Rapport de projet

Schecroun Mathis et Renaudin Ariane

May 2024

Introduction

Dans le cadre de la classe de NSI en terminale, nous avons réalisé un projet visant à programmer sur Python, une simulation de propagation d'incendie. Ce projet, réalisé par les élèves Ariane Renaudin et Mathis Schecroun consiste à programmer deux fonctions visant à simuler la propagation d'un incendie. La première fonction modélise une simulation *simple* tandis que la seconde permet de simuler une propagation plus réaliste avec des probabilités. Le projet inclut également la création d'une fonction permettant de générer une map de taille paramétrable aléatoirement avec des zones hétérogènes. Alors qu'Ariane Renaudin s'est chargé de la partie *création de map* avec l'utilisation du module Python Pygame, Mathis Schecroun a traité la partie simulation.

1 Développement du projet

Dans un premier temps, Ariane a appris à utiliser Pygame pour générer des maps aléatoires non intelligente. Il s'agissait simplement d'un amas de Parcelle de couleurs différentes ne formant pas de zone homogène. Mathis quant à lui a réfléchis à une manière de modéliser une parcelle. Une fois qu'Ariane a appris à utiliser Pygame, elle a tenté de créer une map intelligente avec des zones homogènes de plaine, de foret, d'eau et avec quelques maisons. Mathis quant à lui, a travaillé avec un modèle réduit de map définit manuellement afin de mettre au point la simulation simple et la simulation avec probabilité. Une fois que toutes ces fonctions ont été crée, nous avons relié les deux codes afin que la simulation programmé par Mathis agisse sur la map générée par la fonction de Shoto.

2 Définition d'une grille

Ce code Python vise à générer une carte en utilisant une grille de chiffres pour représenter différents types de parcelles, tels que des plaines, des forêts, de l'eau et des maisons. Voici comment le processus fonctionne :

Tout d'abord, la fonction generer_terrain(grid) initialise une grille remplie de points, puis sélectionne aléatoirement un nombre défini de cellules dans cette grille et les remplace par des chiffres (1, 2 ou 3) en fonction de probabilités données. Ces chiffres représentent différents types de terrain : 1 pour les plaines, 2 pour les forêts et 3 pour l'eau. Ensuite, elle met à jour les cellules voisines

avec le même chiffre pour créer des zones de terrain homogènes. Pendant ce processus, les coordonnées des cellules modifiées sont stockées dans une liste.

Ensuite, la fonction generer_maison(grid) ajoute un nombre aléatoire de maisons à la grille. Elle sélectionne aléatoirement des coordonnées pour chaque maison, en veillant à ce qu'il y ait suffisamment d'espace pour une maison de taille fixe (10x10). Ces maisons sont représentées par le chiffre 4 dans la grille.

La fonction generer_voisin(grid) est utilisée pour étendre les zones de terrain en remplaçant les cellules vides voisines des cellules déjà remplies avec le même chiffre. Elle parcourt la liste des coordonnées des cellules modifiées lors de la génération du terrain initial, et pour chaque cellule, elle explore les cellules voisines vides et les remplit avec le même chiffre.

Enfin, la fonction creation_terrain(grid) convertit les chiffres de la grille en instances de la classe Parcelle correspondantes, en fonction de leur valeur numérique. Par exemple, un chiffre de 1 représente une parcelle de plaine, un chiffre de 2 une parcelle de forêt, etc. La grille finale représente ainsi une carte composée de différentes parcelles de terrain, prête à être utilisée pour des simulations ou des visualisations.

3 Utilisation de Pygame

Ce code est conçu pour être une partie d'une application utilisant Pygame pour afficher une simulation graphique basée sur une grille de parcelles de terrain. Voici un paragraphe qui explique son fonctionnement :

La fonction affichage(grid) est essentielle pour rendre visuellement la grille de parcelles sur la fenêtre Pygame. Elle parcourt chaque cellule de la grille et dessine un carré de taille fixe à l'emplacement correspondant sur l'écran en fonction de la nature de la parcelle et de ses caractéristiques telles que l'intensité, l'estompage et la vie.

