Teodoro Douglas, Stettler Christian, Soukouti Nader et Barreiro Lindo Flávio

Projet d'Algorithmes et structures de données

Modalité du projet

Durée

- 6 semaines (jusqu'à la semaine 14 inclue)

Groupes

- Max 2 personnes

Évaluation :

L'évaluation sera décomposée en deux parties : qualité et algorithmes.

Voici les sous-parties et le poids de chacune :

- Qualité (25%)
  - Docstring (5%)
  - Typage (variables et types de retour) (5%)
  - Documentation du code complexe (3%)
  - Décomposition de fonctions (12%)
- Algorithmes (75%)
  - Fonctionnement correct (40%)
  - Algorithme approprié (25%)
  - Performance (via Profiling de PyCharm) (10%)

Bonus

- 1ère place : +30%
- 2ème place : +20%
- 3ème place : +10%

Temps à disposition

- Heures de cours
- Heures de laboratoire

Contraintes sur le code :

Toutes les fonctions et tous les algorithmes doivent être codés par vous-mêmes ! Seule l'utilisation du package *numpy* est autorisée. Ce package n'est pas obligatoire et c'est à chaque équipe de décider si cela apportera un gain supplémentaire de performance.

Planning

Semaine 9

- Présentation du sujet
- Manipuler PyRat pour se familiariser avec le logiciel
- Programmer quelques fonctions simples
- Programmer une première IA un peu meilleure que l'aléatoire

Semaine 10

- Programmer un (ou des) programmes pour ramasser un morceau de fromage dans le labyrinthe
- Vous serez guidés pour l'écriture du parcours en largeur
- Comment les deux parcours se comparent ? Regarder le temps de calcul en fonction des paramètres du labyrinthe

Semaine 11

- Contrôle continu

Semaine 12

- Ramasser plusieurs morceaux de fromage dans le labyrinthe
- Comparer le temps de calcul de Roy-Warshall et Dijkstra
- Regarder l'influence des paramètres du labyrinthe sur la taille des chemins trouvés ou le temps de calcul
- Regarder l'impact des choix d'implémentation sur le temps de calcul

Semaine 13

- Ramasser plusieurs morceaux de fromage dans le labyrinthe
- Tester différentes heuristiques ou algorithmes approchés
    - Regarder l'impact sur le temps de calcul
    - Regarder l'impact sur le résultat
    - Comparer avec l'algorithme exhaustif

Semaine 14

- Rendu du projet le lundi 06.01.2020 à 17h00 via GitHub
- Championnat le jeudi 09.01.2020 à 19h15

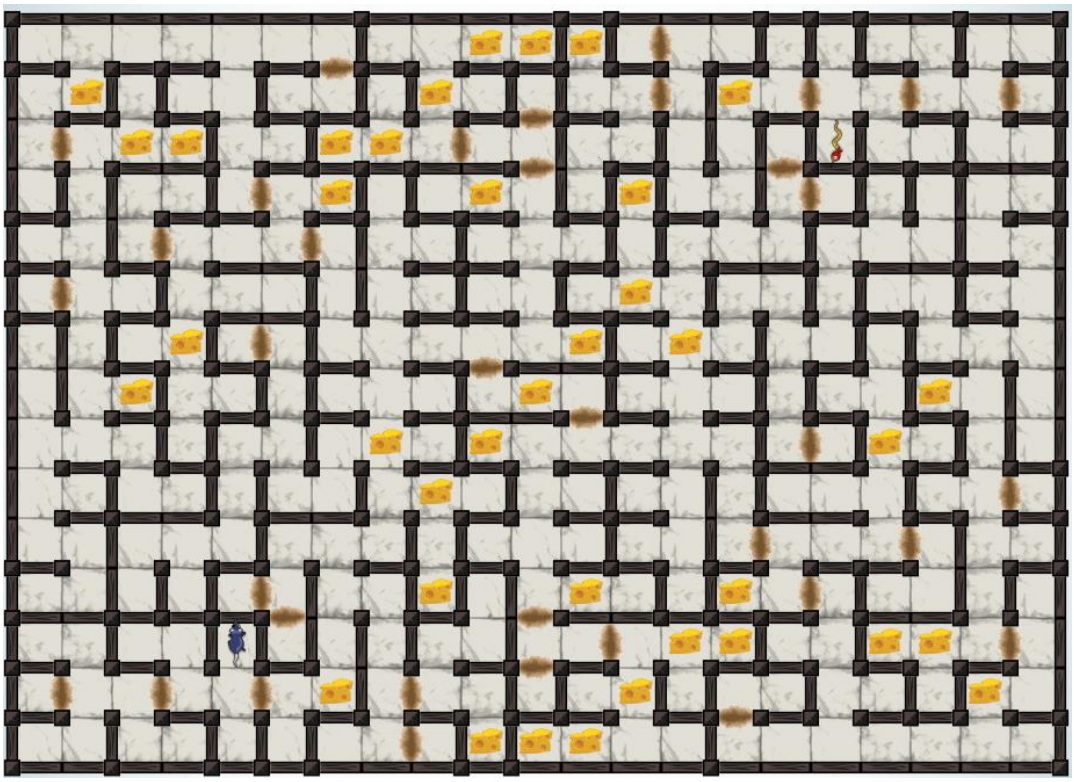
Teodoro Douglas, Stettler Christian, Soukouti Nader et Barreiro Lindo Flávio

