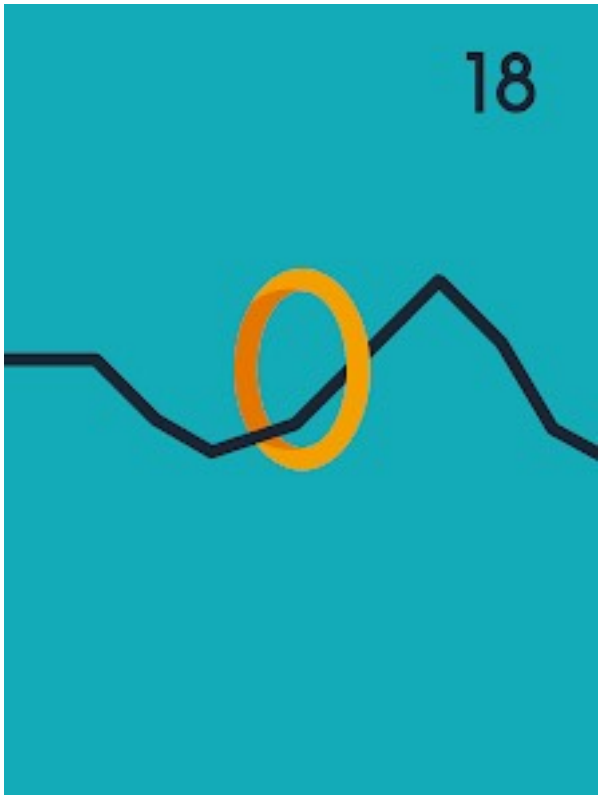
Rapport de projet : Circle

1. Introduction : Cahier des Charges

Objectif du projet : Nous voulons réaliser un mini-jeu inspiré de Ketchapp Circle où une ovale de déplace le long d'une ligne brisée générée de manière aléatoire. Le but du jeu est d'éviter de sortir de la ligne. Le joueur peut cliquer sur l'écran pour faire monter l'ovale, l'ovale descend au fur et à mesure du temps. On accède au jeu avec un menu.



2. Analyse globale :

Principales fonctionnalités à développer avec leur niveau de difficulté et niveau de priorité :

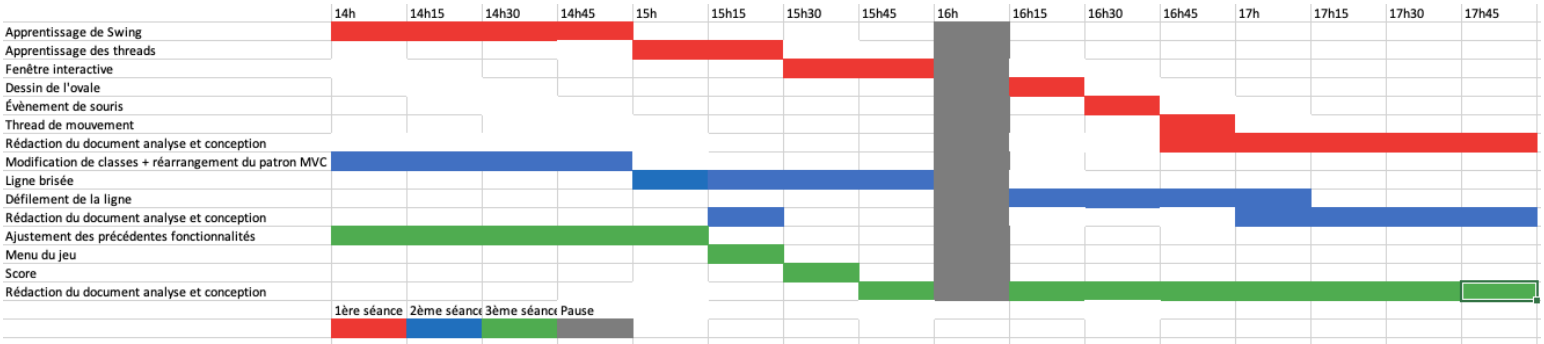
- Fenêtre interactive : difficulté moyenne, priorité 1
- Dessiner un ovale dans une fenêtre graphique: difficulté basse, priorité 1
- Contrôle du mouvement de l'ovale : difficulté basse, priorité 1
- Descente de l'ovale au fil du temps : difficulté basse, priorité 1
- Génération de la ligne brisée : difficulté moyenne, priorité 1
- Défilement de la ligne brisée : difficulté haute, priorité 1
- Le score : difficulté basse, priorité 3
- Menu du jeu : difficulté moyenne, priorité 2

