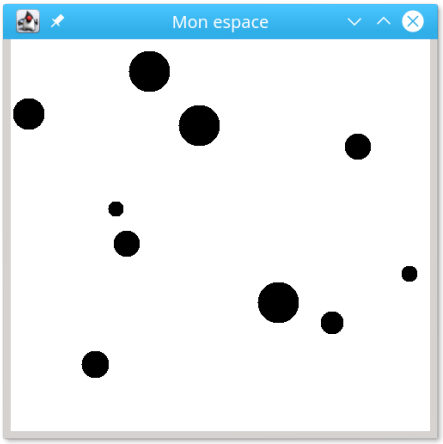
Sujet 1 : Implémentation d'un jeu vidéo

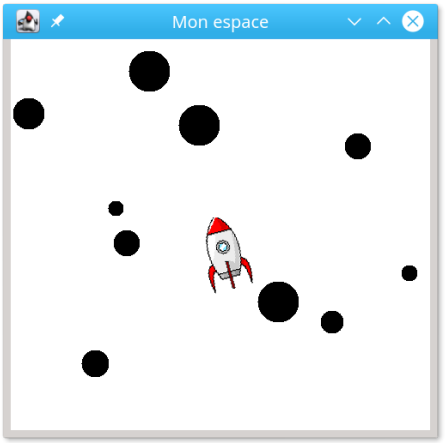
On a choisi comme sujet pour votre dernier projet l'implémentation d'un jeu vidéo du style des années 80. Vous pouvez penser que mettre en œuvre un jeu de ce type revient à réinventer la roue, et vous avez probablement raison. Cependant, c'est en suivant le chemin que les pionniers de l'informatique ont parcouru que vous serez en mesure d'apprendre les fondements et complexité de ce métier. Bien évidemment, de nos jours, on a plus de puissance de calcul et de nouvelles technologies logicielles qui nous permettent de faire des travaux plus impressionnants, mais les fondements mathématiques et physiques de l'informatique n'ont pas beaucoup changé au cours de ces dernières années.

1. Préliminaire

Pour ce dernier projet, on souhaite recréer un espace interplanétaire rempli de mondes différents. Par conséquent, on vous propose de créer un plateau (par exemple de 1000 par 1000 pixels) avec un nombre paramétrable de mondes, chaque monde pouvant être défini par la position de son centre (c_x, c_y) et son diamètre. Vous pouvez générer les deux, la position du centre et du diamètre, de manière aléatoire.



Évidemment, si vous voulez jeter un coup d'œil dans l'espace interplanétaire, vous aurez besoin d'un vaisseau spatial. On va vous fournir des designs de vaisseaux spatiaux au format png mais, évidemment, on invite chaque équipe à concevoir son propre design.



Ce premier sujet porte sur le système de navigation du vaisseau spatial que vous devrez implémenter. Plus particulièrement, on va se concentrer sur deux aspects du système de navigation : **le mouvement du vaisseau spatial et les collisions avec les planètes.**

2. Mouvement du vaisseau spatial

Pour recréer le mouvement du vaisseau spatial, bien évidemment, on va se baser sur les lois de la physique. Néanmoins, on ne va pas essayer de modéliser des conditions réalistes de l'espace (hey ! on veut mettre en œuvre un jeu vidéo et non le système de navigation du Falcon 9 !).

Étape 1 : Quelques attributs de base :

Positionnement : De toute évidence, pour pouvoir se déplacer dans l'espace, notre vaisseau spatial disposera d'un système de positionnement. Dans notre jeu on utilise des coordonnées cartésiennes en deux dimensions, donc, la position du vaisseau sera donné par des coordonnées (x,y) qu'on peut appeler **posX** et **posY** (ou quel autre nom de votre choix).

