrables (nombre de vies, désactiver collision, couleurs, nombre de joueurs, IA ...)



https://services.emi.u-bordeaux.fr/projet/git/m1-pdp-15/src/slimevolleygym_multiagent/test_state.py

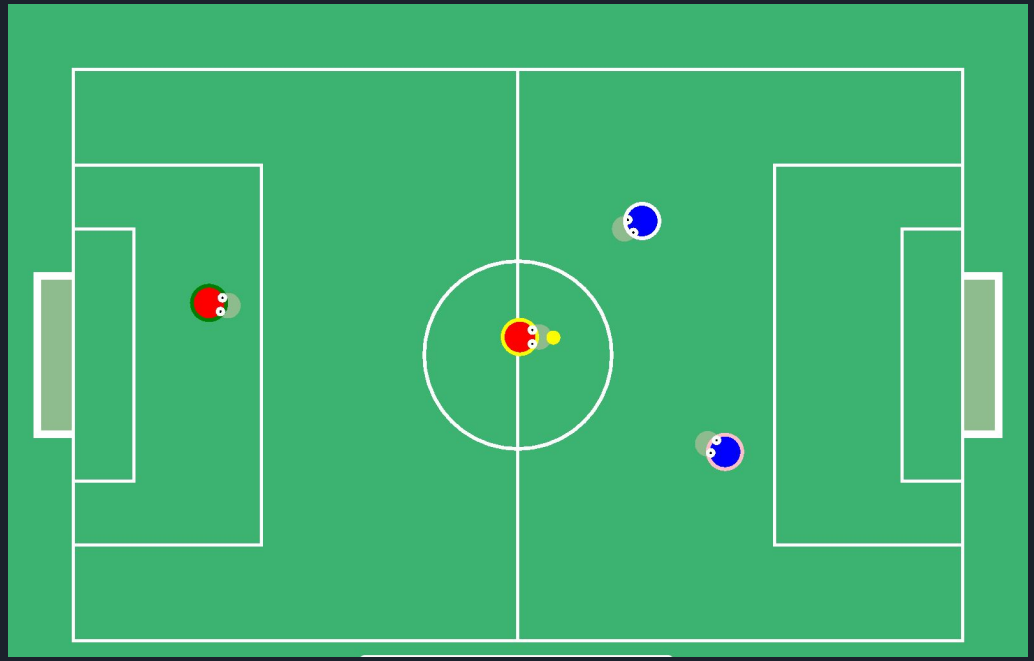
Description des besoins Robocup

Objectif : Créer une simulation de la Robocup fournissant un environnement Gym paramétrable dans lequel on peut entraîner un agent.

