POLYTECH ORLÉANS
École d'Ingénieurs de l'Université d'Orléans

UNIVERSITE D'ORLEANS

PROJET INFORMATIQUE - S4 PEIP2

RAPPORT DE PROJET

REDACTEUR

Killian SINGLA

PeiP 2 - TP2

RESPONSABLE DU PROJET

M. Remy LECONGE

TUTEUR ACADEMIQUE

M. Remy LECONGE

2019 – 2020

REMERCIEMENTS

Je tiens tout d'abord à remercier Monsieur Remy LECONGE, mon tuteur académique durant ce projet informatique, également enseignant-chercheur en informatique à Polytech Orléans, pour son encadrement hors pair, sa disponibilité, sa bienveillance et son aide précieuse lors de la réalisation de ce projet. Je tiens également particulièrement à le remercier de m'avoir laissé le choix concernant le sujet dudit projet et pour son implication la plus totale lors de ces deux années à Polytech Orléans que ce soit pour les cours d'informatique ainsi que pour son travail de directeur des études.

Je tiens également à remercier tous mes professeurs d'informatique durant ce PeiP que sont Madame Kaouther TABIA, Monsieur Christophe LEGER, Monsieur Dian BAH, Monsieur Aladine CHETOUANI, ainsi que Madame Asma BOUGRINE. Leurs enseignements et leur envie de transmettre m'ont confortés dans l'idée que l'informatique est fondamentalement la discipline pour lequel j'éprouve un très grand attachement.

Je tiens également à remercier Monsieur Clément CAPART, game développeur et fondateur du studio indépendant de jeux-vidéo High Tea Frog , pour ses recommandations, sa bienveillance et ses conseils précieux lors de la réalisation de ce projet et durant ces deux années de cours en informatique.

Enfin je tiens également à remercier à la fois mes parents, ainsi que tous mes camarades de Polytech Orléans. Ceux-ci ont été à la fois d'un grand soutien durant ces années, mais également des conseillers particulièrement avisés durant la réalisation de ce projet.

A tous ces intervenants que j'ai eu la chance de côtoyer, que je continuerais de côtoyer à l'avenir et surtout à qui je dois beaucoup, je présente mes remerciements les plus sincères, mon respect ainsi que ma profonde gratitude. Je leur souhaite à tous le meilleur que cela soit d'un point de vue personnel et d'un point de vue professionnel.

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

INTRODUCTION

Asteroids est un jeu Américain édité et développé par Atari Inc. sorti en 1979 sur bornes d'arcade. A l'instar de son collègue *Space Invaders* sorti un an plus tôt, *Asteroids* est ce qu'on appelle en jargon jeu vidéoludique un « shoot'em up ». Il s'agit d'un jeu vidéo dans lequel le joueur dirige un véhicule ou un personnage devant détruire un grand nombre d'ennemis à l'aide d'armes de plus en plus puissantes, au fur et à mesure des niveaux, tout en esquivant les projectiles adverses pour rester en vie ». La particularité de celui-ci réside dans le fait que les déplacements du joueur sont multidirectionnels contrairement à *Space Invaders* où seuls les déplacements horizontaux sont autorisés. A sa sortie, le succès est quasi phénoménal et représente à ce jour le plus grand succès commercial d'Atari.

De nos jours la création de jeux-vidéo n'a jamais été aussi simplifiée. On ne dénombre plus les aides et les outils quasiment gratuits à la disposition des autodidactes et développeurs confirmés pour développer des jeux-vidéo :

- **Les tutoriels/streams de développement**
  - *Chaîne YouTube de AskGamedev*
  - *Chaîne Twitch de SirLynixVanFrietjes*
  - *Chaîne Twitch de Aherys*
  - *etc.*
- **Les forums en ligne/sites en ligne**
  - *OpenClassRoom*
  - *UnityLearn*
  - *Scratch*
  - *etc.*
- **Les game engines**
  - *Unity, l'un des plus appréciés par ses nombreuses fonctionnalités et sa simplicité d'utilisation*
  - *Unreal Engine*
  - *CryEngine*
  - *etc.*
- **Les frameworks/libraries**
  - *MonoGame (framework C# utilisé pour réaliser ce projet)*
  - *Raylib (librarie écrite en C)*
  - *SDL (librarie)*

C'est ainsi que tout naturellement, en grand fan de jeux-vidéo et avec un certain attrait pour le pixel-art, les jeux 8-bit, et ce bien que très éloignés des générations de jeux avec lesquels je suis né et j'ai grandi, j'ai décidé de réaliser une version remasterisée et plus moderne du célèbre jeu *Asteroids* lors de ce projet informatique de PeiP2. Ce projet s'étale sur deux semaines complètes et a été réalisé grâce au framework C# MonoGame.

Ce rapport a pour but de décrire en détail le projet en question, d'énoncer et justifier les choix techniques mis en œuvre, donner le cahier des charges, d'énoncer son déroulé ainsi que ses calendriers (prévisionnel, réel), ainsi que fournir une analyse argumentée de celui-ci avec des pistes d'améliorations.

DEVELOPPEMENT

1.1 TRAVAIL PRELIMINAIRE ET CHOIX TECHNIQUES

1.1 Pourquoi *Asteroids* ?

Lorsque l'on réalise pour la première fois un projet, et ce pour pleins de disciplines techniques autres que l'informatique, on commence par refaire ce que nos prédécesseurs ont déjà fait avant nous. Cela fait office de base pour ensuite évoluer et réussir à faire de nouveaux projets bien plus compliqués et bien plus ambitieux.

Dans le cas des jeux-vidéo, le développement de jeux-vidéo d'arcade représente bien ces fameuses bases. Pourquoi les jeux-vidéo d'arcade ? L'explication est très simple : ceux-ci datent d'il y a 40 ans, à une époque où l'informatique n'en était qu'à ses débuts, et la documentation y est très riche. Ces dits types de jeux-vidéo nécessitent que peu de mémoire, ne sont pas très riches graphiquement (jeux principalement en 2D, 8-bit), et leur fonctionnement n'est pas très compliqué à mettre d'un point de vue technique.

Ainsi cela permet entre autres à des jeunes étudiants de mettre en pratique leurs connaissances dans un langage informatique sur un cas très concret et d'apprendre à résoudre des problèmes techniques en un temps quasi-record. Ces mêmes jeux d'arcade sont d'ailleurs souvent à réaliser lors de tests techniques dans de grands studios comme Ubisoft afin de recruter des programmeurs.

Ayant réalisé durant mon année de Terminale S un projet *Space Invaders* en Python avec la librarie TKinter (ToolKit interface), j'ai décidé durant ce projet de 2ème année de PeiP à Polytech Orléans, de poursuivre sur cette lancée et de réaliser un projet plus ambitieux et ce en temps bien plus réduit. Après avoir joué à plus d'une dizaine de jeux arcade disponibles sur Internet, mon choix s'est porté sur *Asteroids*. Son style graphique minimaliste avec ses graphismes vectoriels noir et blanc ainsi que son gameplay assez novateur m'ont tout de suite séduit.

Le principe du jeu est très simple : le joueur contrôle un vaisseau spatial qui peut tourner sur lui-même (à gauche et à droite) et voler sur la map. Le but est de tenir le plus possible face à l'abondance des astéroïdes et les soucoupes volantes qui lui arrivent dessus à pleine vitesse. Pour cela, il a le choix de les éviter en usant de ses déplacements et du concept innovant de wraparound (quitter la map par le haut nous téléporte immédiatement sur la partie basse de celle-ci, de même pour les côtés gauche et droit, ce qui est très utile lorsque nous n'avons pas une grande marge de manœuvre) ou de les détruire grâce à des tirs. Le score du joueur peut ainsi y être référencé avec ses initiales et ensuite recommencer le jeu autant qu'il le souhaite afin d'obtenir le score le plus haut possible.

Figure 1 : Logo du jeu original Asteroids

1.2 Choix techniques concernant le développement du jeu

Le temps alloué au développement du projet est somme toute réduit. Ce temps est de 14 jours à temps complet pour réaliser un programme informatique fonctionnel, un poster A0 et ce dit rapport. Cela impose ainsi des choix techniques pour la réalisation de la partie informatique qui représente à elle seule, 2/3 du déroulé total du projet :

- Résolution d'écran fixe : chaque logiciel s'adapte plus ou moins en fonction de l'écran de l'utilisateur, et c'est évidemment bien plus le cas concernant les jeux-vidéo afin de profiter d'une expérience la plus optimale possible. Afin de ne pas trop s'encombrer de contraintes techniques concernant le positionnement et l'échelle des éléments graphiques, les collisions, la taille des polices d'écriture utilisées, la résolution sera ici fixe. **Mon choix s'est porté sur la résolution 1080p soit 1920x1080 pixels qui offre une bonne qualité visuelle au jeu**. Ce choix n'est pas anodin car il s'agit ici, selon la plateforme Steam, de la résolution la plus massivement utilisée sur la plateforme (voir figure). Même au-delà des jeux-vidéo, on estime que la résolution d'écran la plus utilisée en avril 2020 en France est de 1920x1080 par 20,74% des utilisateurs. Mon ordinateur étant un Surface Book 2 avec une résolution de 3000x2000 pixels, cela me permettra de travailler avec une représentation somme toute assez fidèle du produit final.

- Utilisation d'un framework C# : le développement de jeux-vidéo est désormais de plus en plus simplifié au fil des années. On ne dénombre plus le nombre de game engines et autres frameworks/libraries disponibles gratuitement pour développer facilement. Unity est un game engine particulièrement utilisé dans ce type de projet informatique étant donné sa simplicité, son ergonomie et ses fonctionnalités particulièrement puissantes. Cependant ce projet est avant tout un projet de programmation et le peu de scripts à réaliser sur Unity qui jalonnent un tel jeu ne permettent pas de mettre totalement en pratique ses connaissances en programmation (ici le C#). Un langage qui a ici été préféré face au C++ pour sa gestion dynamique de la mémoire et son orientation beaucoup plus POO**. Afin de trouver un bon compromis entre un game engine complet, et une librarie classique, mon choix s'est porté sur un framework. Le framework contrairement à la librarie permet au codeur de donner certain un style d'architecture, un squelette à son programme.** Cela est plus avantageux lors de la réalisation d'un projet aussi complexe qu'un jeu. **Mon choix s'est porté sur le MonoGame C#, un framework très connu du fait de sa simplicité d'utilisation et ses fonctionnalités complètes.**

- Plateforme de développement « universelle » : lors de la création d'un jeu sur MonoGame, le type de projet désiré est obligatoirement à indiquer. Plusieurs choix s'offrent à nous : Windows 10 Universal project, Windows project, Android project, iOS project, ou encore Cross-platform Desktop project (Linux, MacOS**). Ici mon choix s'est porté sur Windows project afin de créer un jeu qui soit fonctionnel sur la plupart des versions courantes de Windows (notamment Windows 7, Windows 8/8.1 et Windows 10).** Ce choix a bien été fortement influencé par les parts de marché que possède Windows dans le monde des systèmes d'exploitation sur PC. 90% des ordinateurs dans le monde possèdent Windows en guise d'OS contre 8% pour MacOS et 2% pour Linux.

1.3 L'histoire et le fonctionnement du framework C# MonoGame

Afin d'accompagner le lancement de sa nouvelle console et aider les programmeurs dans le développement de leurs jeux-vidéo sur les différentes plateformes de l'entreprise (Windows NT, Windows Phone, Xbox 360…), Microsoft lance une série d'outils gratuits nommée XNA (pour Xbox Next-Generation Architecture, bien que cela ne soit pas clairement défini). Cette série d'outils est composée entre-autres d'un framework nommé XNA Framework, d'un IDE (integrated development environment) nommé XNA Game Studio, et d'un logiciel permettant la définition, le débogage et l'optimisation des différents assets du jeu nommé XNA Build.

Après plusieurs années de loyaux services et des milliers de jeux indépendants développés grâce à ces outils, Microsoft décide en 2011 de changer de stratégie et d'abandonner XNA.

C'est ainsi qu'est né le framework C# MonoGame qui est une implémentation OpenSource de XNA 4.0. Celui-ci est désormais mis à jour au fur et à mesure et permet le développement de jeux-vidéo sur les plateformes classiques (Windows, macOS, Android…) et les plateformes de jeux-vidéo de dernières générations comme la Xbox One, la PS4 ou la Nintendo Switch.

MonoGame est une API multi-plateforme permettant la gestion et l'affichage de graphismes, la lecture de sons, la gestion d'évènements dans le déroulé du jeu, la gestion des entrées et fournissant un logiciel de pipeline de contenu pour l'importation d'assets. Celui-ci utilise soit la librarie OpenGL soit Microsoft DirectX en fonction de la plateforme de développement choisie (OpenGL pour macOS/Linux, DirectX pour Windows). Contrairement à la plupart des moteurs de jeu classiques, MonoGame ne fournit ni n'impose aucun véritable modèle ou structure de projet en particulier. Cela signifie que les développeurs sont libres d'organiser leur code comme ils le souhaitent. Cela signifie également que peu de lignes de code de configuration sont nécessaires lors du démarrage d'un nouveau projet. Lors de la création d'un projet, quelques lignes de code sont préalablement tapées, et les structures des fonctions principales du programme sont à remplir (Initialize, LoadContent, Update, Draw, UnloadContent).

Le cycle de vie du fonctionnement d'un jeu sur MonoGame est le suivant :

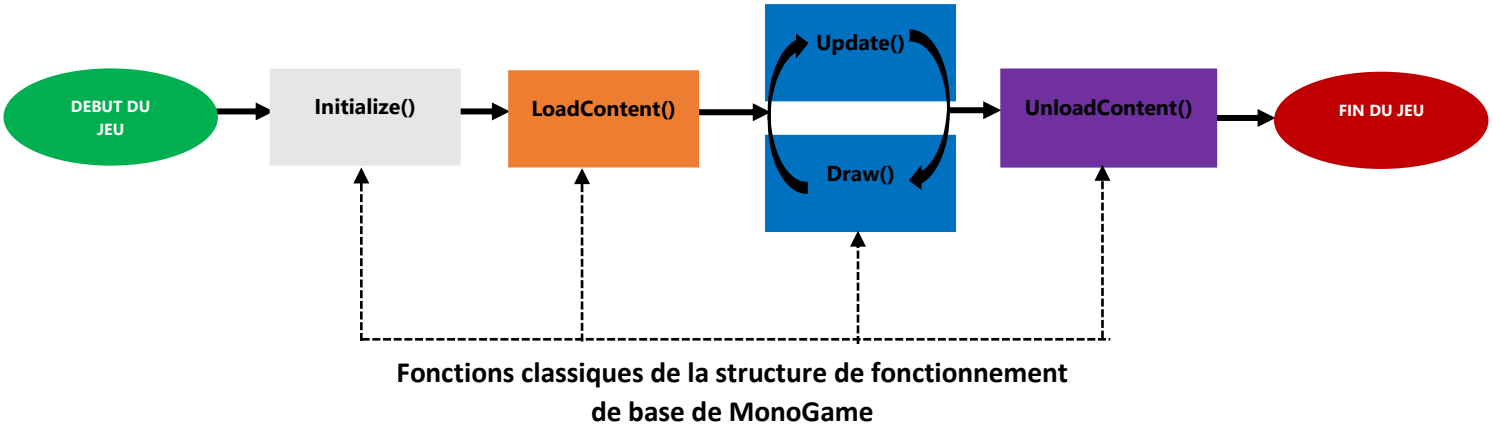

Figure 2 : Cycle de vie du fonctionnement d'un jeu sur MonoGame

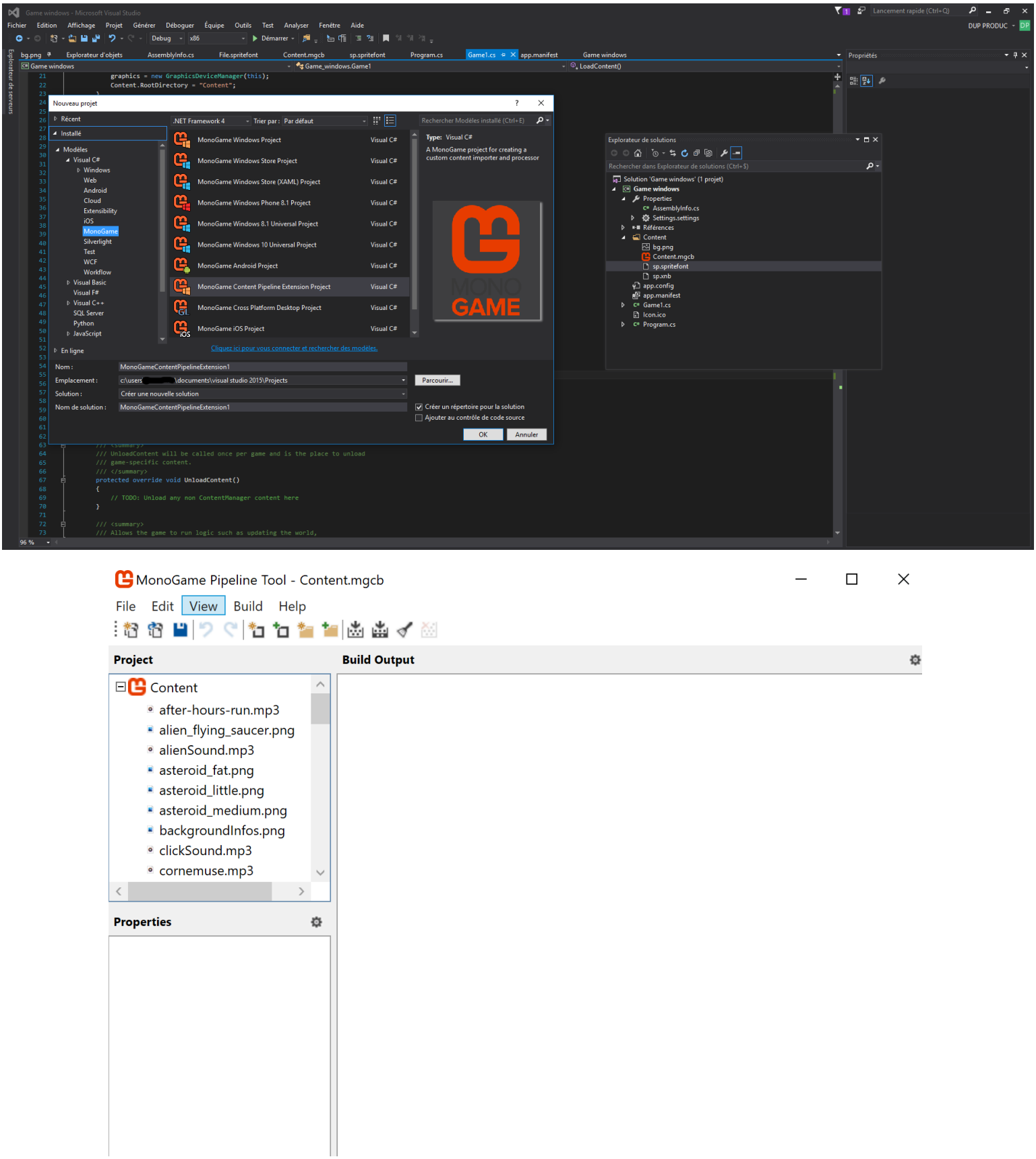

Figure 3 : Interface de MonoGame directement intégrée à Visual Studio et son pipeline tool pour importer et build différents assets du jeu

1.4 Cahier des charges

FICHE DU PRODUIT

- **Nom du jeu :** *Goddamn ASTEROIDS& ALIENS ! REMASTERED*
- **Genre :** Shoot'em up
- **Plate-forme :** Windows
- **Moteur :** MonoGame C# Framework / XNA
- **Joueur(s) :** 1
- **Pegi :** 3+

- **Créer un menu avec les éléments/fonctionnalités suivantes :**
  1. Présence d'un logo
  2. Présence des informations relatives au jeu (crédits, année de développement, remerciements etc.)
  3. Jouer une musique en fond sonore
  4. Permettre de couper/découper le son de la musique
  5. Permettre au joueur d'afficher/ou non une scène permettant le choix de playlists musicales
  6. Permettre au joueur d'afficher/de fermer s'il le désire une scène d'information concernant les commandes du jeu ainsi que son fonctionnement
- **Créer une scène permettant d'informer le joueur sur les commandes du jeu, ainsi que son fonctionnement**
  1. Présence d'un background
  2. Présence d'images
  3. Présences d'informations

- **Créer une scène permettant au joueur de jouer avec de la musique**
  1. Permettre le choix de playlist musical
  2. Permettre la création de ses propres playlists musicales
  3. Permettre l'adaptation du volume en fonction de son choix (80% du volume, 50% du volume ou 25% du volume)
  4. Permettre au joueur de lancer le jeu
- **Créer une scène de zone de jeu avec les éléments/fonctionnalités suivantes :**
  1. Présence d'un background
  2. Présence du joueur
  3. Présence d'un HUD (score, vies du joueur)
  4. Rotation du joueur vers la gauche ou vers la droite
  5. Déplacement du joueur en tenant compte de sa direction
  6. Permettre au joueur de tirer en tenant compte de sa direction
  7. Jouer des sons en fonction des différentes actions de la partie
  8. Jouer la playlist musicale choisie préalablement par le joueur
  9. Génération aléatoire d'astéroïdes de tailles différentes avec des trajectoires aléatoires
  10. Génération d'aliens toutes les X secondes (à définir) avec des trajectoires aléatoires
  11. Génération de tirs aléatoires (aliens)
  12. Gérer les collisions tirs/astéroïdes, tirs/aliens, tirs aliens/joueur
  13. Destruction des astéroïdes
  14. Destruction des aliens

- **Créer une scène de fin de jeu avec les éléments/fonctionnalités suivantes :**
    1. Présence d'un background
    2. Présence de textes d'informations
    3. Jouer une musique en fond sonore
    4. Affichage du score du joueur
    5. Permettre au joueur de rejouer
    6. Enregistrer automatiquement le score du joueur dans un fichier
    7. Permettre au joueur d'afficher le score de ses anciennes parties et de fermer le tableau de scores

1.5 Modification/réalisation d'assets graphiques et maquettes graphiques

Afin d'avoir une idée précise du rendu du jeu avant la partie développement, des maquettes graphiques ont été réalisées. Comme une maquette graphique du menu du jeu et de la map du jeu avec ses différents éléments.

