Rapport du projet d'informatique

Charlier Maximilien – Corentin Ducruet

BA1 Sciences informatiques

UMONS 2012-2013

Présentation

Le projet consiste à réaliser une version du célèbre jeu « Astéroïdes ». La version de « base » de ce jeu a pour but de survivre à bord d'un vaisseau dans l'espace en détruisant les astéroïdes pouvant le percuter.

Ce but est légèrement différent dans le nôtre. En effet, le joueur se trouve toujours dans l'espace, mais il doit sauver ses collègues astronautes perdus dans celui-ci. Cependant le danger est toujours présent puisque le joueur peut se faire percuter par un astéroïde.

Le vaisseau se trouve en permanence au centre de la fenêtre. Au premier niveau, le joueur a une minute trente pour sauver cinq astronautes. Le temps augmente à chaque niveau de vingt secondes et le nombre d'astronautes augmente de un.

La partie s'arrête lorsque le temps pour sauver les astronautes est dépassé ou quand le vaisseau n'a plus de vie.

Points forts

- Le logiciel *ANT* gère les commandes requises, mais aussi *« javadoc »* qui permet de générer la *Javadoc*, « clean » qui supprime le dossier *Build* et « jar » qui crée un dossier avec un fichier exécutable *.jar*.
- Le jeu peut être redimensionné selon différentes tailles prédéfinies. Il y a la possibilité de changer le rafraichissement par seconde.
- Concernant l'aspect graphique, nous avons dessiné toutes les images.
- Le jeu est orienté tactique : il faut prendre des décisions qui faciliteront ou pas le déroulement du jeu.
- Collisions entre astéroïdes : ils rebondissent entre eux.
- Facilité d'utilisation et de compréhension grâce à l'aide complète.

Point faible

- Lenteur d'exécution.

Répartition des tâches.

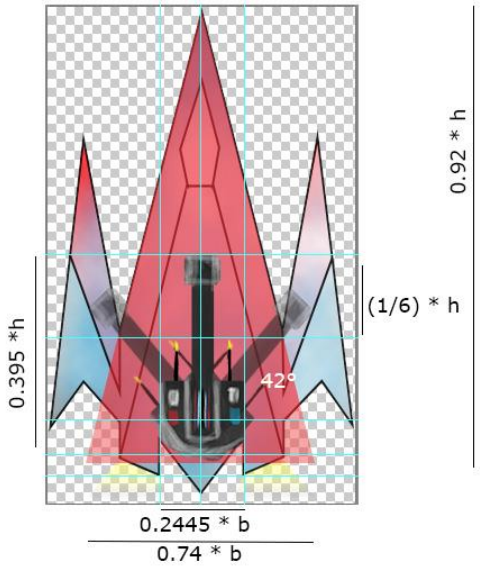
Ducruet Corentin

- La classe *GuiEvent* contenant les sous-classes servant à dessiner les différents fonds : l'aide, l'image d'accueil, l'image de pause, l'image de « Game over »,…
- La classe *Level* contenant le niveau courant, le temps qu'il reste pour finir le niveau, le nombre d'astronautes qu'il reste à sauver. Cette classe se charge aussi d'ajouter les astronautes quand on passe un niveau.
- Les tests unitaires de la classe *Geom* contenant toutes les méthodes liées à la géométrie et de la classe astéroïdes.
- L'affichage de différents éléments à l'écran : le bonus, le malus, les missiles, les informations liées au niveau.
- La classe *ImageGame* permettant de charger toutes les images en début de jeu pour ne plus avoir à les recharger pendant le jeu.
- La classe *Scoring* permettant de calculer le score du jeu en fonction des astéroïdes, des astronautes et du niveau.
- La classe *Bonus* s'occupant de modéliser les différents bonus.
- La classe *Missile* s'occupant de modéliser les missiles lancés par le vaisseau.
- L'implémentation du *Timer*.
- L'implémentation du menu.
- L'implémentation de la possibilité de faire une pause, une nouvelle partie et du « Game over ».

