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

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

Master 1 Informatique
Université de Bordeaux
2020-2021

Sommaire

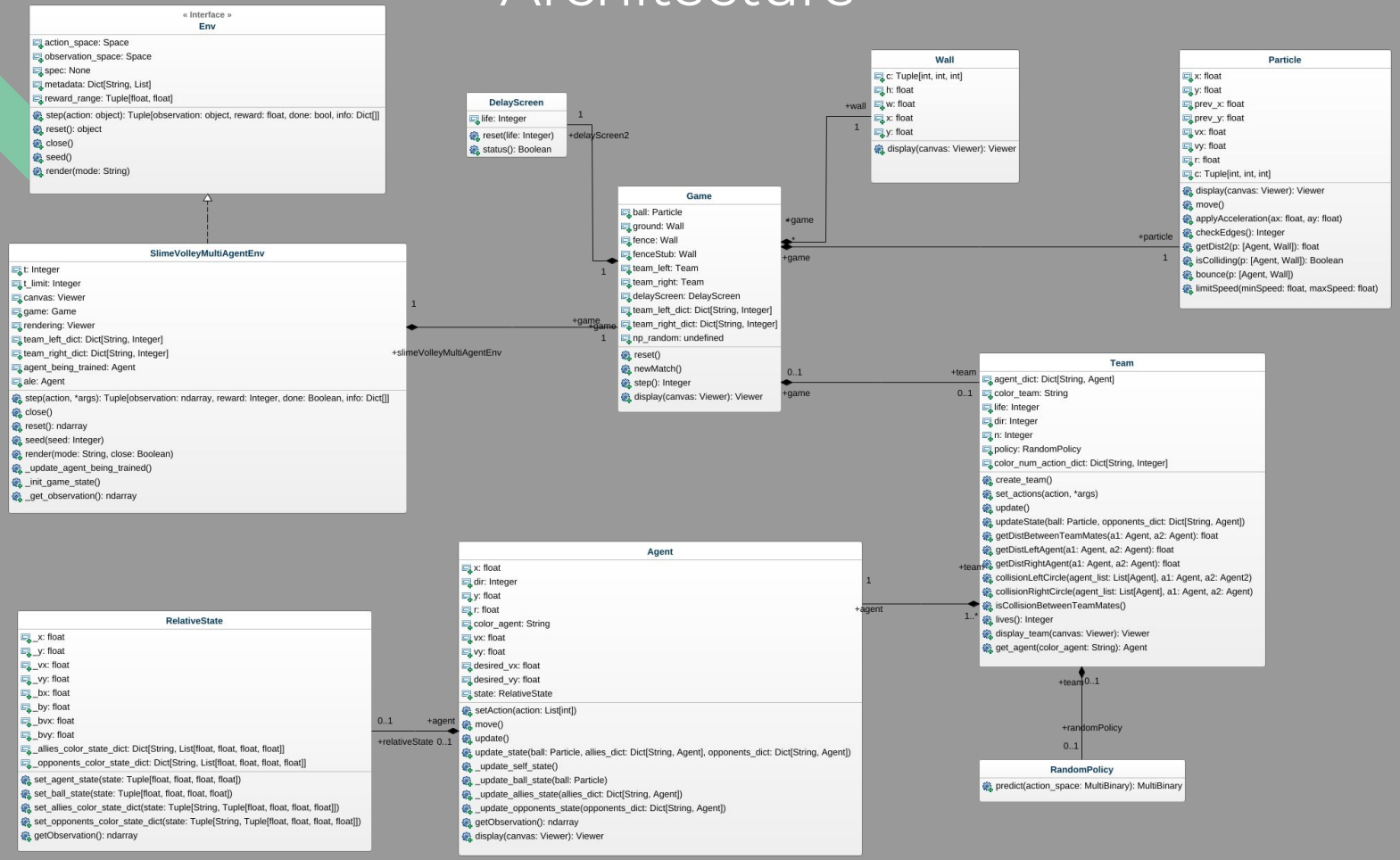
- Présentation du projet
- Besoins principaux
- Architecture SlimeVolleyGym - Multi-Agent
- Architecture Robocup
- Besoins réalisés
  - SlimeVolleyGym - Multi-Agent
    - Implémentation d'une équipe de Slime
    - Adaptation de l'environnement
    - Modification de l'environnement avec JSON
  - Robocup
    - Création d'un environnement
- Objectifs à atteindre

