COLLÈGE DE
BOIS-DE-BOULOGNE

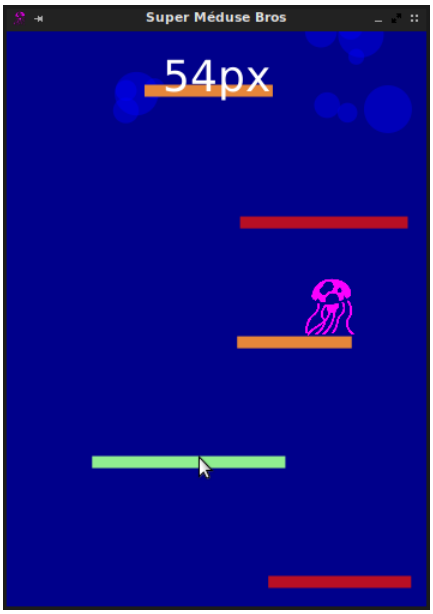
Travail Pratique 2 – Super Méduse Bros.

SIM – A21 – Développement de programmes dans un environnement graphique

Nicolas Hurtubise

1 Contexte

Ce deuxième TP consiste à programmer un jeu en interface graphique avec la librairie *JavaFX*.



Vous incarnez une méduse qui tente de remonter le plus haut possible dans l'océan en sautant de plate-forme en plate-forme.

La caméra monte graduellement automatiquement. L'objectif du jeu est de ne pas tomber au fond de l'océan.

C'est un jeu qui n'a pas de fin : il n'y a pas d'autre but que de monter le plus haut possible, jusqu'à ce que l'océan vous absorbe.

La méduse peut se déplacer de gauche à droite avec les flèches du clavier (gauche/droite) et peut sauter avec la barre espace ou la flèche du haut.

2 Méduse

La méduse est modélisée par un carré de taille $50px$ par $50px$ et est animée avec les images `meduse1.png` à `meduse6.png`. L'animation doit avoir un *framerate* de 8 images par seconde (*à lire là-dessus : notes de cours* ***JavaFX - Animations partie 1***).

Ces images montrent la méduse qui regarde vers la droite. Si la méduse regarde à gauche, on devrait plutôt utiliser les images `meduse1-g.png` à `meduse6-g.png` dans l'animation.

La "gravité" du jeu est de $1200px/s^2$ vers le bas.

Chaque saut avec la barre espace/la flèche du haut donne instantanément une vitesse de $600px/s$ vers le haut.

Lorsqu'on appuie sur les flèches gauche et droite, cela a pour effet de donner instantanément à la méduse une *accélération en x* de $1200px/s^2$ vers la gauche ou vers la droite (respectivement). Si aucune des deux touches n'est appuyée, on doit donner une accélération **en sens inverse** de pour faire en sorte que la méduse ralentisse automatiquement (*à lire là-dessus : notes de cours* ***JavaFX - Animations partie 4***).

La méduse ne peut pas sortir de l'écran par les côtés : lorsque la méduse touche la gauche ou la droite de l'écran, elle **rebondit** dans l'autre direction.

3 Plates-formes

Les plates-formes sont des rectangles de différentes couleurs, toujours espacés de $100px$ verticalement.

Leur hauteur est toujours de $10px$, leur largeur est choisie aléatoirement entre $80px$ et $175px$.

Leur position horizontale (en x) est choisie au hasard de façon à rester dans l'écran.

Pour simplifier les collisions, la méduse est affichée avec des images mais est en réalité modélisée par un carré de taille $50px$ par $50px$.

Il y a **quelques sortes de plates-formes**. Les plates-formes réagissent différemment à la collision avec la méduse. Chaque nouvelle plate-forme ajoutée au niveau est choisie au hasard.

3.1 Plate-forme simple (orange)

La plate-forme simple est une plate-forme qu'on peut traverser depuis le bas mais qui sert de plancher lorsqu'on tombe dessus.

Utilisez la couleur `Color.rgb(230, 134, 58)` pour afficher ces plates-formes.

3.2 Plate-forme rebondissante (vert pâle)

La plate-forme rebondissante a pour effet de faire faire un rebond à la méduse lorsqu'elle tombe dessus.

Lorsque la méduse entre en collision avec la plate-forme (donc en tombant dessus depuis le haut), sa vitesse verticale v_y est inversée comme dans tous les rebonds, mais elle est également amplifiée par un facteur de $\times 1.5$.