Ne voulant pas trop abuser d'assets graphiques disponibles sur Internet et voulant faire en sorte que le jeu soit le plus personnel possible, mon travail a également consisté à créer, en parallèle du projet, des textures 2D de toutes pièces ou à en modifier certaines déjà existantes à ma convenance.

Ces textures, ainsi que ces maquettes graphiques ont été réalisées et modifiées via le logiciel PhotoFiltre Studio X.

Préféré dans le cadre de ce projet du fait de sa simplicité d'utilisation et sa rapidité d'exécution vis-à-vis de ses concurrents certes bien plus complets comme GIMP ou Adobe PhotoShop. Le but est ici de modifier simplement des éléments graphiques et d'en réaliser certains d'une manière efficace sans empiéter sur la programmation du jeu qui est bien plus chronophage.

Figure 4 : Logo du jeu

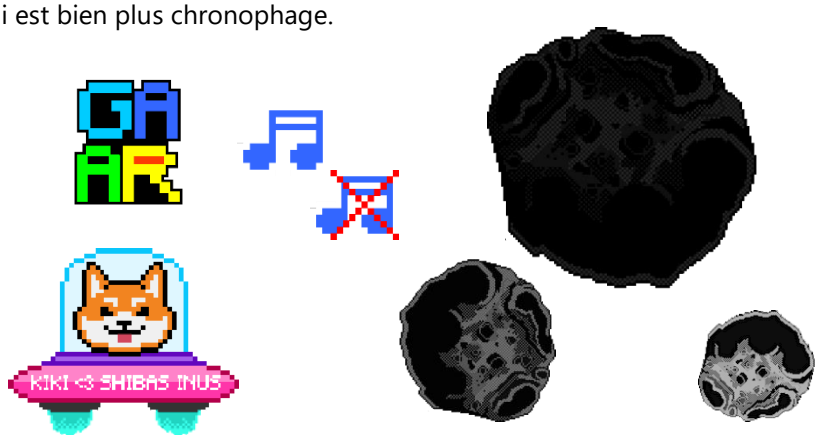

Figure 5 : Différents assets graphiques du jeu réalisés par mes soins

Figure 6 : Maquette graphique du menu du jeu

1.6 Calendrier prévisionnel des tâches à réaliser

M : 9h30-12h30 / AM : 14h-18h30

Tâches à effectuer	18-mai-20		19-mai-20		20-mai-20		21-mai-20		22-mai-20		23-mai-20		24-mai-20		25-mai-20		26-mai-20		27-mai-20		28-mai-20		29-mai-20		AVANCEMENT AVEC COMMENTAIRE(S)		
	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	OK	EN COURS/A PEAUFINER	A FAIRE
Réaliser un rapport de projet		X																	X	X	X	X	X	X			
Réaliser un poster format A0 du projet		X																	X	X	X	X	X	X			
Classe GameMenu - Mise en place des éléments graphiques														X													
Classe GameMenu - Gestion évènements souris (boutons)														X													
Classe GameMenu - Animation (mouvement) de certains éléments															X												
Classe GameMenu - Permettre la lecture/la mise en pause de la musique de fond														X													
Classe PlaylistMenu -> Classe Playlist - Permettre le choix ou non d'une playlist musicale et de couper ou non le son du jeu (bruitages...), lire cette playlist aléatoirement et gérer son volume par rapport à celui du jeu																	X	X									
Classe PlaylistMenu -> Classe Paylist - Permettre au joueur de créer sa propre playlist avec des titres stockés sur son ordinateur et gérer son volume par rapport à celui du jeu																	X	X									
Classe LobbyMenu : Indiquer les commandes jouables et faire un compte à rebours															X												
Classe GameMap : Mettre en place les éléments graphiques de la map		X																									
Classe GameMap -> Classe Player : Mise en place du sprite		X																									
Classe GameMap -> Classe Player : Gestion des évènements claviers			X																								
Classe GameMap -> Classe Player : Gérer le déplacement du joueur			X																								
Classe GameMap -> Classe Player : Gérer les tirs du joueur et leurs collisions			X	X	X																						
Classe GameMap -> Classe Player : Gérer la jauge de vie du joueur et son respawn (collisions)						X				X																	
Classe GameMap -> Classe Player : Jouer des sons à chaque action du joueur (tir, déplacement, destruction)																X											
Classe GameMap -> Classe Player : Jouer des animations à chaque action du joueur (tir, déplacement)																			X	X							
Classe GameMap -> Classe Asteroids : Mise en place des différents sprites					X																						
Classe GameMap -> Classe Asteroids : Gérer le spawn aléatoire des différents types d'astéroides (3-4 types)						X																					
Classe GameMap -> Classe Asteroids : Gérer les trajectoires aléatoires des différents astéroides					X																						
Classe GameMap -> Classe Asteroids : Gérer la division des astéroides en d'autres astéroides plus petits lors qu'ils sont touchés						X	X																				
Classe GameMap -> Classe Asteroids : Gérer la destruction des astéroides (collisions)							X																				
Classe GameMap -> Classe Asteroids : Jouer des sons (destruction, déplacement...)																X											
Classe GameMap -> Class Asteroids : Jouer des animations (destruction, dépacement...)																			X	X							
Classe GameMap -> Classe Aliens : Mise en place des sprites							X																				
Classe GameMap -> Classe Aliens : Gérer le spawn aléatoire des aliens								X																			
Classe GameMap -> Classe Aliens : Gérer les trajectoires aléatoires des aliens								X																			
Classe GameMap -> Classe Aliens : Gérer les tirs des aliens et leurs collisions								X	X																		
Classe GameMap -> Classe Aliens : Gérer le spawn aléatoire des aliens									X																		
Classe GameMap -> Classe Aliens : Jouer des sons (destruction, déplacement, tir...)																X											
Classe GameMap -> Classe Aliens : Jouer des animations (destruction, dépacement, tir...)																			X	X							
Classe GameMap -> Classe HUD : Afficher le score, ainsi que les vies restantes du joueur											X																
Classe EndGame -> Classe EndScore : Récapituler le score du joueur atteint et le rediriger vers une "fenêtre surprise" en fonction de son score												X															
Classe EndGame -> Classe ScoreRecap : Permettre l'affichage et l'enregistrement des scores du joueur à chaque partie												X	X														
Fin de projet : tester, régler certains paramètres, et corriger les bugs																							X	X	X		

Figure 7 : Calendrier prévisionnel

2. RAPPORT TECHNIQUE DU PROJET INFORMATIQUE

2.1 La structure POO du projet GAAR

L'outil principal de la POO réside dans l'utilisation et la création de classe.

Le C# qui est le langage utilisé dans ce projet est un langage de programmation orienté objet (il a été conçu pour, et chaque programme créé en C# est lui-même une classe). La POO (pour programming oriented objects) est massivement utilisée dans les jeux-vidéo. Celle-ci rend la réalisation d'un jeu bien plus simple.

Tout d'abord, il est primordial de rappeler quelle est la structure de code de base de MonoGame. Celle-ci n'est pas très compliquée à appréhender et son principe est quasi instinctif. **Lors de la création d'un projet monogame, un premier code Program.cs est généré. Celui-ci est une classe basique et instancie, à travers une fonction principale Main(), une classe Game1 qui est ensuite lancée. Il s'agit du code principal de notre jeu.**

```
using System;
namespace GAAR_Game
{
#if WINDOWS || LINUX
    public static class Program
    {
        /// The main entry point for the application.
        [STAThread]
        static void Main()
        {
            using (var game = new Game1())
                game.Run();
        }
    }
#endif
}
```

Un code Game1.cs est donc généré [appelé Main.cs dans le projet]. Celui est également une classe dérivée d'une classe nommée Game. Le constructeur de Game1.cs est le suivant :

```
public Game1()
 {
  graphics = new GraphicsDeviceManager(this);
  graphics.PreferredBackBufferWidth = 1920; // Pour modifier la taille
  graphics.PreferredBackBufferHeight = 1080; de la fenêtre
  Content.RootDirectory = "Content"; // Pour indiquer le répertoire où se trouve les
différents fichiers du jeu (hormis le code).
 }
```

Le corps des fonctions sont écrits dans Game1 et leurs appels se trouvent dans Game (il faut presser F12 pour aller voir le code de Game). Voici la classe Game :

```
public class Game : IDisposable
    {
        public Game();
        ~Game();
        public GameComponentCollection Components { get; }
        public TimeSpan InactiveSleepTime { get; set; }
        public TimeSpan MaxElapsedTime { get; set; }
        public bool IsActive { get; }
        public bool IsMouseVisible { get; set; }
        public TimeSpan TargetElapsedTime { get; set; }
        public bool IsFixedTimeStep { get; set; }
```

```
        public GameServiceContainer Services { get; }
        public ContentManager Content { get; set; }
        public GraphicsDevice GraphicsDevice { get; }
        [CLSCompliant(false)]
        public GameWindow Window { get; }
        public LaunchParameters LaunchParameters { get; }

        public event EventHandler<EventArgs> Activated;
        public event EventHandler<EventArgs> Deactivated;
        public event EventHandler<EventArgs> Disposed;
        public event EventHandler<EventArgs> Exiting;

        public void Dispose();
        public void Exit();
        public void ResetElapsedTime();
        public void Run();
        public void Run(GameRunBehavior runBehavior);
        public void RunOneFrame();
        public void SuppressDraw();
        public void Tick();
        protected virtual bool BeginDraw();
        protected virtual void BeginRun();
        protected virtual void Dispose(bool disposing);
        protected virtual void Draw(GameTime gameTime);
        protected virtual void EndDraw();
        protected virtual void EndRun();
        protected virtual void Initialize();
        protected virtual void LoadContent();
        protected virtual void OnActivated(object sender, EventArgs args);
        protected virtual void OnDeactivated(object sender, EventArgs args);
        protected virtual void OnExiting(object sender, EventArgs args);
        protected virtual void UnloadContent();
        protected virtual void Update(GameTime gameTime);
    }
```

Le préfixe override devant la définition des fonctions dans Game1.cs permet d'étendre la méthode virtual de la classe mère. A chaque lancement du jeu, celles-ci sont appelées. Les fonctions sont les suivantes :

`public override void Initialize()` -> Initialize permet au jeu d'initialiser tout ce dont il a besoin avant de se lancer. C'est ici qu'il peut interroger tous les services requis et charger tous les éléments non graphique. Appeler base.Initialize énumèrera tous les composants et les initialisera également.

`public override void LoadContent()` -> LoadContent est appelée une seule fois par lancement. Celle-ci se charge de charger tous les éléments de notre jeu (sons, textures, font…). C'est également ici qu'on charge un objet de type SpriteBatch qui servira au rendu des différents objets sur l'écran.

`public override void Update(GameTime gameTime)` -> Update s'appelle à chaque frame. Elle permet au jeu de s'actualiser afin de détecter des collisions, jouer des sons, gérer des entrées (comme au clavier par exemple). Le paramètre gameTime fournit un instantané de l'état de synchronisation du jeu.

`public override void Draw(GameTime gameTime)` -> Draw permet au jeu de dessiner les objets à l'écran. Comme sa collègue Update, elle possède un paramètre gameTime, celui qui lui permet d'agir en fonction de l'actualisation du jeu. (Par exemple, si un ennemi spawn sur la map, celle-ci le dessine). Draw et Update travaille ensemble en continue.

`public override void UnloadContent()` -> Contrairement à LoadContent, UnloadContent permet de décharger des éléments.

Dans ce projet, on ne cesse d'exploiter des classes, l'héritage et le polymorphisme. Le but est de créer des classes filles qui vont hériter des éléments principaux du jeu (les fonctions précédentes) qui sont dans Game. Ces éléments principaux sont hérités dans le programme Game1 (appelé Main dans mon jeu), qui seront ensuite hérités par une classe abstract nommé Scene (voire la suite) et qui seront encore hérités par des classes filles (comme MainMenu, GamePlay, EndGame) qui dérive de la classe Scene.

Nous allons également faire appel à d'autres classes afin de gérer plus efficacement les évènements du jeu (une classe Player, une classe Asteroid, une classe Alien…). Voici un schéma de la structure POO du projet :

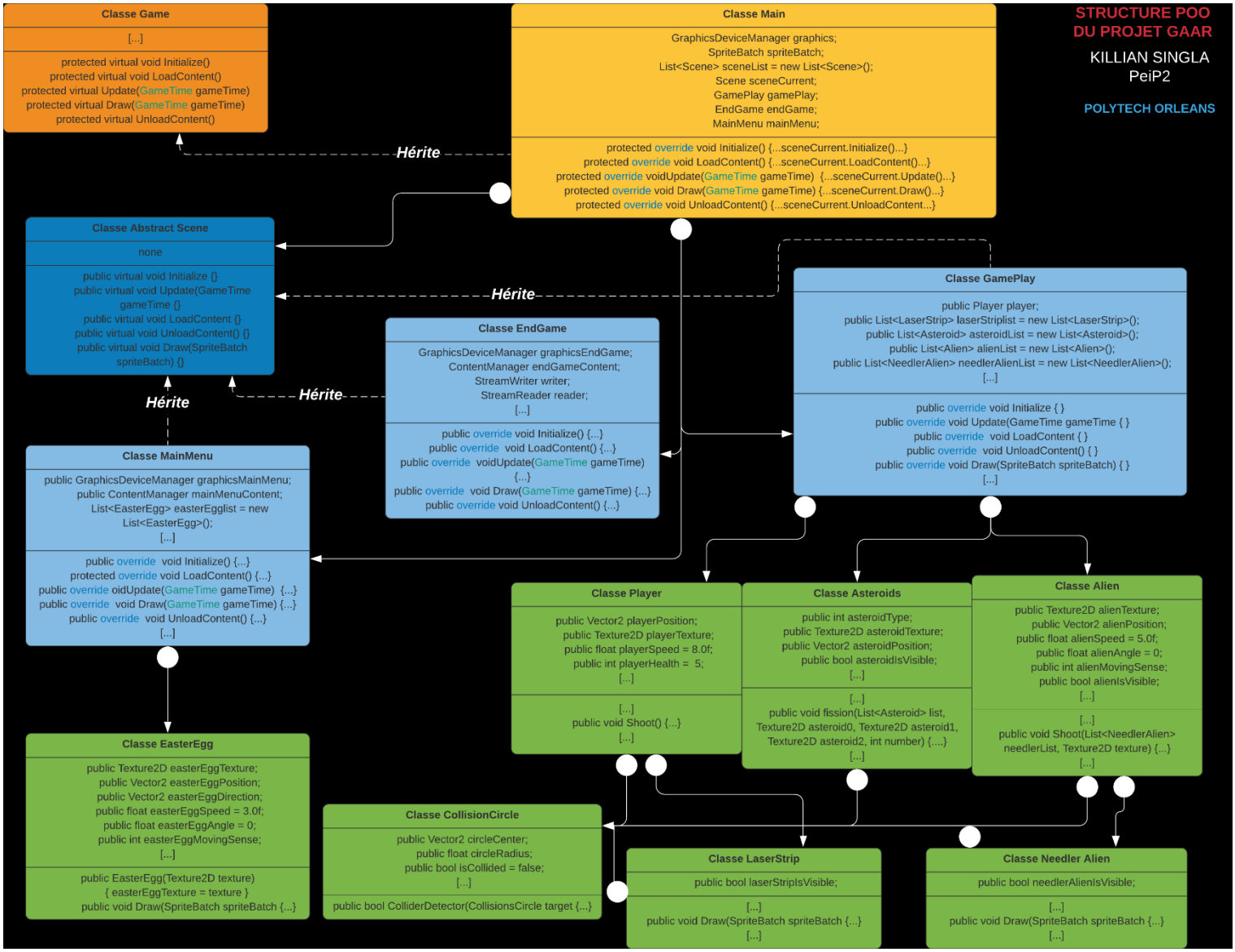

Figure 8 : Schéma de la structure POO du projet

Toute la structure du projet et son fonctionnement sont expliqués plus en détail dans les parties suivantes).

