Introdution à la programmation Objet
Mini projet 2019

Conditions

Le projet doit être réalisé en binôme, au sein d'un même groupe de TP.

Vos codes devront être envoyés à vos chargés de TP pour le **30 novembre au plus tard**, accompagnés d'un bref rapport (d'une demie page à 2 pages) récapitulant brièvement le travail effectué, les points de difficulté rencontrés, les solutions envisagées et/ou réalisées. Des soutenances auront lieu sur vos créneaux de TP, les 5 et 6 décembre. Vous serez principalement notés sur votre réalisation, mais aussi sur le travail de groupe, l'organisation, l'auto-analyse et la présentation.

La partie 1 est relativement simple et devrait être terminée lors de la première séance encadrée. La partie 2 est plus conséquente, il est attendu qu'elle soit fonctionnelle pour la soutenance. La partie 3 ne sera abordée qu'une fois la partie 2 terminée. Vous traiterez ensuite un sujet au choix parmi ceux présentés en partie 4. Divers bonus vous sont présentés, **si vous avez terminé les 4 parties demandées** vous pouvez traiter ceux qui vous plaisent. Tous ne sont pas de difficulté égale, prenez le temps de réfléchir à la conception avant de vous lancer. N'oubliez pas de préciser ce que vous aurez traité ou abordé dans votre rapport et lors de votre soutenance.

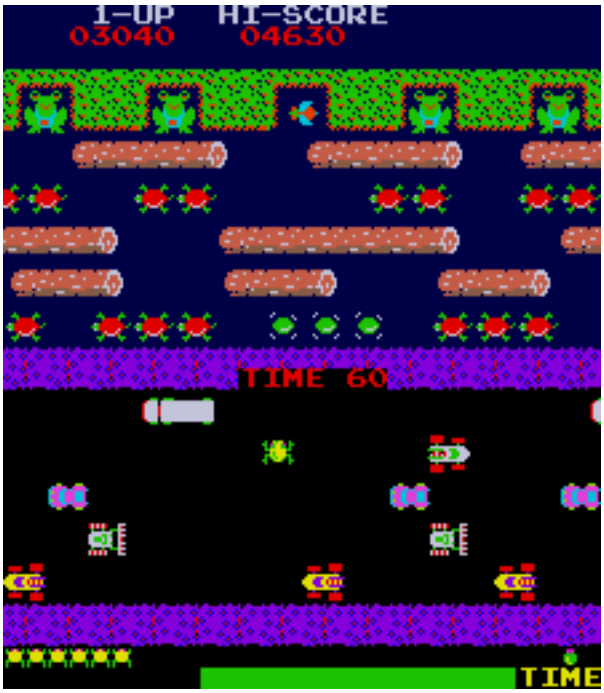

FIGURE 1 – Frogger, 1981 par Konami

Le sujet : Frogger

Frogger est un jeu d'arcade des années 80, ayant connu diverses adaptations depuis. Le but est de faire traverser une rue à une grenouille en évitant les voitures qui y circulent.

Ce projet vous propose de coder un petit jeu basique similaire. Le but sera d'amener une "grenouille" du bas de la fenêtre de jeu au haut de celle-ci. L'écran se composera d'une suite de lignes. La première ligne est la ligne de départ, et ne contient rien d'autre que la grenouille (au début de la partie, et quand elle y revient). La dernière ligne est également vide, et l'atteindre fait gagner la partie. Entre les deux, chaque ligne représente une voie de la route, contenant des voitures de taille 1 à 3 (blocs contigus) se déplaçant de gauche à droite ou de droite à gauche, et toutes les voitures d'une même voie se déplacent à la même vitesse. De nouvelles voitures apparaissent à des instants aléatoires au début de la ligne. Le joueur dirige la grenouille avec les 4 flèches du clavier. Une collision avec une voiture entraine une défaite.

