PROJET D'ALGORITHMIE ET DE STRUCTURES DE DONNÉES

TABLE DES MATIERES

Modalités du projet

DURÉE

Six semaines (jusqu'à la semaine 14 incluse).

GROUPE

Trois personnes par groupe.

ÉVALUATION

L'évaluation de votre projet sera composée de trois parties : qualité, algorithme et présentation. Un tournoi vous permettra d'ajouter un bonus sur cette note.

Qualité (15 %)

- Docstring
- Typage et signature des méthodes
- Documentation du code complexe
- Décomposition en fonctions

Algorithme (55%)

- Fonctionnement correct
- Algorithme approprié
- Complexité

Présentation (30%), en semaine 15

- Forme
- Clarté
- Respect du temps (5 minutes + 2 minutes de questions)

TEMPS À DISPOSITION

- Heures de laboratoire
- Temps personnel chez vous

TOURNOI (BONUS), EN SEMAINE 15

- 1ère place : +15%
- 2ème place : +10%
- 3ème place : +5%

CONTRAINTES

L'entièreté de votre code doit être codé par vous-même. Toute tentative de plagiat sera sévèrement punie. Tous les packages de bases de python peuvent être utilisés (les « built-in » packages, https://docs.python.org/3/py-modindex.html) ainsi que le package NumPy (https://numpy.org/). Tout autre package est proscrit.

NOTES

L'utilisation simple de l'algorithme Dijkstra's ne vous permet pas de dépasser la note de 4.

Toute tentative de plagiat, que ce soit du code sur internet ou du code fourni par un camarade, réduit automatiquement votre note des contrôles continus (CCs + projet) à 1, pour vous et la personne qui vous a donné son code.

Prérequis

Afin de lancer le projet, il vous faut préalablement installer le package pygame (https://www.pygame.org/news) :

Sur Windows :

- Ouvrir un terminal (invite de commande)
- Exécuter *python -m pip install -U pygame –user*
- Exécuter la commande *python -m pygame.examples.aliens* pour vérifier que tout ai bien été installé. Cela devrait ouvrir une fenêtre et lancer un petit jeu

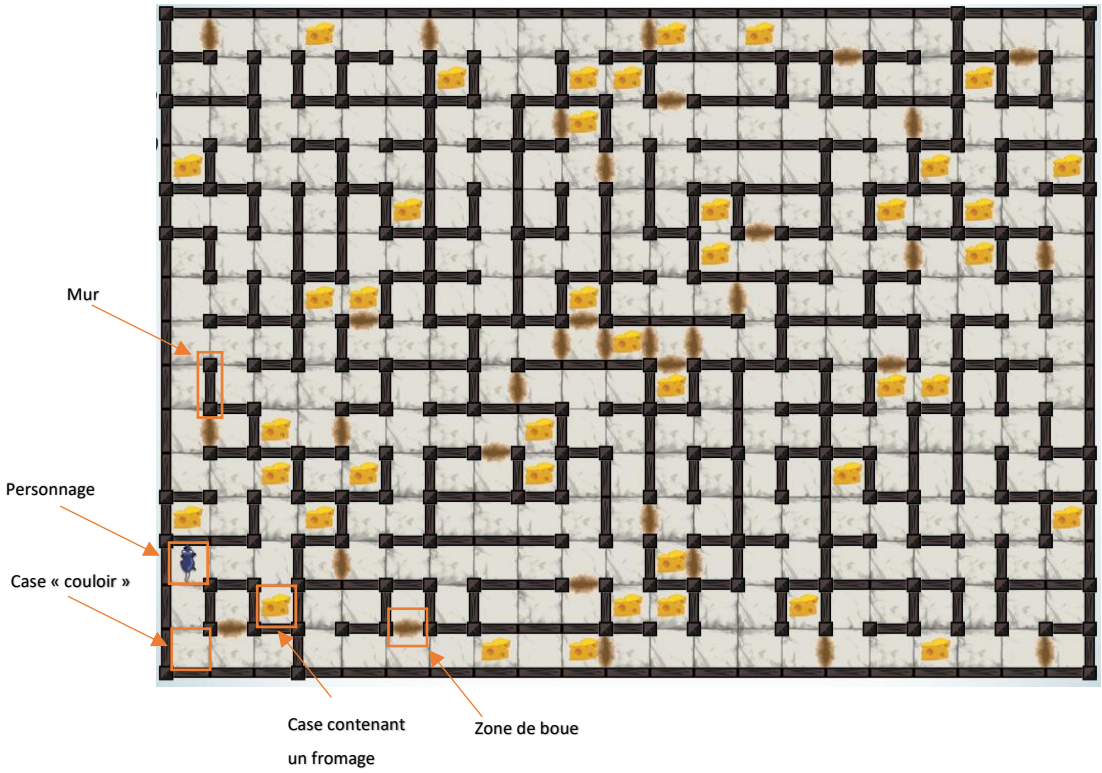
Sur MacOS :

