Tactibrême

Documentation technique

Auteurs: Gros Charlène, Pottier Loïc, Mudoy Jacques,
Horter Louise, Los Thomas, Gambier Clément

Date de rendu: March 18, 2025

Contents

1 Introduction

Tactibreme est un projet dédié à l'implémentation et l'entraînement d'une intelligence artificielle pour le jeu de plateau "Les tacticiens de Brême". L'objectif principal est de développer des agents utilisant l'apprentissage par renforcement pour maîtriser les règles et stratégies de ce jeu.

Le projet propose plusieurs fonctionnalités:

- Entraînement d'agents IA via l'apprentissage par renforcement.
- Possibilité de faire jouer des agents entre eux.
- Génération de statistiques et analyses des parties.
- Interface permettant de visualiser les parties.

2 Technologies et Dépendances

2.1 Langages et Frameworks

- **Python 3.8+**: Langage principal du projet.
- **PyTorch**: Framework d'apprentage automatique utilisé pour l'implémentation des réseaux de neurones.
- **Pygame**: Bibliothèque pour l'interface utilisateur.

2.2 Bibliothèques principales

- **torch**: Pour la création et l'entraînement des réseaux de neurones.
- **pygame**: Interface graphique pour visualiser les parties.
- **tqdm**: Pour l'affichage des barres de progression durant l'entraînement.
- **matplotlib et seaborn**: Pour la génération de graphiques et visualisations.
- **pandas**: Pour la manipulation des données générées.

2.3 Installation des dépendances

Toutes les dépendances nécessaires sont listées dans le fichier `requirements.txt` et peuvent être installées avec la commande:

```
pip install -r requirements.txt
```

Note importante: Le fichier `requirements.txt` installe PyTorch dans sa version CPU uniquement. Pour exploiter les capacités GPU et accélérer considérablement l'entraînement, il est nécessaire d'installer manuellement une version de PyTorch compatible avec votre matériel spécifique (CUDA, ROCm, etc.). Veuillez consulter la documentation officielle de PyTorch (`https://pytorch.org/get-started/locally/`) pour les instructions d'installation correspondant à votre configuration matérielle.

3 Architecture du Projet

3.1 Structure des dossiers

Le projet est organisé selon la structure suivante:

- **/ai**: Contient les implémentations des agents IA et des réseaux de neurones.
- **/csv**: Stockage des données d'entraînement et d'évaluation.
- **/document**: Documentation technique et informations de conception.
- **/logs**: Fichiers journaux générés pendant l'exécution.
- **/tests**: Tests unitaires pour vérifier le bon fonctionnement des composants.
- **/venv**: Environnement virtuel Python (généré lors de l'installation).
- **/checkpoints**: Modèles sauvegardés après entraînement (générés pendant l'exécution).

3.2 Composants principaux

Le projet est divisé en plusieurs composants interdépendants:

- **Moteur de jeu**: Implémentation des règles du jeu, la gestion de l'état du plateau, et la vérification des mouvements valides.
- **Module IA**: Agents intelligents basés sur l'apprentissage par renforcement.
- **Interface utilisateur**: Visualisation du jeu et des interactions.
- **Outils d'analyse**: Génération de statistiques et visualisations pour évaluer les performances.

4 Moteur de Jeu

4.1 Représentation du jeu

Le jeu "Les tacticiens de Brême" est implémenté avec les classes suivantes:

- **Board** (`board.py`): Gère l'état du plateau, les positions des pièces et les mouvements.
- **Paw** (`paw.py`): Représente une pièce du jeu avec son type, sa couleur et sa position.
- **Game** (`game.py`): Orchestrateur principal qui gère le déroulement d'une partie, l'entrainement des agents et la génération de statistiques.

4.2 Règles du jeu

Le jeu implémente les règles suivantes:

- Déplacement spécifique pour chaque type de pièce (Donkey, Dog, Cat, Rooster équivalent à : Tour, Fou, Cavalier, Reine).
- Phase de draft initiale où les joueurs placent une pièce à tour de rôle sur le plateau jusqu'à ce que toutes les pièces soient placées.
- Mécanisme de retraite lorsqu'une pièce est amenée sur la ligne de départ adverse.
- Condition de victoire lorsqu'une pile contient 4 pièces.