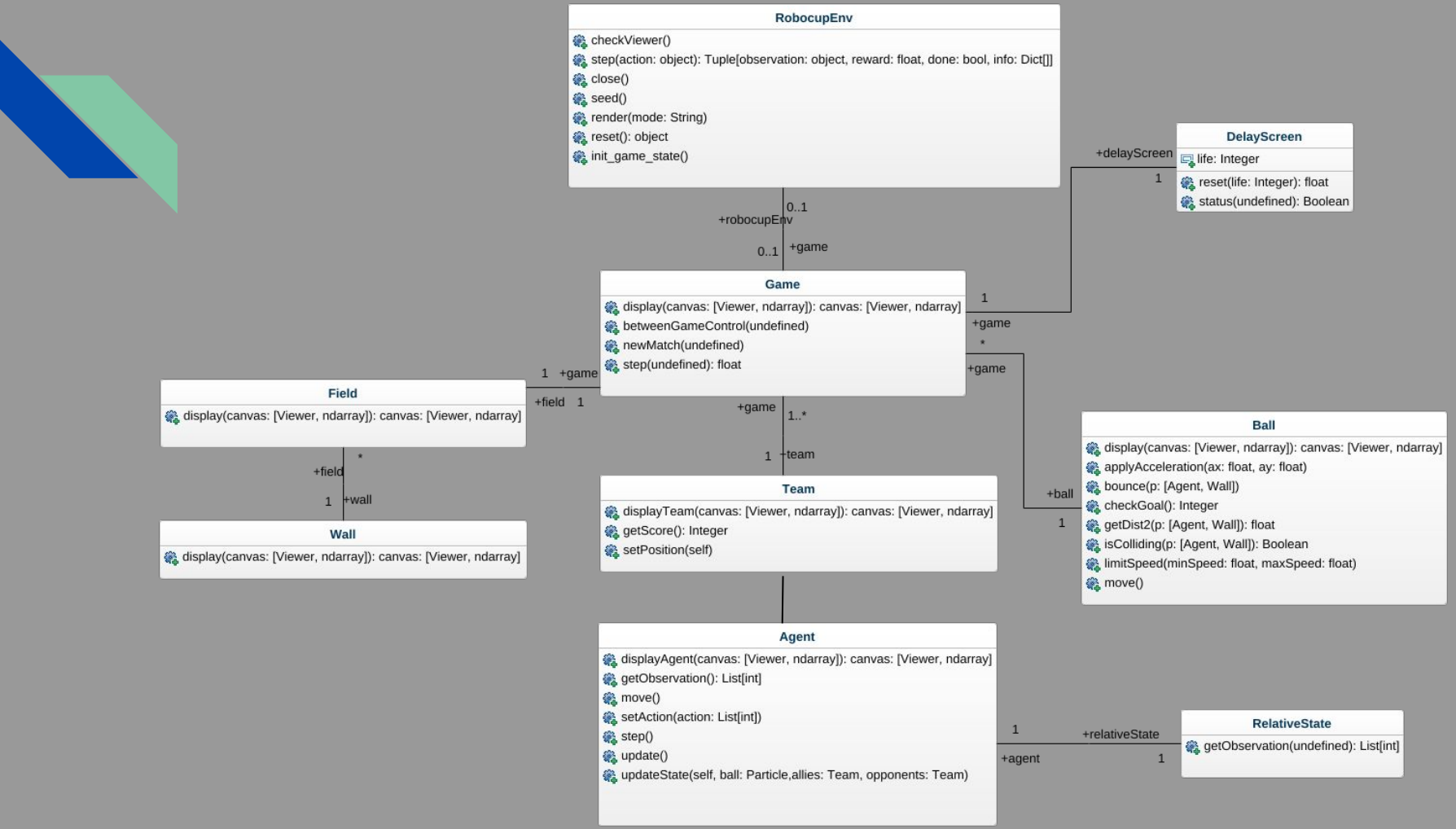
Besoins principaux

- SlimeVolleyGym Implémenter un environnement multi-agent
  - Lecture des paramètres de l'environnement dans un JSON
  - Définir un environnement Multi-agent
    - Implémenter une classe équipe
    - Modifier l'espace d'observation des agents
    - Mettre à jour l'état et l'observation de l'agent
    - Implémenter la collision entre agent de la même équipe
    - Pouvoir paramétrer l'environnement (taille de terrain, système de récompense)