- Ouvrir un terminal
- Exécuter *python3 -m pip install -U pygame –user*
- Exécuter la commande *python3 -m pygame.examples.aliens* pour vérifier que tout ai bien été installé. Cela devrait ouvrir une fenêtre et lancer un petit jeu

Introduction

L'objectif de ce projet est d'analyser un problème et de développer un algorithme permettant l'optimisation de la recherche d'un parcours dans un labyrinthe. Ce dernier doit permettre à un personnage de se déplacer dans le labyrinthe tout en mangeant le maximum de fromages, dispersés aléatoirement dans le labyrinthe.

Ci-dessous, un exemple de labyrinthe :



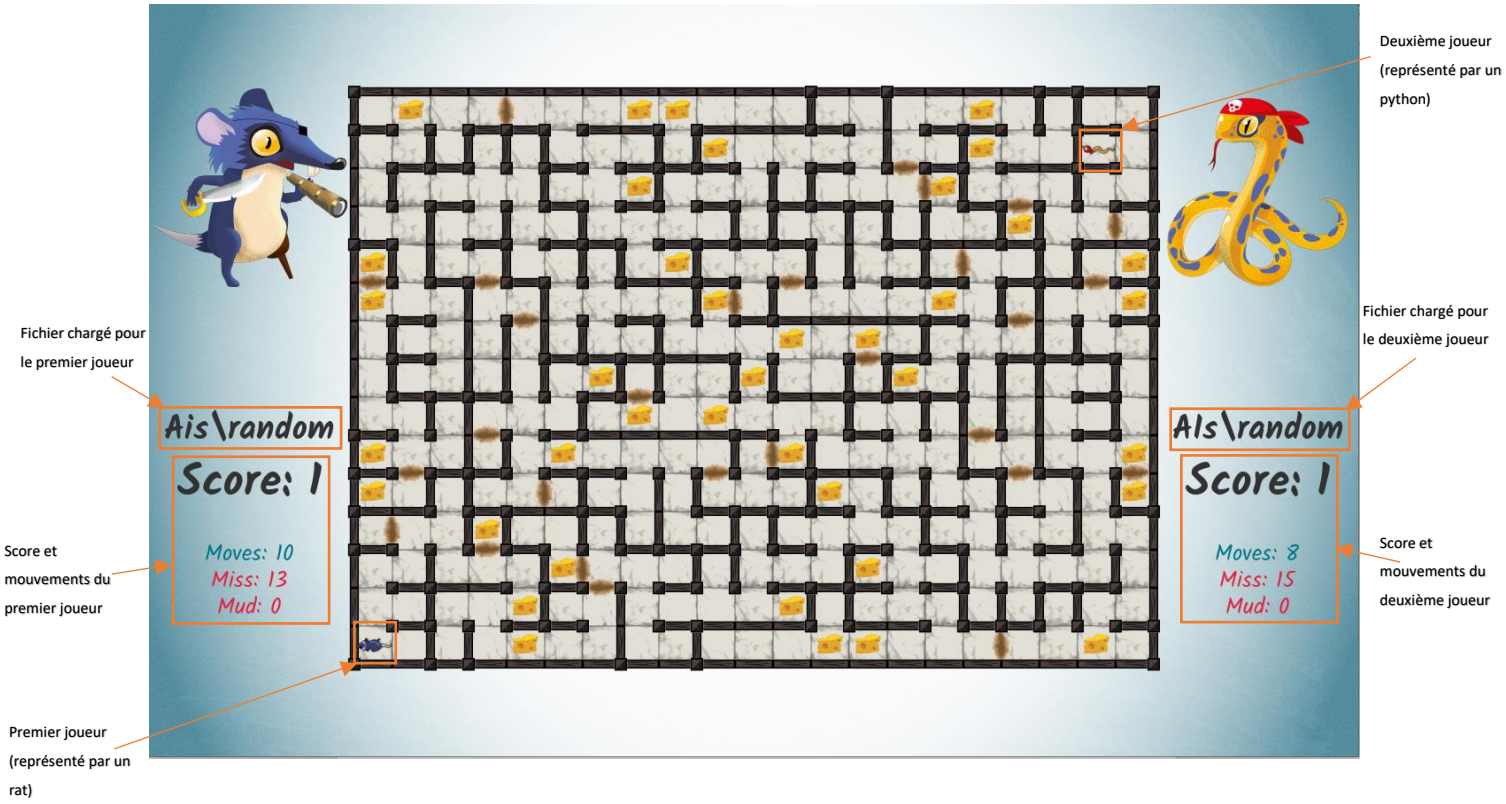
Le labyrinthe est constitué de cases de « couloirs », séparées entre-elles par des murs. Il est possible de se déplacer d'une case de couloir à une autre mais il est impossible de traverser un mur. Chaque déplacement d'une case à une autre coûte un mouvement. Il est également disposé de manières aléatoires des zones de « boues » entre 2 cases de « couloirs ». Ces zones de « boues » augmentent le nombre de mouvements nécessaires pour passer d'une case à un autre (de 2 à 10).