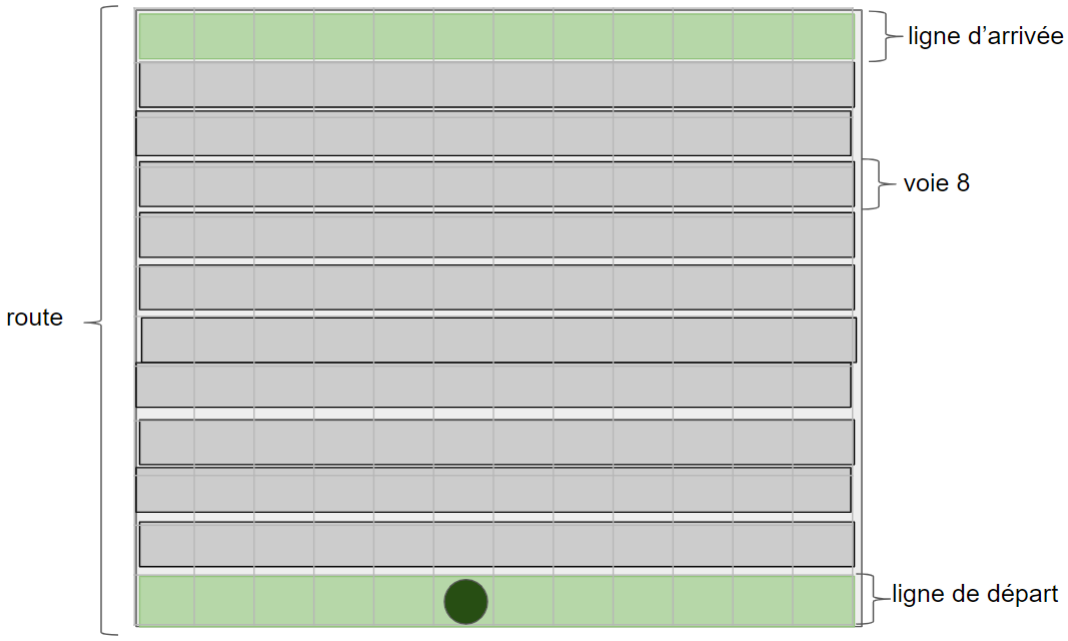

FIGURE 2 – Composition de l'écran de jeu

Vous pouvez voir et tester ce qui est attendu en lançant `FroggerExample.jar`, disponible sur la page du cours.

1 La grenouille

Dans un premier temps, vous allez devoir coder la classe `Frog`, permettant de gérer la grenouille, avec un environnement déjà fourni.

Téléchargez le squelette fourni (`squelPartie1.zip`, à extraire) ainsi que la bibliothèque pour l'environnement (`givenEnvironment.jar`, qui ne doit pas être extrait). Créez un projet, incluez-y le squelette fourni et liez la bibliothèque (copier-coller puis add to build path).

Le squelette est organisé en plusieurs packages :

— `frog` : le package devra contenir la ou les classes gérant la grenouille. Il s'y trouve la classe `Frog` que vous devez compléter.

— `graphicalElements` : contient la gestion de l'interface graphique.
L'interface `IFroggerGraphics` vous donnera les méthodes disponibles pour l'utilisateur. La classe `Element` permet de donner une case et une couleur, utile pour demander d'afficher un carré de cette couleur à des coordonnées données via les méthodes données dans `IFroggerGraphics`. Vous n'avez pas besoin de lire `FroggerGraphic`, la classe concrète qui gère l'affichage basique proposé, et vous l'utiliserez au travers de son interface.

— `gameCommons` : contient les entités permettant de gérer la partie de manière générale. Les interfaces `IFrog` et `IEnvironment` décrivent les actions qui doivent être disponibles pour le contrôle de la grenouille et de l'environnement respectivement. La classe `Game` gère une partie de jeu, vous devrez la compléter ; elle ne dépend que des interfaces pour l'environnement et la grenouille et ne doit pas dépendre directement des classes concrètes que vous implémentez. La classe `Main` permet de lancer le jeu, vous n'avez normalement pas besoin d'y toucher pour les 3 premières parties, si ce n'est pour modifier les classes concrètes de grenouille et d'environnement utilisées ou les paramètres (taille de l'écran, densité,...) du jeu.

— `util` : contient les définition de types `Case` et `Direction`, utiles à travers tout le projet.

Dans cette première partie, vous allez devoir coder `Frog`, ainsi que compléter certaines méthodes de `Game` (les tests de victoire et défaite). Vous devez respecter l'interface donnée, et être compatible avec les éléments fournis. Les interfaces (`IFrog`, `IEnvironment`) ainsi que le package `graphicalElements` **ne doivent pas être modifiés** !

A l'issue de cette étape, vous devez obtenir un jeu fonctionnel, similaire à celui donné en exemple (`FroggerExample.jar`).

Une entité de type *enum*, `Direction` vous est donné : il s'agit d'un type énuméré. Ici, un objet de type `Direction` n'a que 4 valeurs possibles : *up, down, right et left*, désignées par `Direction.up`, `Direction.down`, `Direction.right` et `Direction.left`. C'est un type, des objets de ce type peuvent donc être passés en argument, mis dans des variables ou renvoyés par des méthodes.
Il est également possible de les comparer : `if (myDirection == Direction.up)`.