5 Module d'Intelligence Artificielle

Cette partie du projet est documentée en détail dans un document séparé dédié spécifiquement à l'intelligence artificielle. Voici un bref aperçu des composants principaux :

5.1 Architecture générale

Le système d'IA se divise en deux parties principales :

- Des réseaux de neurones spécialisés pour chaque phase du jeu (phase de draft et phase principale).
- Des agents qui utilisent ces réseaux pour prendre des décisions.

Le projet utilise l'apprentissage par renforcement avec une approche basée sur le Q-learning et l'experience replay.

5.2 Organisation des fichiers

Les composants d'IA sont organisés comme suit dans le dossier `/ai` :

- `base_agent.py` : Classe abstraite pour tous les agents
- `game_agent.py` : Agent pour la phase principale du jeu
- `draft_agent.py` : Agent pour la phase de draft
- `network.py` : Réseau de neurones pour la phase principale
- `draft_network.py` : Réseau de neurones pour la phase de draft
- `network.md` : Documentation détaillée sur l'architecture des réseaux

Pour les détails complets sur l'architecture des réseaux, les mécanismes d'apprentissage, les fonctions d'encodage et les stratégies d'entraînement, veuillez consulter le document dédié à l'IA du projet.

6 Outils et Utilitaires

6.1 Enregistrement et analyse

- **WriterBuffer** (`writerBuffer.py`): Enregistre les données de jeu au format CSV.
- **Logger** (`logger.py`): Système de journalisation pour le débogage et le suivi.
- **Parser** (`parser.py`): Analyse des fichiers CSV générés pendant l'entraînement.
- **Heatmap** (`heatmap.py`): Génération de cartes de chaleur pour visualiser les mouvements.

6.2 Statistiques

La classe `Stats` (`stats.py`) collecte diverses métriques pendant les parties:

- Taux de victoire pour chaque agent.
- Moyenne de mouvements par parties.
- blablabla
- blablabla

7 Utilisation du Projet

7.1 Entraînement d'un modèle

Pour entraîner un nouveau modèle:

```
python main.py train nom_du_modele --count=1000
```

Options disponibles:

- `--count`: Nombre de parties pour l'entraînement (défaut: 1000)
- `--load`: Chemins vers deux modèles existants pour continuer l'entraînement
- `--epsilon`: Taux d'exploration initial
- `--decay`: Taux de décroissance d'epsilon
- `--gamma`: Facteur de remise pour les récompenses futures
- `--lr`: Taux d'apprentissage
- `--ui`: Active l'interface graphique pendant l'entraînement

7.2 Confrontation entre modèles

Pour faire jouer deux modèles l'un contre l'autre:

```
python main.py record --blue=chemin/vers/modele1.pth --red=chemin/
    vers/modele2.pth --count=100
```

Options disponibles:

- `--count`: Nombre de parties à jouer
- `--blue`: Chemin vers le modèle pour le joueur bleu
- `--red`: Chemin vers le modèle pour le joueur rouge
- `--ui`: Active l'interface graphique

7.3 Génération de statistiques

Pour analyser les données générées:

```
python parser.py chemin/vers/fichier.csv
```

Pour générer des cartes de chaleur:

```
python heatmap.py chemin/vers/fichier.csv
```

7.4 Mécanisme d'apprentissage

L'entraînement suit ces étapes:

1. Initialisation des agents et de leurs réseaux
2. Pour chaque partie:
   (a) Phase de draft pour placer les pièces
   (b) Alternance des tours entre agents
   (c) Stockage des transitions (état, action, récompense, nouvel état)
   (d) Entraînement périodique sur des mini-batchs d'expériences
   (e) Mise à jour du paramètre epsilon pour réduire l'exploration
3. Sauvegarde des modèles entrainés

7.5 Améliorations possibles

- Optimisation des architectures de réseaux de neurones.
- Remplacement de l'ui pygame par une interface web (Gradio par exemple) afin de faciliter l'utilisation.
- Parallélisation de l'entraînement pour accélérer l'apprentissage (entrainement sur des batchs, threading pour les parties).