Le labyrinthe est créé de manière aléatoire et sera différent à chaque partie, tout comme la position des fromages et des zones de boue. L'objectif est de manger l'ensemble des fromages avec le moins de mouvement possible.

Mode deux joueurs

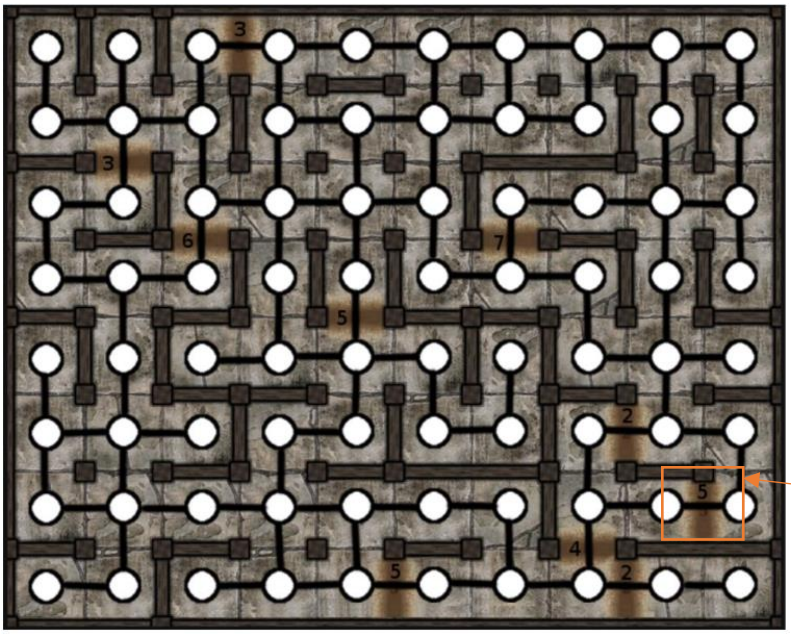
Il est également possible de lancer la partie avec 2 joueurs. Dans ce cas, le premier joueur ayant mangé plus de la moitié des fromages remporte la partie. En cas d'égalité, c'est le nombre de mouvements qui désigne le vainqueur.

