Travail Pratique 3 - Fish Hunt

IFT1025 - Programmation 2

Concepts appliqués – Programmation orientée objet, interfaces graphiques, animation, développement d'application complète

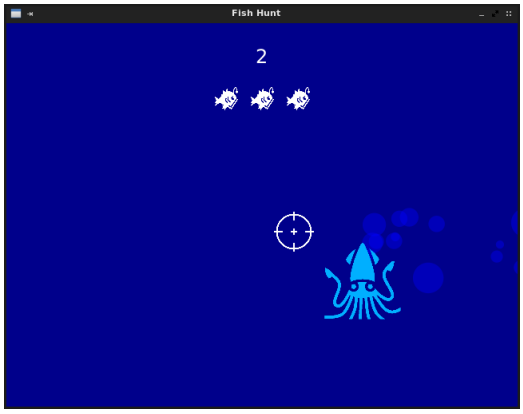
---

Figure 1: Logo du jeu

1 Contexte

Le TP final consiste à programmer un autre jeu en interface graphique, toujours avec la librairie *JavaFX*.

1.1 Description du jeu



Vous incarnez un requin qui chasse des poissons pour son souper.

Étant un requin gourmand, vous ne pouvez pas vous permettre de laisser trop de poissons passer... Au bout de 3 poissons ratés, la partie est perdue.

Le jeu se contrôle à la souris et continue à l'infini, ou jusqu'à ce que 3 poissons aient été ratés (selon lequel arrive en premier).

1.2 Déroulement du jeu

Au début du jeu, le niveau est affiché pendant 3 secondes : **Level 1**.

Une fois les 3 secondes écoulées, le texte disparaît et le premier poisson apparaît à l'écran.

Afin de capturer les poissons, il faut cliquer sur le canvas avec la souris pour leur lancer des balles. Chaque poisson capturé fait monter le score de +1. Cependant, les poissons se déplacent : si un poisson réussit à sortir de l'écran sans se faire capturer, il est considéré comme perdu. Au bout de 3 poissons ratés, le jeu se termine et la partie est perdue.

Le score est affiché à l'écran et le nombre de poissons ratés restants (initialement 3) est représenté par des images de poissons sous le score.

À tous les 5 poissons *capturés*, on augmente le niveau du jeu : les poissons arrêtent d'apparaître pendant 3 secondes et le jeu affiche le prochain numéro de niveau, ex.: **Level 2**.

1.3 Poissons

1.3.1 Poissons normaux

À toutes les 3 secondes, des poissons normaux apparaîssent soit depuis la gauche, soit depuis la droite de l'écran.

La gravité pour les poissons normaux (accélération en Y vers le bas) doit être de $100px/s^2$. La vitesse horizontale des poissons doit avoir une magnitude suivant la fonction :

$$vitesse(level) = 100 \times numeroNiveau^{1/3} + 200 \qquad px/s$$

La vitesse verticale initiale des poissons est tirée au hasard entre 100 et 200 px/s vers le haut de l'écran.

Les poissons qui commencent à droite vont vers la gauche, les poissons qui commencent à gauche vont vers la droite. Leur position y initiale est tirée au hasard entre $\frac{1}{5}$ et $\frac{4}{5}$ de la hauteur du jeu.

L'image des poissons doit être inversée au besoin pour les faire "regarder" dans la direction où ils se déplacent. Chaque nouveau poisson a également une couleur aléatoire. Pour manipuler les images fournies (attribuer une couleur aléatoire et inverser l'image), regardez les fonctions disponibles dans la classe `ImageHelpers` fournie sur StudiUM.

Utilisez les images `fish/00.png` à `fish/07.png` dans le dossier d'images fournies.

1.3.2 Poissons spéciaux[1]

À partir du niveau 2, un poisson spécial doit apparaître à toutes les 5 secondes en plus des poissons normaux. Un poisson spécial est soit un *Crabe*, soit une *Étoile de mer* (50% de probabilité pour chacun).

[1] Bien que techniquement pas des poissons, les crabes et les étoiles de mer sont inclus lorsqu'on parle de "poissons" dans l'énoncé.

Ces poissons ont une façon différente des autres de se déplacer :

Crabe : le crabe est représenté par l'image `crabe.png`. Les crabes vont à une vitesse de $1.3 \times vitesse(level)$ – voir la fonction définie plus haut – et vont en ligne droite (autrement dit, sans gravité, $acceleration_y = 0$).