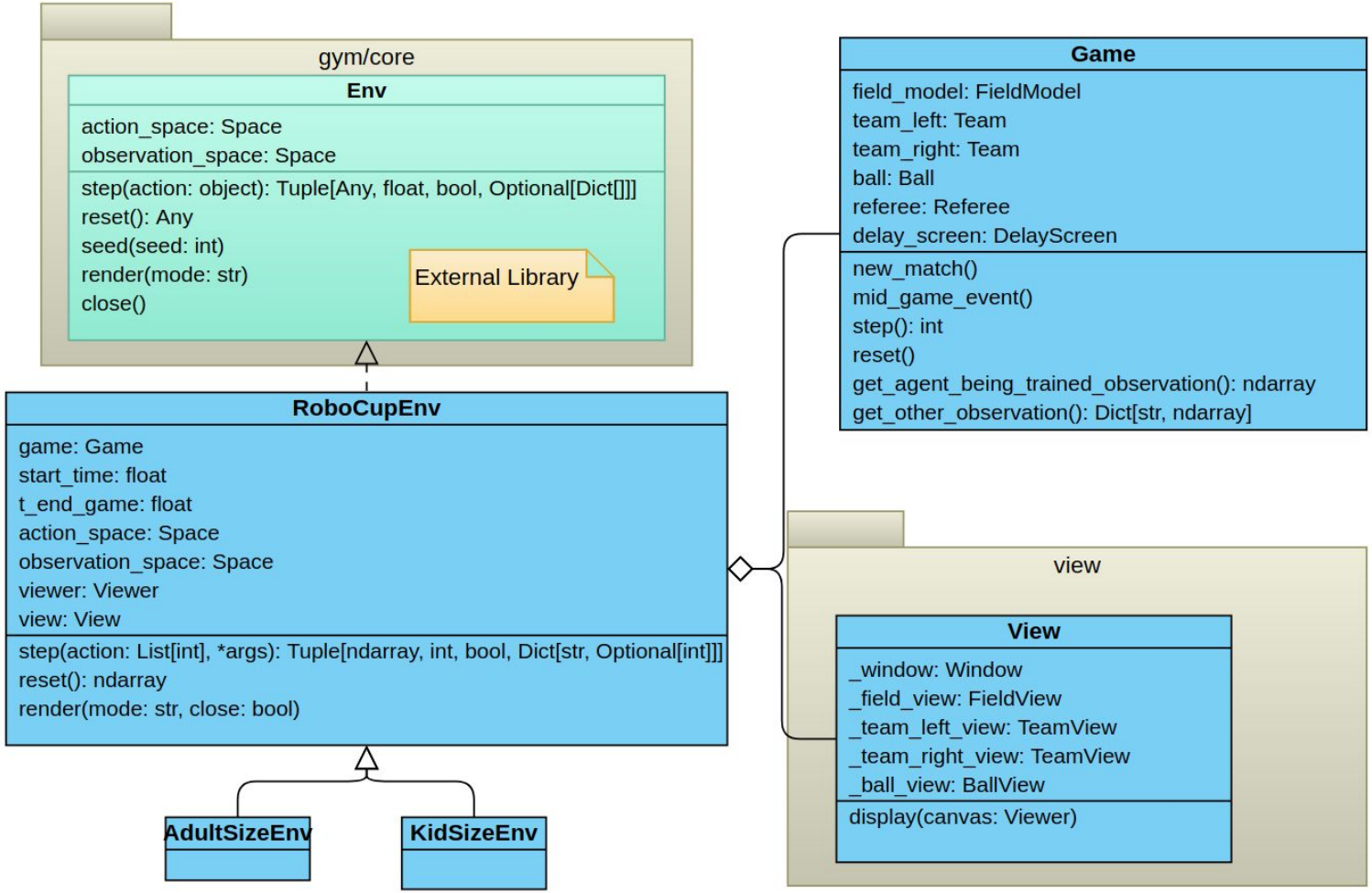
- Créer une simulation de la Robocup
  - Créer le terrain de la robocup
  - Créer et gérer : la balle, les 2 équipes d'agents
  - **Définir un espace d'action pour les agents**
  - **Définir un espace d'observation pour les agents**
  - Gérer le déroulement d'un match
  - **Implémenter les méthodes de la classe env de Gym**
  - Ajouter des règles et des détails : collisions, sorties de balle, fautes
  - Faire un script de test
- Pouvoir entraîner des agents dans l'environnement
  - Pouvoir attribuer une politique aux agents
  - Faire un script d'entraînement et entraîner un agent
- Pouvoir récupérer les résultats des entraînements et des évaluations
- Pouvoir paramétrer l'environnement

Description du logiciel Robocup

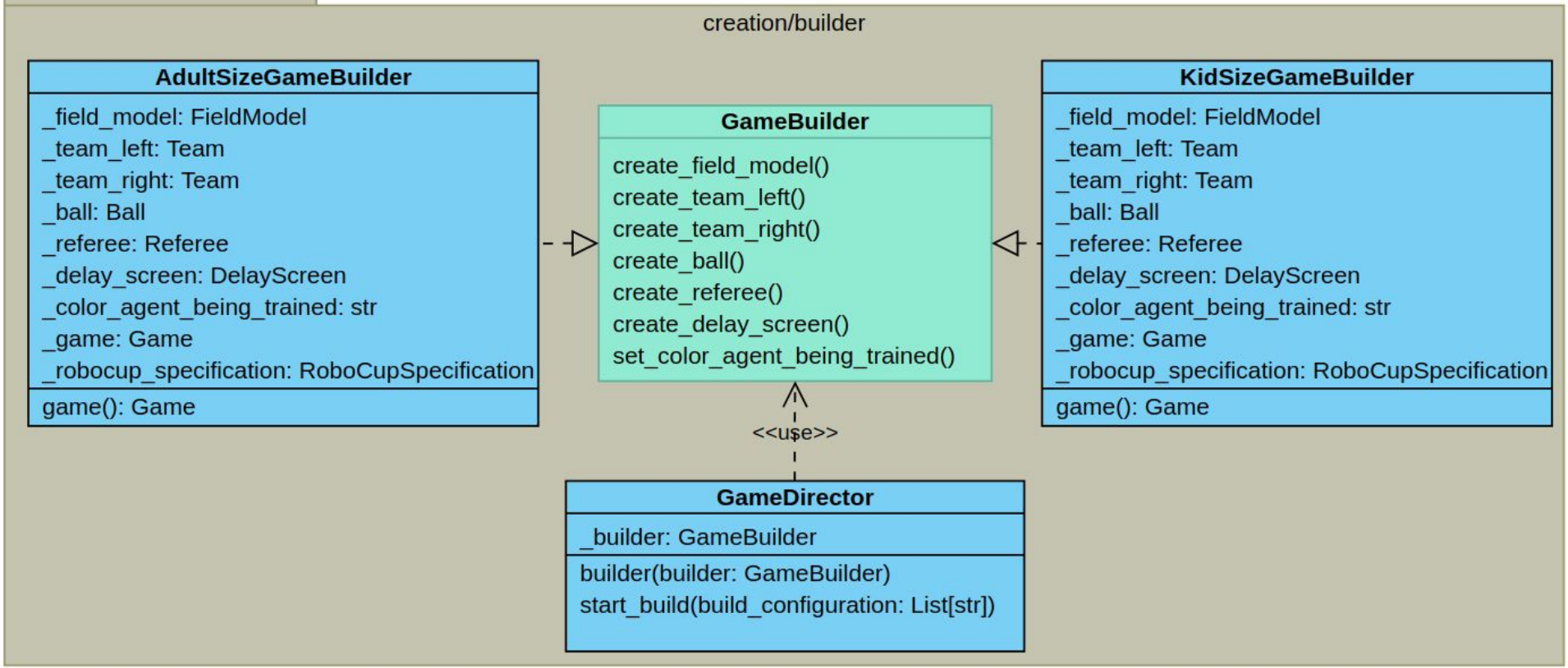
- Environnement de 2vs2
- Règles / phases de jeu : engagement, marquer un but, collisions, sorties de balle
- IA : random
- Paramètres à choisir dans le JSON :
  - taille du terrain
  - couleur des agents
  - activer désactiver collisions
  - modifier l'espace d'observation
- Fichier CSV pour récupérer les résultats des matchs