Pygame offre une interface simple mais puissante pour la création de graphiques en 2D dans Python. Dans ce contexte, la fonction pygame.draw.rect() est utilisée pour dessiner des rectangles (carrés dans ce cas) sur la surface de l'écran. Chaque cellule de la grille est représentée par un carré dont la couleur dépend de sa nature : bleu pour l'eau, vert pour la forêt, jaune pour les plaines et marron pour les maisons. De plus, des nuances d'orange et de rouge sont utilisées pour indiquer différentes intensités de destruction.

Cette méthode de rendu permet de visualiser rapidement et efficacement l'état actuel de la simulation, en permettant aux utilisateurs de comprendre visuellement ce qui se passe dans le modèle de simulation sous-jacent. Cela rend l'expérience utilisateur plus immersive et intuitive, ce qui est essentiel pour les simulations interactives ou les jeux basés sur la grille.

4 Définition d'une parcelle

Comme expliqué précédemment, la grille est composé d'un certain nombre de parcelle, dépendant de la taille de la grille, celle-ci étant paramétrable par l'utilisateur. Il a donc fallu définir un objet Parcelle. Celui ci prend en paramètre un type de terrain ("foret","maison","eau" ou "plaine). Ce type de terrain devient la valeur de l'attribut **nature**. Nous créons également d'autres attributs

tel que **vie** qui est un booléen et qui prend la valeur True si la parcelle n'est pas calciné. Les autres attributs sont **intensité** qui prend la valeur de l'intensité du feu et 0 si la parcelle n'est pas brulée ou est calcinée. Il y a également un attribut **est_estompe** qui prend la valeur Vrai si l'intensité du feu est en phase de diminution et enfin l'attribut **intensite_max** qui dépend de la nature de la parcelle et qui définit l'intensité maximal du feu pouvant exister sur la parcelle avant que le feu diminue. L'objet parcelle possède également 2 méthodes. Un première permettant de passer les paramètres correspondant à une parcelle calciné. La deuxième fonction retourne un booléen qui prend la valeur Vrai si la parcelle est en feu.

5 Simulation simple

La simulation simple suit un principe comme son nom l'indique très simple. Lorsqu'une parcelle est en feu, toutes ces voisines se retrouvent en feu au tour suivant sans contrainte de probabilité ou de condition météorologique. Pour cela, on défini dans un fichier **voisin** une fonction prend en paramètre une liste et les coordonnées d'un élément et retournant les voisins de cet élément dans la liste. Ainsi, nous devons avoir connaissance des coordonnées de la parcelle en question. Pour cela, à chaque fois qu'une parcelle va prendre feu, nous ajouterons dans une liste feu, une nouvelle liste à 3 éléments : l'objet Parcelle et les deux coordonnées de la Parcelle. Ainsi à chaque tour de boucle, nous regardons la totalité des voisins d'un élément n'étant pas eu feu, et si c'est le cas est que la nature de la parcelle n'est pas de l'eau, alors le voisin en question prend feu au tour suivant. Nous avons également crée la fonction **unite()** qui prend en paramètre une parcelle et qui retourne les opérations que celle ci doit subir si elle est en feu. On exécute donc pour chaque tour de boucle la fonction unite() pour toutes les parcelles de la liste **feu**. La fonction de simulation s'arrête lorsqu'aucun élément n'a changé d'état entre deux unités de temps. Pour vérifier cela, nous copions la position de la map à chaque tour de boucle. La boucle s'arrête si la copie de la map et la map après les éventuelles modifications sont identiques.

6 Simulation avec probabilité

Ce type de simulation est beaucoup plus précis et réaliste que la simulation simple dont nous avons parlez précédemment. En effet, la simulation prend des éléments en compte tel que la nature des parcelles, la positions des parcelles les unes par rapport aux autres ou encore les conditions météorologique comme le vent, la pluie ou la foudre.

