Analyse projet : jeu de la vie et jeu du jour et de la nuit

Clément ***** *– L3*

Le but de l'application est de simuler **le jeu de la vie**, **le jeu du jour et de la nuit (Day & Night)** mais aussi le jeu **Quad Life**, tous deux **des automates cellulaires**. L'application peut être réalisée dans le langage de programmation que l'on veut. Cependant, pour la réalisation du projet nous n'aurons pas le droit d'utiliser d'objet, seulement des fonctions. Pour simuler les deux automates, **nous réutiliseront les mêmes fonctions**.

Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps t+1 est fonction de l'état au temps t d'un nombre fini de cellules appelé son « voisinage ».

Les trois automates sont très **similaires**, seul les règles de vie et de mort en fonction du nombre de cellules voisines en vie à l'étape t + 1 changent.

Le jeu de la vie fut imaginé par John Horton Conway en 1970. Il est probablement le plus connu de tous les automates cellulaires.

À l'instar du jeu de la vie, Day & Night est un automate cellulaire bidimensionnel à deux états (« vivant » ou « mort »), où chaque cellule possède huit voisines. Il fut créé par Nathan Thompson en 1997. Les états « vivant » et « mort » sont symétriques : si l'état de toutes les cellules est inversé, l'évolution de la structure sera l'inverse de celui de la structure originale

Quad Life fonctionne exactement de la même façon que le jeu de la vie, à ceci près qu'il possède cinq états, dont quatre « vivants ».

Nous déciderons ici d'utiliser **Python** pour sa simplicité, sa syntaxe, mais aussi pour sa librairie **Tkinter** permettant de faire des interfaces rapidement.

Rappelons les règles du jeu de la vie :

Une cellule X en vie à l'étape t, avec moins de 2 ou plus de 4 cellules voisines vivantes, mourra à l'étape t + 1 pour cause, respectivement de solitude ou de surpopulation.

Une cellule X en vie à l'étape t, restera en vie à l'étape t + 1 si cette cellule X possède 2 ou 3 cellules voisines en vie.

Une cellule X morte à l'étape t, sera de nouveau en vie à l'étape t + 1 si cette cellule X possède 3 cellules voisines en vie.

Maintenant les règles du jeu du jour et de la nuit :

Une cellule X en vie à l'étape t, avec moins de 3 ou avec 1, 2 ou 5 cellules voisines vivantes, mourra à l'étape t + 1.

Une cellule X en vie à l'étape t, restera en vie à l'étape t + 1 si cette cellule X possède 3, 4, 6, 7 ou 8 cellules voisines en vie.

Une cellule X morte à l'étape t, sera de nouveau en vie à l'étape t + 1 si cette cellule X possède 3, 6, 7 ou 8 cellules voisines en vie.

Les règles de l’automate Quad Life :

Une cellule morte y naît à l'étape suivante si elle est entourée de 3 voisines, une cellule vivante survit à l'étape suivante si elle est entourée de 2 ou 3 cellules vivantes.

Lorsqu'une cellule naît, si toutes les cellules qui lui ont donné naissance se trouvent dans des états différents, la nouvelle cellule prend l'état restant.

Dans le cas contraire, elle prend l'état de la majorité des trois cellules.

La première étape est la création d’une grille, qui contiendra les cellules de notre automate, pour la dessiner nous utiliserons une fonction, fonction qui resservira à chaque fois que nous voudrons « nettoyer » l’ancienne simulation et repartir sur de bonne base.

Fonction : CreationGrille()

Elle ne prendra aucuns paramètres et ne retournera aucunes valeurs. La création de grille se basera sur la taille des cellules (taille fixe) ainsi que sur le nombre de cellules par ligne (pour calculer la taille d’une ligne, taille d’une ligne = taille d’une cellule * le nombre de cellule). Pour dessiner la grille nous tracerons des lignes verticales et horizontales.

Qui dit création, dit réinitialisation, comme dit précédemment, nous devrons réinitialiser la grille afin de remettre à 0 l’automate, pour pouvoir redessiner dessus.