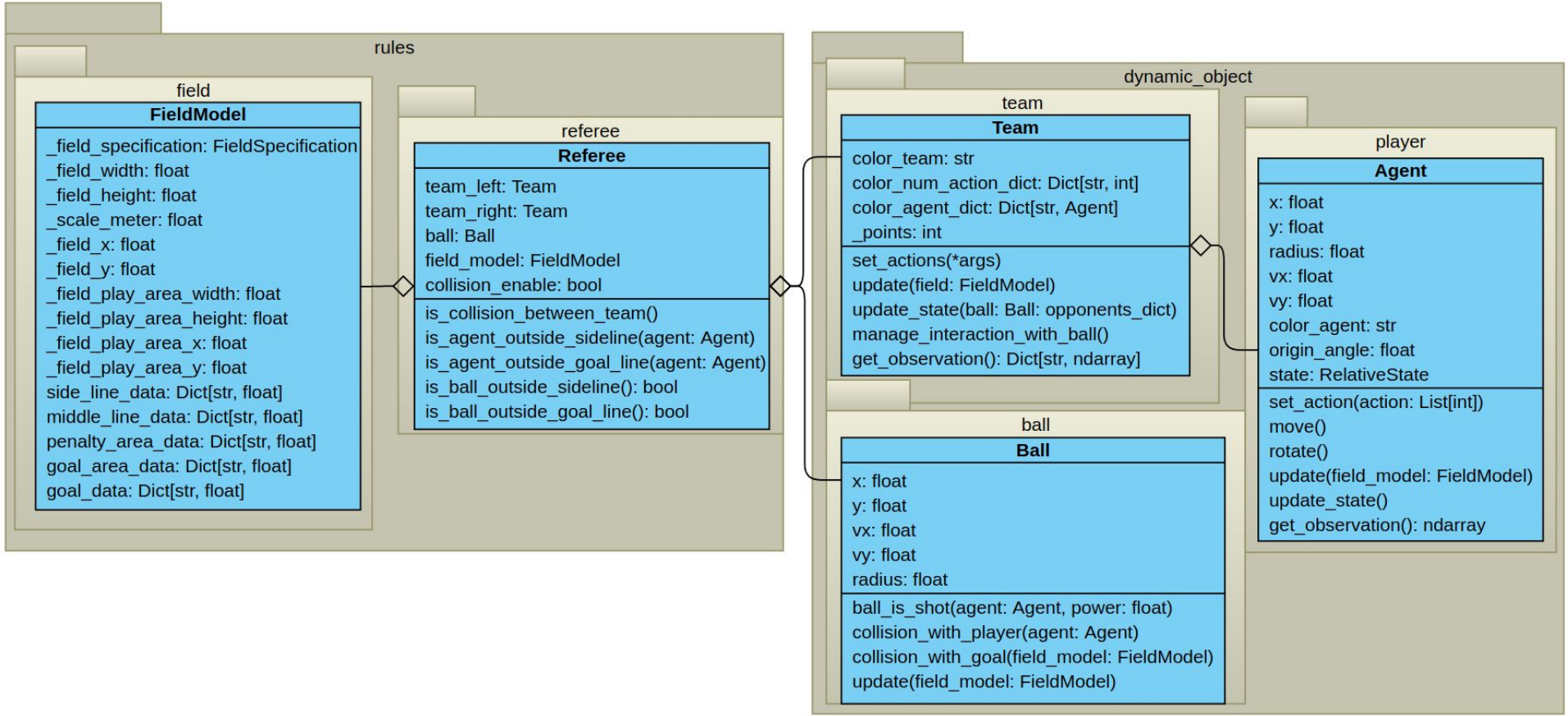
Architecture, Environnement: Model View Controller