Un Crabe avance en oscillant : il avance pendant 0.5s, puis recule pendant 0.25s, puis avance pendant 0.5s, ... jusqu'à avoir dépassé l'autre côté de l'écran.

Étoile de mer : l'étoile de mer est représentée par l'image `star.png`. Elle avance à la même vitesse que les poissons normaux, sans subir de gravité ($acceleration_y = 0$), mais oscille elle verticalement avec une amplitude de $50px/s$.

1.4 Cible & Balles

Pour aider à viser, l'image `cible.png` doit être affichée dans un format $50px \times 50px$ de façon à toujours être centrée sur le curseur.

Lorsqu'on fait un clic gauche avec la souris, une balle est lancée. Pour simuler une balle qui "s'éloigne" et qui part vers les poissons, la balle commence en étant un cercle noir de rayon $50px$ et son rayon **diminue** à une vitesse de $300px/s$. Au bout de $\frac{1}{6}sec$, le rayon devrait donc atteindre $0px$.

Lorsque le rayon est égal à $0px$, on considère que la balle est arrivée à destination : si un poisson se trouve sur ce point, il est considéré comme touché par la balle et est capturé. Il disparaît alors du jeu.

1.5 Bulles

Puisque le jeu se déroule dans l'océan, comme pour le TP2, le décor est le même : à toutes les 3 secondes, des bulles apparaissent dans l'arrière-plan et remontent à la surface.

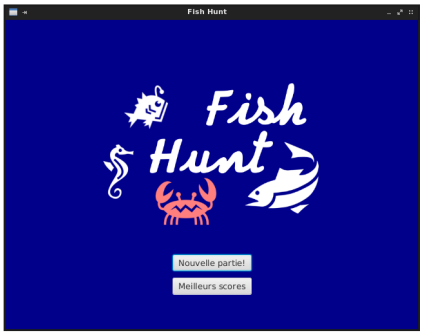
Rapportez-vous à l'énoncé du TP2 pour plus de détails sur le fonctionnement des bulles.

2 Interface

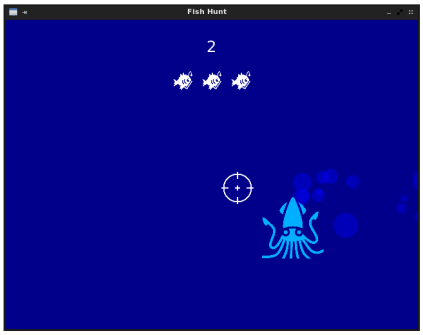
La fenêtre du programme doit avoir une largeur de $640px$ et une hauteur de $480px$.

- La fenêtre ne doit pas être *resizable* : la taille de la fenêtre est fixée au début et elle ne peut pas être redimensionnée avec la souris
- La fenêtre doit porter le titre "Fish Hunt"

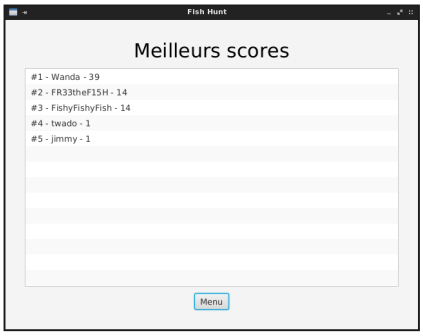
Afin d'avoir un programme plus complet que pour le TP2, ce programme est constitué de 3 scènes (voir **Fig. 2**).



Accueil



Fenêtre de Jeu



Meilleurs Scores

Figure 2: Vue d'ensemble des différentes scènes du jeu

2.1 Écran d'accueil

Lorsqu'on ouvre le programme, la scène affichée doit être l'écran d'acceuil. Cette scène montre :

1. Le logo du jeu (`logo.png`)
2. Un bouton *Nouvelle partie!*
3. Un bouton *Meilleurs Scores*

Le bouton *Nouvelle partie!* passe à la scène du jeu et démarre une nouvelle partie.

Le bouton *Meilleurs Scores* permet de passer à la scène des *Meilleurs Scores*.

2.2 Meilleurs Scores

Cette scène permet d'afficher le top 10 des meilleurs scores réalisés dans le jeu, avec le nom et le nombre de points des personnes qui ont marqués le plus de points pendant une partie.

Lorsqu'on joue au jeu pour la première fois, cette liste doit être vide. Lorsqu'on joue plus que 10 parties, on en garde seulement 10.

