Programmation Bio-Inspirée
TP3 - Tournoi de joueurs

Simon Camille Master 2 IWOCS 22/11/2017

1 Abstract

The purpose of this practical work is to create a model for a player tournament using the Prisoner Dilemma. A player is defined by an energy value and a state.

- The energy value is set to 20 by default. When the value drop to 0 or less, the player is removed from the simulation. To play a match, a player has to pay an energy cost. Depending on the result of the match, the player recieves an amount of energy.
- The state of a player can be C for Cooperate or D for Defect.

There are three ways for selecting players for a match :

1. The players are placed randomly on the world map. Two of them are selected arbitrary to play a match.
2. The players are still placed as the same way, but this time, they move randomly across the world map. When two players are close, they start a match.
3. The players are placed in a spatial network where they form the nodes. In function of the graph type, the model will generate edges between nodes. On each tick, players connected by an edge play a match.

When a player $P1$ match a player $P2$, there are 4 different cases for the couple $(P1, P2)$: (C, C), (C, D), (D, C) and (D, D). This four cases correspond to four different procedures on the model. For each case, the reward for both players can be synthesize in a table like this :

		Player 2	
Player 1		C	D
	C	$(+2, +2)$	$(0, +3)$
	D	$(+3, 0)$	$(+1, +1)$

Figure 1: Prisoner Dilemma payoff table

Also, after the match, the player's state can change. The evolution is in function of the current state of the player and the adversary state.

The probability to change or stay in a state follow a multi-strategy probabilistic automaton like this example :

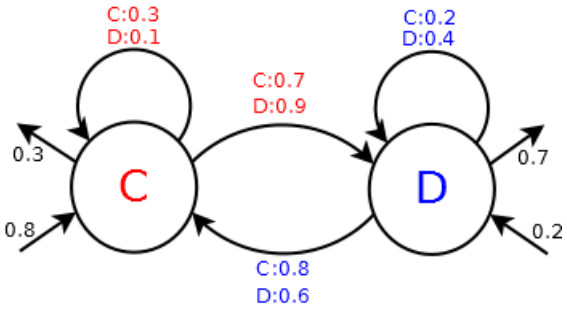

Figure 2: Probabilistic automaton

For each couple of state between $P1$ and $P2$, there are a procedure who determine the new state value according to the probabilistic automaton.

2 Présentations des configurations

Le travaille demandé est découpé sous la forme de trois programmes.
Dans cette partie, nous allons présenter les différentes configurations des programmes. Pour chacune des configurations, nous allons décrire ses entités, ses procédures ainsi que le comportement des entités et de la simulation.

2.1 Analyse de la première configuration

2.1.1 Entités et variables globales

Pour notre tournoi, nous avons besoin d'une population de joueurs. Il est donc nécessaire de créer une espèce ($breed$) de joueurs ($players$).
Chaque joueur à deux attributs : $energy$ et $state$. $Energy$ est la valeur énergétique décrite dans le sujet. $state$ quant à lui, va permettre de modéliser dans quel état se trouve le joueur. Si $state$ vaut C cela signifie que le joueur est dans l'état $cooperate$, si il vaut D cela veut dire que le joueur est dans l'état $defect$.
Nous avons besoin de deux variables globales qui prendront les valeurs des deux joueurs sélectionnés pour jouer un match ($player1$ et $player2$).
Nous avons également besoin d'information données par l'utilisateur pour paramétrer la simulation :

- $num - player$: Le nombre de joueur.
- $max - energy$: La valeur maximum d'énergie d'un joueur.
- $match - cost$: Le coût en énergie pour jouer un match.
- $pts - win$: La valeur en énergie obtenu par un joueur qui gagne son match.
- $pts - lose$: La valeur en énergie obtenu par un joueur qui perd son match.
- $pts - both - cooperate$: La valeur en énergie obtenu par les deux joueurs si ils sont tous les deux dans l'état coopérer.

- $pts - both - defect$: La valeur en énergie obtenu par les deux joueurs si ils sont tous les deux dans l'état $defect$.
- Les probabilités pour les transitions de l'automate sous la forme d'un tableau similaire à ce qui est vu dans le cours.