2.2 Travail réalisé et explication du fonctionnement des éléments principaux

Le développement d'un programme informatique consiste à transposer les éléments du cahier des charges en codes qui permet de réaliser les fonctionnalités et d'afficher les éléments désirés. C'est justement le but de cette partie, expliquer concrètement comment quelques fonctionnalités classiques du jeu ont été réalisées. **/!\ Cette partie est avant tout explicative. Les bouts de codes présents dans cette partie ne représentent que certaines parties du code total du jeu, certaines fonctions et classes n'apparaissent pas.**

- **Afficher un élément (ici le joueur) et permettre sa rotation en fonction des touches du clavier**

Pour afficher un élément à l'écran, il faut tout d'abord lui attribuer une image (ici une texture). Il s'agit ici d'un jeu en 2D, donc il faut lui attribuer une Texture2D. Cette texture 2D est à placer dans le dossier Content (qui contient TOUS les éléments du jeu, comme les images, les sons hormis la partie code). Puis ouvrir le MonoGame Pipeline tool, charger les items en question, et appuyer sur build. Cela permet d'importer nos éléments (police fonts, textures 2D, sons, musiques, effets spéciaux…) et de les transformer en assets directement exploitables dans le jeu (format XNB pour XNA Game Studio Binary Package). Si cette étape n'est pas réalisée, MonoGame refusera d'exploiter les éléments désirés et ce même si ceux-ci sont contenus dans le dossier Content. Les fichiers chagés

Puis, une fois notre texture 2D chargée il faut initialiser une variable du même type, et la charger dans la fonction LoadContent en tenant compte du fait que LoadContent nécessite également un spriteBatch (voir 2.1).

```
public Texture2D playerTexture;

protected override void LoadContent ()
{
	spriteBatch = new SpriteBatch(GraphicsDevice); // Comme indiqué précédemment, on
	initialise également un objet de type spriteBatch pour pouvoir dessiner.
	playerTexture = Content.Load('texture_de_mon_joueur'); //On récupère et charge la
	texture du joueur à partir du dossier Content
}
```

L'image est chargée, il faut désormais lui donner une position, un angle et une origine pour qu'on puisse ensuite la dessiner grâce à la fonction Initialize puis la fonction Draw.

```
public Texture2D playerTexture;

public Vector2 playerPosition;

public Vector2 playerOrigin;

public float playerAngle

protected override void Initialize ()
{
	playerOrigin = new Vector2(playerTexture.Width/2, playerTexture.Height/2); //
	L'origine d'une image c'est son centre. Ainsi sa hauteur/2 et sa largeur /2
	playerPosition = new Vector2(graphicsGamePlay.PreferredBackBufferWidth / 2,
	graphicsGamePlay.PreferredBackBufferHeight/2); // Dans MonoGame, la référence
	pour toutes les positions est le coin supérieur gauche. Ainsi, pour avoir un
	objet centré, il faut diviser la largeur par 2,
	playerAngle = 0;
	base.Initialize();
}
protected override void Draw(GameTime gameTime)
```

```
{
        GraphicsDevice.Clear(Color.Black); //Donne une couleur de Background
        spriteBatch.Begin(); //On commence à afficher des éléments graphiques…
        Rectangle playerRectangleTexture = new Rectangle(0, 0, playerTexture.Width,
        playerTexture.Height); //Tout d'abord on créé un "Rectangle d'affichage". Ce
        rectangle permet d'indiquer à MonoGame quelle partie de la texture est à afficher.
        Ici je rappelle que la référence est le coin supérieur gauche, ainsi (0,0).
        spriteBatch.Draw(playerTexture,      playerPosition,      playerRectangleTexture,
        Color.White, playerAngle, playerOrigin, 1.0f, SpriteEffects.None, 1);
        //Texture, position de la texture, rectangle, Couleur (Afficher avec une lumière
        blanche, angle,origine, facteur d'échelle, ici 1, pas d'effets de sprite,
        profondeur = 1. Le fait d'indiquer l'origine permettra de tourner autour de celle-
        ci.
        spriteBatch.End(); //…et on termine de dessiner
        base.Draw(gameTime);
}
```

L'image est chargée et affichée au milieu de l'écran. Désormais on veut la faire tourner dans un sens, et faire en sorte que sa direction change en fonction de l'angle (sans quoi, malgré la rotation, le déplacement sera le même). On se dirige ainsi vers Update (car l'évolution de l'angle et de la direction doit se faire à chaque image).

```
public Texture2D playerTexture;

public Vector2 playerPosition;

public Vector2 playerOrigin;

public Vector2 playerDirection;

public float playerAngle

public override void Update(GameTime gameTime)
{
        KeyboardState state = Keyboard.GetState(); // On récupère ce que reçoit le clavier
        en entrée

        if (state.IsKeyDown(Keys.Left)) //Si touche gauche appuyée…
        {
          player.rotateLeft(); //…alors on tourne vers la gauche.
        }
}

public void rotateLeft()
{
        playerAngle = playerAngle - 0.06f;
        playerDirection = new Vector2((float)Math.Sin(playerAngle),
(float)Math.Cos(playerAngle));
// Pour obtenir la direction, on utilise la trigonométrie. Or, ici, notre vaisseau pointe
vers le haut lorsque son angle est à 0. Ainsi, si on voulait se déplacer (on rappelle
que dans Asteroids, seul la flèche du haut permet de se déplacer dans la direction que
possède le vaisseau en fonction de son angle), on irait vers le haut, soit vers les Y
décroissant. playerPosition.X += playerSpeed * playerDirection.X et de même pour Y. Il
faut donc que la composante selon Y pour la direction soit à 1, or Sin(0) = 0 mais Cos(0)
= 1, ainsi on met Sin pour la composante en X, et Cos pour la composante en Y.
}
```

- **Générer aléatoirement différents types d'astéroïdes avec des spawns et des trajectoires différentes toutes les X secondes**

Pour cela on commence à créer une classe spéciale Astéroide avec toutes les fonctions classiques.

```
public class Asteroid
    {
        public int asteroidType;
        public Texture2D asteroidTexture;
        public Vector2 asteroidPosition;
        public Vector2 asteroidOrigin;
        public Vector2 asteroidDirection;

        public Rectangle asteroidTextureRectangle;
        public CollisionsCircle asteroidColisionsCircle;

        public float asteroidAngle;
        public float asteroidSpeed;
        public int asteroidcoefTrajectory0;
        public int asteroidcoefTrajectory1;
        public int asteroidMovingSense;

        public bool asteroidIsVisible;

        public Asteroid(int type, Texture2D texture0, Texture2D texture1, Texture2D
texture2)
        {
            asteroidType = type;

            if (type == 0) // En fonction du type renseigné lors de la création d'un
astéroide, celui-ci prend une texture différente.
            {
                asteroidTexture = texture0; //Petit astéroïde
                asteroidSpeed = 8.0f; //Grande vitesse
            }

            if (type == 1)
            {
                asteroidTexture = texture1; //Astéroïde de taille moyenne
                asteroidSpeed = 5.0f; //Vitesse moyenne
            }

            if (type == 2)
            {
                asteroidTexture = texture2; //Astéroïde de grande taille
                asteroidSpeed = 3.0f; //Petite vitesse
            }

            asteroidIsVisible = true;
        }
       public void Draw(SpriteBatch spriteBatch)
        {
            Rectangle asteroidRectangleTexture = new Rectangle(0, 0,
            asteroidTexture.Width, asteroidTexture.Height);
            spriteBatch.Draw(asteroidTexture, asteroidPosition,
            asteroidRectangleTexture, Color.White, asteroidAngle, asteroidOrigin,
            1.0f, SpriteEffects.None, 1);
        }
    }
```

Puis dans la fonction Update de notre programme principal, on fait en sorte de générer à chaque instant t des astéroïdes. Chaque astéroïde généré se trouvera dans une liste d'astéroïdes.

```
double timePassedGenerationAsteroids:

public List<Asteroid> asteroidList = new List<Asteroid>();
public override void Update(GameTime gameTime)
{

timePassedGenerationAsteroids = timePassedGenerationAsteroids +
gameTime.ElapsedGameTime.TotalSeconds; // Grâce à
gameTime.ElapsedGameTime.TotalSeconds on récupère le temps passé depuis la dernière
Update du jeu et on le convertit en secondes (double).
            if (timePassedGenerationAsteroids > 3) //Dès que timePassed dépasse 3, on
génère 7 astéroides de types, de spawn et de trajectoires aléatoires.
            {
                timePassedGenerationAsteroids = 0; //On réinitialise
                for (int i = 0; i < 7; i++)
                {
                    GenerateRandomAsteroid();
                }
            }

}

public void GenerateRandomAsteroid()
        {
            int random_value_texture = randomNumber.Next(0, 3); //Un nombre aléatoire
entre 0 et 3 (3 car sinon 2 n'est jamais prise) pour la texture.
            float random_value_angle = randomNumber.Next(-3, 3); //Pareil pour l'angle

            Asteroid asteroid = new Asteroid(random_value_texture, asteroidTexture0,
asteroidTexture1, asteroidTexture2); //Pour être créé l'astéroïde a besoin des 3
différentes textures. Les différentes textures (de tailles différentes) correspondent
aux 3 types d'astéroïdes : un gros, un moyen, un petit. L'attribution des textures en
fonction de la valeur aléatoire se fait dans la classe Asteroid.
            int random_value_X1 = randomNumber.Next(0, 125); //Un nombre aléatoire
pour savoir en quel x l'astéroide va spawner.
            int random_value_X2 = randomNumber.Next(0, 500); //On exploite un 2ème
nombre aléatoire…
            if (random_value_X2 % 2 == 0) //…si le deuxième nombre est paire, ainsi
l'astéroide spawnera…
            {
                random_value_X1 = -random_value_X1; //…à gauche de l'écran…
            }
            else
            {
                random_value_X1 = graphicsGamePlay.PreferredBackBufferWidth +
random_value_X1; //…ou à droite de l'écran
            }

            int random_value_Y1 = randomNumber.Next(0, 125); //Même principe pour Y.
            int random_value_Y2 = randomNumber.Next(0, 500);
            if (random_value_Y2 % 2 == 0)
            {
                random_value_Y1 = -random_value_Y1;
            }
            else
            {
                random_value_Y1 = graphicsGamePlay.PreferredBackBufferHeight +
random_value_Y1;
            }

            if (-125 < random_value_X1 && random_value_X1 < 125 && -125 <
random_value_Y1 && random_value_Y1 < 125) //Si l'astéroide spawn dans le coin
supérieur gauche…
```

```
            {
                asteroid.asteroidcoefTrajectory0 = randomNumber.Next(80, 120);
                asteroid.asteroidcoefTrajectory1 = 1;
                asteroid.asteroidMovingSense = 1;

                //…Alors sa trajectoire sera différente. Il s'agit ici d'une fonction
croissante. Comme pour une fonction classique, on fera en sorte que x varie, et que y
dépend de lui. Le movingSense permet de savoir s'il s'agit des x croissants ou des x
décroissants.
X(t) = X(t-1) + Sens*Vitesse

Y(t) = C1 * X(t) + C0 ;

            }

            if (-125 < random_value_X1 && random_value_X1 < 125 &&
graphicsGamePlay.PreferredBackBufferHeight - 125 < random_value_Y1 && random_value_Y1
< graphicsGamePlay.PreferredBackBufferHeight + 125) //Si l'astéroid spawn dans le coin
inférieur gauche…
            {
                asteroid.asteroidcoefTrajectory0 =
randomNumber.Next(graphicsGamePlay.PreferredBackBufferHeight - 200,
graphicsGamePlay.PreferredBackBufferHeight + 200);
                asteroid.asteroidcoefTrajectory1 = -1;
                asteroid.asteroidMovingSense = 1;

                //…sa trajectoire est ici une fonction décroissante
             }

            if (graphicsGamePlay.PreferredBackBufferWidth - 125 < random_value_X1 &&
random_value_X1 < graphicsGamePlay.PreferredBackBufferWidth + 125 && -125 <
random_value_Y1 && random_value_Y1 < 125) // Si l'astéroid spawn dans le coin
supérieur droit…
            {
                asteroid.asteroidcoefTrajectory0 =
randomNumber.Next(graphicsGamePlay.PreferredBackBufferWidth - 400,
graphicsGamePlay.PreferredBackBufferWidth + 400);
                asteroid.asteroidcoefTrajectory1 = -1;
                asteroid.asteroidMovingSense = -1;

                //…sa trajectoire est ici une fonction décroissante
            }

            if (graphicsGamePlay.PreferredBackBufferWidth - 125 < random_value_X1 &&
random_value_X1 < graphicsGamePlay.PreferredBackBufferWidth + 125 &&
graphicsGamePlay.PreferredBackBufferHeight - 125 < random_value_Y1 && random_value_Y1
< graphicsGamePlay.PreferredBackBufferHeight + 125) // Si l'astéroid spawn dans le
coin inféiruer droit…
            {
                asteroid.asteroidcoefTrajectory0 = randomNumber.Next(-400, -200);
                asteroid.asteroidcoefTrajectory1 = 1;
                asteroid.asteroidMovingSense = -1;

                //…sa trajectoire est ici une fonction croissante
            }

            asteroid.asteroidAngle = random_value_angle;

            asteroid.asteroidPosition.X = random_value_X1;
            asteroid.asteroidPosition.Y = random_value_Y1;
```

```
            asteroid.asteroidDirection = new
Vector2((float)Math.Cos(asteroid.asteroidAngle),
(float)Math.Sin(asteroid.asteroidAngle)); //Même principe que pour le joueur. On a
besoin d'avoir une direction pour que le mouvement prenne en compte la variation de
l'angle.
            asteroidList.Add(asteroid); //On l'ajoute à la liste.
        }
```

- **Passer d'une scène à l'autre (Menu principal, Jeu, Scène de fin de jeu)**

Afin de passer simplement d'une scène à l'autre sans se poser de soucis, on va exploiter le fonctionnement du programme principal et créer une classe Scene qui sera une classe abstract (une classe qui n'est jamais instancié, elle sert de classe de classe de base pour d'autres classes). Cette classe ne possède que les corps de fonctions basiques.

```
namespace GAAR_Game
{
    public abstract class Scene
    {
        public virtual void Initialize()
        {
        }

        public virtual void Update(GameTime gameTime)
        {
        }

        public virtual void LoadContent()
        {
        }

        public virtual void UnloadContent()
        {
        }

        public virtual void Draw(SpriteBatch spriteBatch)
        {
        }


    }
}
```

Chaque corps de fonction sera écrit en fonction de la Scene dans lequel on se trouve. Pour relier tout ça ensemble, on créé dans le main une Liste de Scene avec une variable représentant la Scène actuelle. Cette scène actuelle permettra de réaliser toutes les fonctions classiques (c-à-d Update, LoadContent, Initialize…) avec le corps de chaque fonction écrite pour les différentes scenes. Voici par exemple la fonction LoadContent de la scene GamePlay :

```
public override void LoadContent() //Le préfixe override permet ainsi de faire en
sorte de pouvoir faire en sorte que lorsqu'on lance la fonction LoadContent à partir
d'une scene, on utilise les éléments de la classe dérivée de scene. Ici GamePlay.
        {
            backgroundTexture = gamePlayContent.Load<Texture2D>("game_map_wallpaper");
            player.playerTexture = gamePlayContent.Load<Texture2D>("player_sprite");
            laserStripTexture = gamePlayContent.Load<Texture2D>("laser_strip");
            asteroidTexture0 = gamePlayContent.Load<Texture2D>("asteroid_little");
            asteroidTexture1 = gamePlayContent.Load<Texture2D>("asteroid_medium");
            asteroidTexture2 = gamePlayContent.Load<Texture2D>("asteroid_fat");
```

```
            alienTexture = gamePlayContent.Load<Texture2D>("alien_flying_saucer");
            needlerAlienTexture = gamePlayContent.Load<Texture2D>("needler_alien");
            HUDfont = gamePlayContent.Load<SpriteFont>("gameFont");
            laserSound = gamePlayContent.Load<SoundEffect>("laserSound");
            alienSound = gamePlayContent.Load<SoundEffect>("alienSound");
            needlerSound = gamePlayContent.Load<SoundEffect>("needlerSound");
            explosionSound = gamePlayContent.Load<SoundEffect>("explosionSound");
            alienIsDestroyedSound = gamePlayContent.Load<SoundEffect>("Killtacular");
            playerIsTouched = gamePlayContent.Load<SoundEffect>("health");
            guile = gamePlayContent.Load<Song>("GUILE'S THEME");

            MediaPlayer.Play(guile);
            MediaPlayer.Volume = 0.7f;

            base.LoadContent();
        }
```

Voici ainsi la structure du code principal (renommé Main pour mon projet) :

```
public class Main : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        List<Scene> sceneList = new List<Scene>();
        Scene sceneCurrent;

        GamePlay gamePlay;
        EndGame endGame;
        MainMenu mainMenu;

        public Main()
        {
            graphics = new GraphicsDeviceManager(this);
            graphics.PreferredBackBufferWidth = 1920;
            graphics.PreferredBackBufferHeight = 1080;
            Content.RootDirectory = "Content";

        }
        protected override void Initialize()
        {
            mainMenu = new MainMenu(graphics, Content);
            gamePlay = new GamePlay(graphics, Content);
            endGame = new EndGame(gamePlay.player.playerScore, graphics, Content);

            sceneList.Add(mainMenu);
            sceneList.Add(gamePlay);
            sceneList.Add(endGame);

            sceneCurrent = sceneList[0]; //Par défaut la scène est celle du menu
            sceneCurrent.Initialize();

            base.Initialize();
        }
        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);

            sceneCurrent.LoadContent();
```

```
        }
        protected override void UnloadContent()
        {
              SceneCurrent.UnloadContent();
        }


        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed
|| Keyboard.GetState().IsKeyDown(Keys.Escape)) //Si la touche ECHAP du clavier est
pressée ou si le bouton back de la manette est pressée, alors le jeu se ferme.
                Exit(); //A

            sceneCurrent.Update(gameTime);
            base.Update(gameTime);

        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.Black);
            spriteBatch.Begin();
            sceneCurrent.Draw(spriteBatch);
            spriteBatch.End();
            base.Draw(gameTime);
        }
    }
```

Il suffira par la suite de rajouter des conditions pour faire en sorte que la scène change en fonction du déroulé du jeu. Par exemple, pour passer du GameMenu au Jeu :

```
protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed
|| Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            if (mainMenu.isGameIsAboutToBeLaunched == true)
            {
                mainMenu.isGameIsAboutToBeLaunched = false; //Sans quoi la condition va
s'appeler éternellement.
                mainMenu.UnloadContent();
                sceneCurrent = sceneList[1];
                sceneCurrent.Initialize();
                sceneCurrent.LoadContent();
            }
```

- **Détecter des collisions**

Comment faire en sorte que lorsqu'on tire sur un astéroïde ou sur un alien, celui-ci meurt ? Pour cela il faut détecter la collision entre l'élément 1 et l'élément 2. Prenons ici le cas où le joueur tire sur un astéroïde, et que le tir laser généré atteint celui-ci.

Pour cela nous allons créer une classe nommée CollisionsCircle (pour cercle de collisions). Chaque élément du jeu possèdera son propre cercle de collision. Ce qui posera parfois quelques soucis (voire partie 3 du rapport). Un cercle c'est : une origine, et un rayon.

```
public class CollisionsCircle
    {
        public Vector2 circleCenter;
        public float circleRadius;

        public bool isCollided = false; //On teste si la collision a eu lieu.

        public CollisionsCircle(Vector2 centre, float radius)
        {
            circleCenter = centre;
            circleRadius = radius;
        }

        public bool ColliderDetector(CollisionsCircle target)
        {
            float distanceBetweenThem =  (float)Math.Sqrt((target.circleCenter.X -
circleCenter.X)* (target.circleCenter.X - circleCenter.X) + (target.circleCenter.Y -
circleCenter.Y)* (target.circleCenter.Y - circleCenter.Y)); //Formule pour calculer la
distance entre les deux centres des deux cercles.
            if (distanceBetweenThem < (circleRadius + target.circleRadius)) //
Théorème de Pytagore pour déterminer s'il y a collision entre les deux cercles
            {
                isCollided = true; //Il y a collision.
            }

            else
            {
                isCollided = false; //Il n'y a pas.
            }

           return isCollided;
        }

    }
}
```

Il suffit désormais à chaque frame (donc dans la fonction Update) de tester si le cercle de collision du tir laser généré par le joueur est en collision avec celui de l'astéroïde.

```
foreach (Asteroid asteroid in asteroidList)
            {
                asteroid.asteroidAngle = asteroid.asteroidAngle - 0.01f;
                asteroid.asteroidColisionsCircle = new
CollisionsCircle(asteroid.asteroidPosition, asteroid.asteroidTexture.Width / 2);
                asteroid.asteroidDirection = new
Vector2((float)Math.Sin(asteroid.asteroidAngle),
(float)Math.Cos(asteroid.asteroidAngle));

                asteroid.asteroidPosition.X = asteroid.asteroidPosition.X +
asteroid.asteroidMovingSense * asteroid.asteroidSpeed;
                asteroid.asteroidPosition.Y = asteroid.asteroidcoefTrajectory1 *
asteroid.asteroidPosition.X + asteroid.asteroidcoefTrajectory0;

                                                [...]
                foreach (LaserStrip laserStrip in laserStriplist) //Pour chaque tir
laser dans la liste de tirs laser
                {
                    if
(asteroid.asteroidColisionsCircle.ColliderDetector(laserStrip.laserStripColisionsCircl
e) == true) //S'il y a collision laser/astéroïde
                    {
                        explosionSound.Play(); // BAM

                        laserStrip.laserStripIsVisible = false; // On le voit plus

                        asteroid.asteroidIsVisible = false; //Dès que l'astéroïde
n'est plus dans la zone visible par le joueur, on fait en sorte qu'il soit false pour
le supprimer. De même pour laserStrip.

                    }

                    if (asteroid.asteroidPosition.X >=
graphicsGamePlay.PreferredBackBufferWidth + 200 || asteroid.asteroidPosition.X <= -
200)
                    {
                        asteroid.asteroidIsVisible = false; //Dès que l'astéroïde
n'est plus dans la zone visible par le joueur, on fait en sorte qu'il soit false pour
le supprimer.
                    }

                    if (asteroid.asteroidPosition.Y >=
graphicsGamePlay.PreferredBackBufferHeight + 200 || asteroid.asteroidPosition.Y <= -
200)
                    {
                        asteroid.asteroidIsVisible = false; //Pareil
                    }
                                                [...]
                }
            }

            for (int i = 0; i < asteroidList.Count; i++) //Pour I allant de 0 au
nombre d'éléments dans la liste – 1
            {
                if (asteroidList[i].asteroidIsVisible == false) //Si l'astéroid n'est
pas visible…
                {
                    asteroidList.RemoveAt(i); //…on le supprime de la liste.
                    i = i - 1; //On décrémente car le count de la liste a changé.
                }
            }
                    Même boucle pour les lasers du joueur.
```