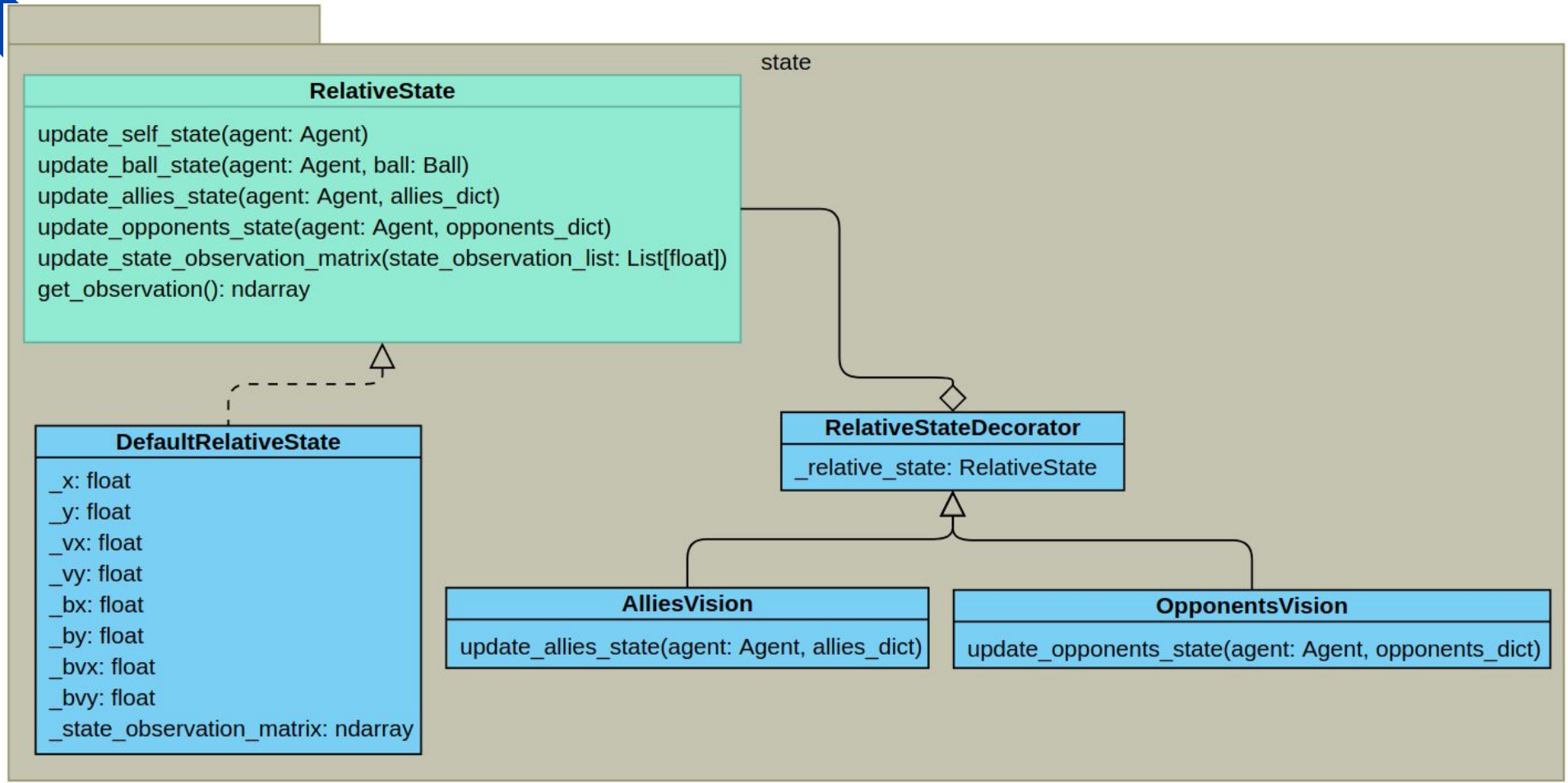
Architecture: Isolation du processus de création



