Rapport du Projet Frogger

Introduction

Nous avons réalisé ce projet en binôme lors des dernières vacances pour la partie programmation, puis après la rentrée, des réunions quotidiennes étaient faites pour discuter des nouvelles idées à mettre en place pour finaliser au mieux ce travail qui nous a permis d'approfondir nos connaissances dans le langage Java. Au début, nous avons rencontré quelques difficultés décrites ci-après, mais au fur et à mesure de l'avancement du projet, la résolution des problèmes devenait de plus en plus facile. Nous avons ainsi pu rajouter plusieurs éléments comme le son et les images pour que le rendu soit plus joli. Ci-après une description résumant ce travail avec les difficultés rencontrées.

1- La grenouille

Nous n'avons pas eu de difficulté à ce niveau-ci, aussi bien dans la conception que dans les déplacements.

2- L'environnement : Suite de routes

Difficultés :

- La principale difficulté consistait à trouver les programmes de fonctionnement des objets (voitures, routes). -> Nous avons alors observé l'exemple du résultat attendu pour comprendre les comportements des différents objets.
- Nous avons eu beaucoup de mal à réaliser la fonction ''update'' pour la route car nous n'avions pas très bien compris le principe du « tick d'horloge ». d'où le déplacement des voitures à la même vitesse avec leur disparition puis leur apparition. -> Nous avons fini par faire le lien entre le temps d'exécution du programme et la vitesse de déplacement des voitures. Quand le tempo est inférieur à la vitesse de la voiture, celle-ci ne doit pas bouger et quand le tempo est égal à celui de la vitesse de la voiture, elle peut avancer d'une case, ce qui donne à la fin l'effet que les voitures se déplacent à vitesses différentes. Donc il fallait donner un paramètre booléen à la méthode ''déplacer'' pour faire avancer ou non la voiture.

3- Jeu infini

Dans cette partie, la principale difficulté était de lier les différents attributs entre les objets. Au départ, nous comptions faire en sorte qu'à chaque déplacement de la grenouille vers le haut, une nouvelle route soit créé et placée en haut de la grille, tandis que l'ancienne route dans laquelle se trouvait la grenouille soit supprimée. Dans ce cas, la grenouille n'aurait pas eu la possibilité de revenir en arrière. Pour cela, il fallait recréer une nouvelle route placée au début du tableau en faisant attention à ce que ce dernier garde une mémoire de routes (les anciennes routes). Nous avons donc introduit 3 postulats:

- Si la grenouille se déplace vers le haut, elle gagne +1 dans son ordonnée mais restera immobile graphiquement.
- La taille du tableau de routes doit augmenter tout en gardant la taille de l'interface graphique stable. L'avantage de cela est que le tableau aura en mémoire les routes précédentes.
- Dans le cas où la grenouille se déplace vers le haut, il faut que les routes se déplacent vers le bas, et qu'une nouvelle route soit rajoutée en haut de la grille.

Pour résoudre le premier point, nous avons modifié notre déclaration « Element » lié à la grenouille, étant donné qu'il y avait deux constructeurs correspondant à cet objet. Pour résoudre le deuxième point, nous avons rajouté une fonction permettant d'avoir de nouvelles routes sur le tableau que nous avions placé par la suite dans le ''move'' vers le haut de ''frog''.

Le dernier point nous posait plus de problèmes, surtout pour faire baisser les routes vers le bas. Nous avons trouvé la solution en jouant sur l'affichage graphique des voitures de sorte que lors du déplacement de la grenouille vers le haut, les voitures baissaient en ordonnée et vice versa. De ce fait, notre tableau contenait dans sa mémoire les anciennes routes.

4 Éléments complémentaires

4.1 Timer

Cette partie a été très simple à réaliser, en cherchant sur internet, on a vu que c'était possible d'avoir l'heure exacte de la machine avec System.currentTime. Nous avons donc créer une classe Chrono avec une méthode start() que nous avoions placé dans notre constructeur Game et qui prenait l'heure où le jeu a commencé, et une méthode stop() que nous avons placé aux testLoose() ou testWin() qui prenait l'heure où le jeu a été stoppé pour ensuite calculer la différence entre le temps du début et le temps de la fin.

