IFT1025 - Exercice 6

RobotFindsKitten - Super Dungeon Master 3000^{Ultra}_{Turbo} Deluxe Edition

Réutilisation, Interfaces Graphiques, Architecture Modèle-Vue-Contrôleur

1 Introduction

Vous avez réalisé un joli petit prototype pour RobotFindsKitten... Le temps est maintenant venu de transformer votre prototype en ligne de commande en un vrai jeu : avec des vraies images, des vraies couleurs et des vrais chatons (ou pas).

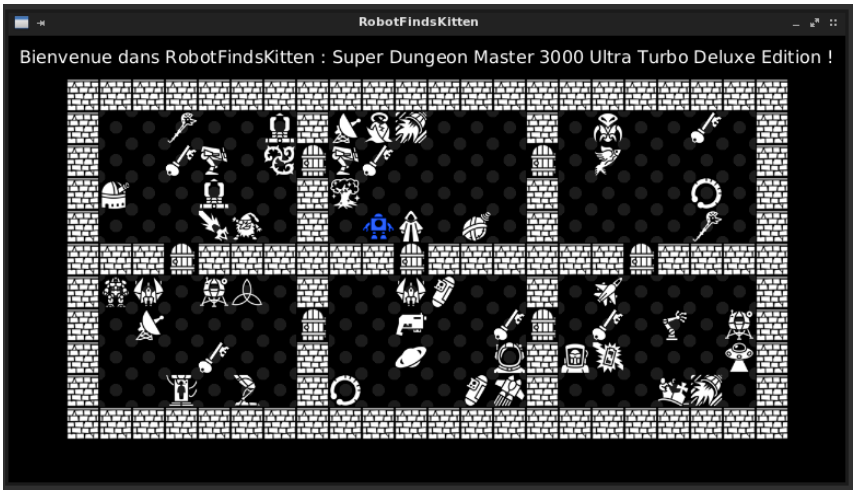

Figure 1: Version graphique du jeu RobotFindsKitten - Super Dungeon Master...

2 Spécification

2.1 Base de code

La seule base de code dont vous disposez est **votre propre code pour le TP1**. Vous avez fait le travail en équipes de deux, vous devrez faire ce travail-ci individuellement à partir de ce que vous avez remis en équipe.

Si pour une raison quelconque vous n'avez pas remis de code au TP1 ou si vous croyez que le code que vous avez remis ne vous permettra pas de réaliser correctement cet exercice, contactez le professeur rapidement pour voir ce qu'on peut faire.

2.2 Disposition des composantes graphiques

Figure 2: Décomposition en composantes graphiques

La *Figure 2* montre les composantes graphiques à utiliser pour afficher le jeu.

Légende :

- Rouge : Un `VBox`, la racine de l'arbre qui représente la scène
- Jaune : Un simple élément `Text` qui sera utilisé pour afficher les différents messages pendant le jeu
- Bleu : Un `GridPane` qui contient des `ImageView` de dimensions 30 pixels par 30 pixels et qui représente les cases de la grille de jeu
- Vert : Un autre `Text`, qui servira à afficher le statut du `Robot` en tout temps (voir la spécification du TP1 là-dessus)

2.3 Exécution du programme

Le déroulement du jeu en tant que tel est exactement comme dans le TP1 : la grille est générée aléatoirement, on commence à un endroit au hasard sur la grille et on se promène en ramassant des clés et un téléporteur, jusqu'à ce que RobotTrouveChaton.

Lorsque la partie se termine (quand RobotTrouveChaton), la `Scene` change pour n'afficher qu'une image attendrissante et le message de victoire, tel que montré à la *Figure 3*.

Lorsque l'écran de victoire est affiché, on peut appuyer sur la touche `Espace` pour démarrer une nouvelle partie.

À tout moment pendant le jeu, on doit pouvoir appuyer sur :

- `F5` pour activer/désactiver le mode FullScreen
- `Escape` pour fermer le programme

