Projet1 – TP3 : Project Specifications

Marin Bougeret, Bogdan Pasca, Kevin Perrot
{ Bogdan.Pasca } @ens-lyon.fr

October 4, 2010

Logistics

- You may work single or in pairs (groups of 2).
- Project is due 11th of October, 11:15AM. We use the second TD hour to evaluate.
- Feel free to ask questions.

1 Poker Texas Hold'em

L'objectif de ce projet est de réaliser un jeu de poker Texas Hold'em. Il est conseillé de réaliser des versions succesives afin de jalonner les étapes du projets. Il n'est pas indispensable de fournir toutes les versions. Vous pouvez privilégier la qualité de certaines versions au détriment du nombre de versions réalisées. Ou vous pouvez privilégier le nombre de versions et aller le plus loin possible mais en réalisant le minimum demandé pour chaque version. A vous de choisir votre approche.

1.1 Les règles

Première étape comprendre les règles que vous trouverez à cette adresse `http://fr.wikipedia.org/wiki/Texas_hold%27em`.

1.2 Versions mode texte

Vous commencerez par une version en mode texte (non graphique).

v0.1 : 2 joueurs "Heads-Up". Avec simplement la distribution des cartes, la carte brulée et l'étalement du flop.

v0.2 : Détection des mains gagnantes.

v0.3 : Table pouvant aller jusqu'au "Full Ring".

v0.4 : Gestion des mises.

1.3 Versions graphiques

Les étapes ci-dessous peuvent être faites après la v0.4 ou alors être entrelacées.

v1.1 : 2 joueurs “Heads-Up”. Avec simplement la distribution des cartes, la carte brulée et l’étalement du flop.

v1.2 : Table pouvant aller jusqu’au “Full Ring”.

v1.3 : Gestion des mises.

1.4 Versions réseau

Et pourquoi pas une version réseau ?

2 Du jeux de la vie au récif corallien

2.1 Le jeu de la vie

Les règles du jeu de la vie sont relativement simples au départ : Les cellules évoluent dans un environnement simplifié : en l'occurrence une grille rectangulaire. Dans un tel environnement, une cellule peut avoir 8 voisins

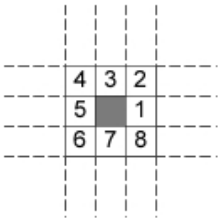
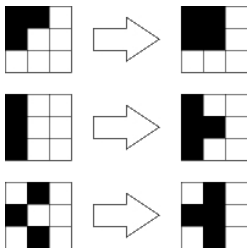


FIGURE 1 – Voisinage d'une cellule sur une grille rectangulaire

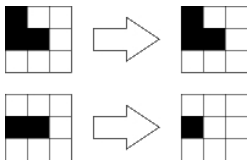
Chaque cellule vit, meurt, ou bien génère une nouvelle cellule (division cellulaire) en fonction du nombre de cellules qui l'environnent. La vie de ces cellules est donc simulée par une boucle infinie. A chaque tour de boucle, on évalue la condition de chaque cellule pour déterminer si elle doit survivre, mourir ou bien générer une nouvelle cellule au prochain tour de boucle.

Par exemple, le comportement typique tel que l'a défini John Conway (inventeur du jeu de la vie) est le suivant :

- Naissance d'une nouvelle cellule : Si un emplacement inoccupé possède 3 emplacements voisins occupés par des cellules, il devient occupé par une nouvelle cellule.



- Vie d'une cellule : Si une cellule possède 2 ou 3 voisines, cette cellule peut survivre.



- Mort d'une cellule : si une cellule ne possède que 0 ou 1 voisine, elle meurt (de solitude). Et si elle plus de 5 voisins (jusqu'à 8), elle meurt également (de surpopulation).

2.2 Récif Corallien

Il existe une infinité de variantes à ce comportement de base, il suffit de changer le comportement associé à chaque nombre de voisins, par exemple.

Dans le cadre de ce projet on va envisager une variante qui consiste lorsqu'une cellule meurt à laisser son squelette en place. Ce comportement rend les cellules similaires à de colonies coralliennes[1] où seule la couche externe des coraux est vivante, l'intérieur n'est fait que de cellules mortes. On peut de la même manière prévoir l'essaimage des nouvelles cellules dans certaines conditions.

On peut aussi imposer des conditions supplémentaires quant à la naissance d'une nouvelle cellule. Par exemple on peut favoriser la direction de croissance des nouvelles cellules, de manière à créer des structures ramifiées, etc.

2.3 Méthode

- Tout d'abord construire les cellules et leur environnement, avec leurs paramètres d'évolution.
- Ensuite, faire évoluer une ou plusieurs population dans un même espace.
- Pour la représentation du jeu de la vie deux solutions sont possibles :
  - on peut faire simplement une représentation sous forme de caractères dans un terminal (texte) ;
  - ou bien une représentation graphique de votre choix.

Votre programme devra permettre la simulation pas à pas (presser une touche ou un bouton pour passer à l'état suivant) et la simulation animée (prévoir un délais entre l'affichage d'un état et celui du suivant).

[1] Longement étudiées de façon non officielle par l'équipe GRAAL dans le cadre de son équipe associée avec l'Université d'Hawai'i à Manoa.

3 Turtle is back

The game starts-off with a rectangular board of size w_x, w_y and two turtles placed randomly on the edges of this area. The turtles move in turn, one after the other. The maximal distance covered at each step is defined at the start of the game (d_{max}). In their movement, each turtle starts installing a defense mechanism (electric wire fence). An area is captured by a turtle if the deployed fence forms a closed perimeter with the edges or another fence of the same turtle.

While the fence in not closed (the second end-point not reached) the fence can be broken by the opposing turtle by intersecting it. A broken fence is automatically removed from the map.

There are at least two different scenarios for ending this game:

- after a predetermined number of steps (p) the turtle having covered the largest area is declared the winner.
- the game ends when there is no more territory to cover. The winner is decided similarly as for first scenario.

Another extension could consist in allowing more than two turtles to play.

The turtles will be instantiated in this object oriented way:

```
from turtle import *
screen = Screen()
turtle1 = RawTurtle(screen)
turtle2 = RawTurtle(screen)
turtle1.forward(100)
turtle2.back(100)
x,y=turtle1.position()
```

Each algorithm will receive the screen, its turtle object and the opponent turtle together with the maximal distance it can cover in each iteration. Once an area is captured by a turtle is the responsibility of that turtle to color the captured area with the turtle's color. Moreover, both turtles should know at each step the area covered by itself and its opponent. When there is no more area left to cover ($a_{me} + a_{opponent} = w_x \times w_y$), the movement functions should return false.

The first test will consist in a fixed number of iterations (scenario 1). The second test will end when there is no more territory to cover(scenario 2). You may build two different functions, one for each scenario.