Introduction

Le projet de ce module est fait en Python et son but est développer, en binômes, un algorithme permettant d'optimiser un parcours dans un labyrinthe.

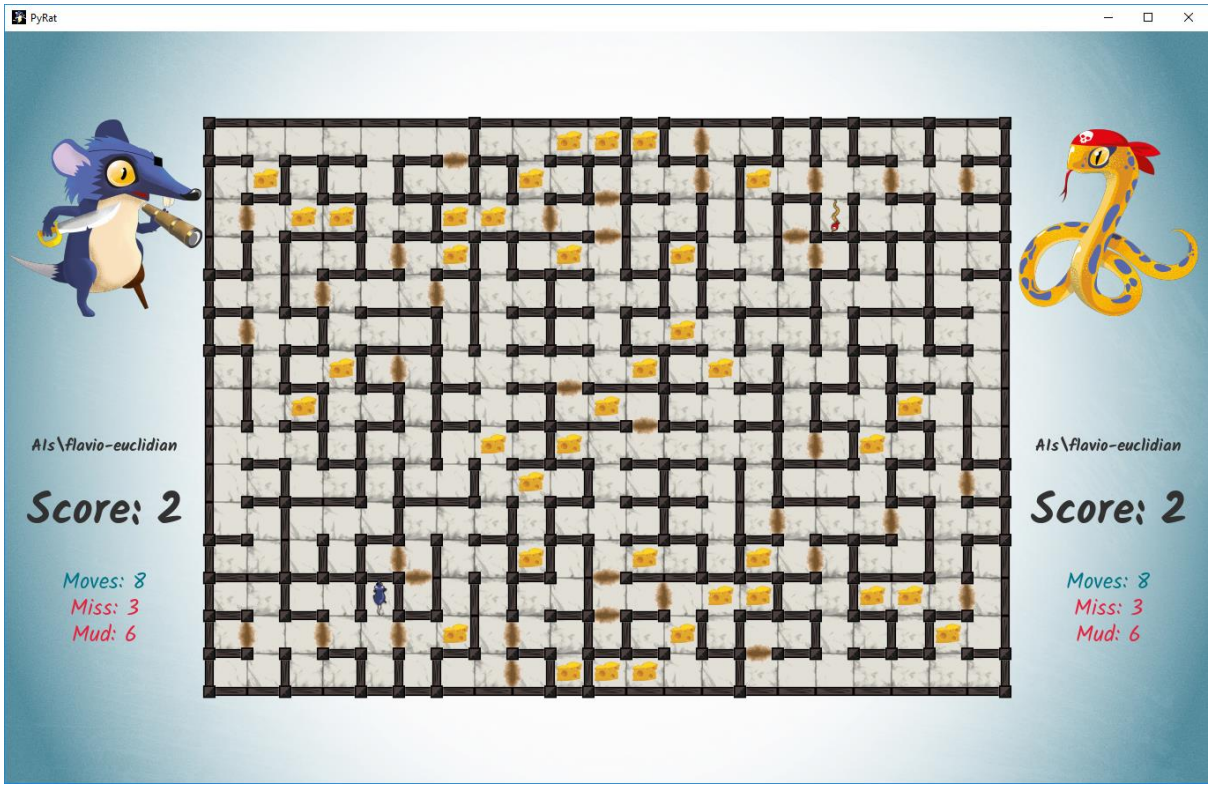
L'algorithme développé doit permettre à un personnage (un rat ou un python) de manger des fromages disposés aléatoirement dans un labyrinthe le plus rapidement possible.