2.1.2 Procédures

- $init - state$: Retourne un état, C ou D en fonction des probabilités données par l'utilisateur.
- $select - player$: Sélectionne aléatoirement deux joueurs en prenant garde qu'il s'agisse de deux joueurs différent.
- $pay - match - cost$: Réduit l'$energy$ des deux joueurs d'une valeur donnée. La valeur correspond au coût que doit payer un joueur pour jouer un match.
- $play - match$: Identifie dans quel état se trouve le couple ($player1$, $player2$). Une fois la situation identifiée, on fait appelle aux procédures permettant de récompenser les joueurs et faisant évoluer leurs états.
- $both - win$: Augmente l'énergie des deux joueurs d'un montant donné en paramètre de la procédure.
- $player1 - win$: Augmente l'énergie du joueur contenu dans la variable $player1$ de la valeur passée en paramètre de la procédure.
- $player2 - win$: Augmente l'énergie du joueur contenu dans la variable $player2$ de la valeur passée en paramètre de la procédure.
- $update - state - c - c$: Fait évoluer l'état du joueur passé en paramètre selon une probabilité fixée par l'utilisateur. Les deux joueurs sont dans l'état C.
- $update - state - d - d$: Fait évoluer l'état du joueur passé en paramètre selon une probabilité fixée par l'utilisateur. Les deux joueurs sont dans l'état D.
- $update - state - c - d$: Fait évoluer l'état du joueur passé en paramètre selon une probabilité fixée par l'utilisateur. Le joueur passé en paramètre est dans l'état C, l'autre joueur est dans l'état D.
- $update - state - d - c$: Fait évoluer l'état du joueur passé en paramètre selon une probabilité fixée par l'utilisateur. Le joueur passé en paramètre est dans l'état D, l'autre joueur est dans l'état C.
- $update - color$: Retourne une couleur pour le joueur en fonction de son niveau d'énergie. Lorsque le joueur à toute son énergie, retourne la couleur verte. Moins le joueur à d'énergie, plus le gradient va vers le rouge.

2.1.3 Comportement des entités

Dans cette première configuration, les joueurs sont fixes dans l'espace. Un joueur interagit avec un autre joueur lorsqu'il est sélectionné pour jouer un match. Jouer un match fait perdre $match - cost$ énergie aux deux joueurs. En fonction de leurs états, les deux joueurs vont recevoir un certain montant d'énergie. Pour un couple de joueur $(Player1, Player2)$ les gains obtenus en fonction de leurs états sont résumés dans le tableau suivant :

		Player 2	
Player 1		C	D
	C	$(pts - both - cooperate, pts - both - cooperate)$	$(pts - lose, pts - win)$
	D	$(pts - win, pts - lose)$	$(pts - both - defect, pts - both - defect)$

Figure 3: Table des gains

Une fois l'énergie reçu, chacun des deux joueurs va appliquer l'automate à multiplicité multi-stratégie. C'est à dire que le joueur, en fonction de son état et de l'état de son adversaire, va avoir une certaine probabilité de rester dans son état ou d'en changer.

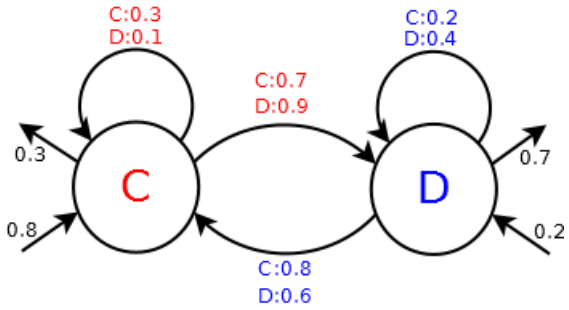

Figure 4: Automate à multiplicité multi-stratégie

Si le joueur est dans l'état C, en rouge sur la figure 4, il a une probabilité de rester dans l'état C de $0,3$ si l'adversaire est dans l'état C et de $0,1$ si l'adversaire est dans l'état D. Il s'agit des deux labels sur la flèche bouclant sur l'état C sur la figure 4.
Par complémentarité des probabilités, le joueur à donc une probabilité de $0,9$ de passer à l'état D si son adversaire et dans l'état D et $0,7$ de passer à l'état D si son adversaire est dans l'état C. Ces deux probabilités sont représentées sur la flèche allant de l'état C à l'état D sur la figure 4.
Si le joueur se trouve initialement dans l'état D, en bleu sur la figure, on s'attarde alors sur les labels bleues représentant les valeurs de probabilité pour un joueur dans l'état initial D.
Après avoir appliqué l'automate, le joueur va changer de couleur en fonction de sa nouvelle valeur d'énergie. Si son énergie est inférieure ou égale à 0, il est retiré de la simulation.

2.1.4 Déroulement de la simulation

On commence par créer les joueurs et on les place aléatoirement dans l'espace. Pour chaque joueur on initialise son énergie, son état et sa couleur.
Une itération de la simulation consiste à sélectionner aléatoirement deux joueurs, chacun d'eux paye le coût d'un match et joue le match. Lorsque deux joueurs jouent un match, ils commencent par récupérer

un montant d'énergie en fonction de leurs états. Puis, ils appliquent l'automate à multiplicité multi-stratégique.
Une fois le match jouer, on met à jour la couleur des joueurs. Si un joueur a une énergie nulle ou négative, il est retiré de la simulation.
La simulation poursuit les itérations jusqu'à se qu'il reste moins de deux joueurs. En effet, dans cette situation il n'est pas possible de jouer un match.