Architecture: Traitements métiers principaux



Architecture: Représentation de l'espace d'observations



Tests
Tests manuels

Tests manuels avec `script_manual.py`

- Résultat visible

Limites

- Pas automatique

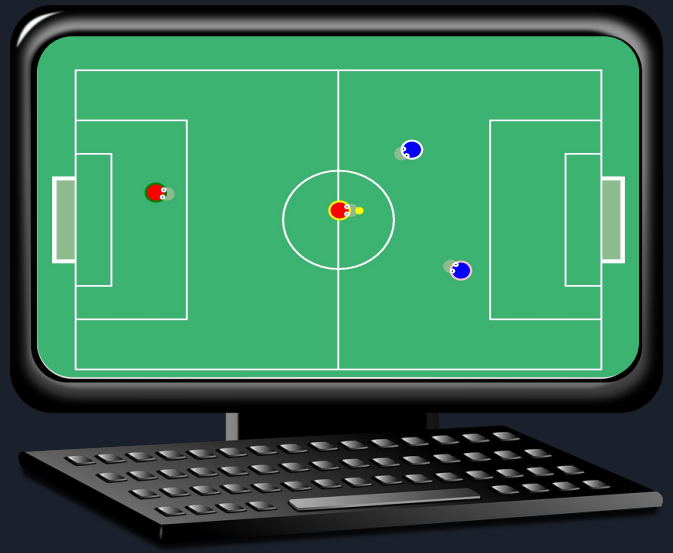


illustration d'une partie jouable et affiché à l'écran

Tests
Tests unitaires

Framework **Unit test**

- Tests unitaires automatiques pour les méthodes de classes
- Multiples paramètres pour tester différentes entrées

liste des tests

Tests

Couverture

Couverture des `tests/` avec **PyCharm Pro**

- Les lignes parcourues pendant l'exécution

Couverture globale **64%**

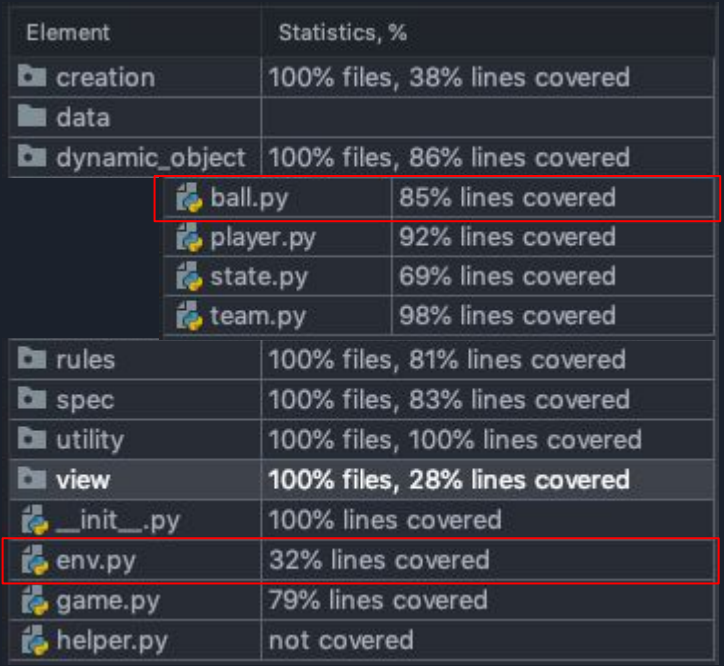
- Fonctions graphiques pas testées

Element	Statistics, %
creation	100% files, 38% lines covered
data	
dynamic_object	100% files, 86% lines covered
rules	100% files, 81% lines covered
spec	100% files, 83% lines covered
utility	100% files, 100% lines covered
view	**100% files, 28% lines covered**
__init__.py	
geom.py	**39% lines covered**
view.py	22% lines covered
__init__.py	100% lines covered
env.py	32% lines covered
game.py	79% lines covered
helper.py	not covered

couverture sur les tests

Tests
Analyse

- Il manque surtout des tests de classes et toutes les méthodes ne sont pas testées
  exemple : env.py
- Tous les cas ne sont pas testés donc pas de couverture à 100%
  exemple : sortie de terrain que sur un côté du terrain
- Code couvert ≠ Code testé
- Pas de garantie que le code n'ait pas de bugs importants



Element	Statistics, %
creation	100% files, 38% lines covered
data	
dynamic_object	100% files, 86% lines covered
ball.py	85% lines covered
player.py	92% lines covered
state.py	69% lines covered
team.py	98% lines covered
rules	100% files, 81% lines covered
spec	100% files, 83% lines covered
utility	100% files, 100% lines covered
view	**100% files, 28% lines covered**
__init__.py	100% lines covered
env.py	32% lines covered
game.py	79% lines covered
helper.py	not covered