- **Faire en sorte que lors qu'un astéroïde est touché, celui-ci se coupe en plusieurs petits astéroïdes en fonction de sa taille s'il est gros, ou moyen, augmenter le score du joueur et jouer un son d'explosion**

Dans le jeu, dès qu'un gros astéroïde ou un astéroïde de taille moyenne, ceux-ci se coupent respectivement en 3 et en 2 petits astéroïdes. On va donc réexploiter la partie concernant les collisions (dans Update) pour faire en sorte que dès qu'un astéroïde est touché par un tir laser, en fonction de son type, on appelle une fonction nommée « fission ». On va également faire en d'ajuster le score en fonction du type d'astéroïde détruit.

```
foreach (Asteroid asteroid in asteroidList)
{

        foreach (LaserStrip laserStrip in laserStriplist)
                 {
                            if
        (asteroid.asteroidColisionsCircle.ColliderDetector(laserStrip.laserStripColision
        sCircle) == true)
                            {
                                explosionSound.Play(); //On fait en sorte de jouer le
        son dès qu'il est touché. Le son est chargé de la même manière qu'une texture2D.
        Pas de dessin ici, juste un sound.Play() dès qu'on a besoin.
                                laserStrip.laserStripIsVisible = false;
                                if (asteroid.asteroidType == 0) //Si c'est un petit
        astéroïde, + 150 en score, félicitation !
                                {
                                    player.playerScore = player.playerScore + 150;
                                }

                                if (asteroid.asteroidType == 1) //Si c'est un astéroïde
        de taille Moyenne, il va se couper en 2 petits astéroïdes et + 50 en score,
        félicitation !
                                {
                                    player.playerScore = player.playerScore + 50;
                                    detectionFission = true;
                                    asteroidSplit = asteroid; // L'astéroïde à couper
        sera celui-là. On ne peut pas faire le split directement car nous sommes en
        train de parcourir la liste. On ne peut pas modifier le nombre d'élément d'une
        liste alors que nous somme en train de la parcourir.
                                    numberOfAsteroidsToGenerateAfterFission = 2;
                                }


                                if (asteroid.asteroidType == 2) //Si c'est un astéroïde
        de grande taille, il va se couper en 3 petits astéroïdes et + 50 en score,
        félicitation !
                                {
                                    player.playerScore = player.playerScore + 10;
                                    detectionFission = true;
                                    asteroidSplit = asteroid;
                                    numberOfAsteroidsToGenerateAfterFission = 3;
                                }
                               asteroid.asteroidIsVisible = false; //Il y a eu
        collision, donc l'astéroïde est détruit, donc il n'est plus visible.
                            }

           }

}
```

```
if (detectionFission == true) //On a terminé de parcourir la liste. Désormais, s'il y
a eu fission pour l'astéroïde…
            {
                asteroidSplit.fission(asteroidList, asteroidTexture0,
asteroidTexture1, asteroidTexture2, numberOfAsteroidsToGenerateAfterFission); //…on
lance la fission.
                detectionFission = false; //Pour éviter que la fonction s'auto appelle
à chaque frame.
            }

            for (int i = 0; i < asteroidList.Count; i++)
            {
                if (asteroidList[i].asteroidIsVisible == false) //Et on n'oublie pas
de rendre les astéroïdes invisibles s'ils sortent du cadre ou s'ils sont détruits.
                {
                    asteroidList.RemoveAt(i);
                    i = i - 1;
                }
            }
```

Allons désormais voir la fonction fission qui se trouve dans la classe Asteroid.

```
public void fission(List<Asteroid> list, Texture2D asteroid0, Texture2D asteroid1,
Texture2D asteroid2, int number)
        {
            Random changeTrajectory = new Random(); //On va ici utiliser un nombre
aléatoire (voir la suite)

            for (int i = 0; i < number; i++) //3 petits sont générés si l'astéroïde
est gros, 2 s'il est de taille moyenne
            {
                Asteroid asteroidResulting = new Asteroid(0, asteroid0, asteroid1,
asteroid2);
                asteroidResulting.asteroidPosition = asteroidPosition;
                asteroidResulting.asteroidAngle = asteroidAngle +
changeTrajectory.Next(-3, 3) ; //On va aléatoirement modifier l'angle…
                asteroidResulting.asteroidOrigin = asteroidOrigin;
                asteroidResulting.asteroidDirection = asteroidDirection;
                asteroidResulting.asteroidcoefTrajectory0 = asteroidcoefTrajectory0 -
changeTrajectory.Next(-400, 400); //…et le coefficient 0 de trajectoire de l'astéroïde
                asteroidResulting.asteroidcoefTrajectory1 = asteroidcoefTrajectory1;
                asteroidResulting.asteroidMovingSense = asteroidMovingSense;
                list.Add(asteroidResulting);
            }

            asteroidIsVisible = false;
        }
```

2.3 Algorigrammes de fonctionnement du programme

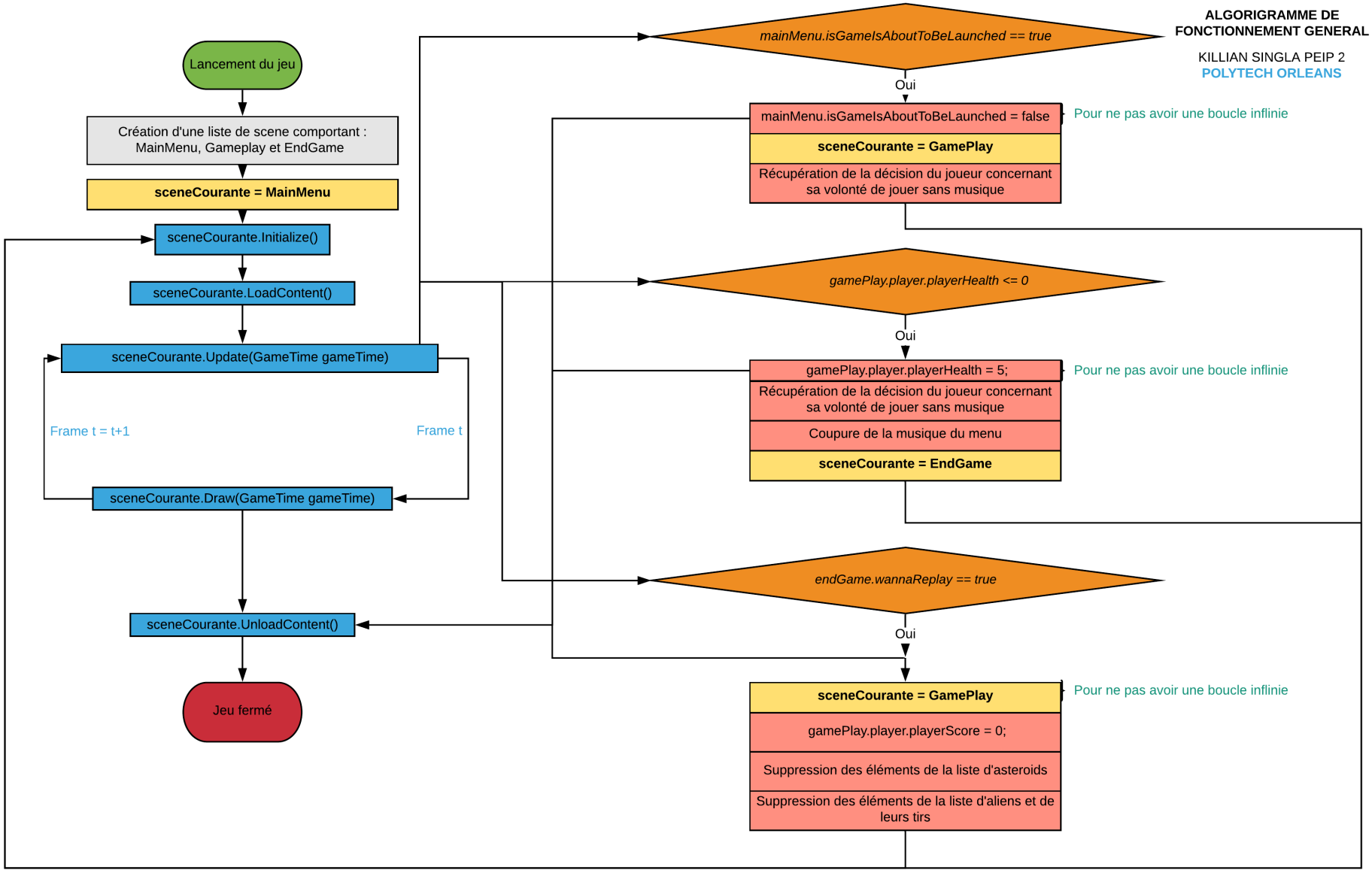

Figure 9 : Algorigramme du fonctionnement général du jeu

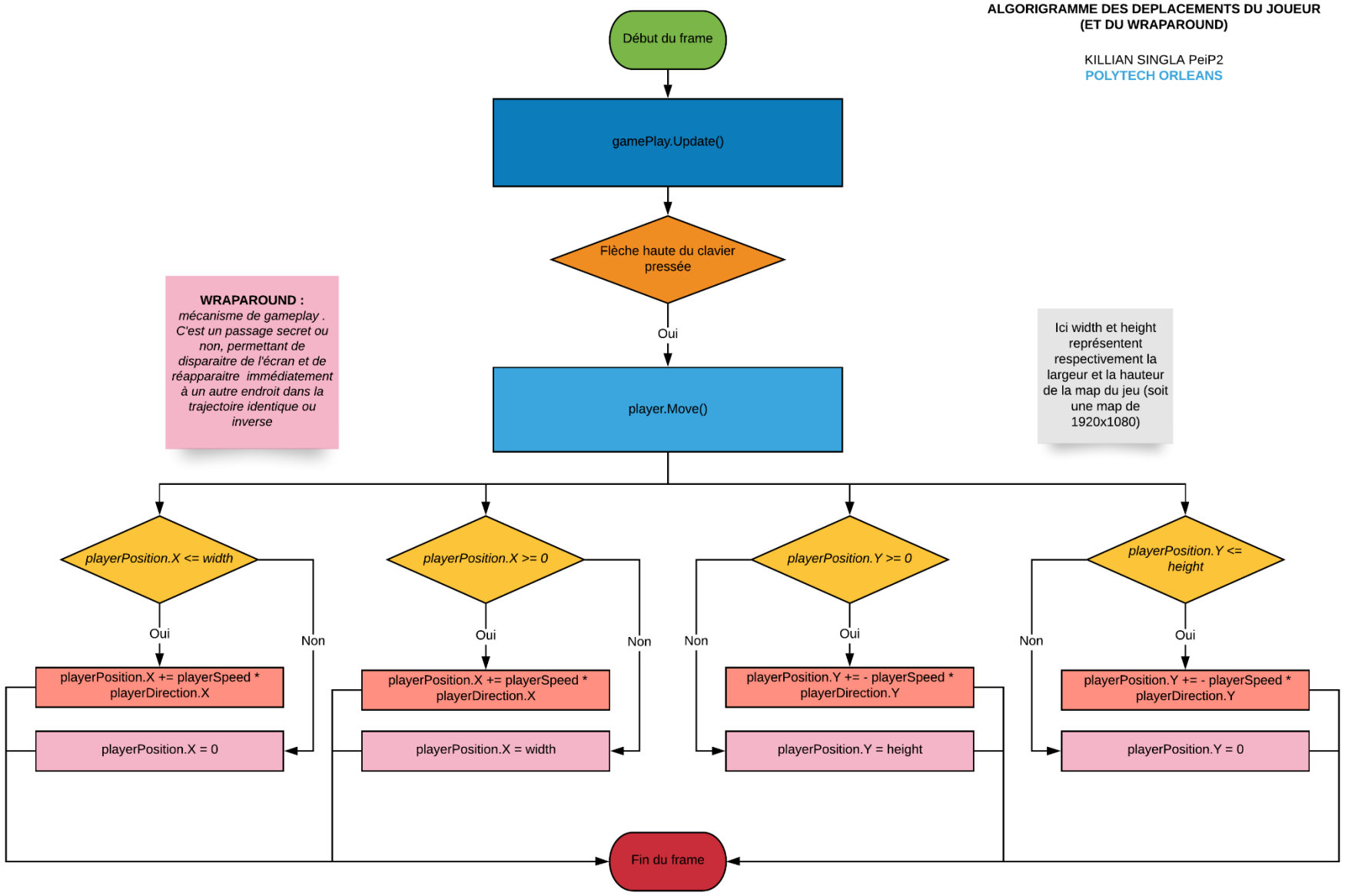

Figure 10 : Algorigramme des déplacements du joueur

Killian SINGLA PeiP2

2.4 Itérations et tests mis en place pendant le développement

Un jeu est un logiciel comme un autre et les tests font partie intégrante du développement. Chaque fonctionnalité créée doit être testée plusieurs fois à la suite pour constater si celle-ci fonctionne bien, si celle-ci ne génère pas de bug, et si celle-ci est suffisamment performante.

Les différents tests se font grâce à la console Windows qui permet d'y envoyer des informations (par exemple, dès qu'un ennemi est touché, un message est envoyé pour constater que la collision a bien été détectée) et grâce au debugger du logiciel Visual Studio est particulièrement puissant dans la traque de bugs en tous genres, et pour suivre le fonctionnement d'un programme informatique.

De nombreuses itérations ont ainsi été mis en place afin de tester chaque fonctionnalité, trouver les bugs, les résoudre et modifier certains paramètres (vitesse du joueur, vitesse des différents astéroïdes, temps de spawn…).

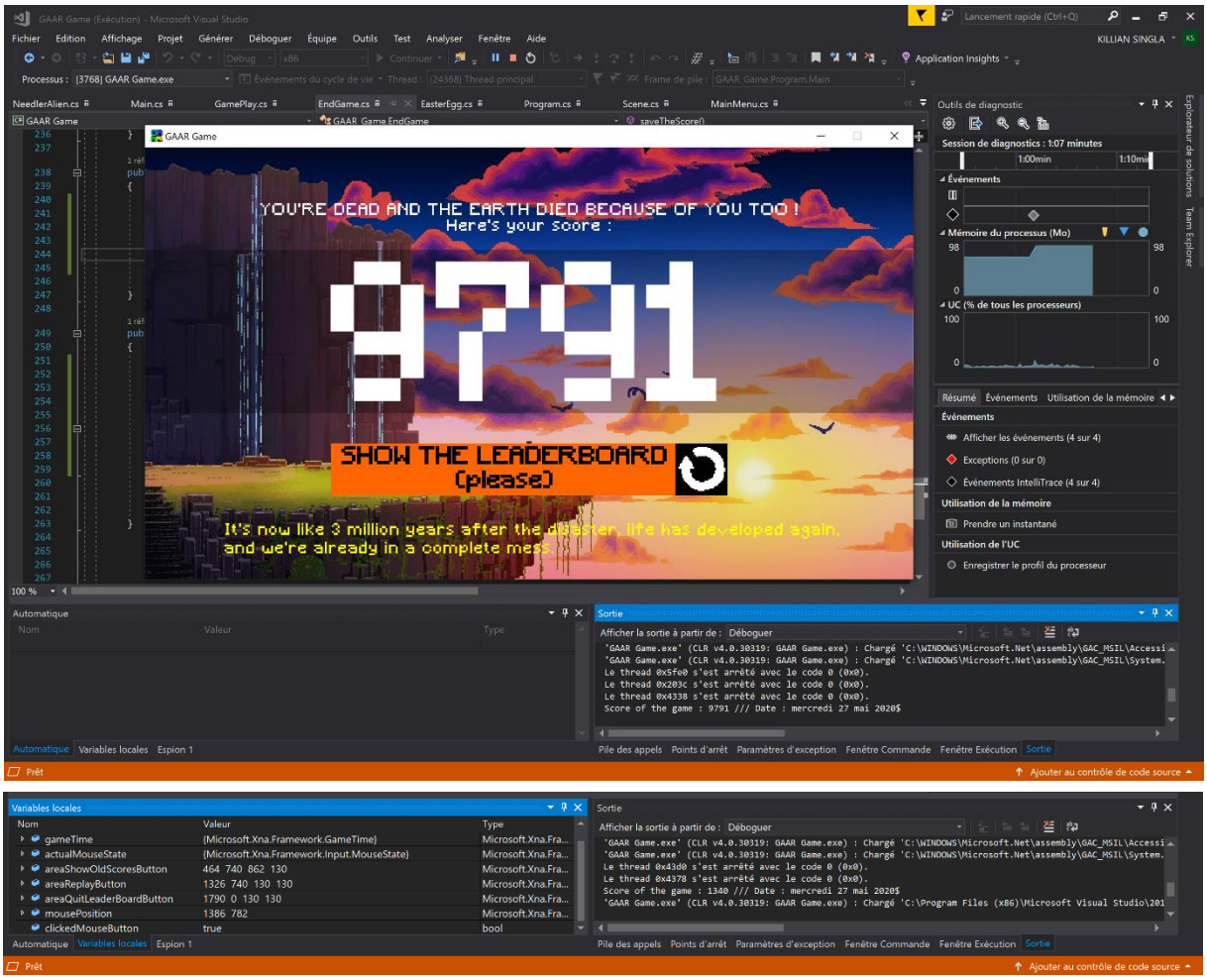

Figure 11 : Affichage des informations dans la console et des variables afin de débugger le jeu

3. ANALYSE ARGUMENTEE DUDIT PROJET ET DE SON DEROULE

3.1 Fiche de suivi réelle et tâches véritablement réalisées

X : créneau supprimé / X : créneau rajouté

Tâches à effectuer	18-mai-20		19-mai-20		20-mai-20		21-mai-20		22-mai-20		23-mai-20		24-mai-20		25-mai-20		26-mai-20		27-mai-20		28-mai-20		29-mai-20		AVANCEMENT AVEC COMMENTAIRE(S)
	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	M	AM	OK / EN COURS/A PEAUFINER / A FAIRE
Réaliser un rapport de projet		X		X		X		X		X		X		X		X		X	X	X	X	X	X	X	
Réaliser un poster format A0 du projet		X						X		X		X				X		X	X	X	X	X	X	X	
Classe GameMenu - Mise en place des éléments graphiques														X		X	X								
Classe GameMenu - Gestion évènements souris (boutons)														X		X	X								
Classe GameMenu - Animation (mouvement) de certains éléments															X	X	X								
Classe GameMenu - Permettre la lecture/la mise en pause de la musique de fond														X		X									
Classe PlaylistMenu -> Classe Playlist - Permettre le choix ou non d'une playlist musicale et de couper ou non le son du jeu (bruitages...), lire cette playlist aléatoirement et gérer son volume par rapport à celui du jeu																	X	X							PAS REALISEE
Classe PlaylistMenu -> Classe Paylist - Permettre au joueur de créer sa propre playlist avec des titres stockés sur son ordinateur et gérer son volume par rapport à celui du jeu																	X	X							PAS REALISEE
Classe LobbyMenu : Indiquer les commandes jouables et faire un compte à rebours															X										REMPLACE PAR UN SIMPLE BACKGROUND SUR GAMEMENU
Classe GamePlay : Mettre en place les éléments graphiques de la map		X																							
Classe GamePlay -> Classe Player : Mise en place du sprite		X																							
Classe GamePlay -> Classe Player : Gestion des évènements claviers			X																						
Classe GamePlay -> Classe Player : Gérer le déplacement du joueur			X																						
Classe GamePlay -> Classe Player : Gérer les tirs du joueur et leurs collisions			X	X	X																				Revoir le sprite du joueur ?
Classe GamePlay -> Classe Player : Gérer la jauge de vie du joueur et son respawn (collisions)						X				X															
Classe GamePlay -> Classe Player : Jouer des sons à chaque action du joueur (tir, déplacement, destruction)																	X	X	X						
Classe GamePlay -> Classe Player : Jouer des animations à chaque action du joueur (tir, déplacement)																				X	X				PAS REALISEE
Classe GamePlay -> Classe Asteroids : Mise en place des différents sprites					X																				
Classe GamePlay -> Classe Asteroids : Gérer le spawn aléatoire des différents types d'astéroides (3-4 types)					X	X																			A peaufiner (lois de proba ?)
Classe GamePlay -> Classe Asteroids : Gérer les trajectoires aléatoires des différents astéroides					X	X	X																		
Classe GameMap -> Classe Asteroids : Gérer la division des astéroides en d'autres astéroides plus petits lors qu'ils sont touchés						X	X																		
Classe GameMap -> Classe Asteroids : Gérer la destruction des astéroides (collisions)						X	X																		
Classe GameMap -> Classe Asteroids : Jouer des sons (destruction, déplacement...)																	X	X	X						
Classe GameMap -> Class Asteroids : Jouer des animations (destruction, dépacement...)																				X	X				PAS REALISEE
Classe GameMap -> Classe Aliens : Mise en place du sprite							X																		
Classe GameMap -> Classe Aliens : Gérer le spawn aléatoire des aliens								X																	
Classe GameMap -> Classe Aliens : Gérer les trajectoires aléatoires des aliens								X																	
Classe GameMap -> Classe Aliens : Gérer les tirs des aliens et leurs collisions								X	X	X															
Classe GameMap -> Classe Aliens : Jouer des sons (destruction, déplacement, tir...)																	X	X	X						
Classe GameMap -> Classe Aliens : Jouer des animations (destruction, dépacement, tir...)																				X	X				PAS REALISEE
Classe GameMap : Afficher le score, ainsi que les vies restantes du joueur											X	X													
Classe EndGame : Afficher les scores des parties précédentes												X	X	X		X	X								
Classe EndGame: Permettre l'affichage du score de la partie réalisé et l'enregistrement de celui-ci.												X	X	X		X	X								
Fin de projet : tester, régler certains paramètres, et corriger les bugs	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	

Figure 12 : Calendrier réel et tâches véritablement réalisées

3.2 Difficultés rencontrées lors du développement

Lorsqu'on développe un projet de cette envergure pour la première fois et ce en un temps limité, on fait face à un certain nombre de difficultés.