- Robocup Implémenter un environnement permettant d'entraîner des agents
  - Robocup V0 version non paramétrable
    - Créer un environnement gym
    - Adapter les classes agent, particule, game
    - Afficher le terrain, les agents, la balle
    - Lancer un match sans arrêt de jeux
  - Robocup V1 version paramétrable
    - Choisir les paramètres de l'environnement avec un JSON
    - Ajouter des paramètres pour faire varier l'environnement
    - Implémenter des règles supplémentaires

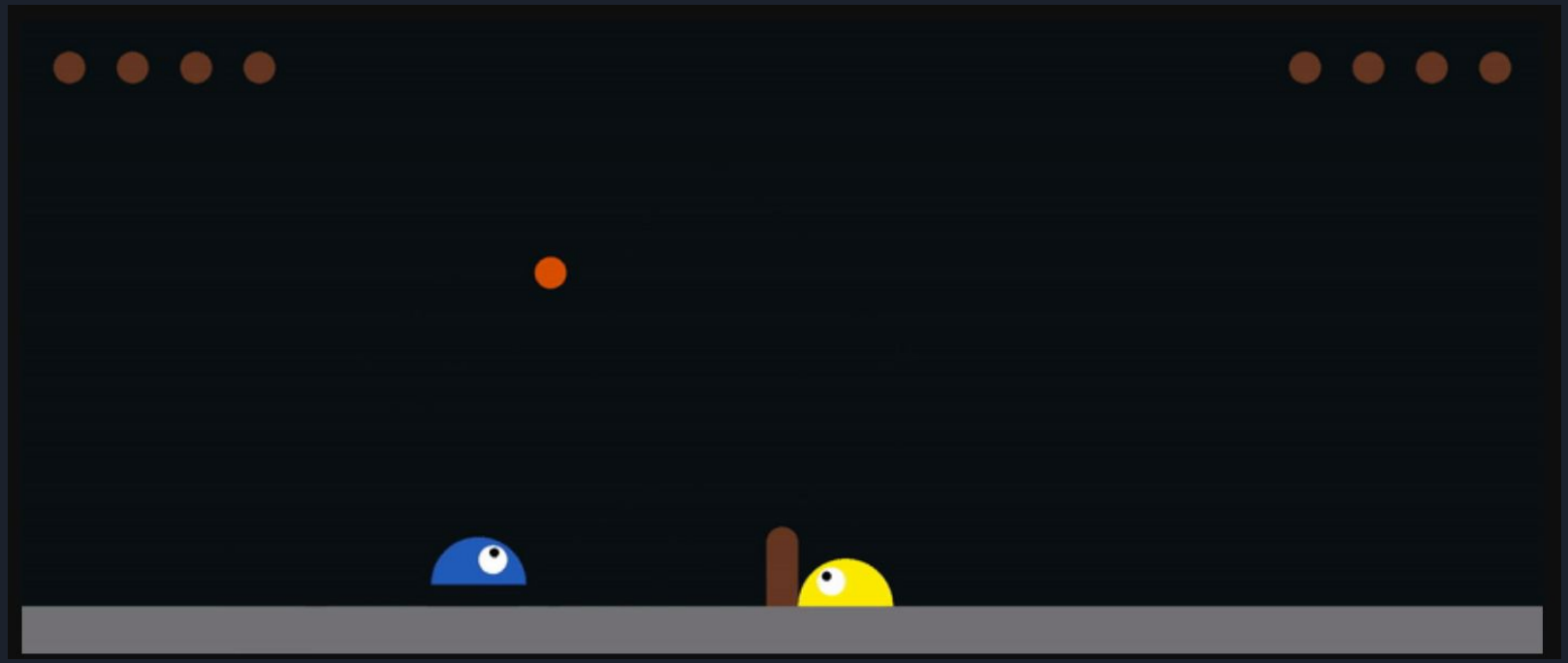
Architecture




RobocupEnv
checkViewer()
step(action: object): Tuple[observation: object, reward: float, done: bool, info: Dict[]]
close()
seed()
render(mode: String)
reset(): object
init_game_state()
DelayScreen
life: Integer
reset(life: Integer): float
status(undefined): Boolean
+delayScreen
1
0..1
+robocupEnv
0..1
+game
Game
display(canvas: [Viewer, ndarray]): canvas: [Viewer, ndarray]
betweenGameControl(undefined)
newMatch(undefined)
step(undefined): float
1
+game
*
+game
1
+game
Field
display(canvas: [Viewer, ndarray]): canvas: [Viewer, ndarray]
+field
1
+game
1..*
1
+team
Ball
display(canvas: [Viewer, ndarray]): canvas: [Viewer, ndarray]
applyAcceleration(ax: float, ay: float)
bounce(p: [Agent, Wall])
checkGoal(): Integer
getDist2(p: [Agent, Wall]): float
isColliding(p: [Agent, Wall]): Boolean
limitSpeed(minSpeed: float, maxSpeed: float)
move()
+ball
1
*
+field
1
+wall
Wall
display(canvas: [Viewer, ndarray]): canvas: [Viewer, ndarray]
Team
displayTeam(canvas: [Viewer, ndarray]): canvas: [Viewer, ndarray]
getScore(): Integer
setPosition(self)
Agent
displayAgent(canvas: [Viewer, ndarray]): canvas: [Viewer, ndarray]
getObservation(): List[int]