Cette scène est composée des éléments suivants :

- Un `Text` qui affiche le titre de la scène
- Un `ListView<String>` qui contient
- Un `Button` *Menu* qui permet de revenir à l'écran d'accueil

Vous pouvez ajouter des éléments au `ListView` avec quelque chose du genre de :

```
ArrayList<String> scores = ...;

ListView<String> list = new ListView<>();
list.getItems().setAll(scores);
```

2.3 Scène de Jeu

La scène de jeu contient seulement un canvas (la fenêtre de jeu), qui doit occuper toute la fenêtre.

En tout temps, on devrait voir le score actuel en nombre de poissons capturés, avec le nombre de poissons qu'il est encore possible de rater juste en dessous, sous la forme de petites images `fish/00.png`.

2.4 Entrer un nouveau *"Meilleur Score"*

Lorsque la partie se termine, on devrait afficher un *Game Over* pendant 3 secondes, puis passer à la fenêtre des *Meilleurs Scores*.

Dans le cas où la partie qui vient d'être jouée doit rentrer dans le Top 10 des meilleures parties, on devrait avoir une ligne supplémentaire affichée dans la scène :

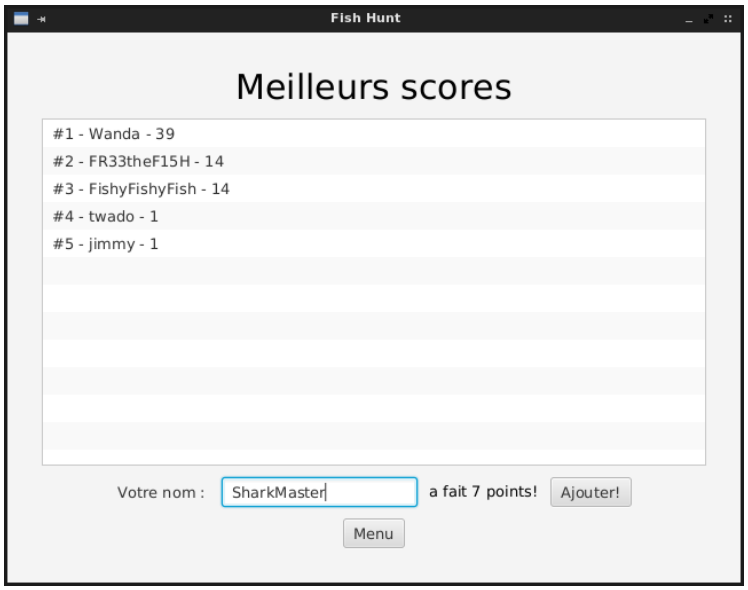


Figure 3:

Lorsqu'on clique sur le bouton *Ajouter!*, le résultat est enregistré.

Notez que **les meilleurs scores doivent être conservés même si le programme est fermé puis rouvert** : utilisez un fichier pour stocker les meilleurs scores entre les différentes exécutions du programme.

3 Debug

Afin de simplifier le débogguage du jeu, on devrait pouvoir :

- Appuyer sur la touche `H` pour faire monter le niveau de +1
- Appuyer sur la touche `J` pour faire monter le score de +1
- Appuyer sur la touche `K` pour faire monter le nombre de poissons restants de +1 (en vous assurant de ne pas dépasser le maximum de 3 poissons), autrement dit, pour donner une chance de plus de rater un poisson avant que la partie soit perdue
- Appuyer sur la touche `L` pour faire perdre la partie

4 Code & Design Orienté Objet

Ce sera à vous de choisir le découpage en classes optimal pour votre programme. Faites bon usage de l'orienté objet et de l'héritage lorsque nécessaire.

Vous **devez** utiliser une architecture du type *Modèle-Vue-Contrôleur* tel que vu en classe et vous serez évalués là-dessus.

Réfléchissez à ce qui constitue votre Modèle pour ce jeu, à comment définir une vue correctement et à comment faire usage du Contrôleur pour isoler complètement les classes de la Vue des classes du Modèle.

La classe principale du programme doit se nommer `FishHunt` et être dans le package par défaut (aka, pas de ligne `package ...` au début du fichier).

Vous pouvez créer des packages pour vos autres classes si vous jugez que c'est nécessaire, mais gardez en tête que le programme ne devrait pas être particulièrement gros et n'en a pas absolument besoin.