Voici un exemple de labyrinthe :



Dans ce labyrinthe, chaque déplacement coûte un mouvement, sauf où il y a de la boue. Si un personnage essaie de traverser de la boue, cela peut coûter entre 2 et 10 mouvements pour aller d'une case à une autre. Aussi, il n'est pas possible de passer à travers un mur.

De plus, vous pouvez jouer en solo ou en duo :

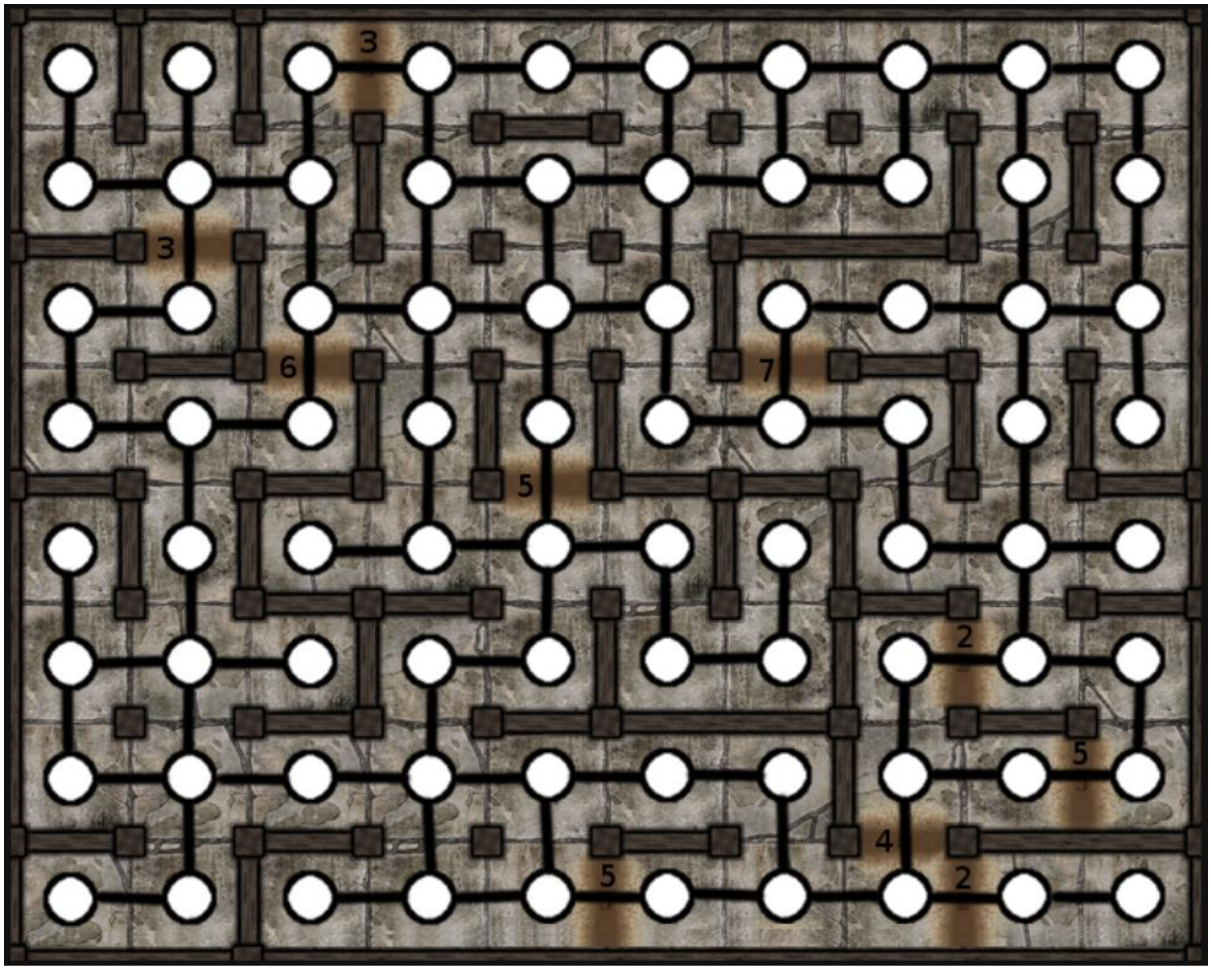


Dans le mode « deux joueurs », le premier qui a mangé plus de la moitié des fromages, gagne la partie.