3. Plan de développement :

Temps de travail en terme d'analyse, conception, développement et test pour chaque fonctionnalité :

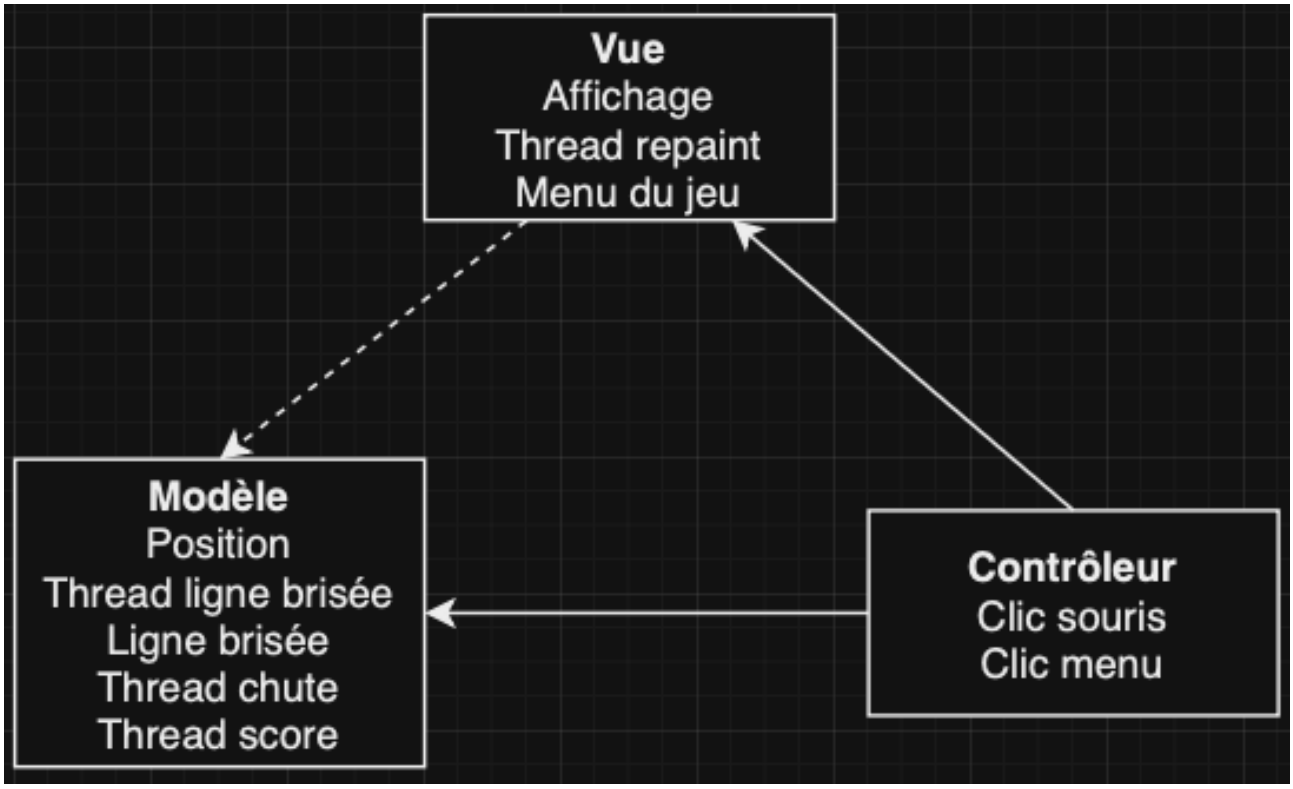
- Apprentissage de Swing : 1h
- Apprentissage des threads : 30 mins
- Fenêtre interactive : 30 mins
- Dessin de l'ovale : 15 mins
- Clic de souris : 15 mins
- Thread de chute 15 mins
- Rédaction du document analyse et conception : 1h15
- Modifications de classes + réarrangement du patron MVC : 1h développement
- Ligne brisée : 1h développement
- Défilement de la ligne : 1h développement
- Rédaction du document analyse et conception : 1h15
- Ajustement des précédentes fonctionnalités : 1h15
- Menu du jeu : 15 mins
- Score : 15 mins
- Rédaction du document : 2h

Une seule ressource, moi.



