Compte rendu du projet du module Info42

Julie Guignabert - Ludivine Roy

7 mai 2014

Sommaire

Rappel du sujet

1.1 Règles de fonctionnement

Pour notre projet, nous avons choisi le sujet n°2 : le jeu Mouse-Trap.
Nous rappelons le sujet : “Le joueur dirige une souris dans un labyrinthe dans le but de manger tous les morceaux de fromage du niveau. Ses ennemis sont des chats et des faucons. La souris peut collecter des os qui lui permettent, à tout moment, de se transformer en chien quelques instants et de chasser les chats. Le joueur dispose de quatre boutons en plus du joystick directionnel. Le premier bouton permet au joueur de se transformer en chien s'il possède, au moins, un os. Les trois autres permettent au joueur de fermer ou d'ouvrir, à tout moment, les portes jaunes, rouges et bleues et ainsi de bloquer ou emprisonner les chats en transformant l'apparence du labyrinthe.“
Pour notre projet, nous nous sommes inspirées du célèbre jeu `PacMan`. En ce qui concerne les règles retenues, nous voulions respecter toutes les règles de base du jeu Mouse-Trap tout en gardant l'esprit ”PacMan“, puis au fur et à mesure nous avons préféré nous focaliser sur les points importants du jeu (tels que les déplacements des personnages ou la possibilité de jouer à plusieurs en réseau), pour qu'ensuite potentiellement, une fois fini, nous apportions des améliorations. Cependant, nous n'avons pas eu la possibilité de faire ces modifications avant la présentation du projet (nous pensons personnellement à nous y intéresser après les examens finaux).
Les principales fonctionnalités du jeu sont la possibilité de jouer à plusieurs niveaux en mode solo (jusqu'à cinq niveaux), ainsi que de jouer à plusieurs personnes en réseau. En mode solo, le joueur controle PacMan et doit ramasser toutes les pièces se trouvant au sol tout en évitant de se faire attraper par les fantomes controlés par l'ordinateur. Lorsque toutes les pièces sont ramassées, le joueur change de niveau. PacMan a également la possibilité de devenir invincible en ramassant l'étoile rouge qui apparait de temps en temps sur la carte, et peut ainsi tuer les fantomes qu'il touche. Lorsque la partie est finie (donc soit que le joueur n'a plus de vie, soit qu'il a réussi à terminer tous les niveaux), le joueur peut entrer son pseudo afin que son score soit enregistré.
En mode multijoueur, le premier joueur connecté controle PacMan, les joueurs suivants controlent les fantomes. PacMan a pour objectif de récupérer toutes les pièces du niveau, les fantomes doivent l'en empecher.
Le menu Options propose d'activer ou de désactiver le son, donne les touches utilisées dans le jeu, et également accès aux meilleurs scores réalisés.

Les touches utilisées pour controler les personnages sont les fleches directionnelles.

1.2 Choix des structures de données

Nous voulions pouvoir travailler chacune sur une partie du code simultanément et pouvoir ensuite complémenter notre code, c'est pourquoi nous avons choisi d'utiliser un gestionnaire de versions. Nous avons choisi `Git` dont nous avions déjà entendu parlé et qui est plutot simple et pratique à utiliser.
Ensuite, lorsque nous avons commencé à réfléchir à notre projet, il nous a fallu trouver une bibliothèque graphique. Nous avons fait quelques recherches : nous avons hésité entre la bibliothèque graphique de base avec Java (AWT) et Slick2D. Notre choix s'est finalement porté sur Slick2D, qui est une bibliothèque complète et simple d'utilisation (et qui avait l'air assez conseillée pour faire des jeux en 2D).

Ensuite, dans le code, nous voulions pouvoir modifier les constantes sans problème, c'est pourquoi nous avons créé des fichiers de configuration dans lesquels se trouvent les constantes (hauteur et largeur de la carte, vitesse des personnages, taille des images du mur et des personnages, positions de départ des fantomes, écart entre le bord de la fenêtre et les bords de la carte etc.). Chaque niveau du mode solo a son propre fichier de configuration.

