WALTER Ferdinand
MARIYANAYAGAM Mickaël
L2 Informatique

RAPPORT Projet FROGGER _ 2019/2020

Pour ce projet, nous avons avancé jusqu'à la partie 4, au timer et une réflexion sur l'implémentation des cases spéciales.

L'ENVIRONNEMENT
Nous avons implémenté les méthodes suivantes Dans le package environment:

////Car.java
isInside(), getLength(), getFrontPos(), getRearPos() sont des méthodes de recherches

////Lane.java
qui résulte de sa classe mère environment

////Environment.java
isSafe() permet de vérifier en plus des lignes d'arrivée, les bordures de la fenêtre.

LE JEU INFINI
Dans cette partie, nous avons crée un nouveau package qu'on a appeler InfiniteFrogger.
Nous y avons implémentées 3 classes: FrogInf, LaneInf, EnvInf.

////FrogInf.java La classe FrogInf hérite de la classe Frog.
Le seul changement s'effectue dans le constructeur: la grenouille est initialisée à l'ordonnée 1 pour laisser de la marge.

////LaneInfinite.java
La classe LaneInf hérite de la classe Lane. On y a ajouté un attribut 'int id' qui est le numéro d'identification de la lane, et deux booléens 'isEmpty' et 'saved'.
La principale difficulté était de faire défiler les lanes au fur et à mesure que la grenouille avance sur l'écran d'affichage.
L'objectif est qu'à chaque action vers le haut du joueur, on crée une lane qu'on ajoute en haut de l'écran ; vers le bas, on retrouve la précédente lane et on la remet dans l'écran. Chaque lane possède une ordonnée mais avant tout un numéro d'identification id.
'isEmpty' est présente pour indiquer si la lane est vide ou pas. Si c'est le cas, on la remplie de Car ou on change la position en ordonnée des Cars présents.
'saved' est présente pour la méthode afficher(), qui affiche la lane selon si elle appartient à l'ArrayList lanes (des lanes visibles par le joueur).

Pour composer la méthode update(), nous l'avons séparer en plusieurs fonctions : des fonctions de recherche (getId(), getOrd(), setOrd()) et des fonctions qui permets de vérifient (saved(), loaded(), isSafe()...)

////EnvInf.java

La classe EnvInf hérite de la classe Environment. Nous y avons notamment ajouté aux attributs une ArrayList lanesSafe en plus, afin de sauvegarder les lanes qui disparaissent de l'écran et pour pouvoir les remettre si nécessaire. S'ajoute également aux attributs la variable double 'trottoire' qui donne la probabilité de tomber sur une route vide. On retrouve leur identifiant grâce à la méthode getLaneId(), pour les ranger soit dans lanes (affichage à l'écran) soit dans lanesSafe.

D'autres méthodes ont été réaliser pour effectuer des opérations au cœur des ArrayLists : des fonctions de recherches (findSavedLane(), getSavedLanes()), et des fonctions de permutation (addNewLane(), saveLane(), restoreLane(), cleanSavedLanes(), shiftOrdLane() et moveLane()).
A noter que nous avions des difficultés à réaliser shiftOrdLane(), pas que le code soit compliqué mais la compréhension de comment fonctionne le déplacement des lanes était lente. Cette méthode permet de décrémenter les lanes présentes pour laisser la place à une autre.
Un détail que nous avions vu en cours de route : les Cars ne bougeaient pas. En outre, nous n'avions considéré que le mouvement des Cars de gauche à droite et réciproquement. Nous avons pu corriger ce problème.

ELEMENTS COMPLEMENTAIRES _ TIMER

Nous avons abordé un des éléments complémentaires, le timer.
Pour l'implanter, nous avons utilisé les classes Instant et Duration.

Avec l'utilisation de Instant.now() appelée deux fois (timerStart et timerStop), qui récupère l'heure à laquelle la méthode est appelée, on utilise Duration.between() pour calculer la durée écoulée entre timerStart et timerStop et ainsi affichée la durée de la partie. Pour l'affichage, on utilise la méthode toStringDuration() qui prend en paramètre la durée. Le tout est converti en heure, minute, seconde.

l'affichage du score et de la durée de la partie sont espacée et centrée.

ELEMENTS COMPLEMENTAIRES _ CASES SPECIALES

A defaut de temps, nous n'avons pas pu mettre en pratique cette partie mais nous avons une esquisse sur la manière d'implémenter les cases spéciales.
L'objectif serait de créer des obstacles en faisant des classes héritant de Car. Ces classes Trapped, Ice, Walls et Bonus auraient leur propre méthode move(). On rajouterait aussi à la classe LaneInf une fonction qui utilise un switch case qui donnerait l'action à effectuer en fonction du type d'obstacle. La fonction isSafe se verrait ajouter des conditions en plus de la présence de voiture sur la case.