4. Conception générale :

On va suivre le patron de conception MVC où on sépare les données, la vue et le contrôleur et on utilisera le langage Java avec le paradigme objet.
Modèle : position, thread défilement ligne brisée, ligne brisée, thread chute, thread score
Vue : affichage, thread repaint, Menu du jeu
Contrôleur : Clic de souris, Clic dans le menu

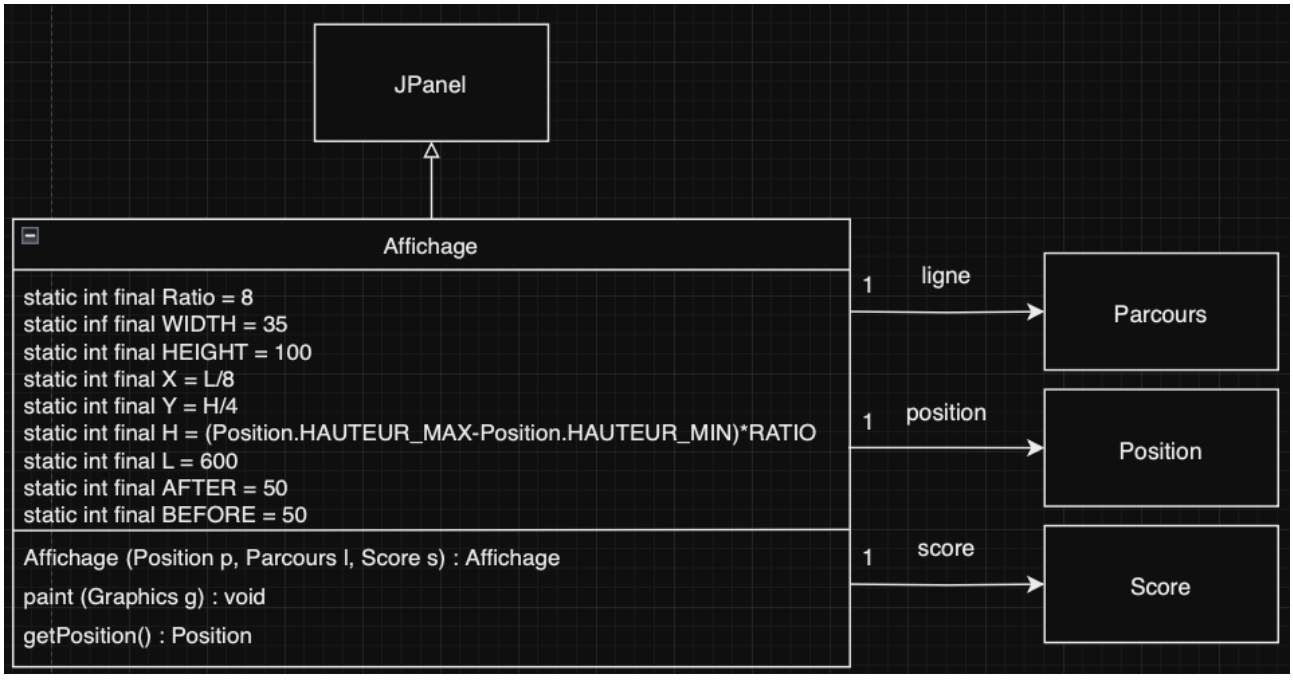


5. Conception détaillée :

Fenêtre interactive et dessin de l'ovale :

Utilisation des API swing, awt et util.ArrayList, et de la classe JPanel.

