DEHEUVELS Joakim
GUIRIMAND Thibaut

Projet Morpion

Introduction

Le morpion est en soit un jeu relativement simple dont les stratégies sont peux nombreuses, ce qui peut le rendre rébarbatif. Une alternative a donc été inventée pour pimenter le jeu, elle consiste à intégrer une grille de morpion dans chaque case d'une grille de morpion principale. La case d'un morpion secondaire dans laquelle joue un des deux adversaires défini la case du morpion principal contenant la grille où jouera le joueur suivant.

Le projet Morpion a consisté à développer un jeu de morpion amélioré en Python.

L'intérêt s'est donc d'abord porté sur le développement d'un moteur de jeu gérant un ensemble de règles et permettant de jouer dans un terminal UNIX.

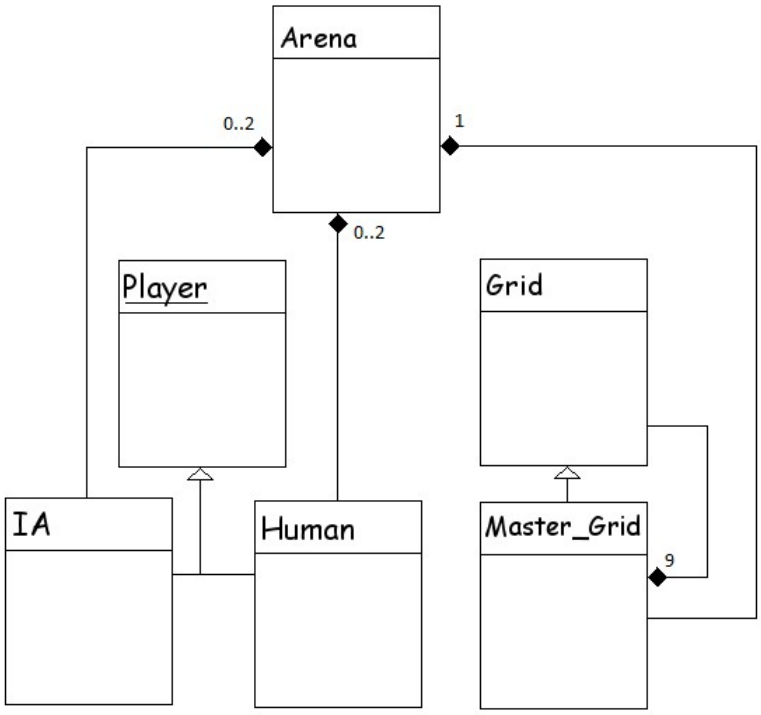
Le deuxième point intéressant correspond au développement d'une interface graphique grâce à un algorithme en programmation événementielle.

Enfin il a fallu créer une intelligence artificielle capable de jouer en respectant les règles.

Matériel et Méthodes

Le développement du projet c'est fait en python. Il a fallu pour cela utiliser plusieurs bibliothèques standards de python : numpy, random, time, threading. Et une bibliothèque non standard : pygame.

Le Python étant un langage orienté objet, plusieurs classes ont été crées selon le diagramme de classes suivant :



DEHEUVELS Joakim
GUIRIMAND Thibaut

Le moteur de Jeu :

Le moteur de jeu définit les différentes règles du jeu, les lie entre elles pour créer un jeu minimaliste fonctionnel.

Pour se faire, différentes classes ont été créées :

- La première est la classe **Grid** qui contient, au départ, une matrice de 0 de taille 3x3. Cette matrice est donc une grille de morpion.

  Grid utilise différentes méthodes permettant d'interagir avec la matrice notamment pour y ajouter un élément lorsqu'un joueur décide d'y jouer ou pour savoir si une ligne gagnante existe.

  Les éléments stockés dans une matrice sont des 0 si la case n'a jamais été jouée, 1 si la case a été jouée par le premier joueur, des 4 si la case a été joué par le second joueur. Ainsi pour obtenir le résultat d'une grille, de tester la somme des éléments d'une grille. Une ligne dont la somme sera multiple de 3 signifiera que la grille est gagnée par le joueur 1 et une somme de 12 pour le joueur 2.
- La seconde classe se nomme **Master_Grid** . Elle hérite de Grid et contient donc une matrice du même genre ainsi que les méthodes qui lui sont dédiées. En plus de cela, elle définit une seconde matrice de taille 3x3 contenant 9 objets de type Grid.

  Master_Grid a des méthodes permettant d'ajouter un élément dans un objet Grid particulier à un emplacement particulier puis de tester si cet objet Grid est gagnant. Il est ainsi possible de remplir la grille héritée de Grid avec les résultat des sous-grilles.
- Il existe une classe abstraite **Player** visant à définir différents type de joueurs.
- La classe **Human** hérite de player elle définit la méthode de réflexion de base d'un joueur, rentrer des coordonnées de cases et les renvoyer.
- Enfin, la classe **Arena** est l'aire de jeu. Elle instancie un élément de type Master_Grid ainsi que deux joueurs qui vont jouer alternativement dans les différentes grilles. Cette classe définit notamment, à partir du coup effectué par un joueur et des sous-grilles gagnées, les sous-grilles dans lesquelles le joueur suivant sera autorisé à jouer.