Pour assurer un rebond minimalement intéressant, la vitesse après rebond est forcée à être *au moins* de 100px/s vers le haut.

Utilisez la couleur `Color.LIGHTGREEN` pour afficher ces plates-formes.

3.3 Plate-forme mouvante (rouge)

Les plates-formes mouvantes sont comme des plates-formes normales, mais qui *se déplacent de gauche à droite*, un peu comme les `FloconsOscillants` de l'exercice fait en classe (*à lire là-dessus : notes de cours `JavaFX - Animations partie 2`*).

Utilisez la couleur `Color.rgb(184, 15, 36)` pour afficher ces plates-formes.

3.4 Plate-forme éphémère (noir)

Lorsque la méduse se pose sur une plate-forme éphémère, elle peut se poser dessus comme sur une plate-forme normale. Tant qu'elle reste dessus, rien ne se passe. Par contre, la seconde où la méduse saute de la plate-forme, la plate-forme *tombe en bas de l'écran* à une vitesse de 200px/s.

Utilisez la couleur `Color.BLACK` pour afficher ces plates-formes.

➢ Pour que le jeu reste amusant, on devrait avoir une **majorité de plates-formes simples**, avec quelques plates-formes spéciales de temps en temps... Vous pourriez par exemple utiliser :

- 50% de plates-formes **simples**
- 20 de plates-formes **mouvantes**
- 15% de plates-formes **rebondissantes**
- 15% de plates-formes **éphémères**

4 Caméra

La caméra monte automatiquement vers le haut (*à lire là-dessus : notes de cours `JavaFX - Animations partie 4`*). Sa vitesse commence à $0px/s$ et accélère graduellement vers le haut à une vitesse de $2px/s^2$. Plus le jeu avance, plus la caméra monte vite d'elle-même, ce qui ajoute de la difficulté à mesure qu'on monte.

Dès que la méduse dépasse 75% de la hauteur de l'écran (à partir du bas) en sautant, cela fait **monter la caméra automatiquement**. La position de la caméra doit donc être ajustée si la méduse monte trop, de façon à ce que le haut de la méduse ne soit jamais affiché plus haut que ça.

Si le *haut du corps* de la méduse se retrouve plus bas que le bas de l'écran (donc si on ne la voit plus du tout), la méduse est *absorbée par l'océan* : la partie est perdue et on affiche l'écran des meilleurs scores.

5 Bulles

Pour donner un peu de vie à l'océan, l'arrière-plan va afficher des bulles qui remontent à la surface de temps en temps.

Des bulles apparaissent dans l'arrière-plan à toutes les 3 secondes :

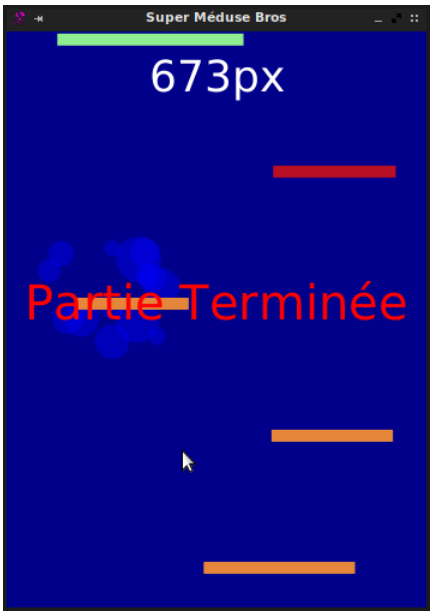
- Les bulles sont des cercles d'un diamètre aléatoire entre $10px$ et $40px$
- Elles sont affichées en bleu avec une transparence de 40%, ce qui peut être obtenu avec : `Color.rgb(0, 0, 255, 0.4)`
- Chaque bulle a une vitesse aléatoire entre 350 et 450 px/s vers le haut
- **Les bulles font partie du décor**, la méduse ne peut pas entrer en collision avec elles
- Pour donner un effet plus réaliste, les bulles sont générées en 3 petits groupes de 5 bulles :
  - Trois coordonnées $base_x$ sont générées aléatoirement entre 0 et la largeur de l'écran
  - Pour chacune de ces coordonnées, cinq bulles sont générées avec comme position x initiale la valeur $base_x \pm 20$
  - La position y initiale des bulles doit être à l'extérieur de l'écran (tout en bas). Les bulles montent jusqu'à ce qu'elles dépassent le haut de l'écran