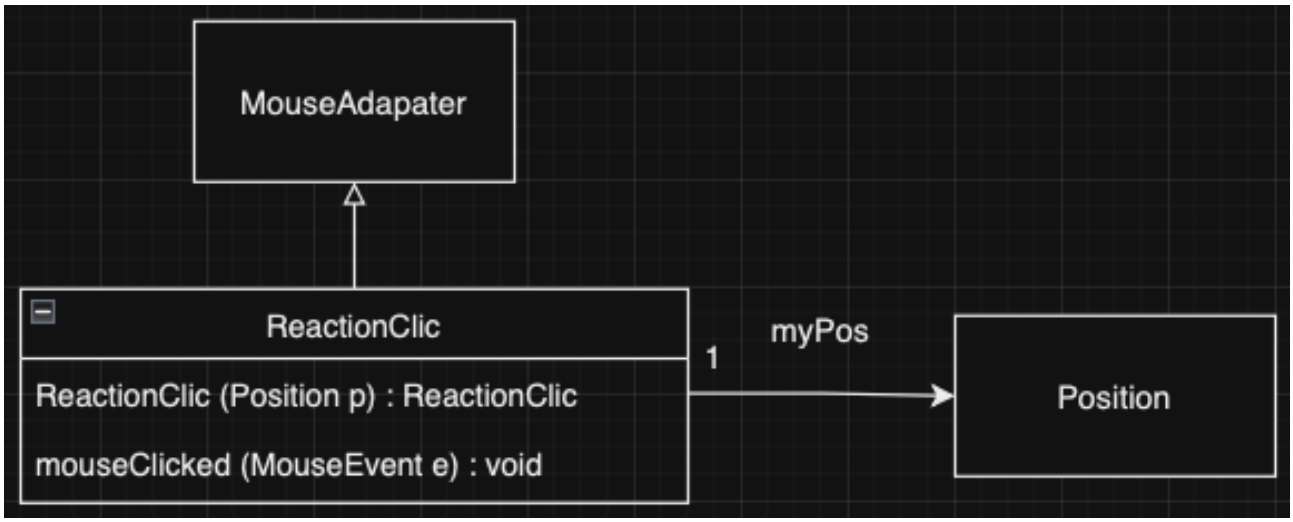
- Structures de données : définition des dimensions de la fenêtre et de l'ovale avec les constantes entières (H, L, HEIGHT, WIDTH, RATIO, AFTER, BEFORE, X, Y, H), position, ligne, score.
- Utilisation par les autres fonctionnalités : Utilisé par le thread Redessine.



Clic de souris :

Programmation événementielle avec l'API MouseEvent et la classe MouseAdapter.

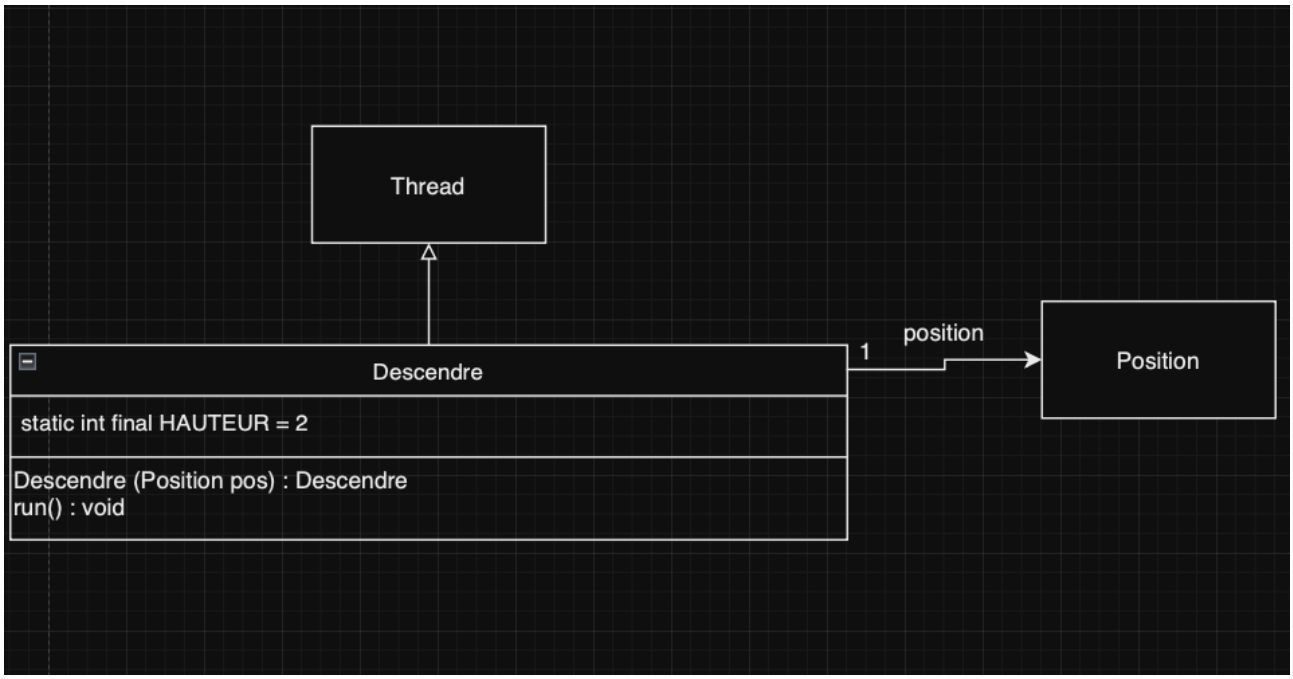
- Structures de données : position (modification de l'attribut hauteur)
- Utilisation par les autres fonctionnalités : thread de mise à jour de l'affichage (délai 50ms dans une constante)



Chute de l'ovale :

Programmation concurrente avec la classe Thread.