Architecture générale

Fonctionnalités en détail

3.3 Affichage des fenetres - Les états

Slick2D nous a permis de créer différents états (que ce soit le menu ou les niveaux de jeu) qui sont continuellement exécutés par le compilateur ce qui permet de les mettre a jour sans arrêts. Ceci est essentiel pour comptabiliser le score et la vie du joueur au fur et à mesure du jeu. Les états doivent hériter d'une classe présente dans Slick2D : `StateBasedGame`, qui surcharge trois fonctions principales qui sont exécutées continuellement : `init`, `render` et `update`. La première fonction permet d'initialiser les variables, la deuxième permet d'afficher tout ce qu'il faut pour le bon fonctionnement du jeu, et la troisième permet la mise à jour de l'état des objets.
La fonction `init` prenant en paramètre le conteneur de jeu (`GameContainer`) et l'état de jeu (`StateBasedGame`) permet d'initialiser les objets du jeu, c'est à dire les images, les cartes, le Pacman, les fantômes et leurs configurations.
La fonction `render` prend les mêmes paramètres que la fonction `init` et un graphisme permettant l'affichage de la carte. C'est dans cette fonction que tous les objets sont affichés a l'écran.
La fonction `update` prend aussi les mêmes paramètres que la fonction `init` mais avec une variable correspondant au temps entre deux mises a jours de l'état. Cette fonction permet le changement d'état et la réinitialisation de la carte après avoir perdu ou changé de niveau, mais aussi le déplacement des personnages ainsi que la gestion des collisions.
Tous les états de jeux comprennent les fonctions renvoyant le score que le joueur a fait dans ce niveau et la vie qu'il lui reste, pour les mettre à jour dans les niveaux suivants. Le score n'est comptabilisé que si le joueur perd la partie ou s'il gagne tous les niveaux ce qui lui permet de voir quel score il fait à chaque niveau, les étoiles rapportant plus que les pièces, et les fantômes "mangés" en mode invincible plus que les étoiles.

Parmi les états présents dans le programme, il n'y a pas que ceux qui font tourner le jeu, il y a aussi les menus. Ces derniers comprennent les mêmes fonctions (ce sont également des états). Les fonctions `init` et `render` sont les mêmes, la fonction `update` en revanche ne sert pas à faire bouger les personnages mais à faire fonctionner les boutons, c'est à dire pour entrer dans les jeux (un joueur ou multi joueurs) ou pour accéder aux options.
Après avoir changé d'état grâce à ces boutons, il faut réinitialiser les coordonnées de la souris pour ne pas que le compilateur mémorise les coordonnées du bouton sinon il reviendra sur l'état de celui-ci sans cesse. Les états menus comprennent donc aussi une fonction permettant de récupérer les coordonnées du clic de souris pour accéder à un autre état.
Tous les états possèdent leurs propres ID (valeur numérique) permettant de les différencier. C'est cet ID qui est appelé à chaque changement d'état et qui caractérise celui-ci. Tous ces états sont initialisés dans une classe (`windowGame`) qui hérite également de `StateBasedGame`. C'est cette classe qui permet l'exécution des états dans le même conteneur de jeu.

3.4 Affichage de la carte du jeu

Nous avons décidé de représenter toutes les cartes dans des fichiers texte. Dans ces fichiers, chaque 1 représente un mur, chaque 2 représente une étoile, chaque 3 représente une pièce, et chaque 4 des cases qui n'affichent rien (c'est à cet endroit que les fantomes apparaissent à chaque début de niveau ou quand ils viennent d'etre touchés par le PacMan invincible).

Dans le constructeur de la classe `Map`, on donne en paramètre un `String` contenant le chemin du fichier texte. Une fois le fichier chargé dans un objet `FileInputStream`, on parcoure tous les caractères un par un et les met dans un tableau d'entiers à deux dimensions : en ligne sont représentées les coordonnées en abscisse, en colonne ce sont les coordonnées en ordonnée.
Ensuite, grace à la fonction `afficheMap`, on parcoure le tableau et on affiche chaque case une par une, en fonction de l'entier contenu dans le tableau.
Au bout d'un certain moment dans le jeu, une étoile apparait aléatoirement dans la carte (c'est l'étoile d'invincibilité), avec la fonction `spawnEtoile`. Dans cette fonction, on prend aléatoirement des coordonnées x et y, puis on vérifie qu'il n'y ni mur, ni étoile à cet emplacement. Si c'est le cas, on reprend aléatoirement des valeurs et on vérifie à nouveau.