Vitesse : En termes physiques, la vitesse est le rapport de la distance parcourue au temps écoulé. On suppose que l'espace est exprimé en pixels et le temps en millisecondes. Donc, notre vaisseau spatial aura :

- Une vitesse maximale de MAX_VITESSE = 0.100. C'est-à-dire cent pixels par seconde.
- Une vitesse minimale de MIN_VITESSE = 0.0001. Cette valeur sera utilisée pour arrêter le vaisseau spatial si sa vitesse est inférieure à elle.
- Un vecteur (v_x, v_y) pour exprimer la vitesse actuelle. Comme vous le savez, un vecteur exprime la direction du mouvement et sa grandeur. La grandeur $||v||$ (que par commodité on va appeler vitesse) peut être obtenue par l'équation : $||v|| = \sqrt{v_x^2 + v_y^2}$ que nous donne la vitesse actuelle exprimée en pixels par milliseconde.

Angle de rotation : Comme vous pouvez déjà l'imaginer, on va utiliser la rotation expliquée dans le sujet 0 pour notre vaisseau spatial (flèche à droite pour augmenter la rotation de 5 degrés en sens horaires, flèche à gauche pour augmenter la rotation de 5 degrés en sens antihoraire). Pour

simplicité on peut garder l'angle en degrés et le convertir en radians à volonté en utilisant la fonction **`Math.toRadians(angle)`**

Accélération : Pour augmenter l'accélération (distance / temps^2) du vaisseau, on va sauvegarder une variable d'accélération. En principe, la valeur d'accélération sera 0. Si l'on appuie sur la flèche vers le haut (prévoir de modifier **`Keylistener`**), cette valeur augmentera de 0.001 tant qu'on ne relâche pas la flèche. Au moment de le relâcher, la valeur passera automatiquement à 0.

Décélération : on viendra à cet attribut plus tard dans l'étape 5. Pour le moment, on dira seulement que le vaisseau spatial subira une décélération de 0.00001 pixels / milliseconde^2.

Étape 2 : Rotation du vaisseau

Pour contrôler la rotation du vaisseau spatial, veuillez vous reporter au sujet 0. La principale différence pour ce sujet 1 est qu'on ne montrera pas le contour du vaisseau spatial mais l'image. À cette fin, il y a des méthodes spécifiques pour faire pivoter une image dans **`Graphics`**.

Étape 3 : Modifier le vecteur vitesse :

Au début du jeu, le vecteur vitesse (v_x, v_y) doit être initialisé à (0,0). Autrement dit, le vaisseau spatial ne bouge pas. Cependant, tant qu'on appuie sur la flèche en haut, l'accélération va augmenter et donc la vitesse du vaisseau. Afin de calculer la bonne vitesse, on devra prendre en compte trois composantes : l'angle du vaisseau spatial, l'accélération et la vitesse actuelle. Ceci est donné par l'équation :

```
vx = vx + acceleration * cos(angle)
vy = vy + acceleration * sin(angle)
```

où l'angle est donné en radians.

Comme vous pouvez le constater, si l'accélération est égale à 0, le vaisseau spatial conserve une vitesse uniforme. D'un autre côté, comme on impose une vitesse maximale de MAX_VITESSE, on devra vérifier la grandeur du vecteur (v_x, v_y). Si la grandeur est supérieure à MAX_VITESSE, on va conserver une vitesse uniforme à MAX_VITESSE.

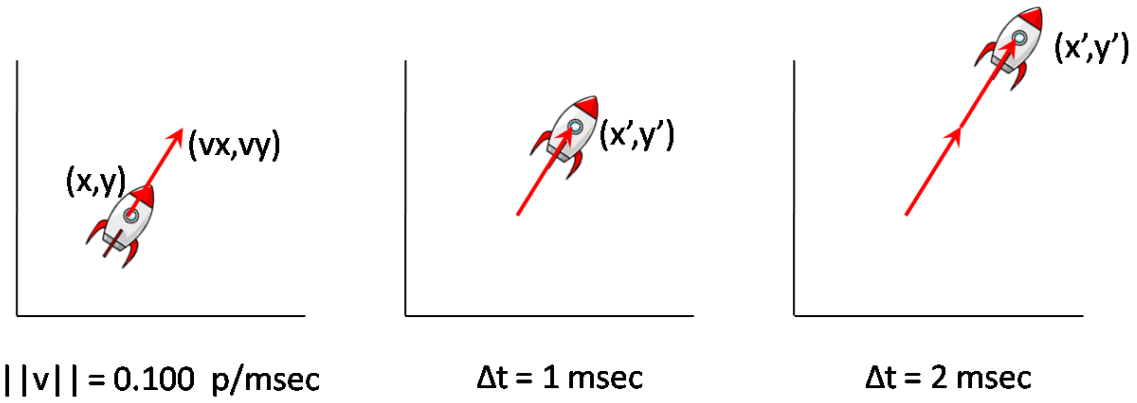
Étape 4 : Modifier la position par rapport au vecteur vitesse :