Sauvegardez une version stable de votre projet une fois cette étape finie.

2 L'environnement : Suite de routes

Le but de cette partie est de recoder votre propre environnement, composé de plusieurs classes, idéalement dans son propre package, sans modifier le reste du code. Un squelette vous est proposé : `squelPartie2.zip`.

Commencez par remplacer, dans la classe `Main`, cette ligne :

```
//Création et liaison de l'environnement
IEnvironment env = new GivenEnvironment(game);
```

par celle-ci :

```
//Création et liaison de l'environnement
IEnvironment env = new Environment(game);
```

Le schéma proposé se compose des éléments suivants :

— un environnement global, composé d'une route de plusieurs voies
— une voie a un sens (de gauche à droite ou de droite à gauche), une densité (qui détermine la probabilité qu'une voiture entre sur la voie) et une vitesse. Elle contient un ensemble de voitures.
— une voiture a une taille, une position, un sens et une vitesse (ces derniers correspondants à ceux de la voie sur laquelle elle est).

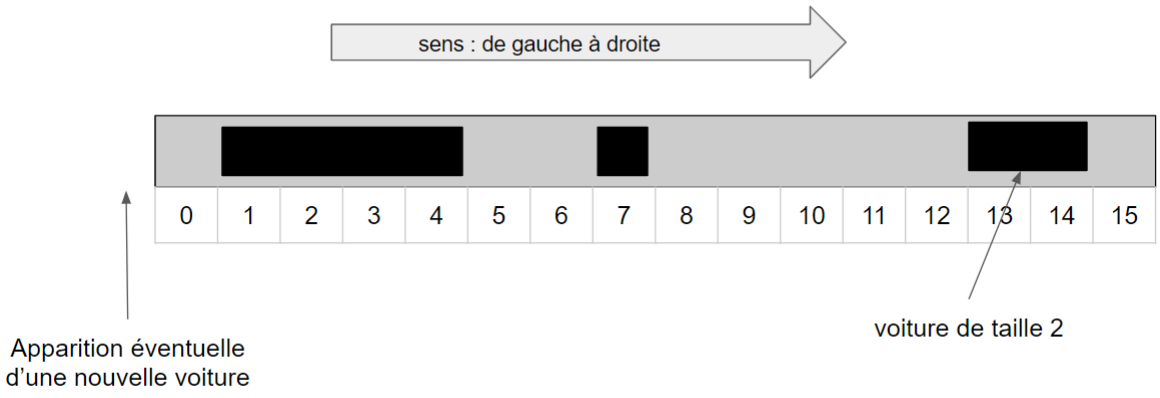

FIGURE 3 – Composition d'une voie

Le squelette fourni contient ces éléments, à compléter, tout en respectant l'interface `IEnvironment` qui ne doit pas être modifiée. Il n'est pas nécessaire de modifier `Game`, `Main` ou la partie `Frog` codée précédemment.

Vous avez plusieurs méthodes à ajouter, et pouvez en créer autant que nécessaire à l'exécution et à la lisibilité de votre programme. Vous pouvez parfaitement ajouter, supprimer ou modifier des attributs.

La méthode `mayAddCar()` de `Lane` (voie) vous est fournie : elle permet de tirer aléatoirement, d'après la densité de la voie, l'apparition d'une voiture "juste avant" la première case de la voie. Vous pouvez l'utiliser pour faire apparaître des voitures lors de l'actualisation du modèle (après le déplacement des voitures déjà en place).

Sauvegardez une version stable de votre projet une fois cette étape finie. Vous devrez présenter une version fonctionnelle de cette partie en soutenance, et présenter vos choix et difficultés.

3 Jeu infini

Vous allez maintenant écrire des variantes des classes précédentes pour jouer à un jeu légèrement différent : il n'y a maintenant plus de ligne d'arrivée, le monde est infini et chaque déplacement vers le haut fait apparaitre une nouvelle voie à l'écran. Pour que la fenêtre reste de taille fixe, votre grenouille sera toujours affichée à la même hauteur et les anciennes lignes (celles trop loin sous la grenouille) ne seront plus affichées. Un exemple de résultat vous est donné sur la page du cours, en tant que jar exécutable : `InfiniteFroggerExample.jar`.

