PROJET ASTEROID BELT

Introduction

Vous avez été fraîchement recrutés par la compagnie Gamelove en tant que programmeurs juniors pour travailler sur un projet dans le créneau du “jeu indé” d’inspiration rétro en style “pixel art”.

Vous avez été longuement briefé par le Lead Game Designer sur le contenu du jeu de ses rêves. Il a un nom : **Asteroid Belt** !

Toutefois comme il s’agit du premier projet de la société de ce type, il n’y a aucune base de code sur laquelle vous appuyer et **vous partez de zéro**.

Le lead développeur étant occupé à porter le moteur de jeu de l’entreprise sur OpenGL 1.0, il vous a confié le développement d’Asteroid Belt ainsi que tous les documents de design du projet (quel leadership !).

Avant de vous lancer tête baissée dans le code, vous devez analyser les exigences du game design et architecturer votre futur projet dans les règles de l’art…

Objectifs

Vous devez donc suivre l’architecture d’un jeu de type “**shoot them up**” avec un déroulement de type “**infinite runner**”. Les règles sont simples : faire en sorte que votre frêle vaisseau survive le plus longtemps possible au milieu d’une ceinture d’astéroïdes dont on ne voit pas la fin…

Il s’agit au premier abord d’un **jeu PC** (mais la production pense à en faire une version **mobile** plus tard, ou même **switch**, si le succès est au rendez-vous…). Après d’âpres débats, l’équipe a retenu le **C++** comme langage de programmation et la **SFML** comme framework choisi pour développer le jeu.

On a pour objectif d’avoir un **rendu 2D** de style pixel art (à débattre avec le lead artist), avec la possibilité de déclencher une grande variété d’**effets visuels** : transparence, particules, trails, animations de textures… Et pourquoi pas des postprocess caméras ou des shaders ?? (ne nous emballons pas.)

<u>Vous devez donc réaliser les tâches suivantes</u> :

- Versionner votre travail sur le dépôt Git de l'entreprise
- Produire un **document technique** écrit dans un français irréprochable (votre patron va le lire avec attention !), appelé **DocumentTechnique.docx**, publié sur le dépôt Git qui doit résumer les points suivants :
    - l'**étude technique** du projet - la SFML vous semble-t-elle adaptée pour ce projet ? Conseilleriez-vous un autre moteur à la place ? Justifiez.
    - l'**analyse logicielle**, qui montre votre réflexion sur l'architecture et sa modélisation à l'aide des diagrammes UML fournis. Décrivez dans les grandes lignes les propriétés de chaque objet (les structures de données choisies, leur complexité...), commentez les points forts et les points faibles de cette architecture.
    - présentez les différents **patterns logiciels utilisés dans ce jeu** qui permettent d'implémenter l'ensemble de manière propre et optimisée.

Ne négligez pas l'élégance et la qualité de votre code, l'équipe technique est très exigeante chez Gamelove :

- Autant de const que possible
- Aucune copie superflue (attention à vos & ...)
- Corollaire du point précédent : vous êtes encouragé à utiliser les features C++ telles que std::move quand c'est possible !
- Aucune action superflue de manière générale (ne recréez pas vos sprites à chaque frame, etc...)
- Une séparation claire entre les classes de "Moteur de jeu" (pouvant resservir pour n'importe quel autre jeu) et celles spécifiques à l'Asteroid Belt. Le bruit court déjà que le moteur sera réutilisé plus tard pour faire un autre jeu. On veut éviter de devoir tout réécrire.

Jeu

Il s'agit d'un jeu au **gameplay 2D**, vu de dessus, avec **scrolling vertical**.
Il s'agit d'un jeu pour **un joueur seulement**.
Le joueur devra survivre le plus longtemps possible au milieu d'astéroïdes qui seront spawnés depuis le haut de l'écran et descendrons vers le bas. Plus la partie durera longtemps, plus le nombre d'astéroïdes spawnés sera important.
Pour survivre, le joueur pourra soit

- **tirer** sur les astéroïdes afin de les détruire
- les **esquiver** en slalomant habilement
- **se protéger** en activant un bouclier temporaire

Les astéroïdes détruits pourront aléatoirement libérer des **bonus** qui, une fois absorbés par le vaisseau du joueur, vont incrémenter un **multiplicateur de score**.
Si le joueur est touché par un astéroïde, le multiplicateur retombe à x1.

