# TP 7 - partie 2 : Space Invaders

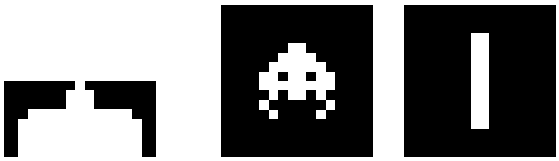
G. Levieux

## 1 Objectif du TP

Dans ce TP, vous êtes supposés travailler dans un studio de développement de jeux vidéo. Il a été décidé de créer une version de Space Invaders pour téléphone mobile, et l'on vous demande de réaliser, en Java, un petit prototype du jeu.

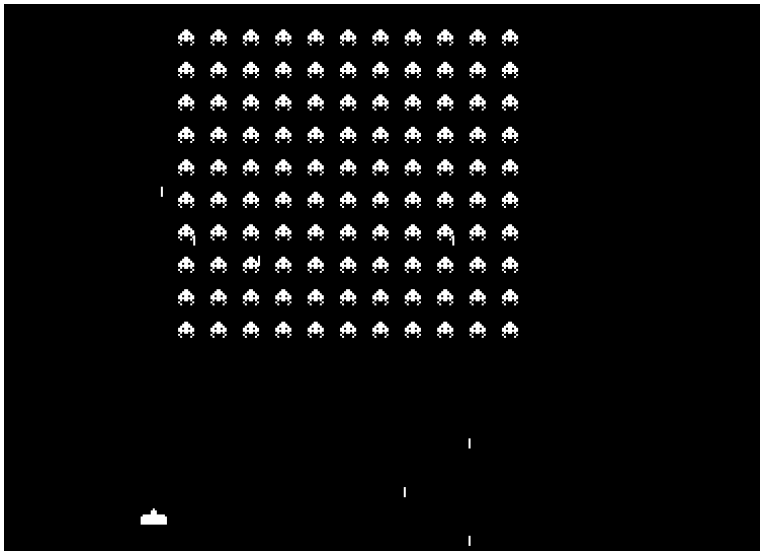
## 2 Ressources graphiques

Vous disposez pour cela de plusieurs ressources. Tout d'abord, les graphistes ont proposé trois images :



Vaisseau du joueur Ennemis Missiles

En accord avec le game designer, concepteur du jeu, voici a quoi doit ressembler votre prototype :



# 3 Règles du jeu

Le game designer vous a expliqué les règles du jeu. Votre application devra respecter ces règles.
— Il y'a une masse de 10 * 11 ennemis au dessus du joueur au départ.
— Toute la masse se déplace chaque seconde de 16 pixels
— Toute la masse se déplace quatre fois à droite, une fois en bas, puis quatre fois à gauche, une fois en bas, puis à nouveau quatre fois à droite, etc...
— Lorsqu'un ennemi atteint le sol, la partie est perdue.
— Toutes les 100 milisecondes, chaque ennemi a 5 chances sur 1000 de tirer un missile vers le sol.
— Chaque ennemi ne peut tirer que si son missile précédent n'est pas déjà visible à l'écran.
— Le missile ennemi tombe de 4 pixels toutes les 30 milisecondes.
— Si un missile ennemi touche le joueur, la partie est perdue
— Le joueur peut se déplacer a chaque pression des touches gauche et droite de 16 pixels, mais il ne va jamais plus loin que les positions horizontales maximales des ennemis.
— Quand le joueur appuie sur espace, il tire un missile vers le haut de l'écran.
— Si le missile du joueur touche un ennemi, ce dernier disparait de l'écran et est considéré comme mort.
— Le joueur ne peut tirer que si son précédent missile n'est pas visible à l'écran.
— Le missile du joueur monte de 16 pixels toutes les 30 milisecondes.

# 4 Ressources logicielles

Votre studio n'en est pas à son premier développement de jeu. Vous disposez d'un tout petit moteur de jeu, spécialement développé pour concevoir des prototypes de jeu en Java. Ce moteur se compose des deux classes suivantes :

## 4.1 La classe GameObject

### 4.1.1 Utilité de GameObject

Tous les jeux ont des objets, qui ont un certain nombre de comportements de base qu'il n'est pas nécessaire de recoder à chaque fois. Vous disposez donc d'une classe `GameObject` qui représente un objet du jeu, comme un missile, un ennemi ou même le joueur. Cette classe permet de tester si deux objets du jeu sont en collision (si ils se touchent), de déplacer un objet, d'afficher un objet en dessinant son image, etc... Tous ces comportements sont fournis par GameObject, et vous aurez juste à les utiliser.

Pour votre jeu, vous allez devoir hériter de cette classe de base, et redéfinir certaines méthodes, pour tous les objets du jeu. Il y'en a quatre types : la classe joueur, la classe missile du joueur, la classe ennemi et la classe missile ennemi. Vous devrez créer chacune de ces classes, et au minimum, redéfinir les méthodes comme expliqué ci-après.

### 4.1.2 Redéfinir le constructeur

Tout d'abord il faudra créer votre propre constructeur, pour initialiser vos attributs et paramétrer votre objet. Il faut par exemple définir la taille et la position du rectangle de collision avec setBbox(), choisir votre image avec setImage().