2.2 Analyse de la deuxième configuration

Dans cette configuration, les joueurs ne sont plus choisis aléatoirement. En effet, les joueurs vont se déplacer dans le monde et affronter les joueurs se trouvant dans leur voisinage.

2.2.1 Entités et variables globales

Dans cette deuxième configuration, les joueurs et leurs attributs ne changent pas.
Les deux variables globales $player1$ et $player2$ ne sont plus nécessaires, elles ont donc été retirées. Une nouvelle variable $vicinity$ est ajoutée, il s'agit du rayon d'un cercle autour du joueur. Les autres joueurs se trouvant dans ce cercle sont considérés comme étant dans le voisinage du joueur. Un joueur ne peut affronter que les joueurs se trouvant dans son voisinage.

2.2.2 Procédures

La sélection des joueurs se faisant différemment dans cette configuration, la procédure $select - players$ a été enlevé. Les procédures $player1 - win$ et $player2 - win$ sont remplacées par une seule procédure $player - win$ prenant en paramètre une valeur et un joueur. Les procédures suivantes ont été modifiées :

- $pay - match - cost$: Réduit l'$energy$ des deux joueurs passés en paramètre d'une valeur donnée en paramètre.
- $play - match$: Prend maintenant deux joueurs en paramètre et les fait s'affronter pendant un match.
- $both - win$: Augmente l'énergie des deux joueurs passés en paramètre d'un montant donné en paramètre de la procédure.

Les autres procédures n'ont pas changées.

2.2.3 Comportement des entités

Dans cette configuration, les joueurs se déplacent dans l'espace. A chaque itération, le joueur compte combien d'autres joueurs se trouve dans son voisinage. Le voisinage est définit par un cercle où le centre est la position du joueur et le rayon est donné par la variable $vicinity$.
Si il y a un ou plusieurs joueurs dans son voisinage, le joueur joue un match contre l'un d'entre eux. Le match se déroule de la même façon que dans la précédente configuration.
Une fois le match terminé et sa couleur mise à jour, le joueur fait un pas en avant dans une direction aléatoire.
Si son énergie est nulle ou négative il est retiré de la simulation.

2.2.4 Déroulement de la simulation

On commence par créer les joueurs et on les place aléatoirement dans l'espace. Pour chaque joueur on initialise son énergie, son état et sa couleur.
Une itération de la simulation consiste à :

1. Demander à chaque joueur de sélectionner un de ses voisins si il en a.
2. Chaque couple de joueur ainsi obtenu paye le coût d'un match.
3. Les joueurs jouent les matchs : ils commencent par récupérer un montant d'énergie en fonction de leurs états. Puis, ils appliquent l'automate à multiplicité multi-stratégique.
4. On met à jour la couleur des joueurs.
5. Chaque joueur prend une direction aléatoire et avance d'un pas.
6. Si un joueur a une énergie nulle ou négative, il est retiré de la simulation.

La simulation poursuit les itérations jusqu'à se qu'il reste moins de deux joueurs. En effet, dans cette situation il n'est pas possible de jouer un match.

2.3 Analyse de la troisième configuration

Cette fois, les joueurs sont positionnés dans un réseau spatial. Cela signifie que l'on va travailler sur un graphe dont les joueurs sont les noeuds. Dans ce graphe, les arêtes vont représenter la relation "faire un match", un joueur ne peut faire un match qu'avec les joueurs auxquels il est relié.

2.3.1 Entités et variables globales

Les entités de cette configuration sont les mêmes que pour la configuration précédente.
De nouvelles variables paramétrable par l'utilisateur sont ajoutés :

- $max - neighbourhood$ permet à l'utilisateur de sélectionner le nombre maximum de voisin des noeuds du graphe.
- $see - link$ est un interrupteur permettant d'afficher ou de cacher les arêtes reliants les noeuds.
- $graph - type$ permet de sélectionner un type de graphe parmi $Random$ et $Circular$.
- $density$ règle la densité du graphe dans le cas où le type de graphe est$Random$.

Il y a également une nouvelle variable globale $num - neighbours$. Cette variable va servir lorsque le type de graphe est $Circular$. Pour ce type de graphe, le nombre de voisin doit être paire, cette variable va donc prendre la valeur inférieure paire maximum de $max - neighbourhood$.