La fenêtre du projet en mode « 2 joueurs » :



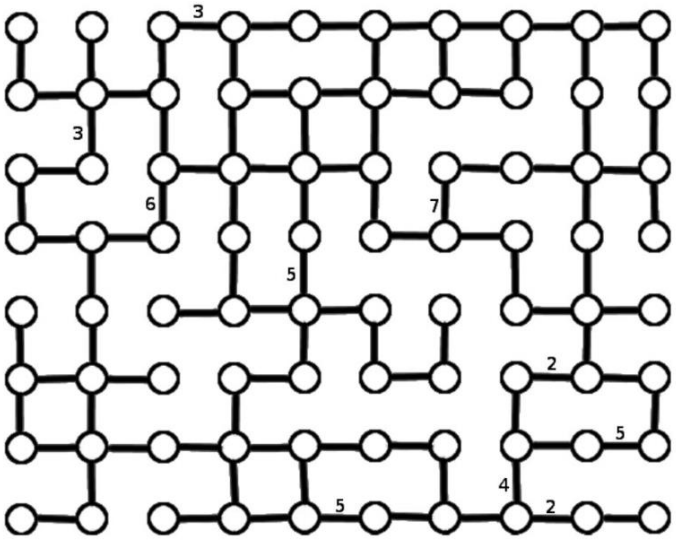
Visualisation en tant que graphe

Le labyrinthe peut être vu comme un graphe. Chaque case représente alors un nœud et chaque mouvement autorisé, une arête :



En regardant de plus près, on voit qu'une arête représentant un passage à travers de la boue contient également un nombre. Ce dernier indique le nombre de mouvements nécessaire pour passer d'un nœud à l'autre avec cette arête. Par défaut, l'ensemble des arêtes ont un coût de 1 mouvement.

En supprimant l'image de fond, on distingue plus facilement le graphe à travers lequel il faudra naviguer.



Par définition, un graphe est une collection de sommets dont certains sont reliés entre eux par des arêtes. Ces arêtes peuvent être pondérées.

Le projet

Commencez par cloner le repository sur votre ordinateur. L'entièreté de votre code et de vos modifications doivent être publiées sur GitHub régulièrement.

LANCER UNE PARTIE

Plusieurs paramètres peuvent être fourni au jeu lors du lancement d'une partie. Voici les paramètres importants ainsi que les valeurs possibles :

- --rat
  - human
  - Chemin du fichier représentant une ia (dans le dossier AIs)
- --python
  - human
  - Chemin du fichier représentant une ia (dans le dossier AIs)
- --desactivate_animations (en cas d'ordinateur lent)
- --fullscreen

Pour lancer une partie, ouvrez un terminal (invite de commande) dans le dossier du projet puis lancez une des commandes suivantes :

Sur MacOS :

Une partie solo avec un humain :

```
python3 pyrat.py --rat human
```

Une partie solo avec une ia :

```
python3 pyrat.py --rat AIs/random.py
```

Une partie à deux joueurs avec des humains :

```
python3 pyrat.py --rat human --python human
```

Une partie à deux joueurs avec un humain et une ia :

```
python3 pyrat.py --rat human --python AIs/random.py
```

Une partie à deux joueurs avec deux ia :

```
python3 pyrat.py --rat AIs/random.py --python AIs/random.py
```

Sur Windows

Une partie solo avec un humain :

```
python pyrat.py --rat human
```

Une partie solo avec une ia :

```
python pyrat.py --rat Ais\random.py
```

Une partie à deux joueurs avec des humains :

```
python pyrat.py --rat human --python human
```

Une partie à deux joueurs avec un humain et une ia :

```
python pyrat.py --rat human --python Ais\random.py
```

Une partie à deux joueurs avec deux ia :

```
python pyrat.py --rat Ais\random.py --python Ais\random.py
```

Vous pouvez également ajouter les drapeaux --desactivate_animations et --fullscreen en cas de besoin.

Si c'est un humain qui joue, vous pouvez contrôler votre personnage avec les flèches du clavier. S'il y a deux humains, le second personnage se contrôles avec le pavé numérique (touches 4,5,6,8).

CRÉER VOTRE « INTELLIGENCE ARTIFICIELLE »

Afin de créer votre propre intelligence artificielle, il vous suffit d'ajouter un fichier python dans le dossier AIs (exemple : AIs/mon_ai.py). Ce fichier python doit obligatoirement contenir 2 méthodes : *preprocessing* et *turn*. Un fichier template.py est disponible dans le dossier AIs. Ce dernier contient toute la structure nécessaire pour créer votre propre intelligence artificielle. Vous pouvez recopier les différentes méthodes et leurs signatures puis y ajouter votre logique.