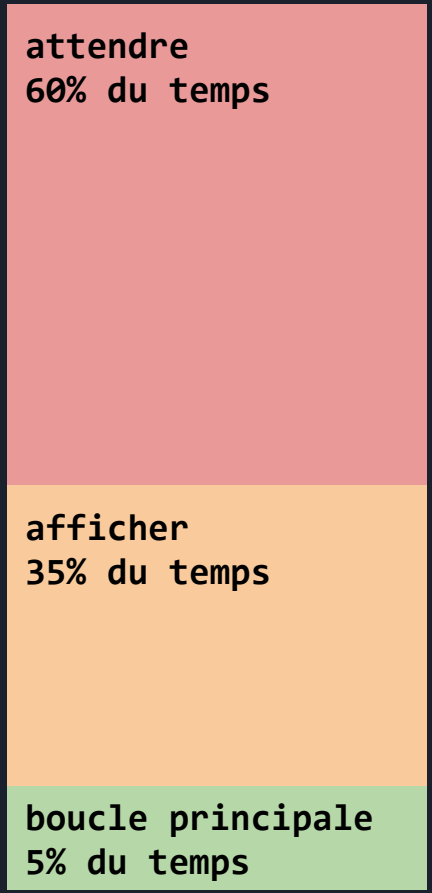
couverture sur les tests

Tests
Profilage

Profilage avec **PyCharm Pro**

- Profilage d'un match manuel avec affichage graphique en temps réel

  `script_manual.py`
- Match ralenti pour qu'il soit jouable par un humain
- Affichage coûteux



résultat du profilage d'un match vs humain avec affichage

Tests
Profilage

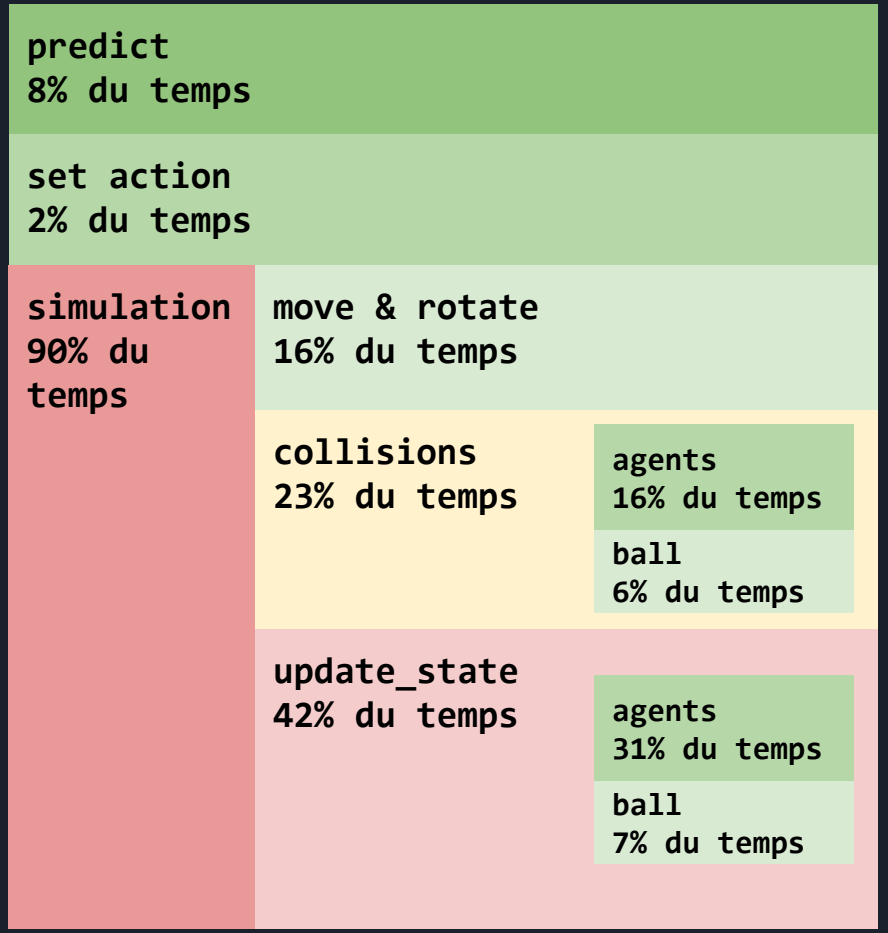
Profilage avec **PyCharm Pro**

- Profilage d'un match d'entrainement sans affichage

  `train_ga_selfplay.py`

Fonctions coûteuses :

- L'actualisation des données pour l'apprentissage (predict & update)
- Complexité collisions



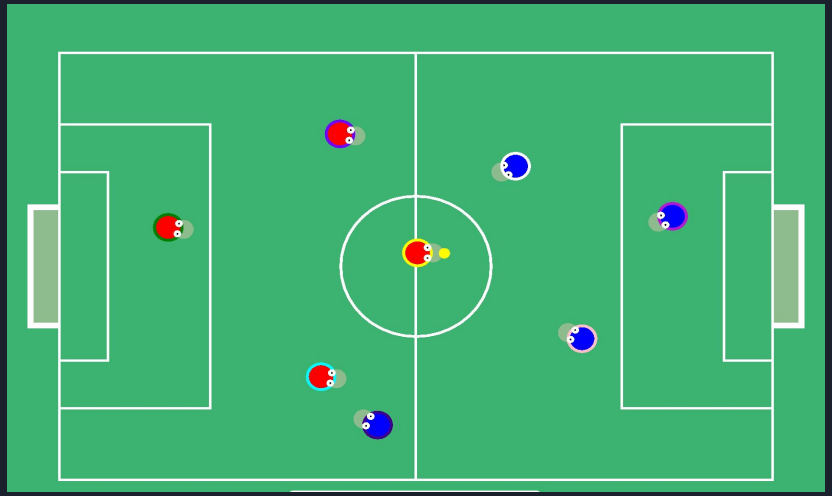
résultat du profilage sur l'entrainement