2.3.2 Procédures

Les procédures de la deuxième configuration sont également présentes dans la configuration trois et n'ont pas changées. Deux nouvelles procédures ont été ajoutées :

- $update - links - color$: Met à jour la couleur des liens en fonction de la variables $see - link$.
- $links - random$: Place les joueurs en cercle sur le monde et crée des liens entre les joueurs de façon aléatoire. Pour crée un lien entre deux joueurs on procède de la façon suivante : on tire un nombre aléatoire entre 0 et 99, si cette valeur est strictement inférieure à $density$ on sélectionne aléatoire un autre joueur et on crée le lien. Chaque joueur répète cette opération autant de fois que $max - neighbourhood$. Enfin, on met à jour la couleur des liens.
- $links - circular$: Le but de cette procédure est de créer un graphe régulier. La structure en anneau consiste à lier un noeud avec ses voisins de gauches et de droites les plus proches. Ici pour une question de lisibilité, on prendra des noeuds plus éloignés. On commence par initialiser $num - beighbours$ pour être sur que le nombre de voisin est pair. Puis on place en cercle les joueurs. Chaque joueur va créer $\frac{num - beighbours}{2}$ liens avec des joueurs choisis précisément pour avoir un beau rendu graphique. Une fois tous les liens créés, on met à jour la couleur des liens.

2.3.3 Comportement des entités

Les joueurs sont placés en cercle dans le monde. En fonction de $graph - type$, $max - neighbourhood$ et $density$ les joueurs vont avoir entre 0 et $max - neighbourhood$ voisins.
A chaque itération, le joueur va affronter tous les autres joueurs auxquels il est relié. Les matchs se déroulent comme dans les configurations précédentes.
Une fois le match terminé et sa couleur mise à jour, on regarde si le joueur a une énergie nulle ou négative, si c'est le cas il est retiré de la simulation.

2.3.4 Déroulement de la simulation

On commence par créer les joueurs et on initialise leur énergie, état et couleur. Puis on les places et on génère les liens en fonction de $graph - type$.
A chaque itération de la simulation, on demande au joueur de jouer un match contre chacun de ses voisins. Après le match, on met à jour la couleur du joueur et on le retire si son énergie est nulle ou négative.
La simulation s'arrête lorsqu'il n'y a plus de lien, en effet les joueurs ne peuvent s'affronter que si il y a un lien entre eux.

3 Analyse des résultats

Observons maintenant le comportement des simulations dans chacune des configurations.

3.1 Première configuration

Pour cette première configuration, comparons deux jeux de tests après les avoir présentés.

3.1.1 Premier jeu de test

Voici un exemple de simulation pour la première configuration :

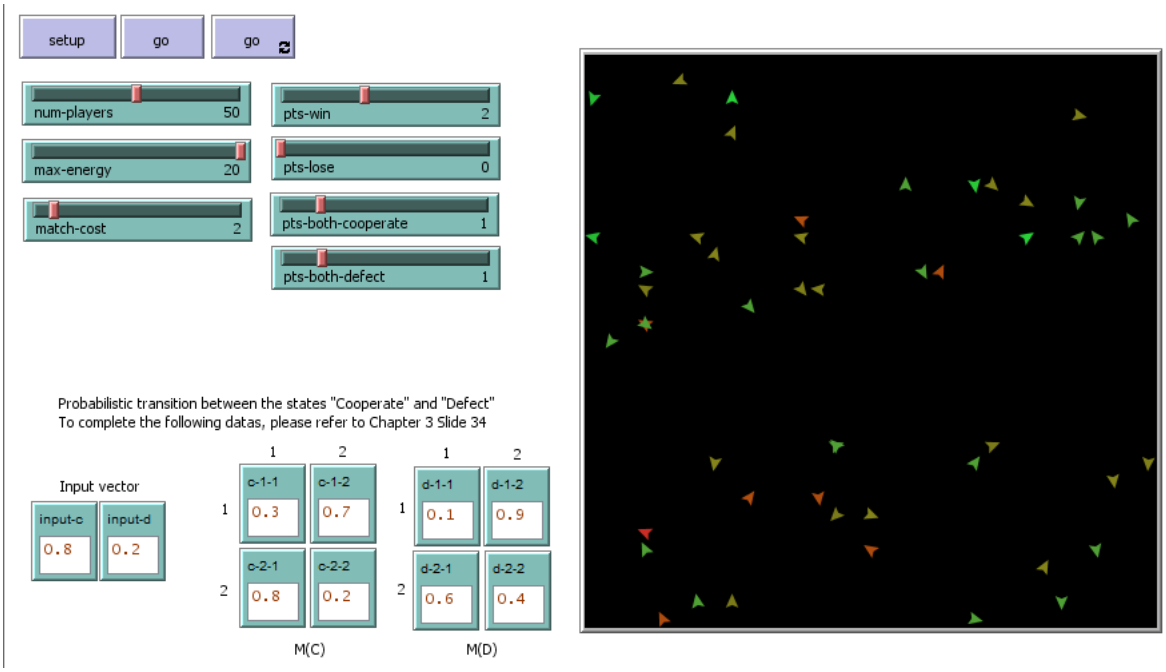

Figure 5: Premier jeu de test avec la configuration 1.