- Structure de donnés : Constante entière HAUTEUR

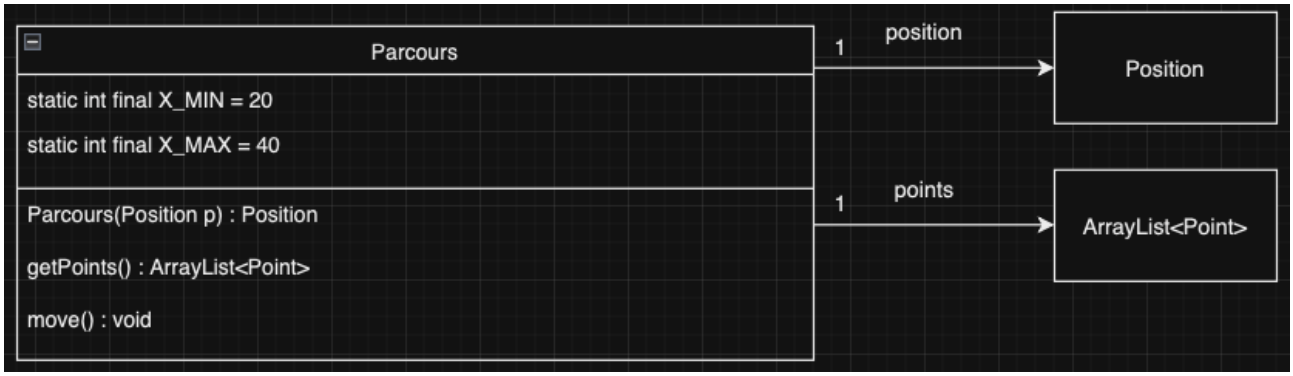


Ligne brisée :
Utilisation des API awt et ArrayList.
Structures de données : constante entières X_MIN et X_MAX, position pos, ArrayList points.

- Entrée : i, taille, x_premierpoint, x, points, BEFORE, X_MAX, X_MIN, HAUTEUR_OVALE, RANDOM, L, HAUTEUR_MAX

  Algorithme :
  i=2, taille = 20, x_premierpoint = BEFORE, x = RANDOM*(X_MAX-X_MIN) + X_MIN + x_premierpoint, points = ArrayList(taille)
  points[0] = Point(0-BEFORE, HAUTEUR_OVALE)
  points[1] = Point(x_premierpoint, points[0].y)
  Tant que x < L && i < taille-2 :
  x = x + RANDOM*(X_MAX-X_MIN) + X_MIN
  Si i modulo 2 = 0 alors points[i] = Point(x, points[i-1].y)
  Sinon points[i] = Point(x, RANDOM*(HAUTEUR_MAX-20)+10)
  i = i+1
  Pour j de i à taille - 1 :
  points[j] = Point(points[j-1].x+1, points[j-1].y)
  points[taille-1] = Point(L+AFTER, points[taille-2].y)

- Utilisation par les autres fonctionnalités : utilisé par Affichage pour afficher la ligne et le thread AvanceLigne pour faire avancer la ligne.