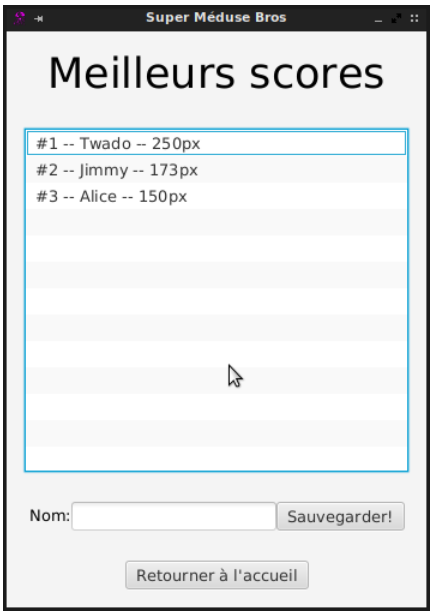
6 Fin du jeu

Lorsque la méduse est engloutie par le fond de l'océan, on doit afficher *Partie terminée* en rouge au milieu de l'écran, pendant 3 secondes.

Au bout des 3 secondes, on doit afficher l'écran des meilleurs scores avec la possibilité d'ajouter son nom.

Les meilleurs scores doivent être gardés d'une exécution à l'autre du programme : vous aurez besoin de les noter dans un fichier.





7 Interface graphique

Quelques précisions sur la fenêtre JavaFX :

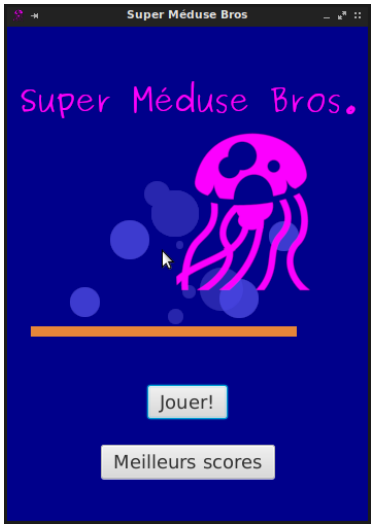
- La fenêtre doit avoir une largeur de `350px` et une hauteur de `480px`
- Le canevas (la fenêtre de jeu) dans l'application doit avoir la même taille et occuper tout l'espace
- La fenêtre ne doit pas être redimensionnable (*resizable*) : la taille de la fenêtre est fixée au début et elle ne peut pas être redimensionnée avec la souris. **Trouvez comment faire ça.**
- La fenêtre doit porter le titre "Super Méduse Bros"
- La fenêtre doit avoir une des images de la méduse en guise d'icône dans la barre de tâches. **Trouvez comment faire ça.**

Pendant le jeu, on devrait voir le score actuel en nombre de pixels montés. Ce nombre commence à zéro et correspond en réalité au nombre de pixels dont la caméra s'est déplacée depuis le début de la partie.

7.1 Scènes

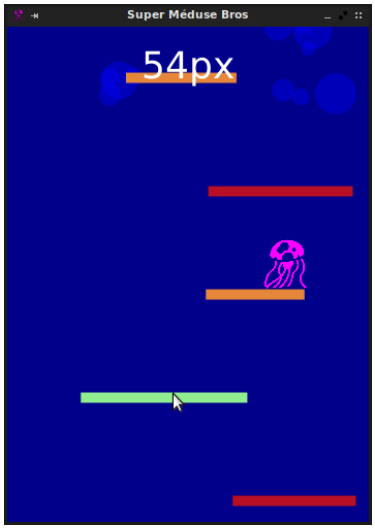
L'application est composée de **trois scènes différentes**

1. La scène d'accueil



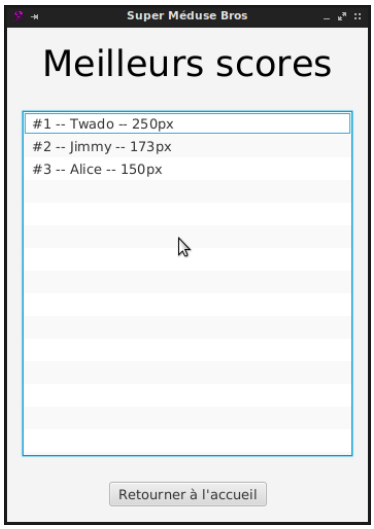
Sur l'accueil, on a un bouton pour commencer une nouvelle partie et un autre pour voir les meilleurs scores. Appuyer sur `Escape` ferme l'application avec `Plateform.exit()`.