3.5 Les personnages - Classes `JoueurPacman` et `Fantome`

Pour chaque personnage nous avons créé une classe correspondante, c'est à dire une classe `JoueurPacman` et `Fantome`, toutes les deux héritant de la classe abstraite `Joueur`. Nous allons d'abord nous intéresser à la classe JoueurPacman.
Il y a plusieurs fonctions importantes dans cette classe :

`JoueurPacman` - Fonctions de déplacement

La fonction `seDeplacer` a son équivalent pour le serveur `seDeplacerServeur`, le principe est le même pour les deux fonctions.
Une suite de conditions teste les entrées clavier (dans la deuxième fonction, la direction à prendre est donnée par un `String` en paramètre), lorsque la direction demandée est trouvée, le programme regarde s'il est possible de se déplacer dans cette direction.
Il vérifie d'abord si la case juste à côté du personnage dans la direction que l'on veut suivre est différente de 1 (1 est utilisé pour représenter les murs dans le fichier texte de la carte, donc par exemple, si le joueur veut se déplacer vers la gauche, on regarde si la case à gauche de celle où est placé le personnage est un 1), sinon il vérifie qu'il y a assez de place pour parcourir 2 pixels (les personnages se déplacent de 2 pixels à chaque fois). Si ce n'est pas le cas, on calcule la distance entre le personnage et le mur, puis on déplace le personnage de cette distance.

`JoueurPacman` - Gestion des collisions

Il s'agit de gérer les moments où le PacMan rencontre un fantôme, une étoile, une pièce ou encore quand il rencontre un mur (la collision avec le mur a été traitée dans la fonction précédente). A chaque collision le programme doit réagir différemment pour, par exemple, ne pas que le PacMan se retrouve sur un mur ou pour qu'il ramasse les pièces et qu'elles disparaissent ensuite.
Le programme ne doit également pas réagir de la même façon si le PacMan est en mode invincible, invisible ou non pour le contact des fantômes. Il doit "manger" les fantômes en mode invincible, passer à travers ceux-ci en mode invisible et se "faire manger" dans le cas contraire. A chaque début de niveau ou "mort" du PacMan, ce dernier est en mode invisible, c'est à dire que les fantômes ne peuvent pas le "manger". Ceci offre au joueur un temps de réaction pour reprendre en main son personnage et permet de ne pas perdre la partie si un fantôme est présent sur le point de départ du PacMan.
La fonction `gestionContact` s'occupe de toutes les collisions avec les fantomes. Elle prend en paramètre un tableau comprenant les fantomes du niveau actuel. Il y atrois cas distincts dans cette fonction : quand PacMan est invincible, invisible ou aucun des deux.
Lorsque PacMan est invincible, on vérifie d'abord que le timer (compteur qui sert à limiter l'invincibilité dans le temps) est supérieur à 0 (et qu'il reste donc du temps d'invincibilité). Ensuite, pour chaque fantome du tableau, on dessine une sorte de "carré" autour du PacMan délimitant ainsi les zones de collision. Si un fantome se trouve dans ce carré, c'est qu'il y a collision, et le fantome touché revient à son point de départ.
Lorsque PacMan n'est pas invisible, le principe est exactement le meme, sauf que s'il y a collision, c'est PacMan qui retourne à son point de départ et il perd une vie. S'il n'a plus de vie, la partie est finie.
Et enfin, lorsque PacMan est invisible, il n'y pas de collision possible avec les fantomes. On décrémente

le compteur, et une fois le compteur arrivé à 0, PacMan n'est plus invisible.

La fonction `setScore` permet de gérer la collision avec les pièces et les étoiles. Si la case sur laquelle se trouve PacMan est un 3, on incrémente le score de 100 points, puis on remplace le 3 par un 0. Si c'est un 2, c'est que c'est une étoile qui est apparue sur une pièce, donc PacMan devient invincible, on rajoute 200 points pour avoir attrapé l'étoile et on incrémente le compteur de pièces ramassées. Si c'est un 5, c'est juste une étoile, donc on rajoute 200 points et on met PacMan en invincible.

Nous passons maintenant à la classe `Fantome`.

`Fantome` - Fonctions de déplacement

Implémentation

Tests et utilisation

Conclusion et évolutions possibles

Bibliographie