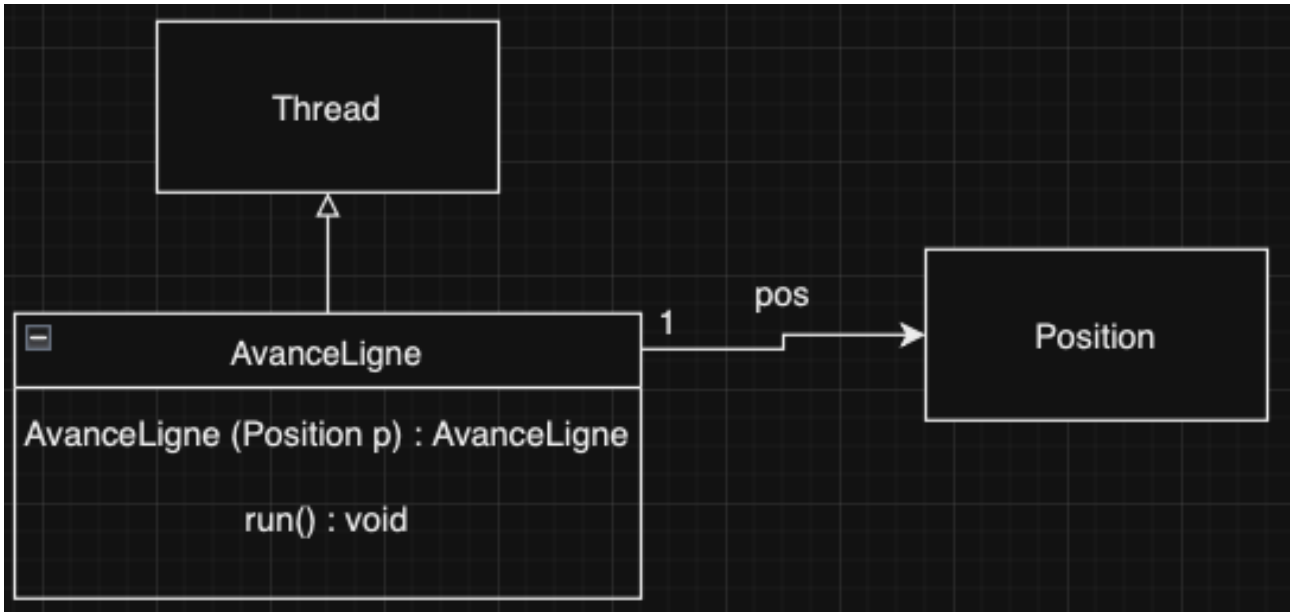


Défilement de la ligne brisée :
Programmation concurrente avec la classe Thread.

- Structure de données : position
- Le mouvement est fait dans la classe Parcours :
  Entrée : points, BEFORE, AFTER, L, RANDOM, HAUTEUR_MAX
  Sortie : points

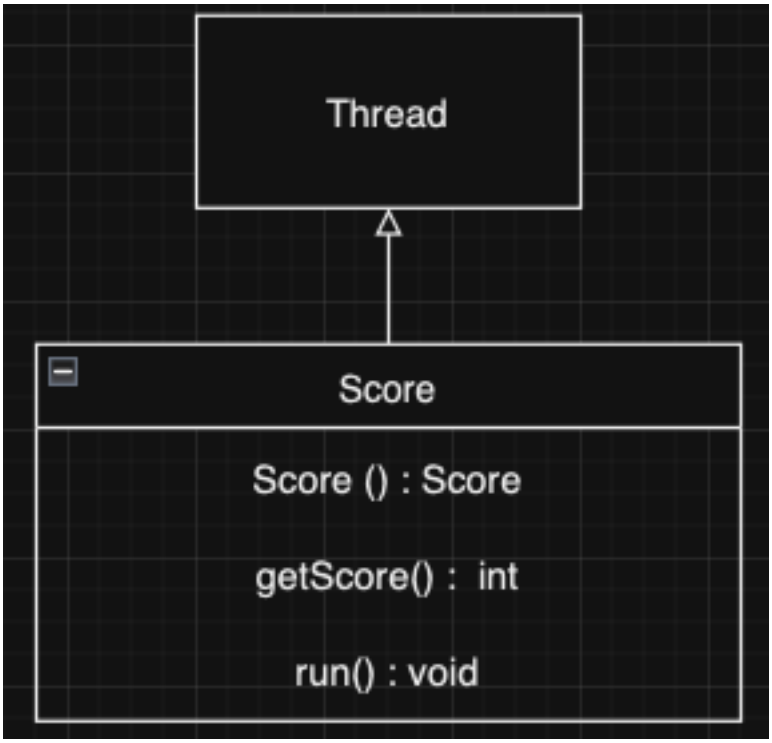
  Algorithme :
  Pour i de 0 à taille(points) :
  points[i].x = points[i].x - 1
  Si points[1].x < -BEFORE alors points.remove(1)
  Si points[taille(points)-1].x < L
  alors points.ajoute(L+AFTER, RANDOM*(HAUTEUR_MAX-20) + 10)

- Utilisation par d'autres fonctionnalités : Utilisé par position pour augmenter la vitesse de défilement de la ligne.