La classe Arena peut donc être instanciée dans le script principal auquel cas, le jeu sera lancé grâce à une méthode nommée play().

L'interface graphique :

Pour simplifier la jouabilité, plutôt que de demander à l'utilisateur de rentrer des coordonnées en console, une interface graphique a été développée. Elle utilise la bibliothèque pygame que l'utilisateur doit installer.

Les méthodes liées à pygame ont été développées dans la classe Arena. Cette bibliothèque permet la création d'un fenêtre graphique dans laquelle pourront être superposés des images (notamment un fond et une grille de jeu).

La partie intéressante de cette interface graphique est la gestion d'événements et en particulier la gestion des clics de la souris.

Une boucle infinie test en permanence si un clic a eu lieu et à quel emplacement. Il a donc été possible de définir si un clic a eu lieu dans une des cases d'une des grilles affichées à l'écran.

Lors d'un clic dans une cellule, les coordonnées de la cellules sont stockées dans une variable de classe de Arena, la récupération de coordonnées dans la grille est suspendue. Les événements et

l'affichage étant gérés dans un thread particulier, le moteur de jeu n'est pas bloqué et peu confirmer, dans une autre variable de classe, que la zone de clic est valable ou non. La boucle gérant les événement et l'affichage va donc tester la variable confirmant ou non le clic jusqu'à son changement de statut par le moteur de jeu.

Si les coordonnées ont été validées le moteur graphique va afficher dans la grille l'élément de jeu correspondant au joueur.

Dans le cas inverse, une erreur pourra être retournée.

Dans tout les cas, les variables (coordonnées et validation) seront réinitialisées pour laisser la place à un nouvel événement.

L'intelligence artificielle :

Pour permettre aux personnes sans adversaires de pouvoir jouer, une intelligence artificielle (IA) a été développées. Bien évidement, il est envisageable de faire jouer deux IA entre elles.

Cette IA est défini grâce à la classe **IA** qui définit une méthode de réflexion particulière. Elle utilise donc des méthode portant le même nom que celles utilisées dans le déroulement du jeu pour la classe Human. Il suffit donc de définir une IA à la place de Human.

Ce qui diffère est la méthode réflexion. Là ou Human devait définir des coordonnées manuellement ou en cliquant, IA va calculer les coordonées.

Deux niveaux ont été implémentés :

- D'abord la génération aléatoire de coordonnées. Cela nécessite donc parfois plusieurs génération de coordonnées jusqu'à en trouver qui soient valables.
- La deuxième solution qui est définit comme la solution par défaut consiste à tester les éléments présents dans la grille. Si des lignes gagnantes peuvent être complétées, une le sera aléatoirement. Si aucune ligne du joueur ne peut être complétée mais qu'une ou plusieurs lignes de l'adversaire peut être bloquée, une des ligne sera bloquée. Enfin si aucune des étapes précédentes ne peut être remplie, une case sera choisie au hasard.

  Cependant pour éviter de prédire les réactions de l'IA, une probabilité d'apparition de 0.8 est appliquée à la première action (gagner une grille), puis de nouveau une probabilité de 0.8 pour la seconde (bloquer l'adversaire).