Teodoro Douglas, Stettler Christian, Soukouti Nader et Barreiro Lindo Flávio

Visualisation en tant que graphe :

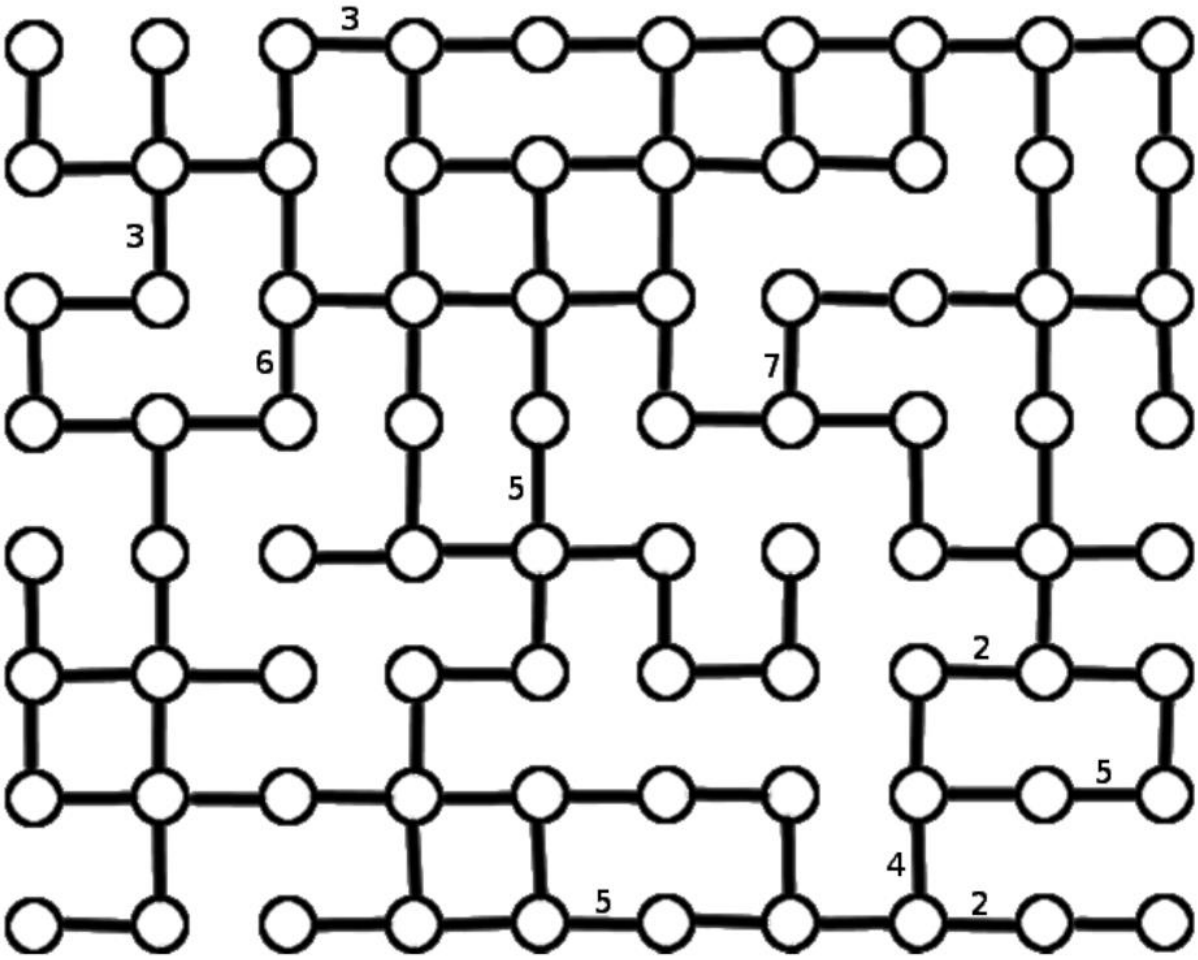
En regardant le labyrinthe, nous pouvons constater que chaque case représente un nœud et que chaque mouvement possible est une arête.