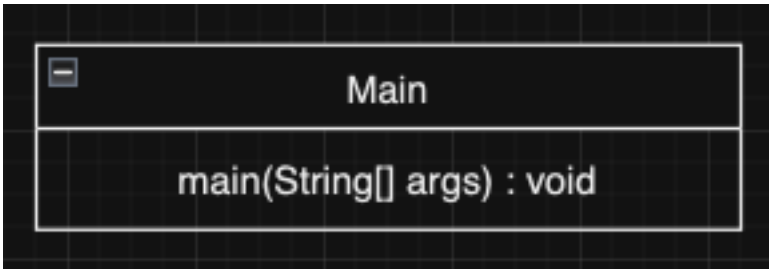


Score :

- Structures de données : entier
- Conditions limites : valeur non négative et qui augmente de manière positive
- Utilisation par les autres fonctionnalités : utilisé par affichage pour afficher le score

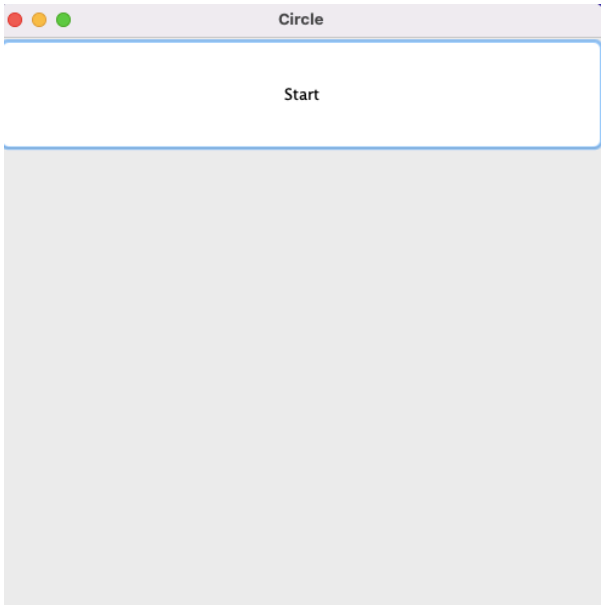


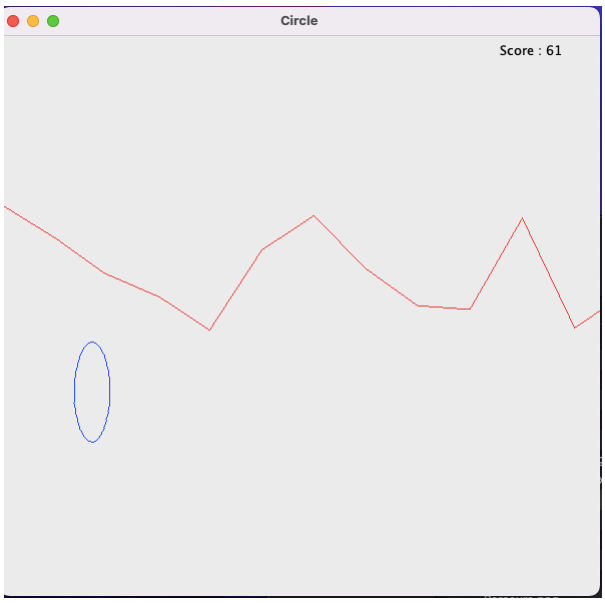
Menu du jeu :
Utilisation des API swing et awt, le code du menu se situe exclusivement dans la fonction main de la classe Main.
On initialise la fenêtre avec JFrame et on fixe les dimensions voulues puis on ajoute un bouton qui va lancer Jeu.main quand on clique dessus (actionListener).



6. Résultat :

Réalisation de la fenêtre interactive et de l'ovale, l'ovale peut sauter avec un clic de souris et descendre au fil du temps. La ligne brisée est présente et peut défiler, les points sont générés à l'infini à l'intérieur de la fenêtre de jeux pour donner l'impression d'une ligne infini. Un score est incrémenté chaque seconde. Le jeu est accessible avec un menu.





7. Documentation utilisateur :

Exécutable jar, il faudra s'assurer que l'utilisateur a bien ce qu'il faut pour exécuter un fichier jar.
On peut exécuter l'application avec : Clic droit sur l'icône du fichier .jar -> Ouvrir ou Double cliquez sur l'icône.
Contrôle de jeux : clic gauche de souris sur la fenêtre.