move()
setAction(action: List[int])
step()
update()
updateState(self, ball: Particle,allies: Team, opponents: Team)
1
+relativeState
+agent
1
RelativeState
getObservation(undefined): List[int]

Besoin réalisé - SlimeVolleyGym multi-agent

1 - Implémentation d'une équipe de Slime



Besoins réalisés - SlimeVolleyGym multi-agent

1 - Implémentation d'une équipe de Slime

Besoins réalisés - SlimeVolleyGym multi-agent

1 - Implémentation d'une équipe de Slime

Création d'une classe Team

- Agent stocké dans un dictionnaire
- Gestion de la vie de l'équipe
- Stocke la couleur et la position de l'équipe
- Attribution d'une politique par défaut à une équipe
- Gestion des collisions entre équipiers

Avantage :

- Équipe plus modulable
- Affichage des Agents plus facile
- Mise à jour de l'état et de la position de l'Agent plus maintenable
- Gestion de la collision et de la vie de l' équipe en diminuant les appels depuis la boucle de jeu
- Modification de la boucle de jeu plus pratique pour le portage vers robocup

Besoins réalisés - SlimeVolleyGym multi-agent

2 - Collision entre Agents

Besoins réalisés - SlimeVolleyGym multi-agent

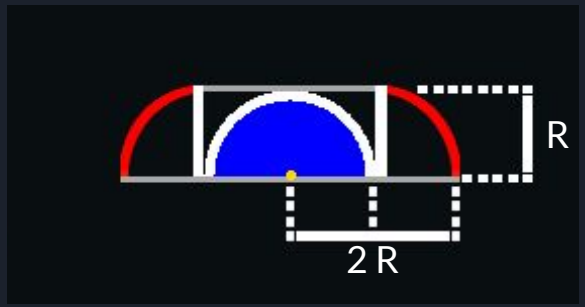
2 - Collision entre Agents

Partie Haute et Basse

- Calcul de la distance entre le centre des deux agents

Partie Droite et Gauche

- on dessine un arc de cercle
- Le centre de l'agent va "glisser" sur ce demi-cercle

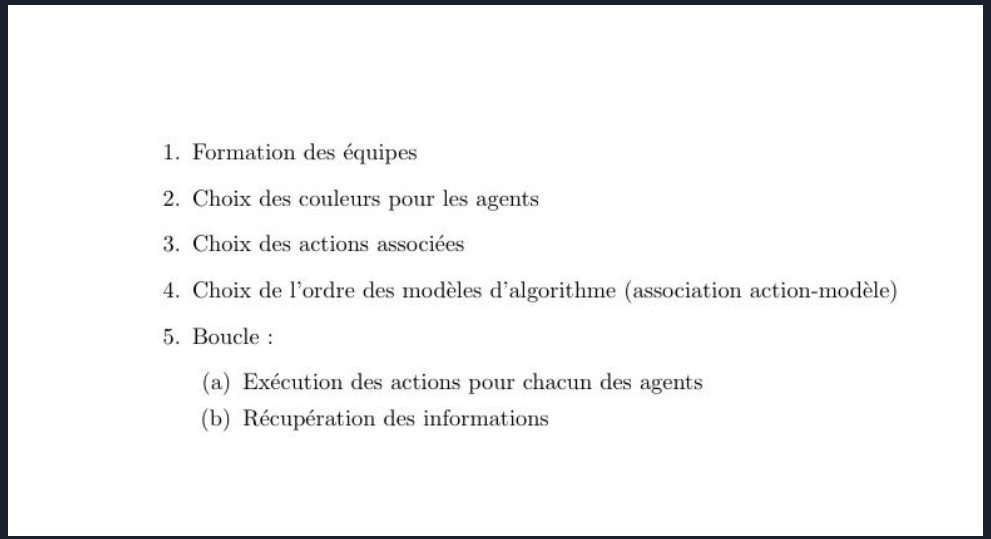


Besoins réalisés - SlimeVolleyGym multi-agent

3 - Pour reconnaître l'agent que l'on entraîne, quels modèles sont utilisés par les agents, il faut pouvoir les différencier dans le terrain en plus de la couleur de leur équipe.