Dans cette exemple les paramètres de la simulation sont les suivants :

$$num - player = 50 \quad pts - win = 2$$
$$max - energy = 20 \quad pts - lose = 0$$
$$match - cost = 2 \quad pts - both - cooperate = 1$$
$$pts - both - defect = 1$$

Les autres données sont les probabilités de l'automate, par défaut elles sont égales à celles du cours. Lorsque l'on exécute cette simulation, on observe que la couleur de tous les joueurs se modifie petit à petit. On a à un instant t au milieu de la simulation (cf. figure 8) des joueurs occupant tout le panel de couleurs de façon homogène.

3.1.2 Deuxième jeu de test

En prenant l'exemple ci-dessus comme expérience témoin, comparons le au jeu de paramètre suivant : Dans ce jeu de test, les points gagnés sont plus important que le coût du match. De plus, $pts - both - cooperate$ et $pts - both - defect$ n'attribuent plu la même récompense.

$$num - player = 50 \quad pts - win = 5$$
$$max - energy = 20 \quad pts - lose = 0$$
$$match - cost = 3 \quad pts - both - cooperate = 3$$
$$pts - both - defect = 1$$

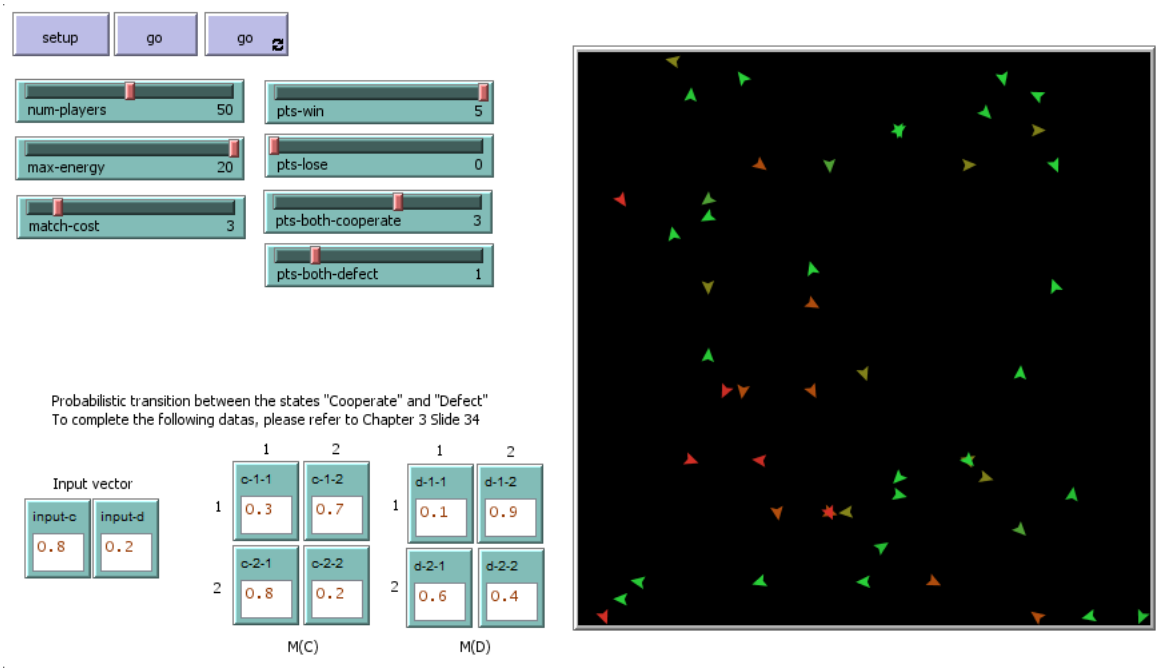

Figure 6: Deuxième jeu de test avec la configuration 1.

Dans cette situation, on voit apparaître des fortes démarcations de couleur entre les joueurs. On voit un groupe de joueur rouge, un groupe de joueur vert et quelques joueurs ayant des couleurs intermédiaires.

3.1.3 Analyse

En faisant varier les points obtenus lors d'un match, on voit les couleurs des joueurs variés différemment. En effet, lorsque l'on choisi une valeur d'énergie pour la victoire plus importante que le coup des matchs, on accroît l'écart d'énergie entre les gagnants et les perdants. Ainsi on crée une répartition hétérogène des couleurs des joueurs avec un groupe de joueur à faible énergie, une groupe plus important de joueur à haute énergie et quelques joueurs entre ses deux paliers.

3.2 Deuxième configuration

Pour cette deuxième configuration, comparons deux jeux de tests après les avoir présentés.