Premièrement, le manque de recul sur le travail effectué et d'expérience. N'ayant pas énormément d'expérience sur le développement d'un projet, j'ai, à de nombreuses reprises douté sur la qualité de mon travail. Un problème possède de nombreuses solutions mais sans expérience, il est impossible de savoir si celle choisie est la meilleure, la plus intelligente, la plus rapide, et la moins gourmande en ressources, et cela génère du doute.

Le peu d'expérience et de confiance en soi rend également la correction de bugs classiques à la fois très compliquée et très chronophage. A de nombreuses reprises, le développement du jeu a été stoppé puisqu'une fonctionnalité importante ne fonctionnait plus à cause d'une simple ligne de code mal placée, ou d'une accolade qui manquait. Une erreur triviale pour un programmeur confirmé qui est parfois compliqué à trouver pour un débutant.

Il était également très compliqué de planifier en amont du projet les tâches à réaliser bien que le planning a globalement bien été respecté, et les travaux livrés en temps et en heure. Deuxième le temps réduit

Puis, bien que très fan des jeux-vidéo avec certaines connaissances dans cette industrie, je suis presque novice dans le développement de jeux-vidéo. Ainsi il a fallu quelque temps pour comprendre comment fonctionnait MonoGame et un jeu-vidéo en général. Il m'est très difficile d'avancer dans un projet ou dans un travail sans véritablement comprendre la tâche réalisée. Ce manque de connaissances en développent de jeux m'a également forcé à devoir me renseigner systématiquement sur les méthodes à utiliser et la documentation technique à fouiller dès que la tâche à réaliser n'était pas évidente pour un jeune programmeur comme moi.

3.3 Points forts et faibles du projet

Points forts du projet :

- Certains éléments pixel-arts sont assez agréables à regarder
- Programmation POO qui permet de réutiliser à souhait chaque élément
- Jeu simple du fait de ses règles mais avec une difficulté assez importante
- Jeu addictif
- Possibilité d'afficher ou non un leaderboard pour voir ses anciens scores
- Possibilité d'afficher ou non une interface permettant d'avoir des informations sur le fonctionnement du jeu et ses commandes
- Interface complète d'un jeu classique
- Possibilité de choisir de couper la musique dans le jeu (choix sur le menu, et cela se répercute sur tout le jeu)
- Easter eggs aléatoires sur le menu avec mouvements
- Fission des astéroïdes et trajectoires aléatoires assez réussies
- Spawn et trajectoire des aliens aléatoires à un certain instant t
- Jeu, code, intégralement en anglais. Ce qui permet à quiconque de comprendre le jeu et de comprendre son code quel que soit sa nationalité.
- Utilisation d'un framework : facilité d'utilisation, léger, gratuit, simple

Points faibles du projet :

- Manque de précision concernant les collisions. Chaque texture n'étant pas particulièrement ronde (cercle de collision), ça rend la détection imprécise (on peut voir à l'écran un élément disparaître car il a été touché or on ne voit pas la collision).
- Affichage du leaderboard imprécis (impossible de voir tous les anciens score quand le fichier possède de nombreux scores)
- Aucune véritable animation
- Aucun effet spécial
- Texture 2D parfois visuellement en deçà.
- Bruitages imprécis et parfois inexistants (déplacements du vaisseau)
- Le fait que le vaisseau est touché n'est pas assez visible (le bruitage ne suffit pas)
- Les transitions sont trop marquées
- Résolution fixe
- Plateforme limitée (Windows)
- Utilisation d'un framework : fonctionnalités limitées face à un game engine comme Unity
- Pas de possibilité de modifier les touches à sa guise et d'utiliser une manette à la place du clavier

3.4 Pistes d'améliorations possibles futures

- Améliorer les collisions en adaptant les figures de collisions en fonction des textures
- Rajouter des niveaux de difficultés
- Rajouter des animations
- Rajouter des effets spéciaux (explosions, destruction de vaisseaux avec débris…)
- Créer un système de playlist musical permettant au joueur de créer et choisir sa propre playlist lorsqu'il joue
- Adapter le jeu en fonction de la résolution de l'écran et faire en sorte que le jeu puisse être lu en 4K
- Adapter le jeu sur d'autres plateformes (LINUX, MacOS, Android, iOS Xbox One, PS4, Nintendo Switch, Stadia voir sur les prochaines consoles)
- Prendre en charge les commandes avec la manette
- Améliorer l'affichage du leaderboard
- Créer des animations pour les transitions
- Composer des musiques et des bruitages adaptés pour le jeu
- Créer toutes les textures du jeu de A à Z
- Créer des aliens de différents types avec des vitesses, armes différentes
- Créer des vaisseaux de différents types (avec des vitesses/dégâts différents) et permettre au joueur de choisir en fonction de son level
- Faire en sorte des sons spéciaux se lancent à chaque pallier de score (comme les séries d'éliminations dans les jeux-vidéo)
- Possibilité de partager son score sur les réseaux-sociaux

CONCLUSION

Que cela soit d'un point de vue scolaire, professionnel et technique, ce projet n'a été que bénéfique.

Ce projet est mon premier projet d'envergure en programmation. Il fait suite à un projet Space Invaders réalisé en Terminale en Python, et à un projet de calculatrice réalisé en C++ lors de ma première année de PeiP à Polytech Orléans. Il s'agit ici de la toute première fois que j'exploite des outils aussi puissants (comme MonoGame) qui n'ont pas été vu durant ma scolarité. Cela m'a permis ainsi de me former aux méthodes de travail d'un ingénieur classique (informatique ou non). En effet, l'exploitation d'outils, de logiciels, de documentations techniques et de méthodes encore étrangères sont des actes quotidiens dans le travail d'un ingénieur informatique. Le but est d'assimiler ces outils, et ces méthodes, ainsi que de les adapter au mieux afin de répondre à un cahier des charges précis établi par le client ou la hiérarchie. Il s'agissait également de la première fois que je mettais mes connaissances en C# en pratique sur un projet hormis les TP réalisés en cours).

Du fait de la durée allouée somme toute assez limitée pour réaliser le développement du programme, ce dit rapport ainsi qu'un poster format A0, cela m'a énormément fait progresser dans la planification, et la gestion de projet. Cela m'a également permis d'estimer au mieux le temps nécessaire pour réaliser certaines fonctionnalités, à concentrer mes ressources sur les tâches les plus complexes, et à faire des choix techniques concernant la réalisation d'une telle tâche en fonction du temps alloué.

Les réunions quotidiennes avec mon tuteur pédagogique Rémy Leconge tendent également à m'empeigner du déroulé classique d'un projet dans une entreprise de service informatique. En effet, les réunions avec sa hiérarchie afin de communiquer sur ledit projet et d'avoir leurs points de vus dessus sont très courantes dans le milieu de l'entreprise.

D'un point de vue personnel j'y ai également appris à me contrôler lors de situations compliquées (bugs difficiles à trouver, fonctionnalité ne fonctionnant toujours pas après plusieurs heures de développement etc.) et à ne pas baisser les bras face à la difficulté.

Je suis extrêmement heureux d'avoir eu cette chance de développer un tel projet qui me tient extrêmement à cœur et je suis particulièrement reconnaissant d'avoir eu la possibilité de choisir mon propre sujet. Malgré les instants parfois compliqués et la fatigue due aux nombreuses heures de travail, je ne retiens que du positif de ce projet.

L'informatique est ma discipline de cœur, et les projets futurs lors de mon cycle ingénieur informatique dans le réseau Polytech à Polytech Tours, continueront, j'en suis persuadé, de m'émerveiller et de m'épanouir.

TABLE DES ANNEXES

ANNEXES

A - Tableau des ressources réalisées par des tiers et utilisées dans GAAR

Auteur de la ressource	Nom réel/Provenance de la ressource	Nom de la ressource dans le code du jeu	Type de ressource	Modification
Mitch Murder	*After Hours Run (titre)*	after-hours-run.mp3	Musique	Non
Pixelartmaker.com	*Shiba Inu*	alien_flying_saucer	Texture 2D	Oui
SoundFishing.net	*teleportation*	alienSound.mp3	Son	Non
X	*X*	asteroid_fat.png	Texture 2D	Oui
X	*X*	asteroid_little.png	Texture 2D	Oui
X	*X*	asteroid_medium.png	Texture 2D	Oui
LaSonotheque.org	*Souris Raspberry, simple clic (nom)*	clickSound.mp3	Son	Non
Royal Scots Dragoon Guards	*Auld Lang Syne (titre)*	cornemuse.mp3	Musique	Oui
Puzzling Dream	*The Way (jeu)*	endGameWallpaper.png	Texture 2D	Oui
SoundFishing.net	explosion2	explosionSound.mp3	Son	Non
Puzzling Dream	*The Way (jeu)*	game_map_wallpaper.png	Texture 2D	Oui
kevinberryman	*Game Gear Style Spartan*	game_menu_surprise_117	Texture 2D	Oui
J.C. Harnell	*Banana Pixel Art Print*	game_menu_surprise_banana	Texture 2D	Oui
david J	Chat en beignet pixel art Poster	game_menu_surprise_cat	Texture 2D	Oui
Freepng.fr	*Crêpe, Le Babeurre, Le Pixel Art*	game_menu_surprise_crepes	Texture 2D	Non
hugolescargot.com	*Ballon de football en pixel art*	game_menu_surprise_foot	Texture 2D	Oui
High Tea Frog	*Logo*	game_menu_surprise_frog	Texture 2D	Oui
Pixelartmaker.com	*Bronze Medal*	game_menu_surprise_medal	Texture 2D	Oui
Pixelartmaker.com	*Ori and sein*	game_menu_surprise_ori	Texture 2D	Oui
Keetay	*German shepherd breed art*	game_menu_surprise_oris	Texture 2D	Oui
Réseau Polytech	*Logo Polytech*	game_menu_surprise_polytech	Texture 2D	Oui
Puzzling Dream	*The Way (jeu)*	game_menu_wallpaper	Texture 2D	Non
Craftron Gaming	*Minecraft*	Minecraft	Police d'écriture	Non
London Music Works	*Guile's Theme (From "Street Fighter II: The World Warrior")*	GUILE'S THEME.mp3	Musique	Non
343 INDUSTRIES	*Killtacular sound from Halo 5 Guardians*	Killtacular.mp3	Son	Oui
SoundFishing.net	*pistoler laser 19*	LaserSound.mp3	Son	Non
pixeljeff1995	*Chill Mario*	leaderBoard_background.jpg	Texture 2D	Oui
343 INDUSTRIES	*Needler weapon sound from Halo 5 Guardians*	NeedlerSound.mp3	Son	Non

B – Code source du jeu

Program.cs

```
namespace GAAR_Game
{
#if WINDOWS || LINUX
    /// <summary>
    /// The main class.
    /// </summary>
    public static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]

        static void Main()
        {
            using (var game = new Main())
                game.Run();

        }
    }
#endif
}
```

Main.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;
using System.Timers;


namespace GAAR_Game
{
    /// <summary>
    /// This is the main type for your game.
    /// </summary>
    public class Main : Game //PRESS F12 ON GAME TO SEE THE CODE OF GAME. GAME IS THE
BASE CLASS. GAME MAP IS AN DERIVED CLASS
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        List<Scene> sceneList = new List<Scene>();
        Scene sceneCurrent;

        GamePlay gamePlay;
        EndGame endGame;
        MainMenu mainMenu;

        public Main()
        {
            graphics = new GraphicsDeviceManager(this);
            graphics.PreferredBackBufferWidth = 1920; // To change the width
resolution
            graphics.PreferredBackBufferHeight = 1080; // To change the height
resolution
            Content.RootDirectory = "Content";

        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting
to run.
        /// This is where it can query for any required services and load any non-
graphic
        /// related content.  Calling base.Initialize will enumerate through any
components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            this.IsMouseVisible = true;
            mainMenu = new MainMenu(graphics, Content);
            gamePlay = new GamePlay(graphics, Content);
            endGame = new EndGame(gamePlay.player.playerScore, graphics, Content);


            sceneList.Add(mainMenu);
            sceneList.Add(gamePlay);
            sceneList.Add(endGame);
```

```
            sceneCurrent = sceneList[0];
            sceneCurrent.Initialize();

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            /// Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            sceneCurrent.LoadContent();

            /// TODO: use this.Content to load your game content here

        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// game-specific content.
        /// </summary>
        protected override void UnloadContent()
        {
            sceneCurrent.UnloadContent();
            /// TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime) //For the keyboard
management
                                                          //We must check the inputs
at every frame
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed
|| Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit(); // If the escape key is pressed down or if the back button of
the controller is pressed, than with we exit.

            sceneCurrent.Update(gameTime);

            if (mainMenu.isGameIsAboutToBeLaunched == true)
            {
                mainMenu.isGameIsAboutToBeLaunched = false;
                mainMenu.UnloadContent();
                gamePlay.isGameMusicStopped = mainMenu.gameMusicIsStopped;
                sceneCurrent = sceneList[1];
                sceneCurrent.Initialize();
                sceneCurrent.LoadContent();
            }


            if (gamePlay.player.playerHealth <= 0)
            {
```

```
            MediaPlayer.Stop();

            gamePlay.player.playerHealth = 5; /* TO NOT HAVE A INFINITE LOOP*/
            gamePlay.UnloadContent();
            sceneCurrent = sceneList[2];
            endGame.isGameMusicStopped = mainMenu.gameMusicIsStopped;
            endGame.scoreToDisplay = gamePlay.player.playerScore;
            endGame.saveTheScore();
            sceneCurrent.Initialize();
            sceneCurrent.LoadContent();

        }

        if (endGame.wannaReplay == true)
        {
            endGame.wannaReplay = false;
            endGame.UnloadContent();
            sceneCurrent = sceneList[1];
            gamePlay.player.playerScore = 0;

            for (int i = 0; i < gamePlay.asteroidList.Count; i++)
            {
                gamePlay.asteroidList.RemoveAt(i);
                i--;
            }

            for(int i = 0; i < gamePlay.alienList.Count; i++)
            {

                gamePlay.alienList.RemoveAt(i);
                i--;
            }

            for (int i = 0; i < gamePlay.needlerAlienList.Count; i++)
            {
                gamePlay.needlerAlienList.RemoveAt(i);
                i--;
            }

            sceneCurrent.Initialize();
            sceneCurrent.LoadContent();
        }
        /// TODO: Add your update logic here

        base.Update(gameTime);

    }



    /// <summary>
    /// This is called when the game should draw itself.
    /// </summary>
    /// <param name="gameTime">Provides a snapshot of timing values.</param>
    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.Black);

        /// TODO: Add your drawing code here

        spriteBatch.Begin(); //Start displaying graphical elements...
```

```
            sceneCurrent.Draw(spriteBatch);

            spriteBatch.End();


            base.Draw(gameTime);
        }
    }
}
```

Scene.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;
using System.Timers;

namespace GAAR_Game
{
    public abstract class Scene
    {
        public virtual void Initialize()
        {
        }

        public virtual void Update(GameTime gameTime)
        {
        }

        public virtual void LoadContent()
        {
        }

        public virtual void UnloadContent()
        {
        }

        public virtual void Draw(SpriteBatch spriteBatch)
        {
        }


    }
}
```

MainMenu.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;
using System.IO;
using System.Timers;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Media;
using System.Threading;


namespace GAAR_Game
{
    public class MainMenu : Scene
    {
        public GraphicsDeviceManager graphicsMainMenu;
        public ContentManager mainMenuContent;

        public Vector2 mainMenuBackgroundPosition;
        public Texture2D mainMenuBackgroundTexture;

        public Vector2 mainMenuLogoPosition;
        public Texture2D mainMenuLogoTexture;

        public Texture2D buttonButtonJumpInTextureDefault;
        public Texture2D buttonButtonJumpInTextureWhenMouseIsOn;
        public Texture2D actualButtonJumpInTexture;
        public Vector2 actualButtonJumpInPosition;

        public Texture2D buttonShowInfosDefault;
        public Texture2D buttonShowInfosCursorOn;
        public Texture2D actualButtonShowInfosTexture;
        public Vector2 actualButtonShowInfosPosition;

        public Texture2D buttonQuitShowInfosSceneTextureDefault;
        public Texture2D buttonQuitShowInfosSceneTextureWhenMouseIsOn;
        public Texture2D actualButtonQuitShowInfosSceneTexture;
        public Vector2 actualButtonQuitShowInfosScenePosition;

        public Texture2D buttonMusicIsOnTexture;
        public Texture2D buttonMusicIsOffTexture;
        public Texture2D actualButtonMusicTexture;
        public Vector2 actuelButtonMusicPosition;
        public bool stateButtonMusic = true;
        public bool isInformationsSceneIsVisible = false;
        public Texture2D backgroundInfosSceneTexture;
        public Vector2 backgroundInfosScenePosition;
        public SpriteFont MainMenuCreditsFont;
        public SpriteFont MenuInformationsFont;

        public Texture2D easterEggTexture0;
        public Texture2D easterEggTexture1;
        public Texture2D easterEggTexture2;
        public Texture2D easterEggTexture3;
        public Texture2D easterEggTexture4;
        public Texture2D easterEggTexture5;
        public Texture2D easterEggTexture6;
        public Texture2D easterEggTexture7;
```

```
        public Texture2D easterEggTexture8;
        public Texture2D easterEggTexture9;
        public Texture2D easterEggTexture10;
        public Texture2D easterEggTexture11;
        public Texture2D easterEggTexture12;

        public SoundEffect clickSound;

        public bool showMainMenuGraphicalElements = true;


        public double timePassedGenerationEasterEgg = 0;

        MouseState lastMouseState;

        public bool isGameIsAboutToBeLaunched = false;
        public bool gameMusicIsStopped = false;

        public Song afterHoursRun;