4.2 Cases spéciales

Nous avons créé principalement trois objets spéciaux: Le premier est l'arbre situé uniquement dans les endroits considérés comme ''safe'' ce qui limite les cases dans lesquels la grenouille peut se poser. Pour cet objet, l'implémentation était simple, sauf lorsque l'arbre était plus volumineux, il fallait que la grenouille le contourne pour pouvoir passer. Le second objet est la toile d'araignée qui ralenti les déplacements de la grenouille. Le joueur devra appuyer trois fois sur la touche correspondant à la direction vers laquelle il souhaite aller pour que la grenouille puisse sortir de la toile. Cet objet est situé uniquement au milieu des routes. Le dernier objet n'est pas un obstacle, c'est une pièce de monnaie qui, lorsque la grenouille passe par-dessus, permet au joueur d'avoir un bonus de 1 point.

4.3 Rivière et Rondins

Pour cette partie, nous avons d'abord créé une nouvelle classe ''rivière'' avec les propriétés de ''Lane'' et une classe ''rondins'' avec les propriétés de ''Car''. Puis nous avons voulu superposer la classe rivière par-dessus ''Lane'', mais on ne pouvais pas déplacer la grenouille de Lane à Rivière. Des erreurs du type ArrayOutOfBands s'affichaient. On a donc décidé de créer la rivière avec la classe Lane en ajoutant un booléen avec ''true'' correspondant à la ligne de route avec les voitures et ''false'' à la rivière avec les rondins. On a ensuite changé les propriétés pour que la grenouille considère les rondins comme safe et l'eau comme dangereux. Il ne restait plus qu'à faire déplacer la grenouille quand elle se trouvait au-dessus des rondins. Pour cela, il y avait plusieurs méthodes. Nous avons utilisé les mêmes méthodes des voitures à grenouilles, c'est à dire move et leftToRight. Puis avons fait en sorte pour que, quand la grenouille est sur un rondin, elle ait les même propriétés que ce dernier, comme si elle devenait rondin elle-même. De ce fait quand elle se trouvait par-dessus, elle se déplaçait à la même vitesse et dans la même direction.

5 Bonus

5.1 Deux Joueurs

Cette partie a été très facile à réaliser, il suffisait de créer une deuxième classe frog2 avec exactement les même propriétés que frog1 mais avec une case de début différentes de frog1. Dans FroggerGraphics, nous avons rajouté dans KeyPressed(KeyEvent e) de nouvelles "case" correspondant aux touches ZQSD. Nous avons aussi fait quelques modifications dans Lane pour que celle-ci tienne compte du deuxième frog. Nous avons ensuite créé des TestLoose() et TestWin() propres au deuxième frog. De ce fait si l'une des grenouilles perd ou gagne, l'écran de fin affiche avec le temps de jeu, le score de chaque frog.

5.3 Meilleur rendu graphique

Au niveau du graphique, nous avons vu sur internet que les images étaient placées à l'intérieur de chaque classe d'objet, mais notre approche était différente: comme chaque élément du jeu avait sa propre couleur (toile d'araignée en blanc, pièce en jaune, grenouille en vert…), nous avons décidé de placer une image par rapport à la couleur de l'objet. Et Pour cela tout se faisait dans le paintComponent() de FroggerGraphics. Au début, on a loadé les images avec un BufferedImage et un ImageIO.read(new File(…)) puis placé dans chaque objet avec un drawImage. Cette façon fonctionne parfaitement bien mais à la fin du projet, en voulant créer un fichier jar correspondant au jeu, nous avons remarqué que les images étaient manquantes. On a donc du modifier la manière dont nous avions loadé les images et utilisé ImageIcon = new ImageIcon(getClass(). getClassLoader().getResource("Image/…")); cette manière de faire garde l'image en mémoire. On a ensuite placé les images dans chaque objet avec la méthode