En supprimant l'image, voici le résultat. C'est à travers ce graphe que vous devrez naviguer pour manger vos fromages.

Par définition, un graphe est une collection de sommets (ou nœuds, ou vertices en anglais) dont certains sont reliés entre eux par des arêtes (ou liens, ou branches, ou arcs, ou edges en anglais).

Le nombre à côté d'une arête représente le coût pour aller entre les deux sommets liés par l'arête. Par défaut, le coût est égal à 1.



Le projet

Comment lancer le projet

Cliquez sur ce lien https://classroom.github.com/g/HcLsJpGX pour créer une équipe et cloner le repository contenant le jeu.

Une fois le repository cloné, vous pouvez lancer le jeu.

Démarrer une partie :

Les étapes décrites ci-dessous sont détaillées dans README.md.

1. Allez dans le dossier racine du jeu (où pyrat.py se trouve)
2. Ouvrez une ligne de commandes et lancez :

Sur MacOS :

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

Sur Windows :

```
python.exe pyrat.py --rat AIs\random.py
```

Sur Ubuntu :

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

Une partie à plusieurs joueurs (humains ou bots) :

```
python pyrat.py --rat AIs\random.py --python  AIs\random2.py
```

Une partie avec un humain :

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

Vous pouvez utiliser les flèches pour contrôler votre personnage. Si vous jouez à deux humains, alors il faudra aussi utiliser le pavé numérique (8, 4, 5, 6).

Créer votre « Intelligence Artificielle » :

Un fichier template.py est disponible dans le dossier AIs. Il possède toute la structure nécessaire pour créer votre algorithme.

Constantes prédéfinies :

Ce sont des constantes utiles. Les mouvements qui y sont définis (`MOVE_XXX`) sont à renvoyer par la fonction `turn` pour indiquer à PyRat comment se déplacer. De plus, le nom de votre IA est à définir dans la constante `TEAM_NAME`.

Fonction de prétraitement :

Au début de toute partie de PyRat, la fonction `preprocessing` est appelée. Cela vous permet de faire quelques calculs en amont, afin de préparer vos décisions lors des tours de jeu.

Fonction de tour de jeu :

Une fois la phase de prétraitement terminée, les tours de jeu s'enchaînent, et la fonction `turn` est appelée régulièrement. A chaque fois que cette fonction renvoie une décision de mouvement, le tour est appliqué.

Attention : Les fonctions `preprocessing` et `turn` ont un temps attribué limité. Si votre prétraitement est trop long, vous allez rater les premiers tours de jeu. De même, si votre fonction de tour est trop longue, il se peut que vous vous déplaciez un tour sur deux !

Pour finir, vous n'êtes pas restreints aux deux fonctions utiles à PyRat (`preprocessing` et `turn`). Vous pouvez créer tout le code que vous voulez, que ce soit des classes ou des fonctions.

Comment les données sont structurées

A chaque appel de fonction (`preprocessing` et `turn`), plusieurs éléments vous seront envoyés :

- Une carte du labyrinthe
- La position du joueur
- La position de l'adversaire
- Les positions des fromages
- La hauteur et la largeur du labyrinthe
- Les points (fromagés mangés) des deux joueurs
- Le temps autorisé pour chaque exécution

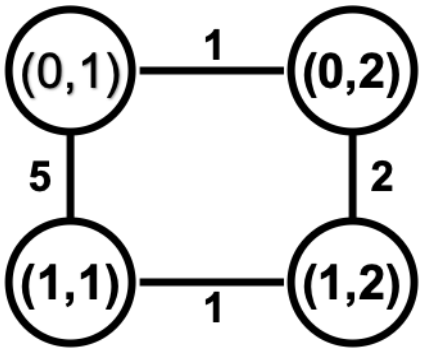
Et voici leur structure :

La carte du labyrinthe est structurée de la façon 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 d'arêtes avec leur coûts respectifs.