Besoins réalisés - SlimeVolleyGym multi-agent

4. Choix dynamique des agents dans chaque équipe, choix des actions qu'ils récupèrent dans step (algorithme d'apprentissage (ou non) associé à l'action).



1. Formation des équipes
2. Choix des couleurs pour les agents
3. Choix des actions associées
4. Choix de l'ordre des modèles d'algorithme (association action-modèle)
5. Boucle :
   (a) Exécution des actions pour chacun des agents
   (b) Récupération des informations

Besoins réalisés - multi-agent

5. Adaptation de la classe SlimeVolleyEnv :

- Modification de l'espace d'observations (observations par états).
- Possibilité de choisir l'agent que l'on entraîne à l'instanciation de l'environnement,
- Récupération des observations relatives par rapport à celui-ci une fois que toutes les actions des agents ont été réalisées.

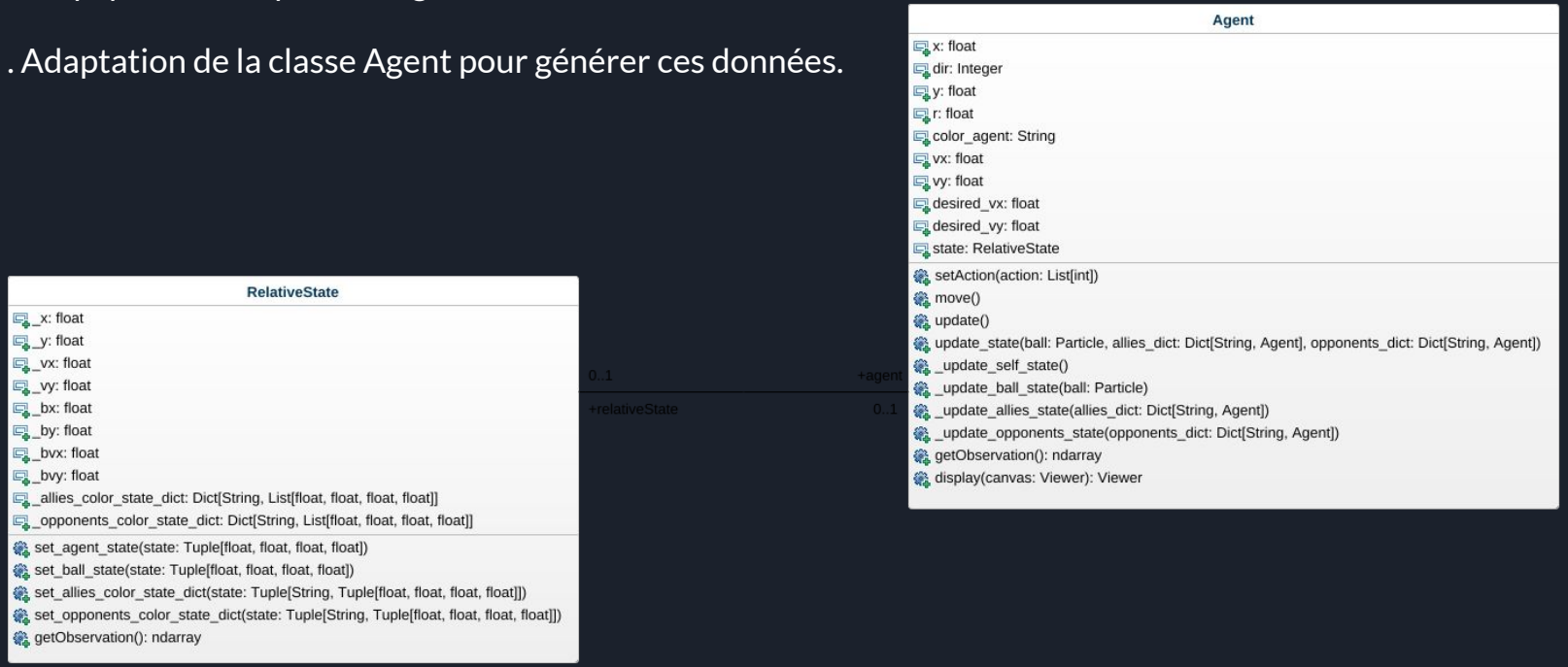
$$\begin{bmatrix} agent\_being\_trained\_x & agent\_being\_trained\_y & agent\_being\_trained\_vx & agent\_being\_trained\_vy \\ ball\_x & ball\_y & ball\_vx & ball\_vy \\ ally1\_x & ally1\_y & ally1\_vx & ally1\_vy \\ \dots & \dots & \dots & \dots \\ opponent1\_x & opponent1\_y & opponent1\_vx & opponent1\_vy \\ \dots & \dots & \dots & \dots \end{bmatrix}$$