        List<EasterEgg> easterEgglist = new List<EasterEgg>();
public MainMenu(GraphicsDeviceManager graphics, ContentManager content)
        {
            graphicsMainMenu = graphics;
            mainMenuContent = content;
        }


        public override void Initialize()
        {
            mainMenuBackgroundPosition = new Vector2(0, 0);
            mainMenuLogoPosition = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 - 290,
graphicsMainMenu.PreferredBackBufferHeight / 2 - 135);
            backgroundInfosScenePosition = new Vector2(0, 0);
        }

        public override void Update(GameTime gameTime)
        {
            MouseState actualMouseState = Mouse.GetState();
            actualButtonJumpInPosition = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 -
actualButtonJumpInTexture.Width / 2 - actualButtonShowInfosTexture.Width / 2,
graphicsMainMenu.PreferredBackBufferHeight / 2 + 250);
            Rectangle areaJumpInButton = new
Rectangle((int)actualButtonJumpInPosition.X, (int)actualButtonJumpInPosition.Y,
actualButtonJumpInTexture.Width, actualButtonJumpInTexture.Height);

            actuelButtonMusicPosition = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth - actualButtonMusicTexture.Width -
10, 10);
            Rectangle areaMusicButton = new
Rectangle((int)actuelButtonMusicPosition.X, (int)actuelButtonMusicPosition.Y,
actualButtonMusicTexture.Width, actualButtonMusicTexture.Height);

            actualButtonShowInfosPosition = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 -
actualButtonShowInfosTexture.Width / 2 + actualButtonJumpInTexture.Width / 2,
graphicsMainMenu.PreferredBackBufferHeight / 2 + 250);
            Rectangle areaShowInfosButton = new
Rectangle((int)actualButtonShowInfosPosition.X, (int)actualButtonShowInfosPosition.Y,
actualButtonShowInfosTexture.Width, actualButtonShowInfosTexture.Height);
```

```
            actualButtonQuitShowInfosScenePosition = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth -
actualButtonQuitShowInfosSceneTexture.Width, 0);
            Rectangle areaQuitLeaderBoardButton = new
Rectangle((int)actualButtonQuitShowInfosScenePosition.X,
(int)actualButtonQuitShowInfosScenePosition.Y,
actualButtonQuitShowInfosSceneTexture.Width,
actualButtonQuitShowInfosSceneTexture.Height);


            var mousePosition = new Point(actualMouseState.X, actualMouseState.Y);
            bool clickedMouseButton = actualMouseState.LeftButton ==
ButtonState.Pressed && lastMouseState.LeftButton == ButtonState.Released;

            if (areaJumpInButton.Contains(mousePosition))
            {
                actualButtonJumpInTexture = buttonButtonJumpInTextureWhenMouseIsOn;

                if (clickedMouseButton)
                {
                    clickSound.Play();
                    isGameIsAboutToBeLaunched = true;
                    MediaPlayer.Stop();
                }
            }

            else
            {
                actualButtonJumpInTexture = buttonButtonJumpInTextureDefault;
            }

            if (areaMusicButton.Contains(mousePosition) &&
isInformationsSceneIsVisible == false)
            {
                if (clickedMouseButton == true)
                {
                    clickSound.Play();
                    if (actualButtonMusicTexture == buttonMusicIsOnTexture)
                    {
                        actualButtonMusicTexture = buttonMusicIsOffTexture;
                        MediaPlayer.Volume = 0.0f;
                        gameMusicIsStopped = true;
                    }
                    else
                    {
                        actualButtonMusicTexture = buttonMusicIsOnTexture;
                        MediaPlayer.Volume = 0.8f;
                        gameMusicIsStopped = false;
                    }

                }
            }

            if (areaShowInfosButton.Contains(mousePosition))
            {
                actualButtonShowInfosTexture = buttonShowInfosCursorOn;

                if (clickedMouseButton)
                {
                    clickSound.Play();
                    showMainMenuGraphicalElements = false;
```

```
                    isInformationsSceneIsVisible = true;
                    clickedMouseButton = false;
                }

            }
            else
            {
                actualButtonShowInfosTexture = buttonShowInfosDefault;
            }

            if (isInformationsSceneIsVisible == true)
            {
                actualButtonQuitShowInfosSceneTexture =
buttonQuitShowInfosSceneTextureDefault;
                if (areaQuitLeaderBoardButton.Contains(mousePosition))
                {
                    actualButtonQuitShowInfosSceneTexture =
buttonQuitShowInfosSceneTextureWhenMouseIsOn;

                    if (clickedMouseButton)
                    {
                        clickSound.Play();
                        isInformationsSceneIsVisible = false;
                        showMainMenuGraphicalElements = true;
                        clickedMouseButton = false;
                    }
                }
                else
                {
                    actualButtonQuitShowInfosSceneTexture =
buttonQuitShowInfosSceneTextureDefault;
                }

            }

            timePassedGenerationEasterEgg = timePassedGenerationEasterEgg +
gameTime.ElapsedGameTime.TotalSeconds;

            if (timePassedGenerationEasterEgg > 6) //I call that " the falling stars "
or " the falling easters eggs
                {
                    GenerateEasterEgg();
                    timePassedGenerationEasterEgg = 0;
                }

                UpdateEasterEgg();

                lastMouseState = Mouse.GetState();
        }
public override void LoadContent()
        {
            mainMenuBackgroundTexture =
mainMenuContent.Load<Texture2D>("game_menu_wallpaper");
            mainMenuLogoTexture = mainMenuContent.Load<Texture2D>("GAAR_logo");
            afterHoursRun = mainMenuContent.Load<Song>("after-hours-run");

            buttonButtonJumpInTextureDefault =
mainMenuContent.Load<Texture2D>("jump_in_button_static");
            buttonButtonJumpInTextureWhenMouseIsOn =
mainMenuContent.Load<Texture2D>("jump_in_button_cursor_on");
            actualButtonJumpInTexture = buttonButtonJumpInTextureDefault;
```

```
            buttonMusicIsOnTexture = mainMenuContent.Load<Texture2D>("music_on_icon");
            buttonMusicIsOffTexture =
mainMenuContent.Load<Texture2D>("music_off_icon");
            actualButtonMusicTexture = buttonMusicIsOnTexture;

            buttonShowInfosDefault =
mainMenuContent.Load<Texture2D>("showInfos_default");
            buttonShowInfosCursorOn =
mainMenuContent.Load<Texture2D>("showInfos_cursoron");
            actualButtonShowInfosTexture = buttonShowInfosDefault;

            buttonQuitShowInfosSceneTextureDefault =
mainMenuContent.Load<Texture2D>("quitLeaderboard_default");
            buttonQuitShowInfosSceneTextureWhenMouseIsOn =
mainMenuContent.Load<Texture2D>("quitLeaderboard_cursoron");
            actualButtonQuitShowInfosSceneTexture =
buttonQuitShowInfosSceneTextureDefault;
            backgroundInfosSceneTexture =
mainMenuContent.Load<Texture2D>("backgroundInfos");
            MainMenuCreditsFont = mainMenuContent.Load<SpriteFont>("MainMenuCredits");
            MenuInformationsFont =
mainMenuContent.Load<SpriteFont>("MenuInformations");

            easterEggTexture0 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_117");
            easterEggTexture1 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_banana");
            easterEggTexture2 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_linearsystems");
            easterEggTexture3 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_polytech");
            easterEggTexture4 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_kiki");
            easterEggTexture5 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_frog");
            easterEggTexture6 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_oris");
            easterEggTexture7 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_crepes");
            easterEggTexture8 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_ori");
            easterEggTexture9 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_tudal");
            easterEggTexture10 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_medal");
            easterEggTexture11 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_foot");
            easterEggTexture12 =
mainMenuContent.Load<Texture2D>("game_menu_surprise_cat");

            clickSound = mainMenuContent.Load<SoundEffect>("clickSound");

            MediaPlayer.Play(afterHoursRun);
            MediaPlayer.Volume = 0.8f;

        }

        public override void UnloadContent()
        {
        }
public override void Draw(SpriteBatch spriteBatch)
        {
```

```
            if (showMainMenuGraphicalElements == true)
            {
                spriteBatch.Draw(mainMenuBackgroundTexture, new
Rectangle((int)mainMenuBackgroundPosition.X, (int)mainMenuBackgroundPosition.Y,
mainMenuBackgroundTexture.Width, mainMenuBackgroundTexture.Height), Color.White);
                Rectangle mainMenuLogoRectangleTexture = new Rectangle(0, 0,
mainMenuLogoTexture.Width, mainMenuLogoTexture.Height);
                Vector2 mainMenuLogoOrigin = new Vector2(mainMenuLogoTexture.Width /
2, mainMenuLogoTexture.Height / 2);
                spriteBatch.Draw(mainMenuLogoTexture, mainMenuLogoPosition,
mainMenuLogoRectangleTexture, Color.White, 0, mainMenuLogoOrigin, 1.55f,
SpriteEffects.None, 1);
                spriteBatch.Draw(actualButtonJumpInTexture, new
Rectangle((int)actualButtonJumpInPosition.X, (int)actualButtonJumpInPosition.Y,
actualButtonJumpInTexture.Width, actualButtonJumpInTexture.Height), Color.White);
                spriteBatch.Draw(actualButtonMusicTexture, new
Rectangle((int)actuelButtonMusicPosition.X, (int)actuelButtonMusicPosition.Y,
actualButtonMusicTexture.Width, actualButtonMusicTexture.Height), Color.White);
                spriteBatch.Draw(actualButtonShowInfosTexture, new
Rectangle((int)actualButtonShowInfosPosition.X, (int)actualButtonShowInfosPosition.Y,
actualButtonShowInfosTexture.Width, actualButtonShowInfosTexture.Height),
Color.White);

                string MessageCredits = " ALL RIGHTS RESERVED TO THE ORIGINAL CREATOR
OF THE GAME 'ASTEROIDS'. FOR EDUCATIONAL PURPOSE ONLY. POLYTECH ORLEANS - 2020 ";
                Vector2 sizeOfMessageCredits =
MainMenuCreditsFont.MeasureString(MessageCredits);
                Vector2 positionMessageCredits = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 - sizeOfMessageCredits.X / 2,
graphicsMainMenu.PreferredBackBufferHeight - 25);
                spriteBatch.DrawString(MainMenuCreditsFont, MessageCredits,
positionMessageCredits, Color.Gray);

                string MessageSongPlayed = " Song Played : AFTER HOURS RUN - MITCH
MURDER ";
                Vector2 sizeOfMessageSongPlayed =
MenuInformationsFont.MeasureString(MessageSongPlayed);
                Vector2 positionMessageSongPlayed = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 + sizeOfMessageSongPlayed.X / 2
+ 250, +27);
                spriteBatch.DrawString(MenuInformationsFont, MessageSongPlayed,
positionMessageSongPlayed, Color.CadetBlue);

                string developedBy = " A (awesome) game developed by Killian SINGLA ";
                Vector2 sizeOfDevelopedBy =
MenuInformationsFont.MeasureString(developedBy);
                Vector2 positionDevelopedBy = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 - sizeOfDevelopedBy.X / 2,
graphicsMainMenu.PreferredBackBufferHeight / 2 + 425);
                spriteBatch.DrawString(MenuInformationsFont, developedBy,
positionDevelopedBy, Color.LightSkyBlue);

                string thanksTo = " Special thanks to Remy Leconge, all of my friends
from Polytech Orleans and elsewhere, and my family for their help and their support ";
                Vector2 sizeThanksTo = MenuInformationsFont.MeasureString(thanksTo);
                Vector2 positionSizeThanksTo = new
Vector2(graphicsMainMenu.PreferredBackBufferWidth / 2 - sizeThanksTo.X / 2,
graphicsMainMenu.PreferredBackBufferHeight / 2 + 445);
                spriteBatch.DrawString(MenuInformationsFont, thanksTo,
positionSizeThanksTo, Color.WhiteSmoke);

                foreach (EasterEgg easterEgg in easterEgglist)
```

```
                {
                    easterEgg.Draw(spriteBatch);
                }

            }

            else if (isInformationsSceneIsVisible == true)
            {
                spriteBatch.Draw(backgroundInfosSceneTexture, new
Rectangle((int)backgroundInfosScenePosition.X, (int)backgroundInfosScenePosition.Y,
backgroundInfosSceneTexture.Width, backgroundInfosSceneTexture.Height), Color.White);
                spriteBatch.Draw(actualButtonQuitShowInfosSceneTexture, new
Rectangle((int)actualButtonQuitShowInfosScenePosition.X,
(int)actualButtonQuitShowInfosScenePosition.Y,
actualButtonQuitShowInfosSceneTexture.Width,
actualButtonQuitShowInfosSceneTexture.Height), Color.White);
            }
        }
public void GenerateEasterEgg()
        {
            Random rand = new Random();
            int randomTexture = rand.Next(0, 39);
            int randomAngle = rand.Next(-3, 3);
            int movingSense;

            int random_value_Y1 = rand.Next(50, 100);
            int random_value_Y2 = rand.Next(0, 500);
            if (random_value_Y2 % 2 == 0)
            {
                random_value_Y1 = -random_value_Y1;
                movingSense = +1;
            }
            else
            {
                random_value_Y1 = graphicsMainMenu.PreferredBackBufferHeight +
random_value_Y1;
                movingSense = -1;

            }


            if ( 0 < randomTexture && randomTexture < 3)
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture0);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 - 290;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);

            }

            if (3 < randomTexture && randomTexture < 6)
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture1);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 -300;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);
            }
```

```
if (6 < randomTexture && randomTexture < 9)
{
    EasterEgg easterEgg = new EasterEgg(easterEggTexture2);
    easterEgg.easterEggAngle = randomAngle;
    easterEgg.easterEggPosition.X = 1920 - 300;
    easterEgg.easterEggPosition.Y = random_value_Y1;
    easterEgg.easterEggMovingSense = movingSense;
    easterEgglist.Add(easterEgg);
}


if (9 < randomTexture && randomTexture < 12)
{
    EasterEgg easterEgg = new EasterEgg(easterEggTexture3);
    easterEgg.easterEggAngle = randomAngle;
    easterEgg.easterEggPosition.X = 1920 - 300;
    easterEgg.easterEggPosition.Y = random_value_Y1;
    easterEgg.easterEggMovingSense = movingSense;
    easterEgglist.Add(easterEgg);
}

if (12 < randomTexture && randomTexture < 15)
{
    EasterEgg easterEgg = new EasterEgg(easterEggTexture4);
    easterEgg.easterEggAngle = randomAngle;
    easterEgg.easterEggPosition.X = 1920 - 300;
    easterEgg.easterEggPosition.Y = random_value_Y1;
    easterEgg.easterEggMovingSense = movingSense;
    easterEgglist.Add(easterEgg);
}

if (15 < randomTexture && randomTexture < 18)
{
    EasterEgg easterEgg = new EasterEgg(easterEggTexture5);
    easterEgg.easterEggAngle = randomAngle;
    easterEgg.easterEggPosition.X = 1920 - 300;
    easterEgg.easterEggPosition.Y = random_value_Y1;
    easterEgg.easterEggMovingSense = movingSense;
    easterEgglist.Add(easterEgg);
}

if (18 < randomTexture && randomTexture < 21)
{
    EasterEgg easterEgg = new EasterEgg(easterEggTexture6);
    easterEgg.easterEggAngle = randomAngle;
    easterEgg.easterEggPosition.X = 1920 - 300;
    easterEgg.easterEggPosition.Y = random_value_Y1;
    easterEgg.easterEggMovingSense = movingSense;
    easterEgglist.Add(easterEgg);
}

if (21 < randomTexture && randomTexture < 24)
{
    EasterEgg easterEgg = new EasterEgg(easterEggTexture7);
    easterEgg.easterEggAngle = randomAngle;
    easterEgg.easterEggPosition.X = 1920 - 300;
    easterEgg.easterEggPosition.Y = random_value_Y1;
    easterEgg.easterEggMovingSense = movingSense;
    easterEgglist.Add(easterEgg);
}

if (24 < randomTexture && randomTexture < 27)
```

```
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture8);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 - 300;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);
            }

            if (27 < randomTexture && randomTexture < 30)
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture9);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 - 300;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);
            }

            if (30 < randomTexture && randomTexture < 33)
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture10);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 - 300;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);
            }

            if (33 < randomTexture && randomTexture < 36)
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture11);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 - 300;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);
            }

            if (36 < randomTexture && randomTexture < 39)
            {
                EasterEgg easterEgg = new EasterEgg(easterEggTexture12);
                easterEgg.easterEggAngle = randomAngle;
                easterEgg.easterEggPosition.X = 1920 - 300;
                easterEgg.easterEggPosition.Y = random_value_Y1;
                easterEgg.easterEggMovingSense = movingSense;
                easterEgglist.Add(easterEgg);
            }



        }

        public void UpdateEasterEgg()
        {
            foreach (EasterEgg easterEgg in easterEgglist)
            {
                easterEgg.easterEggAngle = easterEgg.easterEggAngle - 0.03f;
                easterEgg.easterEggDirection = new
Vector2((float)Math.Sin(easterEgg.easterEggAngle),
(float)Math.Cos(easterEgg.easterEggAngle));
```

```
                easterEgg.easterEggPosition.Y = easterEgg.easterEggPosition.Y +
easterEgg.easterEggMovingSense * easterEgg.easterEggSpeed;

                if (easterEgg.easterEggPosition.Y >=
graphicsMainMenu.PreferredBackBufferHeight + 200 || easterEgg.easterEggPosition.Y <= -
200)
                {
                    Console.WriteLine("disapear");
                    easterEgg.easterEggIsVisible = false;
                }
            }

            for (int i = 0; i < easterEgglist.Count; i++)
            {
                if (easterEgglist[i].easterEggIsVisible == false)
                {
                    easterEgglist.RemoveAt(i);
                    i = i - 1;
                }
            }
        }


    }
}
```

EasterEgg.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;

namespace GAAR_Game
{
    class EasterEgg
    {

        public Texture2D easterEggTexture;
        public Vector2 easterEggPosition;
        public Vector2 easterEggOrigin;
        public Vector2 easterEggDirection;


        public Rectangle easterEggTextureRectangle;

        public float easterEggSpeed = 3.0f;
        public float easterEggAngle = 0;
        public int easterEggMovingSense;

        public bool easterEggIsVisible = true;


        public EasterEgg(Texture2D texture)
        {
            easterEggTexture = texture;
        }


        public void Draw(SpriteBatch spriteBatch)
        {
            easterEggTextureRectangle = new Rectangle(0, 0, easterEggTexture.Width,
easterEggTexture.Height);
            spriteBatch.Draw(easterEggTexture, easterEggPosition,
easterEggTextureRectangle, Color.White, easterEggAngle, easterEggOrigin, 1.0f,
SpriteEffects.None, 1);
        }
    }
}
```

GamePlay.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;
using System.Timers;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Media;
using System.Threading;

namespace GAAR_Game
{
    public class GamePlay : Scene
    {
        public Vector2 backgroundPosition;
        public Texture2D backgroundTexture;
        public Texture2D laserStripTexture;
        public Texture2D asteroidTexture0;
        public Texture2D asteroidTexture1;
        public Texture2D asteroidTexture2;
        public Texture2D alienTexture;
        public Texture2D needlerAlienTexture;
        public SpriteFont HUDfont;

        public double timePassedGenerationAsteroids = 0;
        public double timePassedGenerationAliens = 0;
        public double timePassedGenerationEasterEgg = 0;
        public double timePassedNeedlerShoot = 0;

        public Player player;

        KeyboardState pastState;

        public List<LaserStrip> laserStriplist = new List<LaserStrip>();

        public List<Asteroid> asteroidList = new List<Asteroid>();

        public List<Alien> alienList = new List<Alien>();

        public List<NeedlerAlien> needlerAlienList = new List<NeedlerAlien>();

        public Random randomNumber = new Random();
        public bool detectionFission = false;
        public int numberOfAsteroidsToGenerateAfterFission;
        public Asteroid asteroidSplit;

        GraphicsDeviceManager graphicsGamePlay;
        ContentManager gamePlayContent;

        public SoundEffect laserSound;
        public SoundEffect alienSound;
        public SoundEffect needlerSound;
        public SoundEffect explosionSound;
        public SoundEffect alienIsDestroyedSound;
        public SoundEffect playerIsTouched;

        public Song guile;

        public bool isGameMusicStopped;
```

```
        public GamePlay(GraphicsDeviceManager graphics, ContentManager content)
        {
            graphicsGamePlay = graphics;
            gamePlayContent = content;
            player = new Player();
        }

        public override void Initialize()
        {
            backgroundPosition = new Vector2(0, 0); // 0,0 for the upper-left corner
of the window
            Vector2 position = new Vector2(graphicsGamePlay.PreferredBackBufferWidth /
2, graphicsGamePlay.PreferredBackBufferHeight / 2);
            player.playerPosition = position;
            player.playerAngle = 0;
            base.Initialize();
        }

public override void Update(GameTime gameTime)
        {
            KeyboardState state = Keyboard.GetState();

            if (state.IsKeyDown(Keys.Left)) //LEFT KEY : Rotation to the left
            {
                player.rotateLeft();
            }

            if (state.IsKeyDown(Keys.Right)) //RIGHT KEY : Rotation to the right
            {
                player.rotateRight();
            }

            if (state.IsKeyDown(Keys.Up)) //UP BUTTON : MOVE
            {
                player.move(graphicsGamePlay.PreferredBackBufferWidth,
graphicsGamePlay.PreferredBackBufferHeight);
            }

            if (state.IsKeyDown(Keys.Space) && pastState.IsKeyUp(Keys.Space)) //SPACE
BUTTON : SHOT
            {
                laserSound.Play();
                player.shoot(laserStriplist, laserStripTexture);
            }

