Implémentation d'une version multi-agent de slimevolley

Besoin fonctionnel = toutes les fonctionnalités dont le client à besoin pour son activité.
Besoin non-fonctionnel = tout le reste.

Besoins fonctionnels

Coder un script d'un joueur humain contre un joueur IA (déjà fait)
Coder un script de 2 joueurs humains contre un joueur IA
Coder un script de 2 joueurs humains contre 2 joueurs IA (centralisé ou décentralisé)
Coder un script de 2 joueurs IA contre 2 joueurs IA (centralisé ou décentralisé)

Gestion des collisions entre joueurs d'une même équipe, définir si c'est une pénalité dans le système de récompenses ?

Créer des paramètres influant sur le jeu :

- Pouvoir modifier la taille du terrain
- Pouvoir modifier la hauteur du filet
- Modifier la vitesse de la balle
- Modifier la vitesse des joueurs
- Modifier la vitesse des joueurs en fonction de leur équipe
- Modifier la taille de la balle
- Modifier la taille des joueurs

Adaptation des processus markovien <S,A,R,T> state action reward translation, modifier l'environnement slimevolley SlimeVolleyEnv (qui implémente l'interface Env) pour prendre en considération le rajout des 2 nouveaux slimes, l'espace d'observation va être différent. Vecteur d'observations à 18 paramètres : (slime1x, slime1y, slime1vx, slime1vy, slime2x, slime2y, slime2vx, slime2vy, slime3x, slime3y, slime3vx, slime3vy, slime4x, slime4y, slime4vx, slime4vy, ballx, bally, ballvx, ballvy).

Modifier la classe Game :

- Mise à jour des nouveaux agents (slimes) à chaque pas de temps, réadapter la classe Game (qui représente les différents paramètres du jeu, objets fixes (murs), dynamiques(slimes) etc..).
- Mettre à jour la position des nouveaux agents.
- Synchroniser le résultat des agents dans la même équipe, "l'émotion".

  - Prendre en compte dans la position des 2 nouveaux agents, la position relative d'un agent donné (par rapport à la balle et aux autres slimes).

  - afficher les deux nouveaux slimes.

Synchroniser le score des agents (et autres paramètres communs), dans la même équipe.

Stocker les états (dans un fichier ou en local ?) de chaque agent à partir des données fournies par la classe getObservation()

Réfléchir et implémenter les concepts de comportement centralisé ou décentralisé (centralisé : robot dirigé par une seul IA. Décentralisé : chaque robot avec sa propre IA)

Concernant l'IA : qu'elle puisse faire fonctionner l'équipe ensemble et qu'elle apprenne en même temps.
Elle pourra soit faire des passes, se replacer, établir des stratégies à plusieurs, qui doit tirer, occuper chacun sa partie du terrain.

Pouvoir faire des observation précise sur le match (possession de balle, nombre de points, possession de balle par chaque joueur, agressivité du joueur) (faire des stats)

Besoins non-fonctionnels

Quantification :

Eléments de faisabilités :
- Logiciel SlimeVolley

Contraintes et difficultés techniques :
- Compréhension du code de slimeVolley
- Compréhension du la bibliothèque gym
- Compréhension de la bibliothèque stable_baselines

- Comprendre les algorithmes d'apprentissage par renforcement (PPO, CMA-ES, GA)
- Implémentation des processus markovien
- Définir une architecture cohérente pour que les objets dans la même équipe communiquent entre-eux, se synchronisent.
- Sauvegarde de chaque action de chaque joueur (stockage)
- Développement de statistique par rapport à ces actions
- Implémentation d'un multitude de paramètres (nombre de joueur, vitesse de la balle, vitesse des joueurs, limitation du temps de réflexion de l'IA, taille de la balle, ...)

Risques et parades

- impossibilités de récupérer toute les valeurs intéressantes pour faire des stats ou justement sauvegarder trop de valeurs
- solution : Définir des valeurs intéressantes à sauvegarder (temps de la balle dans un camp, nombre de points, précision des joueurs, vitesse max de la balle pour chaque joueur)

Faire un code facilement compréhensible (beaucoup de commentaires et division en fonction, lisibilité, cohérence des attributs et des méthodes dans une même classe, attribution des rôles cf. Clean code)
Faire des fonctions facilement adaptable lors du passage à la fusion
Spécification des tests de validation et contrôle

Faire un diagramme UML
Développement format papier de l'écran d'accueil (Jouer une partie normal, modifier les paramètres d'une partie)

- partie normal : 1 humain contre 1 IA
- modification de la partie (nb de joueur, vitesse de la balle, …. voir dans les besoins non fonctionnels)