3.2.1 Premier jeu de test

Voici un exemple de simulation pour la deuxième configuration :

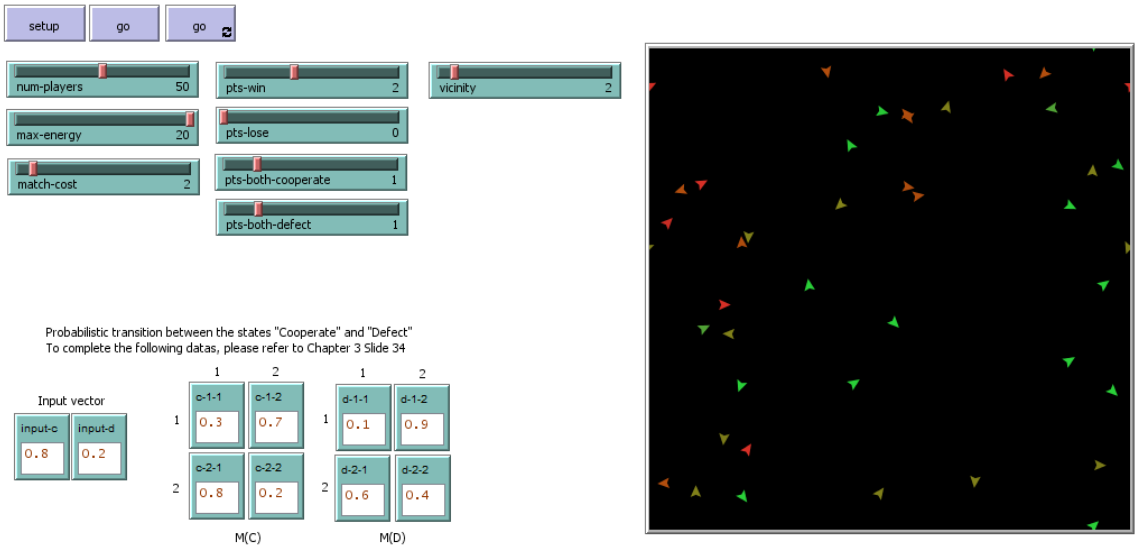

Figure 7: Premier jeu de test avec la configuration 2.

Dans cette exemple les paramètres de la simulation sont les suivants :

$$\begin{array}{lll} num - player = 50 & pts - win = 2 & vicinity = 2 \\ max - energy = 20 & pts - lose = 0 & \\ match - cost = 2 & pts - both - cooperate = 1 & \\ & pts - both - defect = 1 & \end{array}$$

Les autres données sont les probabilités de l'automate, par défaut elles sont égales à celles du cours. Lorsque l'on exécute cette simulation, on observe que la couleur de tous les joueurs se modifie petit à petit. On a à un instant t au milieu de la simulation (cf. figure 7) des joueurs occupant tout le panel de couleurs de façon homogène.

3.2.2 Deuxième jeu de test

En prenant l'exemple ci-dessus comme expérience témoin, comparons le au jeu de paramètre suivant :

$$\begin{array}{lll} num - player = 50 & pts - win = 5 & vicinity = 5 \\ max - energy = 20 & pts - lose = 0 & \\ match - cost = 3 & pts - both - cooperate = 3 & \\ & pts - both - defect = 1 & \end{array}$$

Dans ce jeu de test, le voisinage $vicinity$ des joueurs est plus important. C'est à dire que les joueurs peuvent sélectionner un adversaire qui se trouve plus loin d'eux. Ce qui augmente, par la même, la probabilité d'avoir un joueur dans son voisinage.

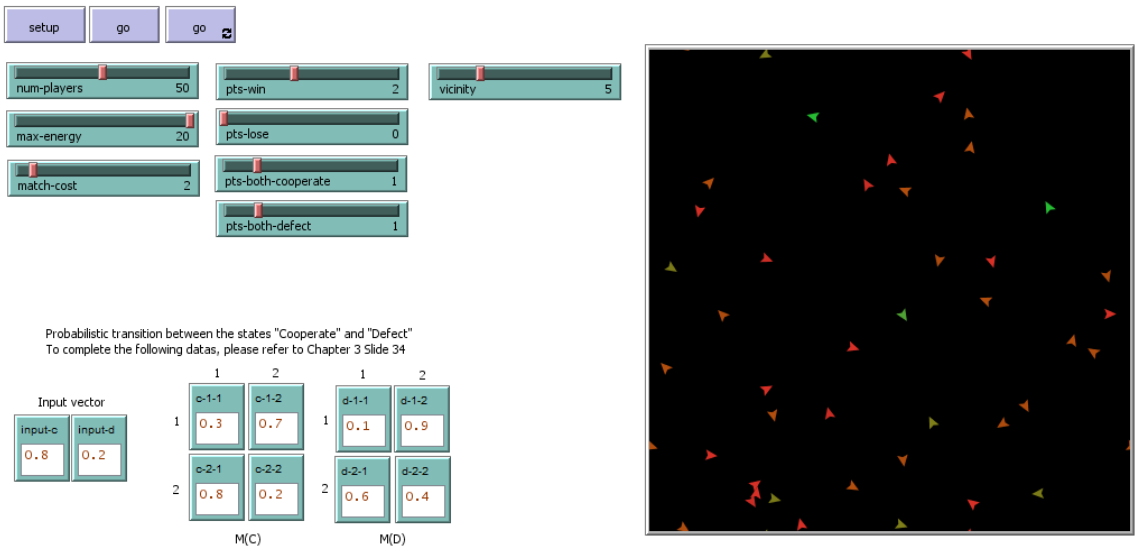

Figure 8: Deuxième jeu de test avec la configuration 2.