Conclusion
Plan

- La partie technique
  - Améliorer la création dynamique d'équipe.
  - Remplacer l'identification par couleur des agents par un numéro.
- La physique de jeu
  - Ajouter une incertitude de direction lors d'un tir.
  - Ajouter une physique plus réaliste à l'herbe.
- Les règles
  - Ajouter des phases de jeu (penalty, coups francs, touche).
- L'interface utilisateur
  - Adapter nos fonctions pour qu'elles puissent fournir des vitesses en cm/s à l'utilisateur.

Conclusion
La partie technique

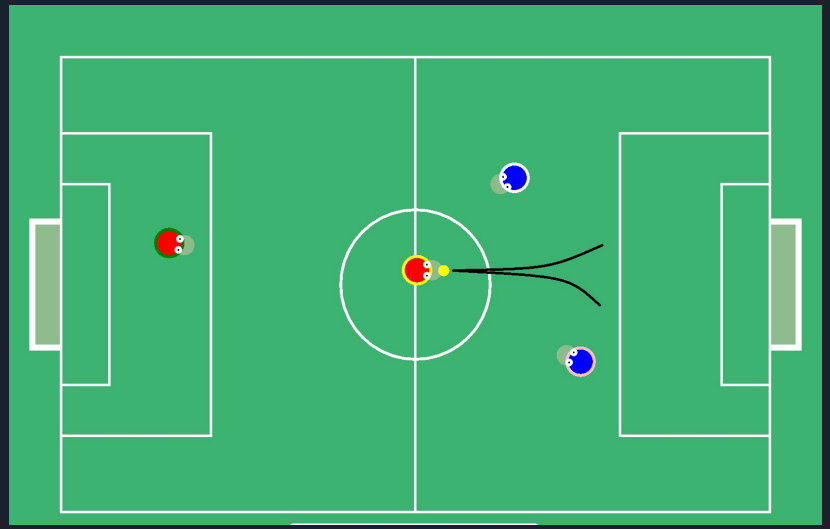
- Améliorer la création dynamique d'équipe.
- Remplacer l'identification par couleur des agents par un numéro.
- Optimiser les collisions entre agent.
- Implémenter la "chute" d'un agent lors d'une collision.
- Avoir des agents rectangulaires.
- Modifier la hitbox des agents (rectangulaire)
- Pouvoir choisir l'agent que l'on veut entraîner.
- Faire des phases de remise en jeu fluide.



Montage photo représentant deux équipes de quatre joueurs

Conclusion
La physique

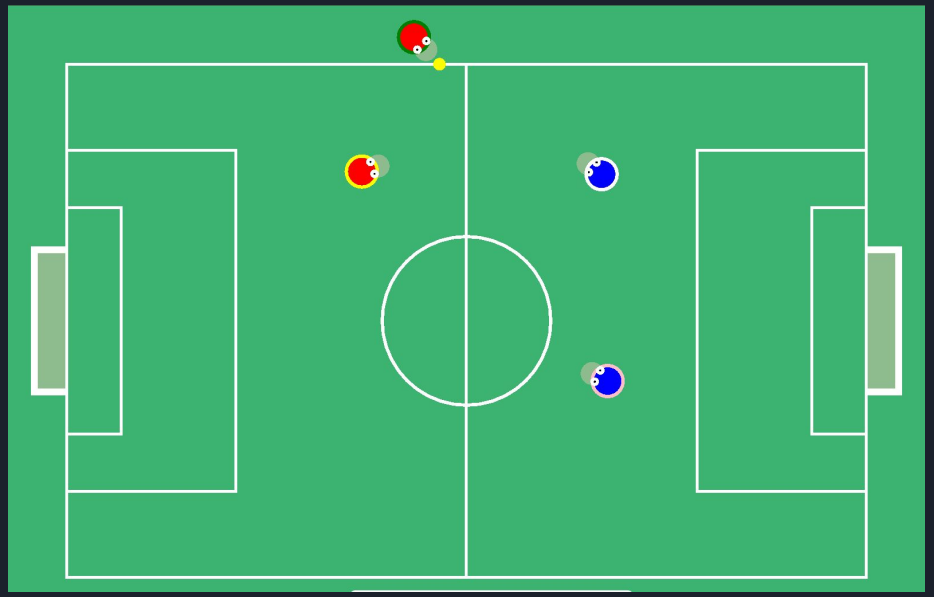
- Ajouter une incertitude de direction lors d'un tir.
- Ajouter une physique plus réaliste à l'herbe.
- Pouvoir moduler la puissance de tir d'un agent pendant la partie.
- Ajouter des effets de balle.
- Avoir des vitesses de déplacement réaliste.