Constantes prédéfinies

Quatre constantes sont prédéfinies (MOVE_DOWN, MOVE_LEFT, MOVE_RIGHT, MOVE_UP) et représentent les mouvements possibles (il ne faut pas modifier leurs valeurs !). Une de ces constantes doit être retourné par la fonction *turn* afin d'informer le jeu du mouvement sélectionné par votre ia. Il est également possible de créer une constante TEAM_NAME pour définir le nom de l'intelligence artificiel (par défaut, c'est le nom du fichier).

Fonction de prétraitement (preprocessing)

Lors du lancement de la partie, la fonction *preprocessing* est appelée. Cette dernière permet de faire quelques calculs avant le départ de la partie afin de préparer vos prises de décisions lors des tours de jeu.

Fonction de tour de jeu (turn)

Une fois la phase de prétraitement terminée, les tours de jeu s'enchainent et la fonction *turn* est appelée régulièrement. A chaque fois, cette fonction renvoie une décision de mouvement et cette dernière est ensuite appliquée. C'est ici que vous devrez prendre la décision en se basant sur les paramètres fournis par le jeu (position des fromages, position de l'autre joueur, etc).

Attention, un temps maximum d'exécution est défini pour les méthodes *preprocessing* et *turn*. Si vous dépassez le temps imparti, votre tour sera alors sauté.

Il vous est également possible (et conseillé) de créer des classes et d'autres fonctions pour structurer votre code.

Structuration des données

Lors des appels de *preprocessing* et de *turn*, les paramètres suivants sont fournis :

- La carte du labyrinthe
- La largeur du labyrinthe
- La hauteur du labyrinthe
- La position du joueur
- La position de l'adversaire
- Votre score (uniquement pour *turn*)
- Le score de l'adversaire (uniquement pour *turn*)
- Les positions des fromages
- Le temps alloué à l'exécution de la méthode.

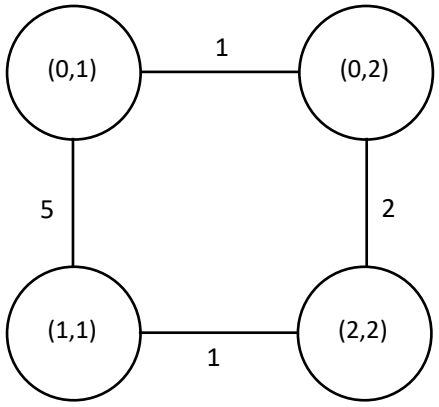
Le labyrinthe

La carte du labyrinthe est structurée de la manière suivante :

$$G(S, A) := dict(tuple(int, int), dict(tuple(int, int), int))$$

où la clé du dictionnaire ($tuple(int, int)$) représente un sommet dans le graphe G, et la valeur associée ($dict(tuple(int, int), int)$) représente l'ensemble des arêtes de ce sommet avec leur coût de déplacement respectif.

Exemple :



(0,1) -----1----- (0,2)
(0,1) -----5----- (1,1)
(0,2) -----1----- (0,1)
(0,2) -----2----- (1,2)
(1,1) -----5----- (0,1)
(1,1) -----1----- (1,2)
(1,2) -----2----- (0,2)
(1,2) -----1----- (1,1)

Ce graphe sera alors représenté de la manière suivante dans le dictionnaire :

```
labyrinthe : dict = {
            (0,1) : {(0,2) : 1, (1,1) : 5},
            (0,2) : {(0,1) : 1, (1,2) : 2},
            (1,1) : {(0,1) : 5, (1,2) : 1},
            (1,2) : {(0,2) : 2, (1,1) : 1}
      }
```

Un sommet, représentant une case dans la carte, est défini par un tuple contenant la position x et y de ce dernier. Pour créer une tuple, il suffit d'écrire :

```
position: tuple = (0, 1)
```

Pour accéder aux éléments d'un tuple, vous pouvez utiliser les indices, comme sur une liste :

```
x: int = position[0]
y: int = position[1]
```

Il est également possible de récupérer l'ensemble des arêtes associées à un sommet (et donc à une position) en utilisant le tuple précédemment créé :

```
chemins: dict = labyrinthe[position]
```

Voici un exemple (seulement les quelques premières lignes) d'un vrai labyrinthe :

```
labyrinthe : dict = {
          (0,8): {(0,9): 10, (1,8): 1},
          (0,9): {(0,8): 10},
          ...
    }
```

Ici, le sommet à la position (0,8) est connecté aux sommets (0,9) et (1,8). Pour passer du sommet (0,8) au sommet (0,9), cela coûte 10 mouvements alors que le coût pour aller au sommet (1,8) est seulement de 1 mouvement.