Exemple : Pour le graphe ci-dessous,



(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)

La carte sera représentée par la structure :

```
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 tuple (sommet), représentant une position dans la carte, est un ensemble de deux valeurs. En Python, pour créer un tuple, il suffit d'écrire

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

Pour accéder aux éléments d'un tuple, vous pouvez utiliser les indices

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

Nous pourrions utiliser la variable `position` pour obtenir un élément stocké dans le dictionnaire du labyrinthe.

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

Voici un autre exemple d'un vrai labyrinthe (seule une petite partie de l'affichage est montrée ici) :

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

Par exemple, la position `(0,8)` est connectée aux positions `(0,9)` et `(1,8)`. Aussi, cela coûte 10 mouvements pour aller de `(0,8)` à `(0,9)` alors que le coût pour aller à `(1,8)` est de seulement 1.

Les positions des joueurs sont également des pairs. Et pour finir, les positions des fromages sont contenues dans une liste de pairs.

Tous les éléments mentionnés ci-dessus sont déjà affichés dans la fonction `preprocessing` du fichier template.py. Vous pouvez supprimer cet affichage dans le rendu final.

Étapes à développer

Semaine 9

- Manipuler PyRat pour se familiariser avec le logiciel
- Programmer une première IA un peu meilleure que l'aléatoire

Tâche 0

- Modifier le nom de l'équipe dans la constante `TEAM_NAME`

Tâche 1

- Comprendre les paramètres de la fonction `turn`
- Coder une fonction permettant de renvoyer les bonnes directions au programme dans la fonction `turn` à partir d'une position initiale et une position finale

Tâche 2

- Réutiliser le code du fichier random.py pour la fonction `turn`
- Modifier la fonction `turn` pour stoker les positions déjà visités

Semaine 10

- Programmer un (ou des) programmes pour ramasser un morceau de fromage dans le labyrinthe

Tâche 3

- Parcours en profondeur

Tâche 4

- Parcours en largeur

Semaine 12

- Ramasser plusieurs morceaux de fromage dans le labyrinthe
- Regarder l'influence des paramètres du labyrinthe sur la taille des chemins trouvés ou le temps de calcul
- Regarder l'impact des choix d'implémentation sur le temps de calcul

Tâche 5

- Implémenter l'algorithme de Roy-Warshall

Tâche 6

- Implémenter l'algorithme de Dijkstra

Tâche 7

- Comparer les temps de calcul

Semaine 13

- Ramasser plusieurs morceaux de fromage dans le labyrinthe
- Tester différentes heuristiques ou algorithmes approchés
    - Regarder l'impact sur le temps de calcul
    - Regarder l'impact sur le résultat
- Comparer avec l'algorithme exhaustif (semaine 12)

Tâche 8

- Se déplacer globalement dans la direction du fromage (en ignorant la présence de murs/obstacles)

Tâche 9

- Aller systématiquement aux k voisins les plus proches

Semaine 14

- Livrer le projet

Tâche 10

- Upload la version finale sur GitHub

Tournoi

À la fin du semestre, un tournoi sera réalisé qui va mettre à l'épreuve les algorithmes des étudiants.

Puisqu'il y aura beaucoup d'équipes, il n'est pas possible d'organiser une phase de poule. Les matchs seront tirés aléatoirement et le meilleur à 2 victoires, passe à la phase suivante.

Une victoire est déclarée lorsqu'un algorithme arrive à manger plus de fromages que l'autre. En cas d'égalité, le nombre de mouvements est utilisé. Si cela ne permet toujours pas de départager les deux équipes, alors un match *humain* vs. *humain* décidera du vainqueur.

Dans le cas où un algorithme possède des bugs ou ne bouge pas lors du tournoi, il est considéré comme disqualifié. L'équipe disqualifiée sera remplacée par la meilleure équipe éliminée, en prenant en compte les facteurs utilisés pour déclarer une victoire. Une disqualification n'impacte pas la note attribuée pour le projet, seul le bonus est annulé.

Voici les paramètres utilisés lors du tournoi :

- Largeur 25, hauteur 23
- Probabilité de mur à 0.7
- Probabilité de boue à 0.1 et pénalité maximum à 10
- 41 morceaux de fromage (victoire à 21)
- 3s de temps de préparation et 100ms par tour.

Ces paramètres vous sont déjà fournis par défaut mais vous pouvez les modifier dans imports/parameters.py. Pensez bien à modifier les paramètres pour vos tests mais à vérifier que votre algorithme est également performant avec les paramètres ci-dessus.

Quelques liens utiles

Cours PyRat :

http://formations.telecom-bretagne.eu/pyrat/?page_id=37

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://youtu.be/aKYlikFAV4k

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://docs.scipy.org/doc/numpy/user/quickstart.html

Crédits

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