Programmation 2: Projet jeu de Hex

Ce document est téléchargeable à cette adresse.

1 Modalités et dates de rendu

Le projet est à réaliser seul ou en binôme ; un binôme est constitué au maximum de deux personnes.

Votre projet doit être terminé au plus tard le vendredi 13 décembre 2019; date à laquelle vous auraient à présenter votre travaille et à me l'envoyer par mail à l'adresse suivante: florian.bridoux@lis-lab.fr

contenant

- en sujet : "[prog2] PROJET NOM1 NOM2"

Remplacez NOM1 et NOM2 par vos noms propres ; si vous êtes seul, remplacez NOM2 par "SEUL".

- dans le corps du message :
- mettez vos noms et prénoms ;
- expliquez comment utiliser votre programme ;
- décrivez ce que vous avez fait en plus ou en moins par rapport à l'énoncé.
- en attaché, votre programme Python 3

2 Objectifs

L'objectif du projet est de réaliser une implémentation python du jeu d'Hex à l'aide de pyGtk. Si vous ne connaissez pas ce jeu, vous pourrez en trouver une description ici.

Votre objectif principale est de créer un programme permettant à deux joueurs de s'affronter sur un "tablier" de taille 11x11 initialement vide jusqu'à ce que l'un des deux joueurs gagne. Le jeu indiquera alors le gagnant et proposera une nouvelle partie.

Des objectifs bonus sont:

- la possibilité d'abandonner et de recommencer une partie;
- la possibilité de changer la taille du "tablier" sur lequel les deux joueurs jouent;

- la possibilité de chronométrer et de limiter le temps entre chaque tours;
- la possibilité de jouer contre une IA;
- toute autre amélioration à laquelle vous pouvez penser.

Ce projet et à réaliser en TP et éventuellement chez vous. Une série d'exercice vous sera proposé pour vous aider à réaliser ce TP.

Exercices

Exercice 1 : Reflexion sur le système de coordonnées

Dans une grille classique (composé de $n \times n$ carrés), chaque case a au plus 4 voisins: une case (x,y) (abscisse x et ordonné y) a comme voisin:

1. $(x-1, y)$
2. $(x+1, y)$
3. $(x, y-1)$
4. $(x, y+1)$

En revanche, dans un tablier d'hexagone, chaque case à jusqu'à 6 voisins potentiels: 1 pour chaque côté de l'hexagone. Par chance, il existe un système de coordonnées pratique pour associer à chaque hexagone une coordonnée (x, y). Dans ce système, une position (x, y) a comme voisin à droite la case de coordonnée $(x+1, y)$ et comme voisin en bas à droite la case de coordonnée $(x, y+1)$.

Quel sont les coordonnées associé à tous les voisins d'une case de coordonnée (x, y)? En supposant que la case la plus en haut à gauche a une coordonnée $(0, 0)$, quel est la coordonnée de la case la plus en bas à gauche? La plus en haut à droite? La plus en bas à droite?

Exercice 2 : Classe Plateau

Créez une classe *Plateau* dans un fichier plateau.py. Cette classe permettra de représenter la situation courante du jeu. Pour rappel, vous pouvez placer votre code pour tester votre classe *Plateau* vous pouvez ajouter la condition __name__ == "__main__". Pour utiliser votre classe Plateau depuis un autre fichier python vous devrez faire: from plateau import Plateau

Implémentez les méthodes suivantes dans votre classe Plateau et n'oubliez pas de tester.

1. Un constructeur qui prend en paramètre un entier n et qui initialise votre Plateau comme un tablier de taille $n \times n$ composé uniquement de 0 (= aucun joueur n'a encore joué).
2. Des méthodes get(self,coo) , set(self,coo,val) et afficher(self) tel que vu en cour avec coordonnée donné sous la forme (x, y).
3. Une méthode case_libre(self,coo) qui indique si la case à la coordonnée *coo* est libre (= aucun joueur n'a joué dessus encore).

4. Une fonction voisins(self,coo) qui indique toutes les coordonnées voisine d'une case à la coordonnée coo qui appartiennent bien à au tablier.

5. Une fonction gagnant(self) qui retourne 0 si aucun des deux joueurs n'a gagné, 1 si le joueur 1 a gagné (= qu'il y a un chemin de 1 entre le bord gauche et le bord droit du tablier), 2 si le joueur 2 a gagné (= qu'il y a un chemin de 2 entre le bord haut et le bord bas),

Exercice 3 : Hexagone

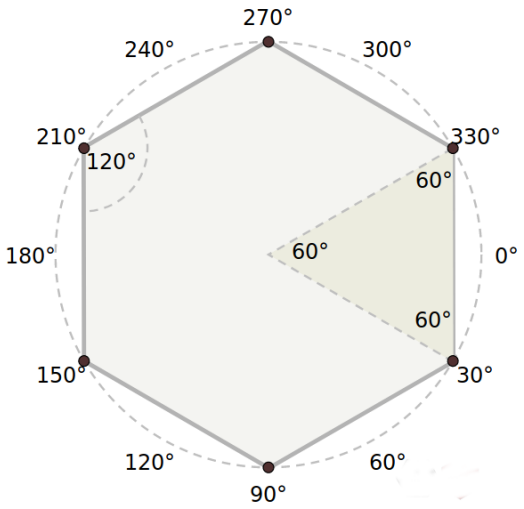
Créez un fichier hexagone.py et définissez une classe *Hexagone* qui modélise un hexagone.
Cette classe aura les méthodes suivantes.