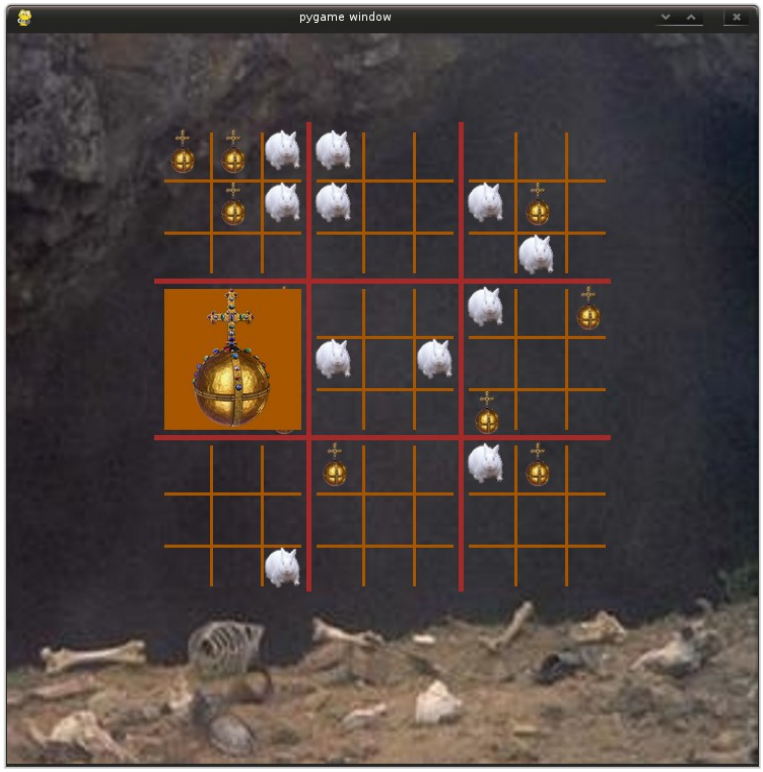
Résultats et Discussions

Déroulement du Jeu

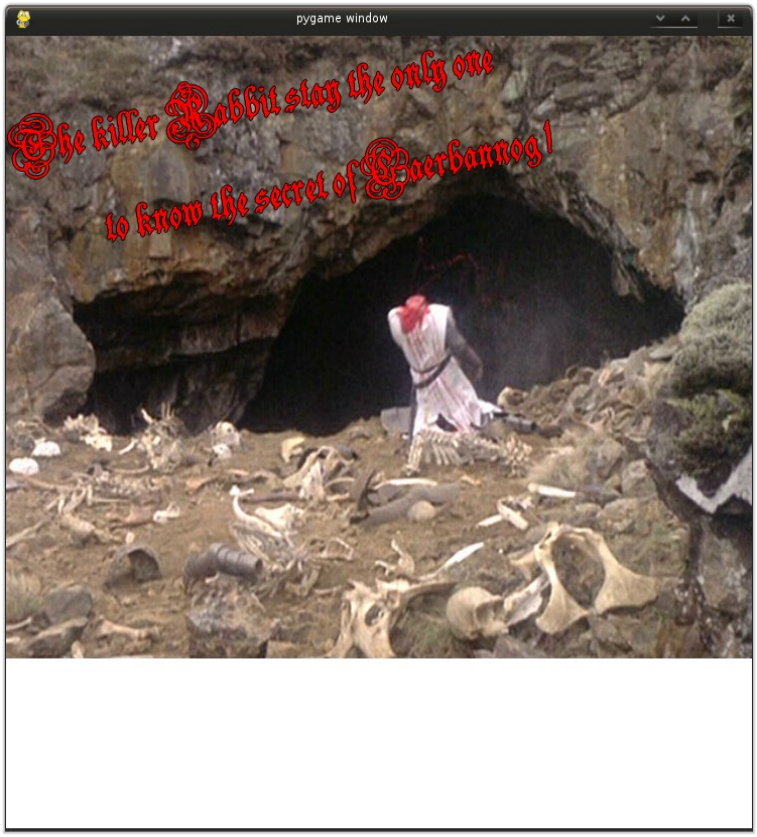
Le résultat final est donc la visualisation d'une interface graphique contenant un menu proposant différents types de jeux : Joueur contre Joueur, Joueur contre IA ou IA contre IA.

DEHEUVELS Joakim
GUIRIMAND Thibaut

Après avoir choisis le type d'interface, il s'en suis une étape de remplissage des grilles selon les règles de jeu.



Enfin lors d'une partie terminée, une fin adaptée au vainqueur est affichée.



Les différents événements sont par ailleurs ponctués de sons en rapport avec leur apparition.

Conclusion et Perspectives

Le jeu conçu est donc fonctionnel, il nécessite évidemment l'installation de la bibliothèque pygame avant d'être utilisé. Il permet le jeu face à un joueur ou face à une IA, qui utilise un minimum de stratégie, ou de faire jouer deux IA.

Le jeu étant codé en python, nous avons créé une interface graphique en rapport avec les Monty Python.

Difficultés de conception

Il s'avère que pour avoir un code plus propre, une classe Interface devais être initialement créée permettant de bien différencier le moteur de jeu développé par Thibaut GUIRIMAND et le moteur graphique développé par Joakim DEHEUVELS.

Devant créer deux thread séparés il est devenu compliqué de les faire communiquer. La solution a donc été de créer un thread dans Arena même pouvant donc ainsi accéder à des variables de classe communes. Cependant, prendre en compte les variables et les méthodes des deux éléments a contribué à rendre la classe Arena plus brouillon puisqu'essayant de regrouper les deux travaux.

Nous nous somme retrouvé notamment face à des problèmes de synchronisation des deux boucles infinies qui ont pu être résolus grâce à des pauses lors de leur exécution.

Une surcharge de la mémoire a aussi pu être observée à cause des boucles ce qui a été résolu en limitant le nombre de tours à 30 par secondes pour la recherche d'événements.

La bibliothèque Pygame n'était peut-être pas le meilleur choix car elle ne gère pas les calques.

Quand une image est dessinée sur la fenêtre, elle écrase tout ce qu'elle recouvre.

Perspectives

L'interface se limite actuellement au strict minimum. Différentes améliorations peuvent être à envisager :

- Mettre en place un menu déroulant pour recommencer la partie.
- Permettre l'annulation d'un mouvement.
- Rendre l'IA plus intelligente pour lui permettre de calculer la position de son pion en fonction de la grille principale plutôt qu'uniquement en fonction du contenu de la sous-grille dans laquelle elle joue.