2. La scène du jeu



Pendant le jeu, on a seulement un canevas qui affiche la partie. Appuyer sur `Escape` fait revenir à l'accueil. Quand la partie se termine, on affiche **Partie Terminée** sur l'écran pendant 3s, puis on va à la fenêtre des meilleurs scores.

3. La scène des meilleurs score



Sur les meilleurs scores, on a un bouton pour revenir à l'accueil.

Si on arrive sur la scène des meilleurs scores après avoir perdu la partie, on doit également pouvoir **ajouter son propre nom avec son score**.

7.2 Mode `Debug`

Pour faciliter les tests, vous devez inclure un mode `debug` à votre jeu, activable/désactivable en appuyant sur la touche `t` à n'importe quel moment.

Le mode `debug` permet de tester les plates-formes et l'avancement dans le jeu : lorsqu'il est activé, la fenêtre arrête de monter automatiquement et monte seulement lorsque la méduse dépasse les 75% de hauteur en sautant.

Ce mode aide à valider les collisions :

- Un carré rouge est dessiné dans la boîte englobante de la méduse. Cela correspond à toute la zone 50 par 50 qui compte dans une détection de collision entre la méduse et une plate-forme
- Lorsqu'une plate-forme est en collision avec la méduse, on change la couleur de cette plate-forme et on la dessine plutôt en jaune (`Color.YELLOW`)

De plus, des informations doivent être affichées en haut à gauche de l'écran. On doit afficher :

- La position (x, y) de la méduse *dans les coordonnées du monde* (et non de l'écran)
- Affichez juste en dessous de ça la vitesse de la méduse
- Son accélération
- Si oui ou non la méduse touche présentement le sol (soit parce qu'elle est sur une plate-forme, soit parce qu'elle est tout en bas du niveau, au début du jeu)

8 Conseils pour commencer

Le TP a **beaucoup d'étapes**, mais ce sont toutes des étapes que vous savez faire, qu'on a déjà fait en classe ensemble.

Allez-y **une étape à la fois**.

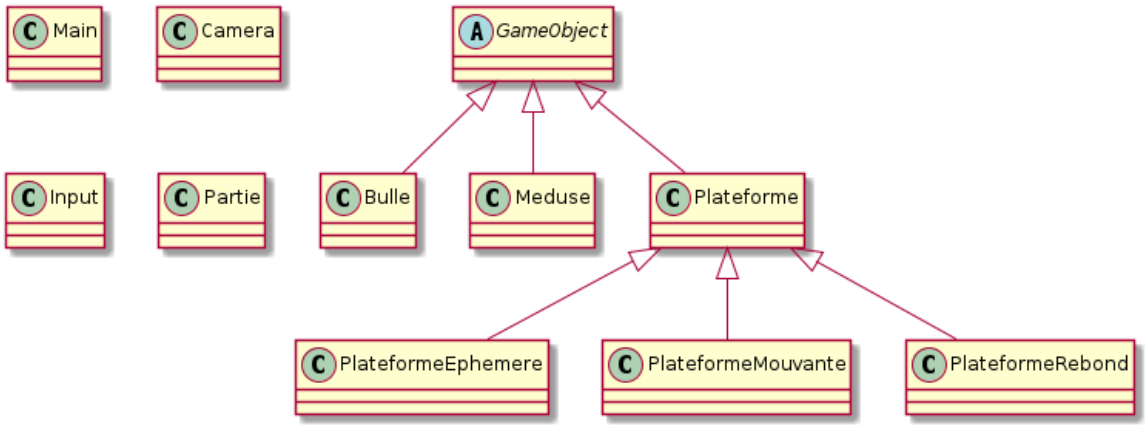
- Commencez très très simple, avec seulement une scène pour le jeu.
- Faites fonctionner la *physique* de la méduse.
- Une fois que ça marche, ajoutez **une** plate-forme **simple**.
- Quand vous avez déjà une plate-forme, essayez d'en rajouter **une autre sorte**
- Une fois que vous avez des plates-formes et une méduse, essayez d'ajouter la **caméra qui monte**
- ...
- Quand votre jeu marche, vous pouvez regarder comment on ajoute un écran d'accueil
- Quand vous arrivez à changer de scène, ajoutez une gestion des meilleurs scores

Vous allez vous rendre compte qu'en y allant avec une petite chose à la fois, le projet n'est pas si compliqué que ça finalement `:)`

➢ Si jamais vous n'arrivez pas à tout faire, je vais vous évaluer sur ce que vous avez **terminé**.