Configuration

Pour commencer, les game designers ont bien précisé qu'ils voulaient pouvoir "tweaker" le jeu et ses paramètres. Et comme ce sont des designers, mieux vaut leur donner des outils simples à utiliser.

Vous implémenterez un "Config Reader" qui va venir lire ligne par ligne un fichier de configuration passé sur la ligne de commande. Le programme devrait être capable de détecter le premier mot de chaque ligne, et selon ce mot-clé, aller lire des arguments sur les lignes suivantes. Il devra gérer les types de valeur suivants :

- valeur string, comme "assets/sounds/blip.mp3"
- valeur float, comme "3.2"

Il s'arrête de lire les arguments pour un mot-clé quand il lit le mot "**end**".

Les 3C (Control, Camera, Character)

Contrôles

Le joueur se **déplacera** sur les axes X et Y, au clavier avec les touches **ZQSD** ou les **flèches.**

Il pourra **tirer** avec la barre d'espace. A chaque tir, un projectile sera lancé vers le haut de l'écran.

Il pourra **switcher entre deux états** à l'aide de la touche SHIFT.
Il pourra **activer son bouclier** avec la touche LEFT CTRL

Character - le vaisseau joueur

Le vaisseau aura une **jauge de vie**, qui lui permettra d'encaisser plusieurs impacts avec les astéroïdes.

Le vaisseau pourra tirer une **infinité de munitions**, à une **cadence** donnée, qui seront spawnées du canon et partirons selon une vitesse constante vers le haut de l'écran. Une munition qui entre en collision avec un astéroïde sera détruite.

Le vaisseau aura aussi la possibilité de **changer de mode** manuellement.
Il aura deux modes à sa disposition :

- un **mode heavy** (par défaut) avec vitesse de déplacement standard mais la possibilité de tirer avec une cadence soutenue.
- un **mode light**, avec vitesse de déplacement accrue mais une puissance de feu amoindrie (cadence de tir réduite).

Tous les **paramètres** du vaisseau (points de vies max, vitesses de déplacement, cadences de tir...), devront être **réglables** ultérieurement par les game designers.

En bonus : pour plus de liberté dans le gameplay, on pourrait faire en sorte que le joueur soit capable de tourner sur lui-même pour bouger et tirer dans toutes les directions. Mais par défaut, il ne sera pas capable de sortir des limites de l'écran (c'est un autre bonus).

Caméra

Il s'agit d'une simple **caméra en vue de dessus**.

Les astéroïdes

Ils auront un comportement très simple : ils seront spawnés en haut de l'écran et auront une trajectoire linéaire afin de traverser l'écran du haut vers le bas. On leur ajoutera un effet de rotation aléatoire sur la texture, pour leur donner plus de "vie", mais cela n'aura aucun impact sur les collisions. On pourra approximer leur **collision** avec un **cercle**.

Ils auront **3 tailles possibles** : petit, moyen et grand.
- Les petits seront détruits avec une seule munition.
- Les moyens seront détruits avec deux munitions.
- Les grands seront détruits avec quatre munitions.

Ils auront une **fréquence de spawn différente**, du plus courant au plus rare (réglable)
Ils seront **spawnés aléatoirement** sur la largeur de l'écran, mais de manière à ne pas les superposer.

Les grands astéroïdes pourront libérer, après destruction, un power-up aléatoire que le joueur pourra récupérer en passant dessus. On pense à au moins :
- un power-up multiplicateur qui incrémente le multiplicateur de score du joueur (qui peut aller de 1x à 5x)
- un power-up qui redonne de la vie au joueur.
- un power-up qui recharge le bouclier du joueur.

En bonus 1 : pour encore plus de challenge, on pourrait faire en sorte que les astéroïdes ne viennent pas que du haut de l'écran, mais peuvent aussi venir des côtés.

En bonus 2 : pour ENCORE plus de challenge, on pourrait imaginer que les astéroïdes collisionnent entre eux pour ricocher, se “casser” en plusieurs petits astéroïdes, etc...

Level

Outre le menu, il n’y aura qu’**un niveau de jeu principal**.

Le fond du décor sera constitué de **textures** pouvant être **animées** et **superposées** avec de la transparence. L’animation des objets de décor sera implémentée de façon à donner un effet “Scrolling parallax” (fausse impression de profondeur).

Les textures utilisées pourront évoluer en cours de partie pour plus de variété.

