Projet GI

2021-2022

Zaouche Djaouida

Enoncé

Vous contrôlez un personnage sur une carte qui doit rejoindre sa maison. Le joueur peut déplacer son personnage sur une carte $2D$ de dimensions fixées. Dans chaque case de la grille, des éléments comme des obstacles (rocher, arbre) ou des bonus (nourriture seront placés). Une case peut être vide. Le principe du jeu est très simple : il s'agit de déplacer le personnage sur la carte. Initialement, le personnage se trouve à la case 0 (i.e. la première ligne, la première colonne). Il doit atteindre la case finale (i.e. la dernière ligne, la dernière colonne). Une distance est associée à chaque couple de cases adjacentes x et y. Elle représente la distance à parcourir pour atteindre la case y en partant de la case x. Le personnage dispose d'une énergie initiale fixée à l'avance.

Le personnage ne peut pas faire de saut, il ne se déplace que vers une case adjacente. A chaque déplacement, il perd une unité de son énergie indépendamment de la distance parcourue. Partant de la case x vers la case y, nous avons les cas suivants :

- y est une case vide, le personnage est autorisé à accéder à la case y.
- y est un obstacle, le personnage rebrousse son chemin et revient à la case x. Malheureusement il perd encore 10 unités de son énergie.
- y est un bonus, son énergie est augmentée de 10 unités.

Tant que le personnage n'a pas rejoint sa destination finale et que le personnage n'est pas mort (énergie tombée à 0), le joueur continuera à déplacer son personnage en appuyant sur les touches →, ←, ↑ et ↓ en utilisant le clavier.

Après chaque saisie autorisée (le personnage ne doit pas sortir de la grille), la carte est actualisée suivant le type de case sur laquelle s'est arrêté le personnage :

- vide : pas de changement sur cette case,

- un obstacle, le personnage n'a pas pu aller sur cette case qui reste inchangée,
- de la nourriture : la nourriture disparait (puisque le joueur la prend) et la case devient vide.

La carte mise à jour avec la nouvelle position du joueur doit être réaffichée à chaque fois.

Travail demandé

1. Implémenter les classes nécessaires afin de permettre au joueur d'atteindre sa destination finale.
2. Pour chaque déplacement, afficher une alerte si le joueur revient sur une case x déjà visitée, la boucle parcourue de x vers x est affichée.
3. Rajouter une méthode qui permet au joueur de défaire un ou plusieurs déplacements, un par un. Un joueur peut défaire au plus 6 déplacements.
4. A la fin du jeu et en cas de succès, vous devez :
   - Afficher la distance parcourue, l'énergie restante, l'énergie gagnée et l'énergie perdue.
   - Afficher le chemin emprunté par le personnage.
   - Afficher le meilleur chemin à prendre en terme d'énergie.
   - Afficher le meilleur chemin à prendre en terme de distance.
5. Donner au joueur la possibilité d'arrêter une partie et de la restaurer par la suite. Les informations de la partie doivent être sauvegardées dans une base de données (la date de la partie interrompue, l'heure de début et de la fin de la portion jouée, l'énergie perdue, l'énergie gagnée, ….).
6. Donner au joueur la possibilité de voir l'historique des parties **jouées**. On suppose qu'on a un seul joueur XXX. Le joueur pourra choisir une partie donnée et revoir, en dessin animé, le personnage emprunté le chemin défini dans la partie. Ici considérez deux types de parties : partie interrompue et partie terminée. Le joueur ne pourra faire un *replay* que pour les parties terminées. Prévoir trois vitesses pour l'animation (lente, moyenne et rapide).
7. Générer la carte de manière aléatoire et l'énergie du personnage. Vérifier qu'il existe au moins un chemin qui permettrait au personnage, partant de la position initiale, d'atteindre la position finale. Dans le cas négatif, regénérer la carte.
   Considérer deux cas :
   - Un chemin sans tenir compte du paramètre énergie.
   - Un chemin en tenant compte de ce paramètre.

8. Tester l'algorithme des k-proches-voisins sur les données "data.csv", Utiliser 80% des données pour l'apprentissage et 20% pour les tests. Calculer le taux de fiabilité du modèle obtenu. Conclure.

Analyse UML

Votre code doit être bien conçu. Il ne s'agit plus seulement de coder un jeu en 2 heures, mais de le concevoir, c'est-à-dire de réfléchir à sa structure et à son architecture, avant de le développer. Vous devrez donc fournir une analyse UML complète du projet : diagramme de cas d'utilisation, diagramme de classes, diagramme de séquences…

Interface graphique

Vous devrez implémenter une interface graphique en JavaFX permettant un meilleur contrôle de la partie. Les éléments graphiques peuvent être :

- Des boutons pour créer, recommencer, sauvegarder, charger ou quitter la partie;
- Un contrôle à la souris, ….
- Une fenêtre de jeu entièrement graphique, avec une représentation du joueur, des obstacles, bonus,…