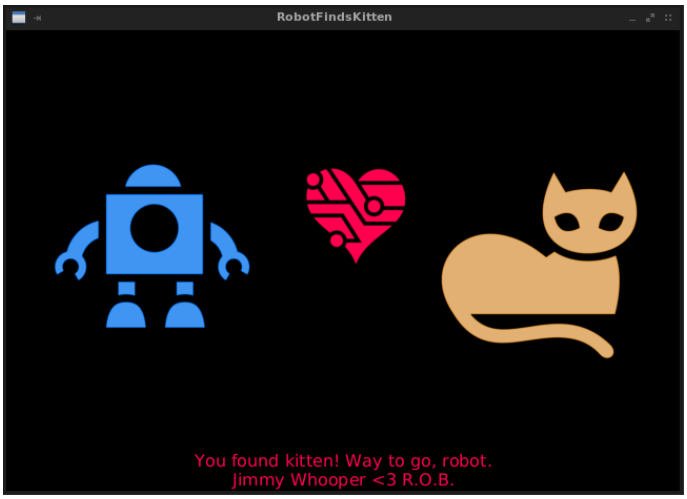

Figure 3: Screenshot attendrissante de ce qui est affiché lorsque RobotTrouveChaton

2.4 Architecture *Modèle-Vue-Contrôleur*

Vous devrez utiliser l'architecture *Modèle-Vue-Contrôleur* montrée en classe pour structurer votre code.

Quelques conseils :

- Puisque le point d'entrée de l'application est une *Vue*, votre *Vue* sera chargée d'instancier l'objet qui sera votre *Contrôleur*
- Le *Contrôleur* sera en charge d'instancier les classes qui font partie des *Modèles*
- Le *Contrôleur* ne devrait contenir **aucune logique mis à part celle qui fait le lien entre l'affichage et le modèle**
- Les *Modèles* de l'application incluent :
  - Le `Robot`
  - La `Grille`
  - Les différents types de `Case`s
  - La logique du jeu : le code qui permet au `Robot` de se déplacer sur la `Grille` (conceptuelle, pas celle affichée), incluant la validation des mouvement (valides ou non), le fait de pouvoir ramasser des clés, de marcher sur un NonKittenItem, de passer à travers une porte, etc.
- Rappelez-vous que le découpage original du code *ne respecte pas* cette architecture. Par exemple, la `Grille` avait une fonction `afficher()`, ce qui ne devra pas être le cas dans ce travail. Vous devrez refactoriser et réécrire des bouts de code pour que l'architecture soit correcte.
- Vous pouvez modifier l'attribut `private char representation` dans `Case` pour en faire un attribut de type `String` qui contiendra le **nom de l'image** à utiliser pour représenter cette case (ex.: `"door.png"`)

3 Images fournies

L'archive `images.zip` est disponible sur StudiUM et contient des images que vous pouvez utiliser pour remplir la grille. Les images sont dérivées des icônes trouvables sur le site http://game-icons.net/.

Vous avez les images pour les différents types de cases (vide/porte/robot/...) + 82 images numérotées de `1.png` à `82.png` pour les *NonKittenItems*.

Les images des *NonKittenItems* doivent être choisies aléatoirement lors de la génération de la grille et ne doivent pas changer en cours de partie.

Notez que vous pouvez utiliser vos propres images si vous le voulez.

4 Évaluation

L'exercice est à faire individuellement et compte pour 5% de la session.

La remise est le 26 mars à 23h55, il y a une pénalité de 25% par jour de retard.

Notez bien la pondération : vous aurez plus de points pour un TP qui fonctionne correctement qu'un TP qui ne marche pas comme demandé mais dont le code est joli et bien commenté (bien que pour avoir 100%, vous devrez faire les deux)

Remettez l'intégralité de votre projet, incluant les images que vous avez utilisées

4.1 Barème

- **70%** ~ Le jeu marche comme demandé
  - Respect de la spécification : toutes les fonctionnalités demandées sont présentes
  - Pas de bugs, ni problèmes d'affichage graphique, ni erreurs de logique
- **30%** ~ Qualité du code
  - Découpage des classes en suivant l'architecture *Modèle-Vue-Contrôleur*
  - Code bien commenté, bien indenté
  - Performance du code raisonnable : pas d'objets ou de composantes graphiques créées et/ou modifiées inutilement, pas d'images allouées un nombre déraisonnalb de fois (par exemple : vous n'avez pas besoin d'allouer plus qu'une seule instance de votre `Image` pour les cases vides, vous pouvez réutiliser la même dans tous les `ImageView` qui affichent une case vide)

5 Perception du monde

L'exercice a l'air relativement gros et c'est voulu : vous devrez découper la grosse tâche en sous-tâches plus simples par vous-mêmes.

Une fois que vous aurez pris le temps de réfléchir à comment organiser votre code, vous devriez vous apercevoir que l'exercice est plus simple que ce qu'il en a l'air à première vue.