5 Note sur la mémoire utilisée

Si votre code crée des instances d'objets pour représenter des éléments qui peuvent sortir de l'écran (ex.: bulles, poissons, ...), assurez-vous ne de pas garder en mémoire des objets qui ne sont plus utilisés.

Autrement dit, assurez-vous de ne pas garder dans un tableau ou dans un ArrayList une référence à une plateforme qui ne sera plus jamais affichée de tout le reste du jeu.

6 Éléments fournis

Aucun code n'est fourni, mais vous pouvez vous inspirer des exemples de code donnés dans le chapitre sur les GUIs et (surtout) dans le chapitre sur l'infographie 2D.

Les images nécessaires au programme sont fournis sur StudiUM et sont basées sur différentes images du site https://game-icons.net/.

7 Bonus

7.1 Améliorer le jeu (jusqu'à 5%)

Ajoutez des fonctionnalités de votre choix au programme pour en faire un jeu plus intéressant. Soyez créatifs, rendez le jeu amusant, amusez-vous!

Le pourcent bonus sera donné sur l'originalité (autrement dit, si vous faites le strict minimum simplement pour avoir votre point bonus, ça ne sera pas compté) et sur la complexité de ce que vous avez fait. Quelque chose de relativement simple à ajouter ne vaudra pas 5%.

Si vous faites ce bonus, ajoutez un fichier `BONUS.txt` à votre remise contenant la liste de ce que vous avez ajouté et expliquant ce que vous avez dû faire pour y arriver.

7.2 Version multijoueurs (10%)

Faites une version multijoueur du jeu, **en ligne**, en plus du mode solo. Ajoutez un bouton supplémentaire à l'écran d'accueil.

Le jeu doit se jouer à N personnes et avoir une mécanique de jeu différente du mode solo. Inventez de quoi.

Si vous choisissez de faire ce bonus, ajoutez un fichier `MULTIJOUEUR.txt` expliquant ce que vous avez fait.

Si vous souhaitez déployer un serveur sur internet une fois que votre code est terminé (après la date de la remise), contactez `nicolas.hurtubise@umontreal.ca` pour arranger de quoi.

7.3 Version mobile multijoueurs (5%)

Portez le jeu sur Android, **avec la version multijoueurs**.

Vous devez absolument avoir fait le bonus précédent pour faire ce bonus-ci.

Si vous choisissez de faire ce bonus, ajoutez un `.apk` à votre remise en plus de remettre un `.zip` contenant le code source du projet.

8 Barème

- **50%** ~ Exécution (programme conforme à ce qui est demandé)
- **30%** ~ Découpage en classes et utilisation judicieuse de l'héritage
- **10%** ~ Découpage du programme en suivant l'architecture MVC
- **10%** ~ Qualité du code
  - Code bien commenté (*JavaDoc*, mais pas besoin d'en mettre pour les accesseurs/mutateurs)
  - Code bien indenté, bon découpage en fonctions, encapsulation, noms de variables bien choisis, performance du code raisonnable, camelCase, pas trop compact, pas trop espacé... bref, du code clair et lisible en suivant les principes que vous connaissez
  - Limitez vos lignes de code à 120 caractères de large

9 Indications supplémentaires

- La date de remise est le *2 mai 2020 à 23h59*. Un travail remis le lendemain avant 23h59 aura une pénalité de *-33%*. Un travail remis passé le lendemain se verra attribué la note de zéro.
- Vous **devez** faire le travail par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code, dans le fichier principal (`FishHunt.java`).
- Une seule personne par équipe doit remettre le travail sur StudiUM
- Un travail fait seul engendrera une pénalité. Les équipes de plus de deux ne sont pas acceptées.
- Remettez tous les fichiers nécessaires à l'exécution de votre code dans une archive `.zip` (pas de `.rar` autorisés)
- De plus :
  - La performance de votre code doit être raisonnable
  - Chaque fonction devrait être documentée en suivant le standard `JavaDoc`
  - Il devrait y avoir des lignes blanches pour que le code ne soit pas trop dense (utilisez votre bon sens pour arriver à un code facile à lire)
  - Les identificateurs doivent être bien choisis pour être compréhensibles (évitez les noms à une lettre, à l'exception de i, j, ... pour les variables d'itérations des boucles for)
  - Vous devez respecter le standard de code pour ce projet, soit les noms de variables et de méthodes en camelCase