6.1 Fonction de probabilité

Nous avons donc crée une fonction calculant la probabilité qu'une parcelle prenne feu d'abord sans prendre en compte le climat. La formule prend en compte les éléments suivants : la distance entre les deux parcelles (coté ou diagonale), la nature du voisin (plaine, foret, maison...), l'intensité du feu sur la parcelle et l'intensité maximal qu'il peut y avoir sur cette parcelle. On crée d'abord un premier coefficient qui dépend de la distance et qui prend la valeur

75 si les deux éléments sont cote à cote et qui prend la valeur 25 si les deux parcelles sont en diagonal l'une par rapport à l'autre. On crée un deuxième coefficient prenant la valeur 1 si la nature de la parcelle voisine est une forêt et 0.5 si c'est une maison ou une plaine. On calcule la première probabilité avec la formule :

$$P = coef\_dis \times coef\_nat \times 0.75^{intensite\_max - intensite}$$

P est un pourcentage représentant la probabilité qu'une voisine prenne feu. Nous devons maintenant ajouter à ça le facteur météorologique

6.2 Vent

Nous générons d'abord un sens de vent aléatoirement selon des propriétés paramétrables définis au préalable. Ce sens du vent se matérialise pas une chaîne de caractère de deux caractères : la première lettre du point de départ du vent et la première lettre du point d'arrivée du vent. Par exemple, si le vent va de droite à gauche, la chaîne de caractère sera : 'dg'. La fonction retourne également le nombre de tour durant lequel le vent aura cette direction. Ainsi avec ce sens du vent, si la parcelle voisine à la parcelle en feu se situe dans le sens du vent par rapport à la parcelle en feu, on multiplie par 1.25 la probabilité que la voisine prenne feu. Dans le cas contraire, on établie ce coefficient à 0 car la parcelle voisine n'a aucune chance de prendre feu si le vent va dans le sens contraire

6.3 Pluie

La pluie quant à elle est binaire : soit il y en a soit il y en a pas. Tout comme le vent, on crée une fonction qui définit si la pluie tombe ou non ainsi que le nombre de tour où l'on effectuera cette action. Si la pluie tombe, on diminue la probabilité que les voisins prennent feu de 0.25. On obtient donc grâce au vent et à la pluie la probabilité finale. Pour que la propagation vérifie ces probas, on utilise la fonction random() qui génère un réel entre 0 et 1. Si la probabilité que l'on a calculé au préalable est inférieur au réel tiré au sort, alors le voisin prend feu

6.4 Foudre

La foudre tombe potentiellement les 10 tours selon une probabilité paramétrable. Si la foudre tombe sur une parcelle qui n'a pas pris feu, celle ci prend feu automatiquement. Néanmoins, plus un nombre important de parcelle brûle, moins la foudre a de chance de frapper une parcelle qui brûle. On tire ainsi au sort une parcelle : si elle n'a jamais pris feu elle prend feu, si elle a pris feu on ne fait rien.

6.5 Simulation

Ainsi, pour effectuer la simulation avancée, on suit le même principe que pour la simulation simple sauf que l'on utilise les probabilités calculés comme vu précédemment pour définir si oui ou non une parcelle prend feu. La manière de brûler des parcelles n'est quant à elle pas différente de la simulation simple.

7 Assemblage des deux parties

Lors de la tentative d'assemblage des deux parties, Pygame ne répondait pas sans qu'il soit possible de savoir pourquoi. Nous ne savons pas si le problème est lié au code ou à l'ordinateur.

8 Conclusion

Nous avons donc programmé deux modèle de simulation de propagation d'incendie : un simple et un utilisant des probabilités. Malheureusement, nous avons été en incapacité de lié la simulation à la map Pygame pour des raisons qui nous sont encore inconnus.