1. Un constructeur qui prend les paramètres suivants:

   (a) *centre*: le centre de l'Hexagone donné sous la forme d'un couple (x,y) avec x et y un abscisse et un ordonné donné en pixel. (attention: le coin (0,0) est le point le plus en haut à gauche de la zone de dessin).

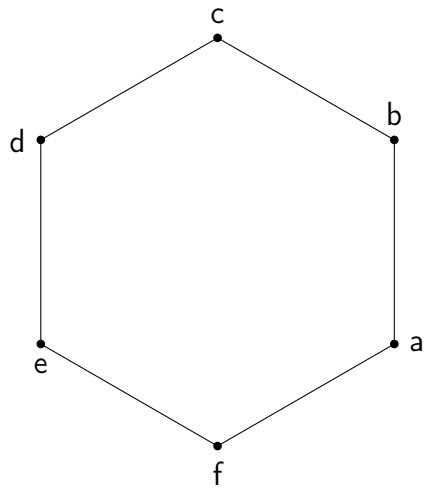
   (b) *rayon*: le rayon de l'Hexagone, c'est-à-dire la distance ("en pixel") entre le centre de l'hexagone et chacun de ses sommets.

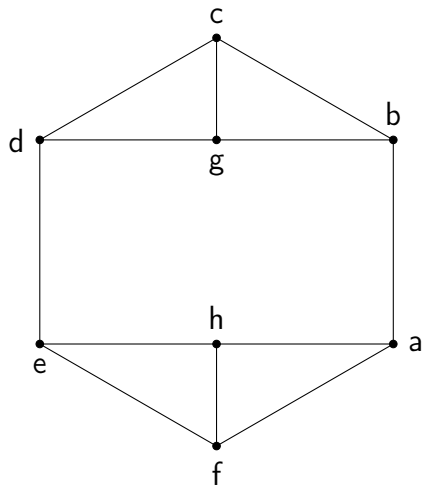
2. Une méthode *positions_sommets* qui retourne la liste des 6 positions des sommets de l'hexagones donnés sous la forme de tuples (x,y) (donnés en pixel). Vous pouvez vous aider de la figure suivante, des fonctions *cos* et *sin* du module *math* ainsi que vos connaissances en géométrie pour calculer les positions des sommets.



3. Une méthode *interieur*(point) qui retourne True si le point donné en paramètre (sous la forme d'un couple (x,y)) est à l'intérieur de l'hexagone et False sinon.

   Indication: observez la figure suivante.





Remarquez que tout le point appartenant à l'hexagone sont compris:

- dans le rectangle (a, b, d, e), ou
- dans l'un des triangles (d, g, c), (c, g, b), (e, h, f) ou (f, h, a).

(Vous pouvez vous faciliter la vie en utilisant les symétries.)

Exercice 4 : Tablier

Créez un fichier tablier.py et définissez une classe *Tablier* qui modélise un tablier. La classe Tablier permet de représenter un tablier de taille $n \times n$ à l'aide de $n \times n$ hexagones.

La classe Tablier devra disposer des méthodes suivantes.

1. Un constructeur avec comme paramètre n,centre_haut_gauche et rayon; centre_haut_gauche étant la position de l'hexagone du tablier le plus en haut à gauche et rayon étant le rayon des hexagones qui constituent le tablier.
2. Une méthode hexagones() qui renvoie la liste de ses hexagones.
3. Une méthode get(coo) qui retourne l'hexagone en coordonnée coo.
4. Une méthode hexagone_qui_contient(point) qui retourne l'hexagone qui contient le point *pont* s'il existe et None sinon.

Exercice 5 : Interface graphique

Vous pouvez maintenant écrire le programme principal dans un fichier jeu.py. Vous allez réaliser ici l'interface graphique. Comme vu dans le CM5, vous pouvez créer une classe MyData qui permet de sauvegarder toutes les informations relatives au déroulement du jeu:

1. le joueur dont c'est le tour
2. l'état du plateau,

3. la zone de dessin et le label qui doivent être modifié régulièrement,

4. ...

Plus précisément, vous pouvez créer une fenêtre avec un bouton "Quitter", un bouton "Recommencer", un label et une zone de dessin en dessous (voir les CM4 et 5). Le bouton "Quitter" fermera le programme. Le bouton "Recommencer" permettra de recommencer la partie. Le label permettra d'afficher le joueur dont c'est le tour ou le joueur gagnant s'il y'en a un. La zone de dessin permettra d'afficher le plateau de jeu et de recevoir les clics des utilisateurs sur le plateau.

Pour ça vous pouvez commencer par:

1. Créer une fonction dessiner_hexagone(cr, hexagone, occupation) qui dessine un hexagone dans la zone de dessin à l'aide de cr. La variable occupation vaudra 0,1 ou 2 selon si l'hexagone est inoccupé, si le joueur 1 a joué dedans ou si le joueur 2 a joué dedans. On voudra afficher l'hexagone différemment dans ses 3 cas de figures. Vous pouvez éventuellement ajouter des paramètres à la fonction pour dessiner différemment les hexagones sur les bords (pour rendre plus claire le fait que le joueur 1 doit tracer un chemin entre la gauche et la droite du tablier).

2. Créer une fonction dessiner_tablier(cr, tablier,myData) qui dessine un tablier dans la zone de dessin à l'aide de cr.

3. Connecter des fonctions aux évènements liés aux clics sur les boutons.

4. Connecter une fonction à l'évènement "clic dans la zone de dessin" pour permettre aux joueurs de jouer et de faire les modifications tant visuelle que sur les données.