Dans cette situation, on voit apparaître des fortes démarcations de couleur entre les joueurs. On voit beaucoup de joueur devenir près du rouge très vite. Le nombre de joueur de couleur clair diminue très vite.

3.2.3 Analyse

En faisant varier la voisinage des joueurs, on voit la couleur de la population varier significativement.
En effet, lorsque l'on choisi une valeur de voisinage faible, il faut que les joueurs soit très proche pour engager un match. Comme leurs déplacements sont aléatoires, certain joueurs peuvent mettre du temps avant de voir leur énergie décroître. On a alors, une répartition des couleurs des joueurs homogène entre les nuances de rouge et de vert.
Lorsque l'on augmente la valeur du voisinage, la probabilité qu'un joueur ai un autre joueur dans son voisinage augmente. Ainsi, tous les joueurs ont plus de chance de jouer des matchs et ainsi de faire varier leur énergie. Ce qui entraîne l'apparition rapide d'un grand nombre de joueur à faible énergie.

3.3 Troisième configuration

Pour cette troisième configuration, comparons les deux types de graphe après les avoir présentés.

3.3.1 Graphe de type aléatoire

Voici un exemple de simulation pour la troisième configuration avec un graphe de type aléatoire :

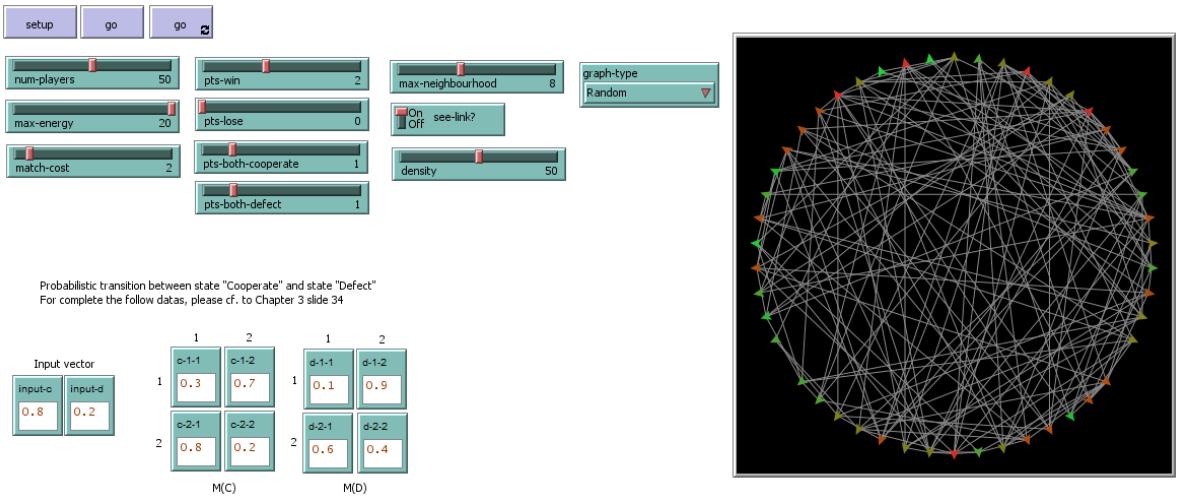

Figure 9: Premier jeu de test avec la configuration 3 avec un graphe de type aléatoire.

Dans cette exemple les paramètres de la simulation sont les suivants :

$$num - player = 50 \quad pts - win = 2 \quad max - neughbourhood = 8$$
$$max - energy = 20 \quad pts - lose = 0 \quad see - link? = true$$
$$match - cost = 2 \quad pts - both - cooperate = 1 \quad density = 50$$
$$pts - both - defect = 1 \quad graph - type = "Random"$$

Les autres données sont les probabilités de l'automate, par défaut elles sont égales à celles du cours. Lorsque l'on exécute cette simulation, on observe que les joueurs disparaissent très rapidement. A la première itération, la couleur des joueurs varie de façon homogène entre le rouge et le vert. L'itération suivante diminue drastiquement le nombres de joueurs. Enfin, la simulation s'arrête au bout de quelques itérations supplémentaires. Les joueurs restants sont majoritairement rouge ou de couleur foncée.