            UpdateLaserStrip();

            timePassedGenerationAsteroids = timePassedGenerationAsteroids +
gameTime.ElapsedGameTime.TotalSeconds;
            if (timePassedGenerationAsteroids > 3) //Each 3 seconds, 7 asteroids of
random types spawn on the map
            {
                timePassedGenerationAsteroids = 0;
                for (int i = 0; i < 7; i++)
                {
                    GenerateRandomAsteroid();
                }
            }

            timePassedGenerationAliens = timePassedGenerationAliens +
gameTime.ElapsedGameTime.TotalSeconds;
```

```
            if (timePassedGenerationAliens > 15) //Alien Incoming !!!
            {
                timePassedGenerationAliens = 0;
                GenerateAlien();
            }

            if (timePassedGenerationEasterEgg > 60) //I call that " the falling stars
" or " the falling easters egss
            {
                /* GenerateEasterEgg */
            }

            double timePassed = timePassedGenerationAliens +
gameTime.ElapsedGameTime.TotalSeconds;

            if (player.playerHealth <= 0)
            {
                Thread.Sleep(2000);
                player.isDead = true;
                MediaPlayer.Stop();

                /* Console.WriteLine("DEAD"); */
            }

            UpdateAsteroid();
            UpdateAlien(gameTime);
            UpdateNeedlerAlien();

            pastState = Keyboard.GetState();

        }
public override void LoadContent()
        {
            backgroundTexture = gamePlayContent.Load<Texture2D>("game_map_wallpaper");
// It's not mandatory to specify the file extension
            player.playerTexture = gamePlayContent.Load<Texture2D>("player_sprite");
            laserStripTexture = gamePlayContent.Load<Texture2D>("laser_strip");
            asteroidTexture0 = gamePlayContent.Load<Texture2D>("asteroid_little");
            asteroidTexture1 = gamePlayContent.Load<Texture2D>("asteroid_medium");
            asteroidTexture2 = gamePlayContent.Load<Texture2D>("asteroid_fat");
            alienTexture = gamePlayContent.Load<Texture2D>("alien_flying_saucer");
            needlerAlienTexture = gamePlayContent.Load<Texture2D>("needler_alien");
            HUDfont = gamePlayContent.Load<SpriteFont>("gameFont");
            laserSound = gamePlayContent.Load<SoundEffect>("laserSound");
            alienSound = gamePlayContent.Load<SoundEffect>("alienSound");
            needlerSound = gamePlayContent.Load<SoundEffect>("needlerSound");
            explosionSound = gamePlayContent.Load<SoundEffect>("explosionSound");
            alienIsDestroyedSound = gamePlayContent.Load<SoundEffect>("Killtacular");
            playerIsTouched = gamePlayContent.Load<SoundEffect>("health");
            guile = gamePlayContent.Load<Song>("GUILE'S THEME");

            if (isGameMusicStopped == false)
            {
                MediaPlayer.Play(guile);
                MediaPlayer.Volume = 0.7f;
            }

            base.LoadContent();
        }

        public override void UnloadContent()
        {
```

```
            gamePlayContent.Unload();
            base.UnloadContent();
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            //The rectangle indicates what part of the texture we want to draw.
            spriteBatch.Draw(backgroundTexture, new
Rectangle((int)backgroundPosition.X, (int)backgroundPosition.Y,
backgroundTexture.Width, backgroundTexture.Height), Color.White);
            //Give the size of the rectangle who will cover the texture (here the size
of the image)

            player.playerOrigin = new Vector2(player.playerTexture.Width / 2,
player.playerTexture.Height / 2);
            player.Draw(spriteBatch);
//Like in optics

            foreach (LaserStrip laserstrip in laserStriplist)
            {
                laserstrip.laserStripOrigin = new
Vector2(laserstrip.laserStripTexture.Width / 2, laserstrip.laserStripTexture.Height /
2);
                laserstrip.Draw(spriteBatch);
            }

            foreach (Asteroid asteroid in asteroidList)
            {
                asteroid.asteroidOrigin = new Vector2(asteroid.asteroidTexture.Width /
2, asteroid.asteroidTexture.Height / 2);
                asteroid.Draw(spriteBatch);
            }

            foreach (Alien alien in alienList)
            {
                alien.alienOrigin = new Vector2(alien.alienTexture.Width / 2,
alien.alienTexture.Height / 2);
                alien.Draw(spriteBatch);
            }

            foreach (NeedlerAlien needler in needlerAlienList)
            {
                needler.needlerAlienOrigin = new
Vector2(needler.needlerAlienTexture.Width / 2, needler.needlerAlienTexture.Height /
2);
                needler.Draw(spriteBatch);
            }


            Vector2 positionHUD = new Vector2(25, 25);
            spriteBatch.DrawString(HUDfont, "SCORE : " + player.playerScore.ToString()
+ "\n" + "HEALTH : " + player.playerHealth.ToString(), positionHUD, Color.White);
            base.Draw(spriteBatch);
        }

public void GenerateRandomAsteroid()
        {
            int random_value_texture = randomNumber.Next(0, 3); //RANGE TO 3 BECAUSE
WITH (0,2), 2 ISN'T CHOSE AT ALL
            float random_value_angle = randomNumber.Next(-3, 3);

            Asteroid asteroid = new Asteroid(random_value_texture, asteroidTexture0,
asteroidTexture1, asteroidTexture2);
```

```
            int random_value_X1 = randomNumber.Next(0, 125);
            int random_value_X2 = randomNumber.Next(0, 500);
            if (random_value_X2 % 2 == 0)
            {
                random_value_X1 = -random_value_X1;
            }
            else
            {
                random_value_X1 = graphicsGamePlay.PreferredBackBufferWidth +
random_value_X1;
            }
            int random_value_Y1 = randomNumber.Next(0, 125);
            int random_value_Y2 = randomNumber.Next(0, 500);
            if (random_value_Y2 % 2 == 0)
            {
                random_value_Y1 = -random_value_Y1;
            }
            else
            {
                random_value_Y1 = graphicsGamePlay.PreferredBackBufferHeight +
random_value_Y1;
            }

            if (-125 < random_value_X1 && random_value_X1 < 125 && -125 <
random_value_Y1 && random_value_Y1 < 125) //If the asteroid spawn in the upper-left
corner
            {
                asteroid.asteroidcoefTrajectory0 = randomNumber.Next(80, 120);
                asteroid.asteroidcoefTrajectory1 = 1;
                asteroid.asteroidMovingSense = 1;
                //Than this trajectory is different. Here it's an increasing affinity
function
            }

            if (-125 < random_value_X1 && random_value_X1 < 125 &&
graphicsGamePlay.PreferredBackBufferHeight - 125 < random_value_Y1 && random_value_Y1
< graphicsGamePlay.PreferredBackBufferHeight + 125) //If the asteroid spawn in the
bottom-left corner
            {
                asteroid.asteroidcoefTrajectory0 =
randomNumber.Next(graphicsGamePlay.PreferredBackBufferHeight - 200,
graphicsGamePlay.PreferredBackBufferHeight + 200);
                asteroid.asteroidcoefTrajectory1 = -1;
                asteroid.asteroidMovingSense = 1;

                //Than this trajectory is different. Here it's an decreasing affinity
function
            }

            if (graphicsGamePlay.PreferredBackBufferWidth - 125 < random_value_X1 &&
random_value_X1 < graphicsGamePlay.PreferredBackBufferWidth + 125 && -125 <
random_value_Y1 && random_value_Y1 < 125) //If the asteroid spawn in the upper-right
corner
            {
                asteroid.asteroidcoefTrajectory0 =
randomNumber.Next(graphicsGamePlay.PreferredBackBufferWidth - 400,
graphicsGamePlay.PreferredBackBufferWidth + 400);
                asteroid.asteroidcoefTrajectory1 = -1;
                asteroid.asteroidMovingSense = -1;
```

```
                //Than this trajectory is different. Here it's an decreasing affinity
function
            }

            if (graphicsGamePlay.PreferredBackBufferWidth - 125 < random_value_X1 &&
random_value_X1 < graphicsGamePlay.PreferredBackBufferWidth + 125 &&
graphicsGamePlay.PreferredBackBufferHeight - 125 < random_value_Y1 && random_value_Y1
< graphicsGamePlay.PreferredBackBufferHeight + 125) //If the asteroid spawn in the
bottom-right corner
            {
                asteroid.asteroidcoefTrajectory0 = randomNumber.Next(-400, -200);
                asteroid.asteroidcoefTrajectory1 = 1;
                asteroid.asteroidMovingSense = -1;

                //Than this trajectory is different. Here it's an increasing affinity
function
            }

            asteroid.asteroidAngle = random_value_angle;

            asteroid.asteroidPosition.X = random_value_X1;
            asteroid.asteroidPosition.Y = random_value_Y1;


            asteroid.asteroidDirection = new
Vector2((float)Math.Cos(asteroid.asteroidAngle),
(float)Math.Sin(asteroid.asteroidAngle));
            asteroidList.Add(asteroid);
        }
public void GenerateAlien()
        {
            Alien alien = new Alien(alienTexture);
            alienSound.Play();
            Random randomPositionAlien = new Random();
            if (randomPositionAlien.Next(0, 10) % 2 == 0)
            {
                alien.alienMovingSense = -1;
                alien.alienPosition.X = graphicsGamePlay.PreferredBackBufferWidth +
10;
                alien.alienPosition.Y = randomPositionAlien.Next(-200, 200) +
graphicsGamePlay.PreferredBackBufferHeight / 2;
            }
            else
            {
                alien.alienMovingSense = 1;
                alien.alienPosition.X = -10;
                alien.alienPosition.Y = randomPositionAlien.Next(-200, 200) +
graphicsGamePlay.PreferredBackBufferHeight / 2;

            }

            alienList.Add(alien);
            alien.alienIsVisible = true;
        }

        public void UpdateLaserStrip()
        {
            foreach (LaserStrip laserStrip in laserStriplist)
            {

                laserStrip.laserStripColisionsCircle = new
CollisionsCircle(laserStrip.laserStripPosition, laserStrip.laserStripTexture.Width);
```

```
                laserStrip.laserStripPosition.X = laserStrip.laserStripPosition.X +
laserStrip.laserStripSpeed * laserStrip.laserStripDirection.X;
                laserStrip.laserStripPosition.Y = laserStrip.laserStripPosition.Y -
laserStrip.laserStripSpeed * laserStrip.laserStripDirection.Y;

                if (laserStrip.laserStripPosition.X >= backgroundTexture.Width ||
laserStrip.laserStripPosition.X <= 0)
                {
                    laserStrip.laserStripIsVisible = false;
                }

                if (laserStrip.laserStripPosition.Y >= backgroundTexture.Height ||
laserStrip.laserStripPosition.Y <= 0)
                {
                    laserStrip.laserStripIsVisible = false;
                }
            }

            for (int i = 0; i < laserStriplist.Count; i++)
            {
                if (laserStriplist[i].laserStripIsVisible == false)
                {
                    laserStriplist.RemoveAt(i);
                    i = i - 1;
                }
            }
        }

        public void UpdateAsteroid()
        {
            player.playerColisionsCircle = new CollisionsCircle(player.playerPosition,
player.playerTexture.Width / 2);

            foreach (Asteroid asteroid in asteroidList)
            {
                asteroid.asteroidAngle = asteroid.asteroidAngle - 0.01f;
                asteroid.asteroidColisionsCircle = new
CollisionsCircle(asteroid.asteroidPosition, asteroid.asteroidTexture.Width / 2);
                asteroid.asteroidDirection = new
Vector2((float)Math.Sin(asteroid.asteroidAngle),
(float)Math.Cos(asteroid.asteroidAngle));

                asteroid.asteroidPosition.X = asteroid.asteroidPosition.X +
asteroid.asteroidMovingSense * asteroid.asteroidSpeed;
                asteroid.asteroidPosition.Y = asteroid.asteroidcoefTrajectory1 *
asteroid.asteroidPosition.X + asteroid.asteroidcoefTrajectory0;

                if
(asteroid.asteroidColisionsCircle.ColliderDetector(player.playerColisionsCircle) ==
true)
                {
                    explosionSound.Play();
                    playerIsTouched.Play();
                    asteroid.asteroidIsVisible = false;
                    player.playerHealth = player.playerHealth - 1;
                    /* player.playerScore = player.playerScore - 1000; */
                }

                foreach (LaserStrip laserStrip in laserStriplist)
                {
```

```
                    if
(asteroid.asteroidColisionsCircle.ColliderDetector(laserStrip.laserStripColisionsCircl
e) == true)
                    {
                        explosionSound.Play();

                        laserStrip.laserStripIsVisible = false;

                        if (asteroid.asteroidType == 0) //If the asteroid is a tiny
asteroid, + 150 points, gg !
                        {
                            player.playerScore = player.playerScore + 150;
                        }

                        if (asteroid.asteroidType == 1) //If the asteroid is a medium
asteroid, he'll selft destruct in 2 tiny asteroid + 50 points, gg !
                        {
                            player.playerScore = player.playerScore + 50;
                            detectionFission = true;
                            asteroidSplit = asteroid;
                            numberOfAsteroidsToGenerateAfterFission = 2;
                        }


                        if (asteroid.asteroidType == 2) //If the asteroid is a fat
asteroid, he'll selft destruct in 3 tiny asteroids and + 10 points, gg !
                        {
                            player.playerScore = player.playerScore + 10; // + 10
points, gg !
                            detectionFission = true;
                            asteroidSplit = asteroid;
                            numberOfAsteroidsToGenerateAfterFission = 3;
                        }



                        asteroid.asteroidIsVisible = false;

                    }

                    if (asteroid.asteroidPosition.X >=
graphicsGamePlay.PreferredBackBufferWidth + 200 || asteroid.asteroidPosition.X <= -
200)
                    {
                        asteroid.asteroidIsVisible = false;
                    }

                    if (asteroid.asteroidPosition.Y >=
graphicsGamePlay.PreferredBackBufferHeight + 200 || asteroid.asteroidPosition.Y <= -
200)
                    {
                        asteroid.asteroidIsVisible = false;
                    }
                }

            }

            if (detectionFission == true)
            {
                asteroidSplit.fission(asteroidList, asteroidTexture0,
asteroidTexture1, asteroidTexture2, numberOfAsteroidsToGenerateAfterFission);
                detectionFission = false;
```

```
            }

            for (int i = 0; i < asteroidList.Count; i++)
            {
                if (asteroidList[i].asteroidIsVisible == false)
                {
                    asteroidList.RemoveAt(i);
                    i = i - 1;
                }
            }


        }
public void UpdateAlien(GameTime gameTime)
        {
            foreach (Alien alien in alienList)
            {
                timePassedNeedlerShoot = timePassedNeedlerShoot +
gameTime.ElapsedGameTime.TotalSeconds;
                alien.alienColisionsCircle = new CollisionsCircle(alien.alienPosition,
alien.alienTexture.Width / 2);
                alien.alienPosition.X = alien.alienPosition.X + alien.alienMovingSense
* alien.alienSpeed;
                alien.alienPosition.Y = 200 * (float)Math.Sin(2 * Math.PI * 1 / 2000 *
alien.alienPosition.X) + graphicsGamePlay.PreferredBackBufferHeight / 2;

                if (alien.alienPosition.X <= -100 || alien.alienPosition.X >=
backgroundTexture.Width + 100)
                {
                    alien.alienIsVisible = false;
                }

                if (timePassedNeedlerShoot > 0.25) //Each 1/2s, the alien shoot a
needler.
                {
                    timePassedNeedlerShoot = 0;
                    alien.Shoot(needlerAlienList, needlerAlienTexture);
                    needlerSound.Play();
                }

                foreach (LaserStrip laserStrip in laserStriplist)
                {
                    if
(alien.alienColisionsCircle.ColliderDetector(laserStrip.laserStripColisionsCircle) ==
true)
                    {
                        alienIsDestroyedSound.Play();
                        alien.alienIsVisible = false;
                        player.playerScore = player.playerScore + 117;
                    }
                }

                if
(alien.alienColisionsCircle.ColliderDetector(player.playerColisionsCircle) == true)
                {
                    alien.alienIsVisible = false;
                    player.playerHealth = player.playerHealth - 1;
                    playerIsTouched.Play();

                }

            }
```

```
            for (int i = 0; i < alienList.Count; i++)
            {
                if (alienList[i].alienIsVisible == false)
                {
                    alienList.RemoveAt(i);
                    i--;
                }
            }

        }


        public void UpdateNeedlerAlien()
        {
            foreach (NeedlerAlien needler in needlerAlienList)
            {

                needler.needlerAlienColisionsCircle = new
CollisionsCircle(needler.needlerAlienPosition, needler.needlerAlienTexture.Width);

                needler.needlerAlienPosition.Y = needler.needlerAlienPosition.Y -
needler.needlerAlienSpeed;

                if (needler.needlerAlienPosition.X >= backgroundTexture.Width ||
needler.needlerAlienPosition.X <= 0)
                {
                    needler.needlerAlienIsVisible = false;
                }

                if (needler.needlerAlienPosition.Y >= backgroundTexture.Height ||
needler.needlerAlienPosition.Y <= 0)
                {
                    needler.needlerAlienIsVisible = false;
                }


                if
(needler.needlerAlienColisionsCircle.ColliderDetector(player.playerColisionsCircle) ==
true)
                {
                    needler.needlerAlienIsVisible = false;
                    player.playerHealth = player.playerHealth - 1;
                    playerIsTouched.Play();

                }

            }

            for (int i = 0; i < needlerAlienList.Count; i++)
            {
                if (needlerAlienList[i].needlerAlienIsVisible == false)
                {
                    needlerAlienList.RemoveAt(i);
                    i = i - 1;
                }
            }
        }


    }
}
```

Player.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;

namespace GAAR_Game
{
    public class Player
    {
        public Vector2 playerPosition;
        public Texture2D playerTexture;
        public Rectangle playerTextureRectangle;
        public CollisionsCircle playerColisionsCircle;
        public float playerAngle;
        public int playerHealth =  5;
        public int playerScore = 0;
        public Vector2 playerOrigin;
        public Vector2 playerDirection;
        public float playerSpeed = 8.0f;
        public bool isDead = false;

        public Player()
        {
        }

        public Player(Vector2 position)
        {
            playerPosition = position;
        }

        public void rotateLeft()
        {
            playerAngle = playerAngle - 0.06f;
            playerDirection = new Vector2((float)Math.Sin(playerAngle),
(float)Math.Cos(playerAngle));
        }

        public void rotateRight()
        {
            playerAngle = playerAngle + 0.1f;
            playerDirection = new Vector2((float)Math.Sin(playerAngle),
(float)Math.Cos(playerAngle));
        }

        public void move(int width, int height) //I needs the background
        {

            playerDirection = new Vector2((float)Math.Sin(playerAngle),
(float)Math.Cos(playerAngle));

            if (playerPosition.X <= width)
            {
                playerPosition.X = playerPosition.X + playerSpeed * playerDirection.X;
            }
            else
            {
                playerPosition.X = 0;
            }
```

```
            if (playerPosition.X >= 0)
            {
                playerPosition.X = playerPosition.X + playerSpeed * playerDirection.X;
            }

            else
            {
                playerPosition.X = width;
            }


            if (playerPosition.Y >= 0)
            {
                playerPosition.Y = playerPosition.Y - playerSpeed * playerDirection.Y;
            }

            else
            {
                playerPosition.Y = height;
            }

            if (playerPosition.Y <= height)
            {
                playerPosition.Y = playerPosition.Y - playerSpeed * playerDirection.Y;
            }

            else
            {
                playerPosition.Y = 0;
            }
        }

        public void shoot(List<LaserStrip> laserList, Texture2D texture)
        {
            LaserStrip newLaserStrip = new LaserStrip(texture);
            newLaserStrip.laserStripAngle = playerAngle;
            newLaserStrip.laserStripSpeed = 17.0f;
            newLaserStrip.laserStripDirection = new
Vector2((float)Math.Sin(newLaserStrip.laserStripAngle),
(float)Math.Cos(newLaserStrip.laserStripAngle));
            newLaserStrip.laserStripPosition.X = playerPosition.X +
newLaserStrip.laserStripDirection.X * newLaserStrip.laserStripSpeed;
            newLaserStrip.laserStripPosition.Y = playerPosition.Y -
newLaserStrip.laserStripDirection.Y * newLaserStrip.laserStripSpeed;
            newLaserStrip.laserStripIsVisible = true;
            laserList.Add(newLaserStrip);
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            Rectangle playerRectangleTexture = new Rectangle(0, 0,
playerTexture.Width, playerTexture.Height);
            spriteBatch.Draw(playerTexture, playerPosition, playerRectangleTexture,
Color.White, playerAngle, playerOrigin, 1.0f, SpriteEffects.None, 1);

//Texture.....position........rectangle..............Color.......angle.........origin
.......scale factor.............depth
        }

    }

}
```