Position des joueurs et des fromages

La position des joueurs sont également des tuples contenant la position x et y. Les positions des fromages est une liste de tuple contenant à chaque fois la position x et y du fromage.

Tous ces éléments sont également documentés dans le fichier template.py !

Présentation

La présentation se déroulera pendant la semaine 15 durant les cours d'algorithmie avancé pour les pleins temps et durant le laboratoire pour les temps partiels (sauf changements non prévu).

Lors de la présentation, il vous faudra nous expliquer dans un laps de temps réduit le fonctionnement de votre algorithme. Sur quel algorithme le vôtre est basé (Dijkstra's, A*, etc) et comment vous avez dû l'adapter pour que cela fonctionne dans ce projet. Il serait également intéressant de nous faire part des astuces que vous avez utilisé pour améliorer les performances de votre algorithme. Tout élément spécifique au fonctionnement de **votre** algorithme (non de celui de base) peut être intéressant à expliquer.

Faites attention, la présentation ne dure que **5 minutes** (+ 2 minutes de questions). Il ne faudra alors pas commencer à nous montrer en détail votre code mais plutôt nous faire une présentation rapide de tous les systèmes que vous avez mis en place pour que votre algorithme soit le plus performant possible, que ce soit au niveau du score mais également au niveau de la complexité.

Tournoi

Le tournoi se déroulera pendant la semaine 15 durant le cours d'algorithmie avancé des temps partiels (mardi 19h à 20h45).

A la fin du semestre, un tournoi sera organisé, vous permettant d'obtenir un bonus sur votre note. Ce dernier mettra à l'épreuve les différents algorithmes que vous aurez créé. Puisqu'il aura beaucoup d'équipes, il n'est pas possible d'organiser une phase de poule. Les matchs seront tirés aléatoirement et le meilleur à 1 à 5 victoires (selon la phase) passera à la phase suivante.

Une victoire est déclarée lorsqu'un algorithme arrive à manger plus de la moitié des fromages. En cas d'égalité, le nombre de mouvements est utilisé pour départager les deux algorithmes. Si cela ne permet toujours pas de les départager, un match humain vs humain décidera du vainqueur.

Dans le cas où un algorithme possède des bugs ou ne fait pas déplacer son personnage, il est considéré comme disqualifié. L'équipe disqualifiée sera alors remplacée par la meilleure équipe éliminée.

Les paramètres utilisés lors du tournoi :

- Largeur de 25 et hauteur de 23
- Probabilité de mur à 0.7
- Probabilité de boue à 0.1 et pénalité de mouvement maximum à 10
- 41 morceaux de fromages (donc victoire à 21 morceaux mangés)
- 3s de temps de préparation (*preprocessing*) et 100ms par tour (*turn*)

Ces paramètres sont ceux par défaut dans le jeu. Si vous désirez les changer pour vérifier que votre algorithme fonctionne dans d'autres situations, vous pouvez analyser le fichier parameters.py (qui se trouve dans le dossier imports).

Quelques liens utiles :

Cours PyRat :

http://formations.telecom-bretagne.eu/pyrat/

Dijkstra's :

https://brilliant.org/wiki/dijkstras-short-path-finder/

https://www.youtube.com/watch?v=GazC3A4OQTE

A* :

https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

https://www.youtube.com/watch?v=ySN5Wnu88nE

https://www.youtube.com/watch?v=aKYlikFAV4k&feature=youtu.be

Problème du voyageur de commerce :

https://interstices.info/le-probleme-du-voyageur-de-commerce/

Optimisation du code en python :

http://www.xavierdupre.fr/blog/2014-04-12_nojs.html

https://pro-domo.ddns.net/blog/optimiser-son-code-python.html

Utilisation de NumPy :

https://numpy.org/doc/stable/user/quickstart.html

CRÉDITS

http://formations.telecom-bretagne.eu/pyrat/?page_id=264