Montage photo représentant l'incertitude lors d'une tir par un agent

Conclusion
Les règles

- Ajouter des phases de jeu (penalty, coups francs, touche).
- Positionner les agents de manière réaliste sur le terrain.
- Ajouter des nouvelles règles de jeu.
  - Exclusion d'un joueur lors d'une pénalité.
  - Gestion des fautes.
  - Pouvoir faire des touches.
  - Ajouter plus de pénalités.

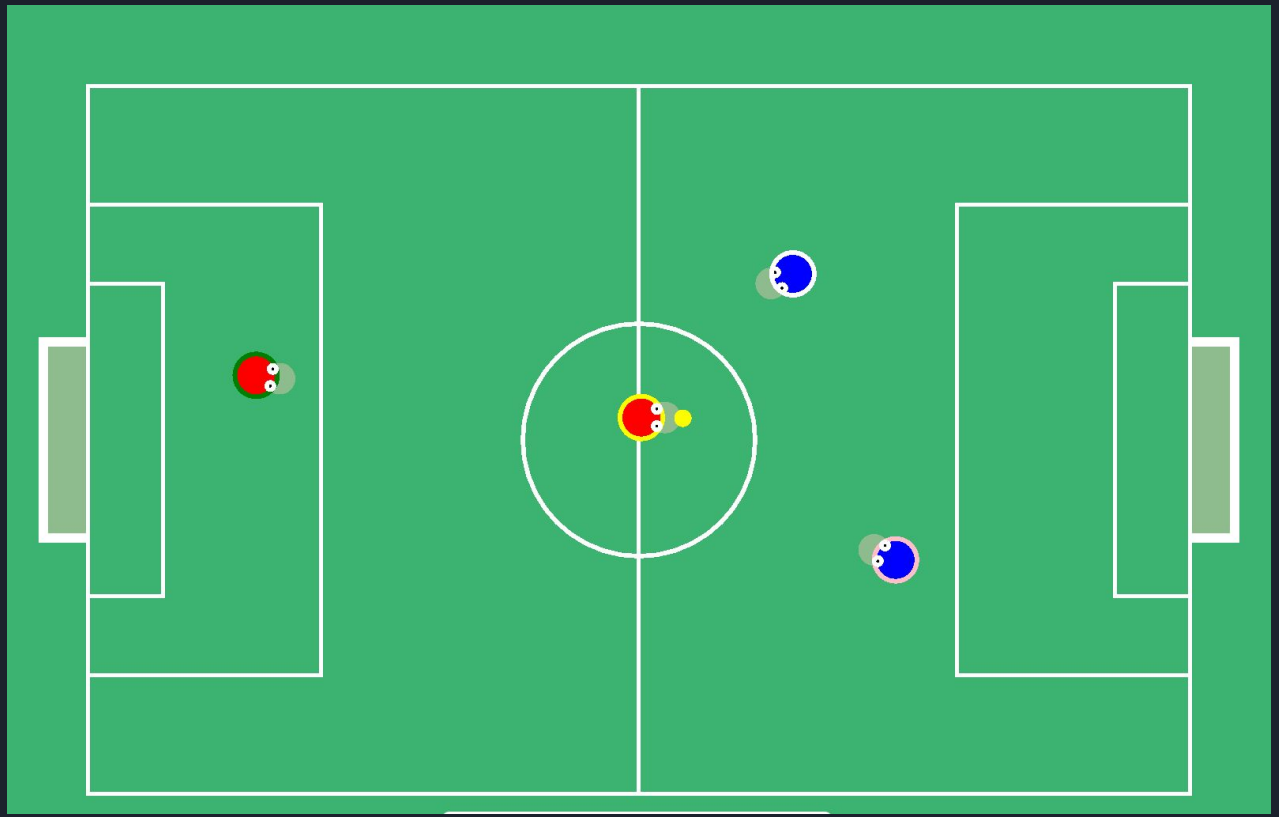


Montage photo représentant une touche effectué par l'équipe rouge

Conclusion
L’interface utilisateur

- Adapter nos fonctions pour qu’elles puissent fournir des vitesses en cm/s à l’utilisateur.
- Faciliter l’implémentation d’algorithmes d’apprentissage par l’utilisateur.
- Pouvoir placer les agents manuellement sur le terrain.
- Permettre l’ajout d’un environnement directement en ligne de commande au lancement.
- Ajouter plusieurs paramètres pour pouvoir créer une multitude d’environnements.

Questions



Capture d'écran d'une partie lors d'un lancement depuis le script script_manual.py