Fonction : ReinitialiserGrille()

Elle ne prendra aucuns paramètres et ne retournera aucunes valeurs. Elle appellera CreationGrille() pour redessiner la grille après avoir réinitialisées, supprimées toutes les cellules vivantes.

Maintenant que nous avons la grille avec les cellules, il faut pouvoir décider quelles cellules nous voulons ressusciter, le clique gauche de la souris est parfaite pour cela.

Fonction : CliqueGaucheSurCellule(unEvenement)

La méthode prendra en paramètre l’évènement du clique, avec ses informations, comme les coordonnées sur l’écran etc. Elle aura pour but de ressusciter la cellule visée. La fonction ne retournera rien.

Nous pouvons créer des cellules vivantes, il faut désormais pouvoir les retirer, pour cela le clique droit de la souris est adapté, cela reste intuitif.

Fonction : CliqueDroitSurCellule(unEvenement)

Comme la méthode CliqueGaucheSurCellule, cette méthode prendra en paramètre l’évènement du clique et ses informations, elle ne retournera rien.

Afin de déterminer de quelle cellule nous parlons, lorsque l’on veut interagir avec elle (avec cliques de souris par exemple), il faut calculer ses coordonnées (ligne : colonne), lorsque nous cliquons, pour rester avec l’exemple précèdent, nous avons juste des coordonnées de l’écran, il faut donc une fonction pour déterminer quelle cellule est visée.

Fonction : CalculCoordEtNumLigne(unEvenement)

En paramètre elle prendra l'évènement, comme pour les cliques, pour avoir toutes ses informations, la fonction retournera la position x et y de l'évènement mais aussi la ligne et la colonne de la cellule, pour avoir ses coordonnées exactes.

Il faut désormais faire en sorte d'appliquer les règles des automates, pour cela nous ferons, tout d'abord, une fonction pour compter le nombre de cellules voisines en vie de chaque cellule.

Fonction : RechercheCellulesVoisines(ligne, colonne)

La fonction prendra en paramètre la ligne et la colonne de la cellule dont on analysera ses voisines. Elle retournera le nombre de voisine en vie d'une cellule.

Nous avons les cellules voisines en vie de chaque cellule, nous pouvons décider de leur futur à l'étape t + 1, de là est née la fonction…

Fonction : FuturDeLaCellule(NbrVoisineEnVie, ligne, colonne)

La fonction prendra en paramètre le nombre de voisine en vie, déterminer avec la fonction précédente RechercheCelluleVoisine, la ligne et la colonne de la cellule dont nous voulons savoir son futur. La méthode ne retournera rien mais modifiera la cellule sur la grille pour l'étape suivante. Les règles des deux modes de jeux y sont inscrites sous forme de condition, les règles qui s'appliqueront dépendront du mode de jeu sélectionné.

Pour changer de mode de jeu il nous faudra une fonction.

Fonction : ChangeModeDeJeu()

La fonction ne prendra aucuns paramètres, étant donné que nous avons trois modes de jeux, nous incrémenterons le mode de jeu de 1, puis s'il dépasse trois retournera à 1 et ainsi de suite.

Maintenant que tout cela est en place, il nous faut un moyen de faire « l'animation », de passer de l'étape t à l'étape t + 1 tout en changeant l'état des cellules en fonction du mode de jeu sélectionné.

Fonction : MiseAJourCellules()

La fonction ne prendra aucuns paramètres et n'en retournera aucuns, elle parcourra la liste des cellules pour les mettre à jour, avec les fonctions créées précédemment. Pour finir, les moyens de commencer et de terminer « l'animation », nous programmerons des fonctions d'arrêt et de lancement.

Fonction : DemarrerBtnEvent()

Fonction : ArreterBtnEvent()

Des fonctions qui permettront de lancer l'animation et d'arrêter l'animation à une certaine étape. Ne prenant et ne retournant aucunes valeurs.