Charlier Maximilien

- La classe *Engine* : gestion des collisions, rafraichissement des différents éléments, mise à jour des calques pour l'affichage, gestion des bonus, division des astéroïdes.
- Tests unitaires de la fonction angle de la classe *Geom*.
- L'implémentation de la classe *Geom*.
- La création des images de l'interface graphique ainsi que leurs placements pour la plupart.
- Le vaisseau, le lance-missile et la vie.
- Le double *Timer* (pour rendre le jeu plus rapide).
- Le radar.
- Les gestionnaires d'évènements (clavier, souris).
- La classe *Moving* permettant de centraliser les fonctions de base des mobiles.
- Le concept de « calque » ainsi que son implémentation.
- Les astronautes.
- La classe *Config* regroupant la majorité des variables modifiables pour configurer plus facilement le Game Play du jeu.
- Les *packages* ainsi que le *ANT*.
- Le fond mobile du jeu.
- Les explosions.
- La gestion du temps.
- L'implémentation de la gestion du rafraichissement de l'affichage et du jeu.
- L'implémentation du redimensionnement de l'affichage.
- L'aide commande, le jeu et l'interface.

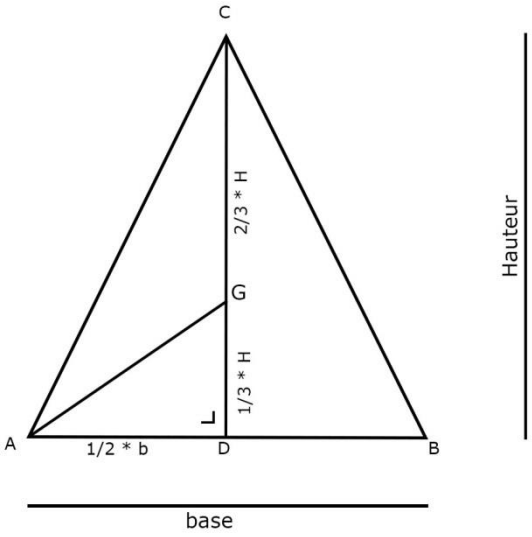
Implémentation.



Représentation du vaisseau

Le vaisseau est identifié dans le jeu par la zone rose (vous verrez plus bas comment elle est construite). Le lance-missile est désaxé de 1/6 par apport au centre de gravité.
Par souci d'esthétique et de Game Play, la tourelle ne peut pas faire un tour complet du vaisseau et est donc bloquée à 42° par rapport à la pointe du vaisseau.
À chaque tir le centre du canon recule d'un pixel par rapport à l'axe de tir. Ensuite, le vecteur d'orientation est réorienté en fonction de l'alignement du centre de gravité du vaisseau et du canon.

Construction du vaisseau



Nous avons décidé de pouvoir construire le vaisseau à partir d'une largeur et d'une hauteur donnée (en fonction d'une image). Les proportions par rapport à l'image ont été calculées avec l'outil de mesure de Photoshop.

Pour ce faire nous avons eu besoin de calculer et mémoriser l'angle AGC, la longueur de GD et le vecteur GC.
Le vecteur GC = 2/3 * H
L'angle AGC = 180- AGD

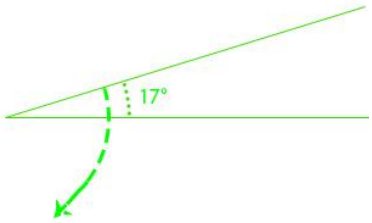
avec $\widehat{AGC} = \tan^{-1} \frac{1}{2} * base \Big/ \frac{1}{3} * hauteur$

La longueur de GD est obtenue via Pythagore.

Pour obtenir les points du triangle à partie de G, il ne reste plus qu'a effectué pour C : additionner G avec GC, pour A : additionner G avec GC tourné de AGC et réduit à la longueur AG, pour B même chose que A mais avec –AGC (car triangle isocèle).