Après avoir réfléchi à comment l'implémenter, créez de nouvelles classes FrogInf et EnvInf pour gérer ce nouveau mode de jeu. Vous pouvez modifier d'autres classes. Vous êtes ici autorisés à modifier les interfaces, à bon escient et en assurant la compatibilité avec les modules précédemment développés ! Il n'est pas nécessaire de modifier le package `graphicalElements`.

On pourra de plus ajouter un score, affiché à la fin de partie, correspondant à la plus haute ligne jamais atteinte.

Sauvegardez une version stable de votre projet une fois cette étape finie, vous devrez la présenter en soutenance.

Vérifiez que vous pouvez inclure les différents modules développés au même projet et lancer les différentes versions en modifiant minimalement le `Main`.

4 Éléments complémentaires

Traitez un élément au choix (ou plus) parmi les 3 suivants. Si les développements sont incomplets, présentez brièvement vos idées, démarches et ce qui vous a empêché d'aller au bout dans votre rapport, et en commentaire des codes incomplets ou buggués. Vous serez évalués sur votre capacité à l'auto-analyse.

4.1 Timer

Difficulté : 1/3
Quantité de code : 1/3

Affichez le temps de jeu à la fin de la partie, en particulier en cas de victoire.

4.2 Cases spéciales

Difficulté : 1/3 ou 2/3
Quantité de code : 2/3

Ajoutez des cases fixes spéciales à votre environnement :
— des pièges qui font perdre la partie si la grenouille avance dessus
— des terrains glissants propulsant la grenouille une case plus loin dans la même direction
— des murs (de tunnel) qui empêchent la grenouille de passer mais pas les voitures
— des cases à traverser pour obtenir des bonus de score

Vous pouvez traiter une ou plusieurs de ces catégories de cases spéciales.

N'hésitez pas à définir une interface commune aux différents types de cases spéciales. Choisissez comment initialiser votre environnement pour en ajouter. Pour l'affichage, vous pouvez dans un premier temps vous contenter de mettre des couleurs différentes sur la case qui les contient.

4.3 Lignes d'eau

Difficulté : 2/3
Quantité de code : 3/3

Le jeu Frogger se compose initialement d'une route à traverser, puis d'une rivière, à traverser en montant sur des rondins flottants. Lorsque ceux-ci se déplacent, ils déplacent la grenouille avec eux. Si la grenouille se déplace dans l'eau ou est attirée au-delà du bord de l'écran, la partie est perdue.

Pour implémenter le déplacement de la grenouille provoqué par celui des rondins, vous aurez besoin d'accéder à la grenouille depuis l'environnement. Pour l'affichage, contentez-vous dans un premier temps d'utiliser une couleur différente pour les rondins.

5 Bonus

Vous pouvez traiter les points suivants de votre choix pour des points bonus. Votre note ne pourra cependant par excéder 20, et ne pourra pas être excellente si les parties obligatoires ne sont pas fonctionnelles.

5.1 Deux joueurs

Difficulté : 2/3
Quantité de code : 2/3
Autonomie et nouveaux concepts : 1/3

Permettre de jouer à deux, sur un clavier (un joueur se déplace avec les flèches, l'autre avec QZSD). Il faudra alors instancier deux objets différents de `Frog`, et modifier la classe `FroggerGraphic` pour écouter d'autres touches, tout en sachant lesquelles correspondent à quelle grenouille.

5.2 Environnement 2D libres

Difficulté : 2/3
Quantité de code 3/3
Autonomie et nouveaux concepts : 1/3 à 2/3

On cherche à généraliser le principe du jeu à d'autres types d'environnements. Le principe reste de traverser l'écran en évitant des obstacles mouvants. Vous pouvez choisir divers types de déplacements pour vos objets, pas seulement de droite à gauche ou de gauche à droite. Et pourquoi pas des nuées d'oiseaux, en réutilisant les concepts et/ou le code du TP BOID ? Vous pouvez garder un affichage en grille simple comme dans les premières parties, n'hésitez pas à en changer la résolution.

5.3 Meilleur rendu graphique

Difficulté : 3/3
Quantité de code 2/3
Autonomie et nouveaux concepts : 3/3

Le rendu graphique est très basique, vous pouvez l'améliorer, minimalement ou énormément. Assurez-vous cependant avant cela d'avoir des parties 1, 2 et 3 fonctionnelles ! Attention, le débuggage peut être potentiellement laborieux et plusieurs notions sortent du cadre du cours.

5.4 Vos idées !

N'hésitez pas à proposer vos propres améliorations. N'oubliez pas de les présenter brièvement dans votre rapport.