Mieux vaut faire un jeu incomplet (ex.: sans les meilleurs scores et sans l'accueil) que de tout faire sans que rien ne marche.

Pour vous aider, voici la décomposition de classes qu'on a suggérée en groupe :



Vous n'êtes pas obligés de la suivre à la lettre : vous aurez peut-être besoin de dévier de ça un tout petit peu.

9 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** mettre un maximum de code dans le *Modèle* de votre application (*à lire là-dessus : notes de cours `Modèle-Vue-Contrôleur`*).

Votre code JavaFX devrait se limiter au strict minimum pour afficher le jeu et les menus. toute la logique de comment la Méduse fonctionne et gagne ou perd la partie devrait être gérée par des classes complètement séparées des définitions de `VBox`, `Button`, `AnimationTimer` et autres.

Réfléchissez aux classes qui constituent votre *Modèle* pour ce jeu.

Comme d'habitude avec JavaFX, on doit pouvoir lancer votre programme avec Gradle.

10 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, plates-formes, ...), 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 plate-forme qui ne sera plus jamais affichée de tout le reste du jeu.

Si je joue à votre jeu pendant 15 minutes, demandez-vous : est-ce que je vais avoir un `ArrayList` de 892,136 plates-formes alors qu'on n'en voit que 5 à l'écran, ou est-ce que la quantité de mémoire utilisée va rester acceptable? :)

11 Éléments fournis

Aucun code n'est fourni, mais vous pouvez vous inspirer des exemples de code donnés et des exercices faits en classe pendant la session.

Les images nécessaires à l'animation de la méduse sont fournies avec l'énoncé et sont basées sur une image par le graphiste Lorc (image originale ici : https://game-icons.net/1x1/lorc/jellyfish.html).

12 Bonus

Seulement une fois que tout le reste marche, vous pouvez aller plus loin si le temps vous le permet.

12.1 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.

12.2 Version mobile (10%)

Portez le jeu sur Android. Faites vos recherches vous-même, si votre découpage MVC est bien conçu, vous ne devriez pas avoir trop de misère à adapter le code.

Vous *devez* utiliser exactement les mêmes classes dans votre projet JavaFX et dans votre projet Android.

Notez que vous ne pouvez **pas** utiliser un outil qui compile du JavaFX directement sur Android, vous devez utiliser la librairie standard d'Android pour définir l'affichage.

Si vous choisissez de faire ce bonus, envoyez-moi votre `.apk` par courriel au `nicolas.hurtubise@bdeb.qc.ca` avant la date de remise.

13 Remise

➢ Le travail est à faire en **équipes de deux**.

Remplissez le formulaire ici pour enregistrer votre équipe :

https://forms.gle/jznqECcueBJWtu726

Une seule remise par équipe est suffisante.

Vous devez remettre sur Léa votre projet IntelliJ (avec la configuration Gradle, le code, les ressources, etc.) **dans un fichier .zip**

La date de remise est spécifiée sur Léa.

14 Barème

- 60% : Fonctionnalités demandées implantées correctement
  - respect des consignes
  - **Pas d'erreurs!**
    Gérez les exceptions correctement
- 40% : Qualité du code
  - Code bien commenté
  - Respect du minusculeCamelCase pour les variables/méthodes, MajusculeCamelCase pour les noms de classes
  - Bon découpage en méthodes
  - Encapsulation : attributs `private`, avec getters/setters au besoin
  - **Pas de variables globales!**
  - **Pas de copier-coller de code!**
- (Facultatif) Jusqu'à 15% bonus

15 Note sur le plagiat

Le travail est à faire **en équipes de 2**. Ne *partagez pas de code avec une autre équipe que la vôtre*, même pas "juste pour aider un ami", ça constituerait un **plagiat**, et **tous les membres des équipes concernées auraient la note de *zéro***.