Radar

Notre première approche était basée sur un radar à impulsion tel un sonar : à un temps t une impulsion est envoyée depuis le vaisseau et les astronautes sont affichés autour de la zone de jeu à partir du moment où l'onde rentre en collision avec l'astronaute. De base cela ne devait pas poser de difficultés, mais le fait d'avoir introduit le module espace impliquait un problème : les astronautes pouvaient se trouver très loin du vaisseau, en conséquence, l'onde devait soit être très rapide, soit ne couvrir qu'une certaine distance et donc ne pas capturer tous les astronautes ce qui rendrait le jeu très difficile à jouer.

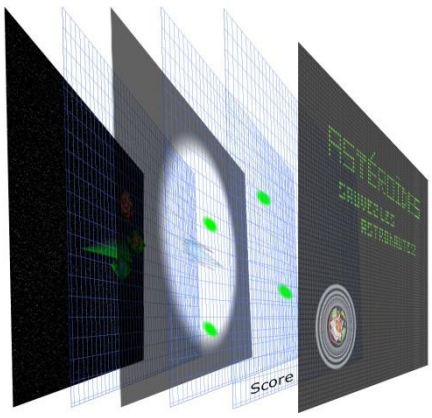


Nous avons donc choisi une autre approche, un radar de type sonar modélisé comme sur les écrans d'observation des sous-marins. Ce type de sonar détecte instantanément les astronautes une fois qu'ils se trouvent dans une zone de 17° par rapport à l'orientation de l'onde du radar qui, elle, tourne autour du vaisseau.

Le bonus radar implique l'accélération de l'onde. Le malus IEM implique que l'onde ne tourne plus et que l'on ne calcule plus les collisions entre l'onde et les astronautes. Concernant la détection, la première version était modélisée par un polygone que l'on faisait tourner autour du centre de gravité du vaisseau, à chaque « tic », nous vérifions si les astronautes étaient en collision avec celui-ci.

Pour améliorer la complexité, nous avons choisi qu'à chaque « tic », l'angle entre le vaisseau et l'astronaute soit calculé et comparer à l'angle du radar. Ce choix a permis de réduire la complexité et s'est révélé astucieux et précis. De plus pour rendre le radar plus réaliste, les informations sont mises à jour à chaque « tic », cela implique que lors de la détection, les informations « changent » pendant une demi-seconde comme un vrai radar.

Affichage principe de « couche »



Nous avons organisé l'affichage via 6 couches superposées que nous avons modélisées via une *arrayList* contenant 6 *listes chainées*. De telle sorte qu'à chaque affichage, les 6 listes sont parcourues dans un ordre voulu pour empiler les éléments afin d'obtenir l'affichage voulu.

Les 6 couches contiennent :

0. Le fond et l'onde radar.
1. Les astéroïdes, les bonus, les missiles et les astronautes.
2. Le fond dégradé et le vaisseau.
3. Les missiles (proches du vaisseau) et les données du radar.
4. La barre de vie, l'échauffement du canon, le score, le niveau,…
5. Les évènements de la *GUI* : « Game over », pause et aide.

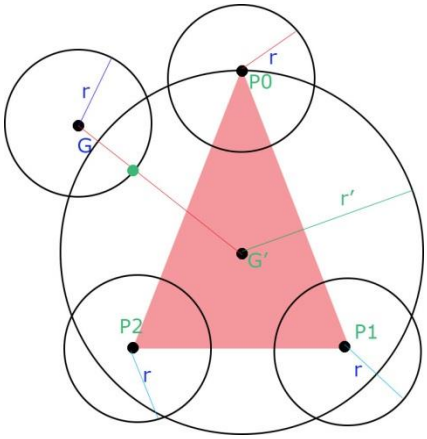
Affichage et scintillement

Nous avons été confrontés à plusieurs problèmes concernant la *GUI* :