Afin de modifier la position du vaisseau spatial, nous devrons appliquer les équations suivantes :

```
x = x + (vx * Δt)
y = y + (vy * Δt)
```

NOTE : Δt (lire temps delta) fait référence au temps réel investi entre deux itérations de la boucle principale et peut être calculé à l'aide de la méthode **`System.currentTimeMillis()`**

Avec les images suivantes, vous pouvez avoir une idée plus claire de ce qui se passe. On va s'imaginer qu'on parte de l'image de gauche, avec une position connue (x, y) du vaisseau spatial et un vecteur vitesse connu (vx, vy) de grandeur 0,100 pixels / milliseconde. Si la boucle principale du programme prend une milliseconde à parcourir, on devra repeindre la nouvelle position du vaisseau spatial (x',y') telle qu'elle apparaît dans l'image centrale. Par contre, si la boucle prend 2 millisecondes, la nouvelle position (x',y') sera celle de l'image à droite.



Étape 5 : Décélération

Malgré la décélération d'un vaisseau spatial dans un espace interplanétaire sans aucune force extérieure enfreint les lois de la physique, on a décidé d'améliorer la jouabilité du jeu. Pour ce faire, le vaisseau spatial subira une décélération continue jusqu'à ce qu'une vitesse minimale MIN_VITESSE soit atteinte. Lorsque cela se produit, le vaisseau spatial s'arrête. Afin de mettre en œuvre cela, on va fixer la décélération à 0.00001 pixels/millisecondes^2.

Si l'on parte d'une vitesse initial de grandeur $||v_i||$et on applique la décélération à vaisseau, on va arriver à une vitesse final dont sa grandeur va être expreser par l'équation :

$||v_f|| = ||v_i||$- (Δt * décélération)

Par conséquent, le vecteur de vitesse (vx,vy) sera modifié par :

$$vx = vx * (||v_f|| \div ||v_i||)$$

$$vy = vy * (||v_f|| \div ||v_i||)$$

3. Detection de collisions

Une fois que vous avez réussi à implémenter le mouvement du vaisseau spatial, la détection de collisions avec les planètes ne doit poser aucun souci. Le secret est d'appliquer toutes les transformations (rotation et translation) à la fois à l'image du vaisseau spatial (que vous avez fait dans la section précédente) et aux contours du vaisseau spatial (que vous avez appris à implémenter dans le sujet 0). En particulier, on va utiliser les transformations appliquées aux contours du vaisseau spatial pour détecter les collisions.

Une fois qu'on connaît les coordonnées des contours du vaisseau spatial après les transformations, l'idée est de parcourir en boucle tous les différents mondes (ou planètes). Ensuite, afin de détecter s'il y a de collision avec une planète donnée centrée en (c_x, c_y), on va calculer la distance euclidienne entre chaque pixel du contour de vaisseau (p_x, p_y) et le centre de la planète :

$$d_{p,c} = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2}$$

si la distance entre un des pixels du contour (p) et le centre de la planète (c) est inférieure ou égale au rayon de la planète, cela signifie qu'il y a une collision entre les deux et le jeu doit donc se terminer. À vous de produire une belle explosion.

4. Exercice à rendre le mardi soir

Votre équipe devra d'implémenter les deux méthodes décrites dans les sections précédentes :
le mouvement du vaisseau spatial et la détection des collisions.

Game Over pour le sujet 1