8. Documentation développeur :

Le code est organisé sous le patron MVC.
Il y a 4 packages utilisés par le jeux qui contient les classes :

- main : contient la classe Main avec le menu du jeu
- model : AvanceLigne, Descendre, Jeu, Parcours, Position, Score
- view : Affichage, Redessine
- control : ReactionClic

Les classes Hello, Test et Tâches sont des exemples pour se former.

Le code à exécuter est dans la classe Main, plus précisément dans l'unique méthode main qui va lancer un menu de jeu et qui va lui même lancer le code du jeu dans Jeu.main.
La classe ReactionClic appelle la méthode jump de position pour sauter.
La classe AvanceLigne est un thread qui fais varier la vitesse de défilement.
La classe Descendre est un autre thread qui va faire chuter l'ovale de HAUTEUR tous les certains temps à définir dans sleep de run.
La classe Jeu contient le code principale du jeu qui va initialisé toutes les instances nécessaire, lancer les threads et afficher la fenêtre du jeu.
La classe Parcours contient les éléments pour créer la ligne brisée et de quoi la faire bouger.
La classe Position manipule la position, les mouvements de l'ovale et contient l'élément qui fait varier le mouvement de la ligne brisé.
La classe Score est Thread pour faire augmenter le score toutes les secondes.
La classe Affichage se charge d'afficher l'ovale selon la hauteur de la position pos, la ligne brisée et le score.
La classe Redessine est un thread qui mets à jour l'affichage toutes les DELAY.
Les constantes/attributs importantes :

- Descendre.HAUTEUR : hauteur de la chute de l'ovale
- Dans Parcours X_MIN et X_MAX : définit l'écart entre chaque point de la ligne brisée
- Dans Position : HAUTEUR_MIN/MAX : définit l'écran de jeu de l'ovale, HAUTEUR_OVALE : définit la hauteur de l'ovale dans cet écran de jeu, IMPULSION : définit la hauteur du saut, avancement : définit la variation de la ligne, saut : démarre la chute
- Score.score : le score du jeu, on peut modifier dans run le temps d'augmentation du score
- Dans Affichage : RATIO : définit le ratio à multiplier aux éléments du modèle et de la fenêtre, H, L : définissent les dimensions de la fenêtre d'affichage du jeu, X, Y, WIDTH, HEIGHT : définissent les caractéristiques de l'ovale, AFTER, BEFORE : définissent les fenêtres maximales en dehors de la fenêtre d'affichage pour la ligne brisée
- Redessine.DELAY : définit le temps d'actualisation de la fenêtre de jeu

Prochaine étape : saut réaliste à faire (basé sur vitesse), leaderboard et animation dans l'accueil
Dans le package model : une nouvelle classe leaderboard qui va écrire dans un fichier txt les scores à chaque partie, modifier position et descendre pour ajouter vitesse et l'utiliser pour faire le saut réaliste.
Dans la classe Main qui contient le menu : ajouter une image puis la faire défiler sur le menu, cela implique de choisir une autre façon de faire le menu avec un JPanel.

9. Conclusion et perspectives :

On obtient au final une fenêtre interactive avec les clics de souris qui permettent de faire sauter l'ovale, elle redescend au fil du temps. On peut se déplacer le long de la ligne brisée qui est générée aléatoirement, le score augmente toutes les secondes et on accède au jeux grâce à un menu.
J'ai appris pas mal de choses : Créer une fenêtre et la rendre interactive, dessiner dans la fenêtre, créer un menu qui lance le jeu et modifier les données avec l'interaction et utilisation de thread. J'ai surtout appris à rédiger un document de conception qui a du sens et qui est utile pour une éventuelle reprise dans le futur.
La plus grande difficulté a été de faire fonctionner et utiliser le thread de hauteur pour faire sauter l'ovale, résolu avec les retours du profs et l'exemple Tâches, comprendre la génération de points a été encore plus corsé mais j'ai fini par comprendre en discutant avec un de mes camarades.
Le menu et le score a été implémenté grâce au copilot.

Il reste à remettre la vitesse, créer une animation dans le menu du jeu et créer un leaderboard. Pour que le leaderboard ait un sens, il faut également implémenter les collisions avec la ligne brisée et la défaite avec un éventuel menu de Game Over ou bien l'écrire dans un fichier à chaque partie.