paintIcon. Ce qui nous a permis de garder les images dans notre fichier jar. Au niveau des images, nous avons choisi principalement des PNG pour représenter les objets que nous avions redimensionnés par rapport à la taille d'une case (16 * 16). Puis des JPEG pour représenter l'environnement. Comme chaque image était placée sur une case, on a eu des difficultés à les placer dans les objets avec des longueurs supérieurs à la surface de la case comme les voitures avec longueur > 1 ou les rondins. Pour résoudre ce souci, on a fait en sorte que la couleur des voitures, soit en fonction de leur taille mais en ne représentant que sa première case et non pas sa case + sa longueur. De ce fait les voitures qui avaient, par exemple, une longueur de 2 étaient représenté avec une couleur différente mais sur une seul case. Puis l'image posé par-dessus l'objet était de longueur 32 ce qui correspondait à la longueur de 2 cases.

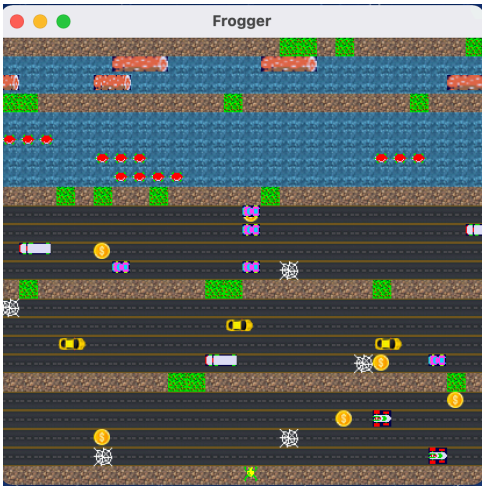
5.4 Vos idées !

Dans cette partie, nous avons décidé de lier tous les éléments (Frogger jeu simple, Frogger jeu à deux et Frogger Infinie) dans un seul et même jeu. Nous avons donc créé un menu d'accueil avec le choix du type de jeu représenté en boutons, nous avons aussi rajouté d'autres boutons comme :

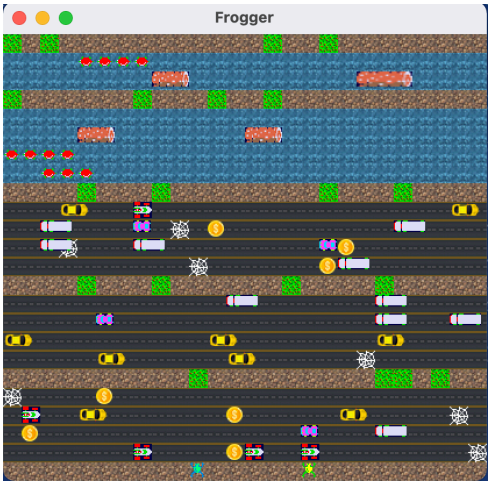
- Restart : pour relancer une nouvelle partie.
- Home : pour aller au menu d'accueil.
- Difficultés : pour avoir des niveaux différents (liés à des densités différentes).

Cette partie était très difficile à réaliser surtout pour lier les trois types de jeu ensembles. Au début, on n'avait pas assez de connaissance dans les swings, mais avec quelques recherches, nous avons compris le fonctionnement.

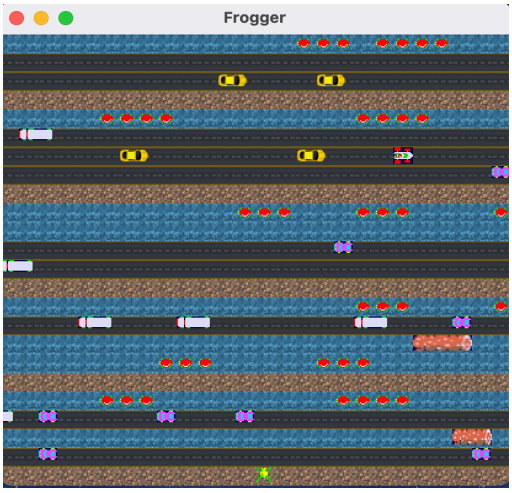
Nous avons aussi rajouté de la musique de fond avec des bruitages quand la grenouille se déplace ou passe par-dessus une pièce par exemple.



Jeu en mode normal



Jeu à deux personnes



Jeu à l'infini