### 4.1.3 Redéfinir update()

Ensuite il faudra redéfinir la méthode `public void update(int time)`. Cette dernière vous permet de mettre régulièrement à jour votre objet. Par exemple, l'ennemi doit se déplacer toutes les secondes, et faire un tirage aléatoire pour savoir si il tire un missile, le missile du joueur doit monter de 16 pixels toutes les 30 milisecondes, etc...

Vous utiliserez les méthodes moveTo() et moveRelative() pour le déplacement des objets, getX() et getY() pour connaitre leur position, etc... Plusieurs fonctions de base de GameObject vous seront très utiles. Faites appel aux javadocs pour en avoir la description.

## 4.2 La classe GameEngine

### 4.2.1 Utilité de GameEngine

Cette classe est écrite pour vous permettre de ne pas avoir a créer de fenêtre, ni à vous occuper des tâches récurrentes que réalise tout moteur de jeu : charger une image, écouter les évènements clavier, mettre à jour les objets du jeu à une certaine fréquence, afficher les objets du jeu sans scintillement, etc...

Votre objectif consiste à développer un jeu particulier à partir de ce moteur, vous allez donc créer votre propre classe, qui héritera de la classe GameEngine, et qui redéfinira uniquement les méthodes nécessaires. Les sections suivantes donnent les méthodes que votre classe de moteur de jeu devra redéfinir.

### 4.2.2 Redéfinir le constructeur

Vous devrez tout d'abord redéfinir le constructeur. Vous allez en effet créer vos propres objets de jeu. Vous utiliserez la méthode addObject() pour ajouter vos objets au moteur de jeu, et startEngine() pour lancer le moteur de jeu une fois tous les objets créés.

### 4.2.3 Redéfinir updateGame()

La méthode `public void updateGame(int time){}` est régulièrement appelée par le moteur de jeu, juste après qu'il ait demandé la mise à jour de tous les objets du jeu. Son paramètre vous permet de savoir combien de temps s'est écoulé depuis le dernier appel, en milisecondes. Il vous faut redéfinir cette méthode pour mettre à jour votre jeu. Vous y vérifierez certaines conditions générales sur les objets du jeu :

- y a-t-il eu collision entre un missile du joueur et un ennemi ? (voir la doc de GameObject pour les collisions)
- y a-t-il eu collision entre un missile ennemi et le joueur ?
- un ennemi a-t-il touché le sol ?

Dans chaque cas, vous prendrez la bonne décision : désactiver un objet avec setAlive(), déclarer la fin du jeu avec setGameOver().

### 4.2.4 Redéfinir traiteTouche()

La méthode `public void traiteTouche(int code){}` est appelée par le moteur de jeu à chaque fois que le joueur presse une touche du clavier. Vous devez la redéfinir, pour modifier vos objets du jeu à chaque événement clavier. Par exemple, si le joueur appuie sur droite (dont le code est `KeyEvent.VK_RIGHT`) alors son vaisseau se déplace de 16 pixels vers la droite.

Pour chaque touche utile (gauche, droite et barre d'espace), vous modifierez l'objet du jeu concerné. Les méthodes moveTo() et moveRelative() setAlive() vous permettront de déplacer des objets et de gérer leur statut.

D'autres fonctions du moteur de jeu vous seront utiles. Pensez à générer les javadocs du moteur et à les consulter.

# 5 En conclusion

Vous allez devoir concevoir une architecture pratique d'objets qui vous permette de coder le prototype de SpaceInvaders. Vous aurez une classe à écrire par objet du jeu (ennemi, missile du joueur, missile alien, et joueur) ainsi qu'une classe pour le moteur de jeu. Utilisez l'héritage pour profiter des fonctionnalités offertes par le moteur de jeu ainsi que pour vos propres objets. Le TP est terminé lorsqu'on peut effectivement jouer

à SpaceInvaders. Pour vous aider nous vous fournissons un .jar du jeu terminé ainsi qu'un projet NetBeans qui contient les images et les classes GameEngine et GameObject.

# 6 Conseil d'optimisation / simplification

Petit conseil supplémentaire : dans un jeu vidéo, il faut écrire du code optimisé, c'est à dire rapide a exécuter par le système. Une bonne manière d'optimiser votre code (et d'éviter un bon nombre de bugs) consiste à éviter de créer / supprimer des objets en permanence. Chaque création d'objet est longue car elle déclenche un appel système pour la gestion de la mémoire. Les destructions / créations sont sources de bugs car elles amènent à manipuler des références nulles. De plus, le ramasse-miette, en se déclenchant, produit un ralentissement très court, mais perceptible, du jeu ; c'est en particulier vrai sous Android, où la machine virtuelle java est moins efficace que son équivalent sur micro.

Solution : le plus simple consiste à créer tous les objets dont vous avez besoin au démarrage, puis à utiliser la méthode setAlive(). Si vous déclarez un GameObject comme mort, il ne sera plus affiché et n'entrera plus en collision, mais conservera sa place en mémoire, prêt à être réutilisé. Par exemple, autant réutiliser toujours le même missile pour le joueur, que d'en créer / détruire un à chaque pression de la barre d'espace. Vous éviterez également les insertions / suppressions dans la liste des objets du moteur.

# 7 Une petite référence bibliographique

Pour java : Andrew Davison, *Killer Game Programming in Java,* O'Reilly. Un peu ancien, mais très pédagogique. Pré-versions Disponibles sur le net : http ://fivedots.coe.psu.ac.th/ ad/jg/