3.4 Graphe de type circulaire

Voici un exemple de simulation pour la troisième configuration avec un graphe de type circulaire :

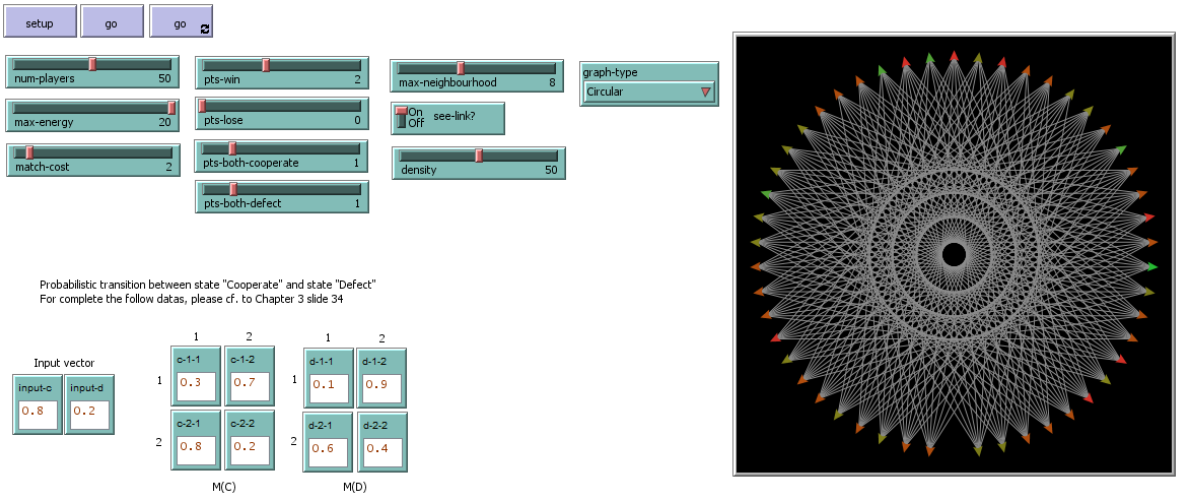

Figure 10: Premier jeu de test avec la configuration 3 avec un graphe de type circulaire.

Dans cette exemple les paramètres de la simulation sont les suivants :

$$num - player = 50 \quad pts - win = 2 \quad max - neughbourhood = 8$$
$$max - energy = 20 \quad pts - lose = 0 \quad see - link? = true$$
$$match - cost = 2 \quad pts - both - cooperate = 1 \quad density = 50$$
$$pts - both - defect = 1 \quad graph - type = "Circular"$$

Les autres données sont les probabilités de l'automate, par défaut elles sont égales à celles du cours.
Lorsque l'on exécute cette simulation, on observe que les joueurs disparaissent très rapidement. A la première itération, les couleurs des joueurs sont variée. Elles vont de façon homogène du rouge au vert. L'itération suivante est généralement la dernière de la simulation. Il ne reste alors que des joueurs de couleur rouge ou très foncée.

3.4.1 Analyse

Le paramètre variant entre ses deux simulation est le type du graphe.
Dans le premier cas le nombre d'arête incidente à un noeud varie de façon aléatoire. Ce qui signifie que les joueurs ne vont pas jouer le même nombre de match.
Dans la deuxième situation, tous les joueurs ont le même nombre d'adversaires. A la première itération ils effectuent tous le même nombre de matchs.
Cette différence entre les deux situations amène à avoir plus de joueur après la deuxième itération dans une simulation avec le graphe aléatoire. De plus, dans le graphe aléatoire, les joueurs restant à la fin de la simulation on des niveaux d'énergie variés. Ce qui n'est pas le cas des joueurs restant à la fin de la simulation dans le cas d'un graphe circulaire. Dans ce deuxième cas, les quelques joueurs restant on toujours un niveau d'énergie faible.

4 Améliorations et perspectives

4.1 Évolution des probabilités

Un des éléments facultatifs du sujet de TP consistait à faire évoluer les probabilités de l'automate multi-stratégique à l'aide d'un algorithme génétique.
Cette implémentation nécessite plus de temps. Afin de palier à ce manque, il est possible de faire varier manuellement les probabilités via l'interface graphique. Cependant, cette méthode n'est pas sans défaut. En effet, il est possible à l'utilisateur d'entrer des valeurs non comprise entre $[0, 1]$ ou de réaliser une somme de probabilité supérieure à 1.

4.2 Types de graphe

Seuls deux types de graphe on été implémenté dans la configuration 3.
Il était nécessaire de présenter plus d'un type de graphe afin de permettre une comparaison entre eux. Le choix d'un graphe en anneau a été faite car il s'agit d'un modèle utilisé pour représenter des réseaux sociaux. Les graphes d'accointances semblent particulièrement adaptés pour cette exercice. En effet, ces graphes sont utilisés pour modéliser des interactions sociales, ce qui est le cas ici car les joueurs ne jouent des matchs qu'avec les joueurs de leur voisinage.