- Un scintillement de l'affichage, que nous avons réglé en activant un double tampon et en demandant d'ignorer les demandes d'actualisation de l'affichage par l'*OS*.
- L'utilisation de *liste chainée* qui entrainait des problèmes de « ConcurrentModificationException». Nous les avons réglés en synchronisant tous les accesseurs liés à cette liste.
- A la base, nous effectuons 100 *repaint* par seconde. Il s'est avéré que les PC de la salle informatique Escher étaient trop peu puissants pour en faire autant. Nous avons donc décidé de descendre à 30 *repaint* par seconde, ce qui limite la lenteur du jeu sur ces PC. Pour conserver une certaine fluidité sur les PC plus puissants, nous avons choisi de permettre à l'utilisateur de pouvoir augmenter le nombre de *repaint*.

Gestion des collisions

Collision entre le vaisseau et les astéroïdes ou astronautes

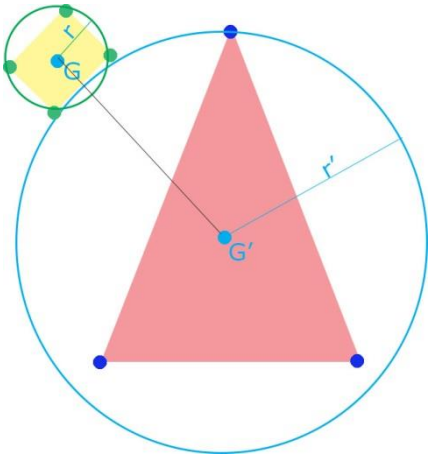


Nous faisons les tests du moins couteux au plus couteux.

Nous évaluons la distance entre G et G' si celle-ci est plus petite que r + r'. Il est alors possible d'obtenir une collision.

Dans ce cas :

- Les distances sont évaluées en G et P0, G et p1, G et P2 si une de ces distances est inférieure à r, une collision se crée automatiquement.
- Sinon, le point sur le vecteur G'G est calculé avec une distance GG' – r en partant de G. Ce qui modélise le point le plus proche de l'astéroïde par rapport au vaisseau et l'algorithme vérifie si ce point appartient au polygone du vaisseau.



Collision entre le vaisseau et les bonus

L'algorithme évalue si la distance entre G et G' est plus petite que r + r' si oui, l'on teste si un des 4 points du bonus appartient à la zone de détection du vaisseau, si oui, la collision s'effectue. Sinon, l'on teste si un des 3 points du vaisseau appartient à la zone de détection du bonus, si oui, la collision s'effectue, si non, les tests sont terminés et il n'y a pas de collisions.

Collisions entre astéroïdes

Nous avons choisi d'implémenter les collisions entre astéroïdes dans un but purement graphique. Nous considérons les collisions comme des collisions inélastiques, ce qui facilite les calculs. En effet, lorsque deux astéroïdes entrent en collision, il suffit d'inverser les vecteurs de direction et de modifier leur vitesse comme suit : les vitesses sont inversées et modifiées avec un ratio entre les deux rayons. De cette manière, les petits astéroïdes reçoivent une vitesse élevée des grands astéroïdes et inversement.
Comme cela est purement visuel, les collisions n'ont lieu que dans la zone d'affichage afin d'éviter des calculs inutiles.

Gestion de l'espace

Les consignes imposaient d'avoir un espace infini pour ne pas avoir un gérer une infinité d'astéroïdes. Nous avons choisi de le gérer dans une zone circulaire par rapport au vaisseau d'un diamètre valant 3 fois la diagonale de la fenêtre. Une fois qu'un astéroïde se trouve trop loin, il est téléporté avec une symétrie par rapport au vaisseau à une distance aléatoire, cela permet d'éviter d'avoir des « champs d'astéroïdes » trop denses et donne plus de dynamique visuelle lorsque nous jouons.