On aimerait qu’un signal quelconque (visuel, sonore...) avertisse le joueur lorsqu’il vient de passer à un niveau supérieur. Idéalement il faudrait **synchroniser** ce changement avec le changement de musique de fond.

Le décor de jeu sera deux à trois fois plus large que la partie visible à l’écran (réglable par le level design).

En bonus : on songe à implémenter un scrolling horizontal de l’écran (de gauche à droite) pour donner une plus grande impression de liberté de déplacement. La caméra devra constamment **suivre les déplacements** horizontaux du joueur, avec un léger retard sur le déplacement. On pense à un effet de “lerping” avec interpolation.
Si le joueur arrive au bord de la zone de jeu, son déplacement sera bloqué et il faudra attendre que la caméra le rattrape. Bien sûr, pour adapter le gameplay à ce nouveau scrolling horizontal, ce serait bien que le bonus 1 des astéroïdes soit implémenté aussi (sinon c’est trop facile !)

Scoring

Le **score va s’incrémenter automatiquement** avec le temps passé à survivre dans le jeu. Un **système de multiplicateur** allant de x1 à x5 sera ajouté, qui pourra être augmenté par la récupération de bonus. Si le joueur est touché par un asteroïd, le multiplicateur retombera à x1.
Le scoring de base sera réglé avec un nombre de points à ajouter par seconde (Ex : 100 pts par secondes). Le multiplicateur servira à augmenter le nombre de points engrangés. Par exemple avec un multiplicateur x3, le joueur gagnera 300 pts par seconde de jeu.

Lorsqu'un score va dépasser un des précédents scores enregistré dans un des mode de jeu (time attack, infinite...), il sera enregistré dans un **fichier de highscores**. L'enregistrement sera effectué au moment du game over.

On veut conserver les **dix meilleurs scores** pour chaque catégorie, avec la possibilité de lui associer le **pseudo** d'un joueur, stocké sous la forme de trois lettres. Ces scores seront consultables depuis un des écrans du menu.

En bonus : dans la grande tradition des shoot em up rétro, vous songez à implémenter un système de **scratching**, qui consiste à récompenser le joueur pour "flirter avec le danger" : un multiplicateur additionnel serait appliqué lorsque le joueur est en train de frôler un astéroïde et que le bouclier n'est pas actif !!

En bonus : implémentez un système de vies pour le joueur.

- Le joueur regagne une vie tous les 100,000 points par exemple (ce serait configurable)
- et tous les trois niveaux, un power-up rapide qui fait regagner une vie si on le touche apparaît

En bonus : implémentez plusieurs joueurs.

En bonus : implémentez la gestion des gamepads.

GUI

La GUI devra rester simple.

On trouvera un **système de menus** classique, avec

- Un écran d'introduction
- Un écran de sélection de mode de jeu
- Un écran d'options avec réglage des volumes sonores et mapping des inputs
- Un écran des high scores dans lequel on peut consulter les meilleurs scores effectués, par catégorie

Pour le **HUD** in-game, on affichera en GUI 2D fixe :

- le score du joueur
- le temps écoulé depuis le début de la partie (minutes : secondes)
- la jauge de bouclier, qui doit indiquer s'il est disponible, en action, ou en train de charger

Modes de jeu

Le joueur pourra choisir, à l'aide du menu, deux modes de jeu :

- “time attack” de 2, 5 ou 10 minutes
- “infinite run” qui durera jusqu’à ce que le joueur meurt

Sons

Des **effets sonores non spatialisés** seront déclenchés lorsque

- un tir est lancé par le joueur
- un tir touche un astéroïde
- un astéroïde est détruit
- un astéroïde touche le joueur
- un astéroïde touche le bouclier du joueur
- le bouclier est déclenché
- le bouclier a expiré
- le joueur switch de mode
- un bonus est ramassé par le joueur
- le multiplicateur retombe à zéro

A cela vont s’ajouter les **effets sonores du menu** (navigation, validation, retour en arrière...).

Le jeu sera aussi accompagné d’une **bande-sonore musicale** “pêchue” de type **synthwave**, qui cyclera parmis une liste de tracks que vous a fournie le lead audio designer. Ces tracks seront constitués de fichiers sonores compressés et lus en streaming, de type mp3, ogg ou autre.

Quelques **jingles musicaux** ou des voix enregistrées seront joués à différents moments du jeu

- Début de partie
- Game over
- lorsque qu’un high score a été dépassé