Besoins réalisés - multi-agent

6. Mettre à jour l'état relatif pour chaque agent

. Adaptation de classe RelativeState pour contenir des observations relatives par rapport à la balle, aux alliés et à l'équipe adverse pour un agent donné.

. Adaptation de la classe Agent pour générer ces données.

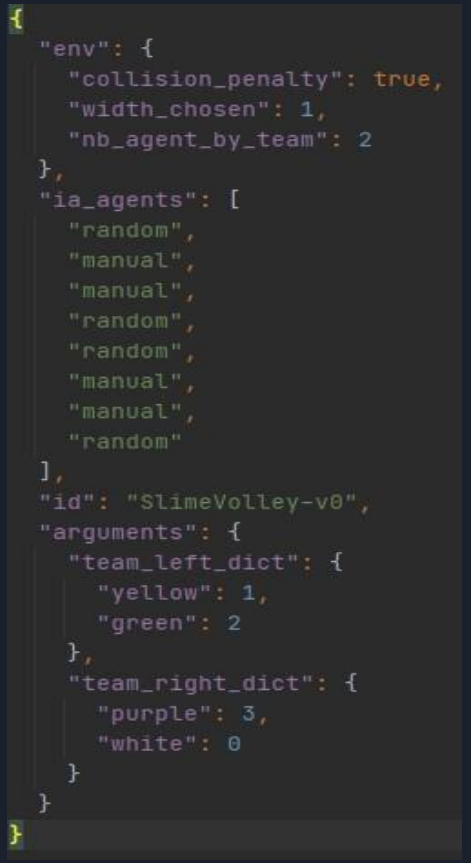


Besoins réalisés - SlimeVolleyGym multi-agent

7. Le fichier de configuration Json :

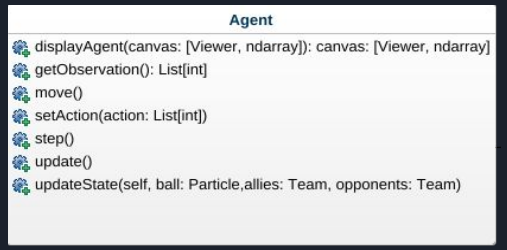
- utilité & contenu
- Choix Techniques de l'implémentation
- Tests unitaires (unittest)

→ Résultats en CSV



Robocup travail réalisé

- Adaptation des classes : `Agent`, `Ball` ,`Game` ,`RelativeState`
- Implémentation des classes `Field` et de `RobocupEnv(incomplet)`
- `testState.py` sortie image cohérente



Robocup travail en cours de réalisation

- Compléter `RobocupEnv.step()` (espace d'action)
- Implémenter classe `Team` et refactoring pour arriver à V0
- `testState.py` à compléter (action et observation)
- Refactoring des constantes et `Settings.py`
- Tests unitaires

Objectifs à atteindre

- SlimeVolley multi-agent (presque fini)
  - Lancer une partie IA depuis JSON
  - Ecrire des résultats dans le CSV
- Robocup
  - priorité :
    - Afficher les éléments correctements
    - Lancer un match version V0
    - Ajouter une classe team
    - Ajouter la collision entre agent
    - Ajouter l'observation des agents
    - Ajouter des politiques pour pouvoir tester les actions
    - Implémenter une couche application pour éviter de lancer avec un script
  - dans un second temps :
    - Ajouter des fautes de jeux (sortie de balle, collision entre agents) et des pénalités
    - Ajouter différentes catégories (Kid/Adult)
    - Choisir les paramètres dans un JSON
    - Ajouter une action "tir"