Développement

Nous avons développé le jeu sous *Windows* avec *Notepad++*, nous avions comme objectif de ne pas utiliser *Eclipse* et nous avons réussi à le tenir.

Pour faciliter le travail de groupe, nous avons utilisé un système de version *GIT* installé sur un serveur dédié *OVH*. Ceci permet de pouvoir faire les tests sur *Linux* sans contrainte et de pouvoir travailler sur plusieurs ordinateurs en même temps et sur les mêmes fichiers étant donné que *GIT* gère la fusion de fichier. De plus comme *OVH* réalise une sauvegarde journalière, nous étions donc assurés de toujours conserver nos données à disposition en cas de problème technique chez eux comme chez nous. Lors de l'apparition de bogue, nous pouvions remonter dans l'historique pour voir à quel moment ils apparaissaient et récupérer d'anciennes versions du code.
Concernant l'aspect graphique, les images ont été réalisées avec *Adobe Photoshop* et pour certaine avec l'utilisation d'une tablette graphique.

Apport du projet.

Ce projet a été très enrichissant. Il nous a permis d'apprendre à travailler, à nous gérer en groupe et acquérir un esprit de groupe, à nous mettre d'accord, à gérer notre temps,… Cela nous a aussi rapproché de la classe, car nous avons pu discuter de nos problèmes et de nos avancements.

Il nous a aussi permis de mettre en pratique concrètement les cours théoriques de programmation et algorithme II tout en apportant un aspect ludique. Nous avons pu apercevoir également les difficultés qu'il peut y avoir dans la conception d'un jeu vidéo ainsi que les contraintes. Nous avons pu apprendre à utiliser un logiciel de version et d'acquérir les commandes de base de Java. Cela nous a aussi appris à utiliser la *Javadoc* et à effectuer des recherches pertinentes.

Problèmes connus.

- Lors de son utilisation sur des PC peu puissants, le jeu peut subir des ralentissements et l'affichage à l'écran peut clignoter.
- Si le jeu est mis en plein écran via un raccourci clavier, il peut se retrouver déformé ou mal affiché.
- Si la bordure en bas de fenêtre est différente de celle de gauche, la gestion de la souris peut s'en retrouver affectée.
- Si on lance le jeu via *Linux* en mode « low graphics », le jeu ne se lance pas.
- À de très rares occasions, une « ConcurrentModificationException» peut faire son apparition.

Recommandation pour l'exécution sur les PC de l'Escher

Le jeu est compatible avec *Linux* comme demander avec comme recommandation d'utilisation :

- Résolution inférieure ou égale à 800x640 (par défaut).
- Rafraichissement 30 IPS (par défaut).

Au-delà, des ralentissements et scintillements de l'écran peuvent apparaitre.

Droit d'auteur.

- Utilisation de pinceaux libres de droit créés par Marcianek issue de deviantart.com pour les astéroïdes. Disponible ici.
- Utilisation du drapeau belge pour l'illustration des astronautes.
- Utilisation du logo de l'UMONS sur le casque de l'astronaute.
- Les collisions entre astéroïdes sont inspirées du projet de Delplanque et Giamello.
- La rotation des mobiles lors de la pause est inspirée du projet de De Weireld et Potie.

Table des matières

Présentation **2**
Points forts 2
Point faible 2
Répartition des tâches. 3
Ducruet Corentin 3
Charlier Maximilien 3
Implémentation. 4
Représentation du vaisseau 4
Construction du vaisseau 4
Radar 5
Affichage principe de « couche » 5
Affichage et scintillement 6
Gestion des collisions 6
Collision entre le vaisseau et les astéroïdes ou astronautes *6*
Collision entre le vaisseau et les bonus 6
Collisions entre astéroïdes 7
Gestion de l'espace 7
Développement 7
Apport du projet. 8
Problèmes connus. 8
Recommandation pour l'exécution sur les PC de l'Escher 8
Droit d'auteur. 8