Asteroid.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GAAR_Game
{
    public class Asteroid
    {
        public int asteroidType;
        public Texture2D asteroidTexture;
        public Vector2 asteroidPosition;
        public Vector2 asteroidOrigin;
        public Vector2 asteroidDirection;

        public Rectangle asteroidTextureRectangle;
        public CollisionsCircle asteroidColisionsCircle;

        public float asteroidAngle;
        public float asteroidSpeed;
        public int asteroidcoefTrajectory0;
        public int asteroidcoefTrajectory1;
        public int asteroidMovingSense;

        public bool asteroidIsVisible;

        public Asteroid(int type, Texture2D texture0, Texture2D texture1, Texture2D
texture2)
        {
            asteroidType = type;

            if (type == 0)
            {
                asteroidTexture = texture0;
                asteroidSpeed = 8.0f;
            }

            if (type == 1)
            {
                asteroidTexture = texture1;
                asteroidSpeed = 5.0f;
            }

            if (type == 2)
            {
                asteroidTexture = texture2;
                asteroidSpeed = 3.0f;
            }

            asteroidIsVisible = true;
        }

        public void fission(List<Asteroid> list, Texture2D asteroid0, Texture2D
asteroid1, Texture2D asteroid2, int number)
        {
            Random changeTrajectory = new Random();
```

```
            for (int i = 0; i < number; i++) //3 tiny asteroids are generated if the
asteroid is a fat asteroid, 2 if it's a medium one
            {
                Asteroid asteroidResulting = new Asteroid(0, asteroid0, asteroid1,
asteroid2);
                asteroidResulting.asteroidPosition = asteroidPosition;
                asteroidResulting.asteroidAngle = asteroidAngle +
changeTrajectory.Next(-3, 3) ;
                asteroidResulting.asteroidOrigin = asteroidOrigin;
                asteroidResulting.asteroidDirection = asteroidDirection;
                asteroidResulting.asteroidcoefTrajectory0 = asteroidcoefTrajectory0 -
changeTrajectory.Next(-400, 400);
                asteroidResulting.asteroidcoefTrajectory1 = asteroidcoefTrajectory1;
                asteroidResulting.asteroidMovingSense = asteroidMovingSense;
                list.Add(asteroidResulting);
            }

            asteroidIsVisible = false;

        }

        public void Draw(SpriteBatch spriteBatch)
        {
            Rectangle asteroidRectangleTexture = new Rectangle(0, 0,
asteroidTexture.Width, asteroidTexture.Height);
            spriteBatch.Draw(asteroidTexture, asteroidPosition,
asteroidRectangleTexture, Color.White, asteroidAngle, asteroidOrigin, 1.0f,
SpriteEffects.None, 1);
        }


    }
}
```

Alien.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;

namespace GAAR_Game
{
    public class Alien
    {
        public Texture2D alienTexture;
        public Vector2 alienPosition;
        public Vector2 alienOrigin;

        public Rectangle alienTextureRectangle;
        public CollisionsCircle alienColisionsCircle;

        public float alienSpeed = 5.0f;
        public float alienAngle = 0;
        public int alienMovingSense;

        public bool alienIsVisible = false;


        public Alien(Texture2D texture)
        {
            alienTexture = texture;
        }

        public void Shoot(List<NeedlerAlien> needlerList, Texture2D texture)
        {
            NeedlerAlien needlerShot = new NeedlerAlien(texture);
            needlerShot.needlerAlienAngle = 0;
            needlerShot.needlerAlienSpeed = 23.0f;
            needlerShot.needlerAlienPosition.X = alienPosition.X;
            needlerShot.needlerAlienPosition.Y = alienPosition.Y;
            needlerShot.needlerAlienIsVisible = true;
            needlerList.Add(needlerShot);
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            alienTextureRectangle = new Rectangle(0, 0, alienTexture.Width,
alienTexture.Height);
            spriteBatch.Draw(alienTexture, alienPosition, alienTextureRectangle,
Color.White, alienAngle, alienOrigin, 1.0f, SpriteEffects.None, 1);
        }

    }
}
```

CollisionsCircle.cs

```
using Microsoft.Xna.Framework;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GAAR_Game
{
    public class CollisionsCircle
    {
        public Vector2 circleCenter;
        public float circleRadius;

        public bool isCollided = false;

        public CollisionsCircle(Vector2 centre, float radius)
        {
            circleCenter = centre;
            circleRadius = radius;
        }

        public bool ColliderDetector(CollisionsCircle target)
        {
            float distanceBetweenThem =  (float)Math.Sqrt((target.circleCenter.X -
circleCenter.X)* (target.circleCenter.X - circleCenter.X) + (target.circleCenter.Y -
circleCenter.Y)* (target.circleCenter.Y - circleCenter.Y));
            if (distanceBetweenThem < (circleRadius + target.circleRadius)) //PYTAGORE
            {
                isCollided = true;
            }

            else
            {
                isCollided = false;
            }

           return isCollided;
        }

    }
}
```

LaserStrip.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace GAAR_Game
{
    public class LaserStrip
    {
        public Vector2 laserStripPosition;
        public Texture2D laserStripTexture;

        public Vector2 laserStripOrigin;
        public Vector2 laserStripDirection;
        public float laserStripAngle;
        public float laserStripSpeed;

        public Rectangle laserStripTextureRectangle;
        public CollisionsCircle laserStripColisionsCircle;


        public bool laserStripIsVisible;

        public LaserStrip(Texture2D texture)
        {
            laserStripTexture = texture;
            laserStripIsVisible = false;
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            laserStripTextureRectangle = new Rectangle(0, 0, laserStripTexture.Width,
laserStripTexture.Height);
            spriteBatch.Draw(laserStripTexture, laserStripPosition,
laserStripTextureRectangle, Color.White, laserStripAngle, laserStripOrigin, 1.0f,
SpriteEffects.None, 1);
        }
    }
}
```

NeedlerAlien.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;

namespace GAAR_Game
{
    public class NeedlerAlien
    {
        public Vector2 needlerAlienPosition;
        public Texture2D needlerAlienTexture;

        public Vector2 needlerAlienOrigin;
        public float needlerAlienAngle;
        public float needlerAlienSpeed;

        public Rectangle needlerAlienTextureRectangle;
        public CollisionsCircle needlerAlienColisionsCircle;


        public bool needlerAlienIsVisible;

        public NeedlerAlien(Texture2D texture)
        {
            needlerAlienTexture = texture;
            needlerAlienIsVisible = false;
        }

        public void Draw(SpriteBatch spriteBatch)
        {
            needlerAlienTextureRectangle = new Rectangle(0, 0,
needlerAlienTexture.Width, needlerAlienTexture.Height);
            spriteBatch.Draw(needlerAlienTexture, needlerAlienPosition,
needlerAlienTextureRectangle, Color.White, needlerAlienAngle, needlerAlienOrigin,
1.0f, SpriteEffects.None, 1);
        }


    }
}
```

EndGame.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input; //This part of the framework manages keyboard
inputs
using System.Collections.Generic;
using System;
using System.IO;
using System.Timers;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Media;


namespace GAAR_Game
{
  public class EndGame : Scene
    {
        GraphicsDeviceManager graphicsEndGame;
        ContentManager endGameContent;

        public Vector2 endGameBackgroundPosition;
        public Texture2D endGameBackgroundTexture;

        public Texture2D buttonShowResultsTextureDefault;
        public Texture2D buttonShowResultsTextureWhenMouseIsOn;
        public Texture2D actualButtonShowResultsTexture;
        public Vector2 actualButtonShowResultsPosition;


        public Texture2D buttonReplayTextureDefault;
        public Texture2D buttonReplayTextureWhenMouseIsOn;
        public Texture2D actualButtonReplayTexture;
        public Vector2 actualButtonReplayPosition;


        public SpriteFont endGameMessageFont;
        public SpriteFont endGameScoreFont;

        public string stringToAddToTheFileScoreCollector;
        DateTime Today = DateTime.Today;
        MouseState lastMouseState;
        StreamWriter writer;
        StreamReader reader;

        bool showEndGameGraphicalElements = true;
        Texture2D leaderBoardBackgroundTexture;
        string leaderBoardString;
        Vector2 leaderBoardBackgroundPosition;

        public Texture2D buttonQuitLeaderBoardTextureDefault;
        public Texture2D buttonQuitLeaderBoardTextureWhenMouseIsOn;
        public Texture2D actualButtonQuitLeaderBoardTexture;
        public Vector2 actualButtonQuitLeaderBoardPosition;

        public SoundEffect clickSound;
        public Song goodbye;


        public bool LeaderBoardisVisible = false;
        public bool isGameMusicStopped;
```

```
        public bool wannaReplay = false;

        public int scoreToDisplay;
        public EndGame(int score, GraphicsDeviceManager graphics, ContentManager
content)
        {
                graphicsEndGame = graphics;
                endGameContent = content;
                scoreToDisplay = score;
        }

        public override void Initialize()
        {
            endGameBackgroundPosition = new Vector2(0, 0); // 0,0 for the upper-left
corner of the window
            leaderBoardBackgroundPosition = new Vector2(0, 0);
        }
  public override void Update(GameTime gameTime)
        {
            MouseState actualMouseState = Mouse.GetState();

            actualButtonShowResultsPosition = new
Vector2(graphicsEndGame.PreferredBackBufferWidth/2 -
actualButtonShowResultsTexture.Width/2 - actualButtonReplayTexture.Width / 2,
graphicsEndGame.PreferredBackBufferHeight/2 + 200);
            actualButtonReplayPosition = new
Vector2(graphicsEndGame.PreferredBackBufferWidth / 2 - actualButtonReplayTexture.Width
/2 + actualButtonShowResultsTexture.Width / 2,
graphicsEndGame.PreferredBackBufferHeight / 2 + 200);
            actualButtonQuitLeaderBoardPosition = new
Vector2(graphicsEndGame.PreferredBackBufferWidth -
actualButtonQuitLeaderBoardTexture.Width,0);

            Rectangle areaShowOldScoresButton = new
Rectangle((int)actualButtonShowResultsPosition.X,(int)actualButtonShowResultsPosition.
Y, actualButtonShowResultsTexture.Width, actualButtonShowResultsTexture.Height);
            Rectangle areaReplayButton = new
Rectangle((int)actualButtonReplayPosition.X, (int)actualButtonReplayPosition.Y,
actualButtonReplayTexture.Width, actualButtonReplayTexture.Height);
            Rectangle areaQuitLeaderBoardButton = new
Rectangle((int)actualButtonQuitLeaderBoardPosition.X,
(int)actualButtonQuitLeaderBoardPosition.Y, actualButtonQuitLeaderBoardTexture.Width,
actualButtonQuitLeaderBoardTexture.Height);

            var mousePosition = new Point(actualMouseState.X, actualMouseState.Y);

            bool clickedMouseButton = actualMouseState.LeftButton ==
ButtonState.Pressed && lastMouseState.LeftButton == ButtonState.Released;

            if (areaShowOldScoresButton.Contains(mousePosition))
            {
                actualButtonShowResultsTexture =
buttonShowResultsTextureWhenMouseIsOn;

                if (clickedMouseButton)
                {
                    clickSound.Play();
                    readTheScore();
                    clickedMouseButton = false;
                }

            }
```

```
            else
            {
                actualButtonShowResultsTexture = buttonShowResultsTextureDefault;
            }


            if (areaReplayButton.Contains(mousePosition))
            {
                actualButtonReplayTexture = buttonReplayTextureWhenMouseIsOn;

                if (clickedMouseButton)
                {
                    clickSound.Play();
                    wannaReplay = true;
                    clickedMouseButton = false;
                }
            }
            else
            {
                actualButtonReplayTexture = buttonReplayTextureDefault;
            }

            if (LeaderBoardisVisible == true)
            {
                actualButtonQuitLeaderBoardTexture =
buttonQuitLeaderBoardTextureDefault;

                if (areaQuitLeaderBoardButton.Contains(mousePosition))
                {
                    actualButtonQuitLeaderBoardTexture =
buttonQuitLeaderBoardTextureWhenMouseIsOn;

                    if (clickedMouseButton)
                    {
                        clickSound.Play();
                        LeaderBoardisVisible = false;
                        showEndGameGraphicalElements = true;
                        clickedMouseButton = false;
                    }
                }
                else
                {
                    actualButtonQuitLeaderBoardTexture =
buttonQuitLeaderBoardTextureDefault;
                }


            }

            lastMouseState = Mouse.GetState();
        }

        public override void LoadContent()
        {
            endGameBackgroundTexture =
endGameContent.Load<Texture2D>("endGame_wallpaper"); // It's not mandatory to specify
the file extension
            endGameMessageFont = endGameContent.Load<SpriteFont>("gameFont");
            endGameScoreFont = endGameContent.Load<SpriteFont>("endGameScoreFont");
            buttonShowResultsTextureDefault =
endGameContent.Load<Texture2D>("showOldScores_default");
```

```
            buttonShowResultsTextureWhenMouseIsOn =
endGameContent.Load<Texture2D>("showOldScores_withcursoron");
            buttonReplayTextureDefault =
endGameContent.Load<Texture2D>("replay_default");
            buttonReplayTextureWhenMouseIsOn =
endGameContent.Load<Texture2D>("replay_cursoron");

            actualButtonShowResultsTexture = buttonShowResultsTextureDefault;
            actualButtonReplayTexture = buttonReplayTextureDefault;

            leaderBoardBackgroundTexture =
endGameContent.Load<Texture2D>("leaderBoard_background");

            buttonQuitLeaderBoardTextureDefault =
endGameContent.Load<Texture2D>("quitLeaderboard_default");
            buttonQuitLeaderBoardTextureWhenMouseIsOn = endGameContent.Load
<Texture2D>("quitLeaderboard_cursoron");
            actualButtonQuitLeaderBoardTexture = buttonQuitLeaderBoardTextureDefault;

            clickSound = endGameContent.Load<SoundEffect>("clickSound");
            goodbye = endGameContent.Load<Song>("cornemuse");

            if (isGameMusicStopped == false)
            {
                MediaPlayer.Play(goodbye);
                MediaPlayer.Volume = 0.5f;
            }

        }

public override void UnloadContent()
        {
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (showEndGameGraphicalElements == true)
            {
                spriteBatch.Draw(endGameBackgroundTexture, new
Rectangle((int)endGameBackgroundPosition.X, (int)endGameBackgroundPosition.Y,
endGameBackgroundTexture.Width, endGameBackgroundTexture.Height), Color.White);
                spriteBatch.Draw(actualButtonShowResultsTexture, new
Rectangle((int)actualButtonShowResultsPosition.X,
(int)actualButtonShowResultsPosition.Y, actualButtonShowResultsTexture.Width,
actualButtonShowResultsTexture.Height), Color.White);
                spriteBatch.Draw(actualButtonReplayTexture, new
Rectangle((int)actualButtonReplayPosition.X, (int)actualButtonReplayPosition.Y,
actualButtonReplayTexture.Width, actualButtonReplayTexture.Height), Color.White);

                string Message1 = " YOU'RE DEAD AND THE EARTH DIED BECAUSE OF YOU TOO
! ";
                string Message2 = " Here's your score : ";
                string Message3 = " It's now like 3 million years after the disaster,
life has developed again,\n and we're already in a complete mess.";
                Vector2 sizeOfMessage1 = endGameMessageFont.MeasureString(Message1);
                Vector2 sizeOfMessage2 = endGameMessageFont.MeasureString(Message2);
                Vector2 sizeOfMessage3 = endGameMessageFont.MeasureString(Message3);


                Vector2 sizeOfScoreString =
endGameScoreFont.MeasureString(scoreToDisplay.ToString());
```

```
                Vector2 positionEndGameMessage1 = new
Vector2(graphicsEndGame.PreferredBackBufferWidth / 2 - sizeOfMessage1.X / 2,
graphicsEndGame.PreferredBackBufferHeight / 2 - 400);
                Vector2 positionEndGameMessage2 = new
Vector2(graphicsEndGame.PreferredBackBufferWidth / 2 - sizeOfMessage2.X / 2,
graphicsEndGame.PreferredBackBufferHeight / 2 - 360);
                Vector2 positionEndGameMessage3 = new
Vector2(graphicsEndGame.PreferredBackBufferWidth / 2 - sizeOfMessage3.X / 2,
graphicsEndGame.PreferredBackBufferHeight / 2 + 400);

                Vector2 positionEndGameScore = new
Vector2(graphicsEndGame.PreferredBackBufferWidth / 2 - sizeOfScoreString.X / 2,
graphicsEndGame.PreferredBackBufferHeight / 2 - sizeOfScoreString.Y / 2);


                spriteBatch.DrawString(endGameMessageFont, Message1,
positionEndGameMessage1, Color.White);
                spriteBatch.DrawString(endGameMessageFont, Message2,
positionEndGameMessage2, Color.White);
                spriteBatch.DrawString(endGameScoreFont, scoreToDisplay.ToString(),
positionEndGameScore, Color.White);
                spriteBatch.DrawString(endGameMessageFont, Message3,
positionEndGameMessage3, Color.Yellow);
            }

            else
            {
                spriteBatch.Draw(leaderBoardBackgroundTexture, new
Rectangle((int)leaderBoardBackgroundPosition.X, (int)leaderBoardBackgroundPosition.Y,
leaderBoardBackgroundTexture.Width, leaderBoardBackgroundTexture.Height),
Color.White);
                spriteBatch.Draw(actualButtonQuitLeaderBoardTexture, new
Rectangle((int)actualButtonQuitLeaderBoardPosition.X,
(int)actualButtonQuitLeaderBoardPosition.Y, actualButtonQuitLeaderBoardTexture.Width,
actualButtonQuitLeaderBoardTexture.Height), Color.White);

                string MessageA = " LEADERBOARD ";
                Vector2 sizeOfMessageA = endGameMessageFont.MeasureString(MessageA);
                Vector2 sizeOfleaderBoardString =
endGameScoreFont.MeasureString(leaderBoardString);
                Vector2 positionLeaderBoardMessageA = new
Vector2(graphicsEndGame.PreferredBackBufferWidth/2 -sizeOfMessageA.X/2, 10);
                Vector2 positionLeaderBoardString = new
Vector2(graphicsEndGame.PreferredBackBufferWidth/2 - sizeOfleaderBoardString.X/20,
50);

                spriteBatch.DrawString(endGameMessageFont, MessageA,
positionLeaderBoardMessageA, Color.Yellow);
                spriteBatch.DrawString(endGameMessageFont, leaderBoardString,
positionLeaderBoardString, Color.White);

            }


        }

        public void saveTheScore()
        {
            stringToAddToTheFileScoreCollector ="Score of the game :
"+scoreToDisplay.ToString()+" /// Date : "+Today.ToLongDateString()+"$";
            Console.WriteLine(stringToAddToTheFileScoreCollector);
```

```
            writer = new StreamWriter("SCOREFILEDONOTOPENANDMODIFY.txt", true); ; //
By default, when we write a file, it just replace the file.
                                                                    // That's why we put " true
" to say that we APPEND a text.
            writer.WriteLine(stringToAddToTheFileScoreCollector);
            writer.Close();
        }

        public void readTheScore()
        {
            reader = new StreamReader("SCOREFILEDONOTOPENANDMODIFY.txt");
            leaderBoardString = reader.ReadToEnd();
            char[] separateur = { '$' };
            String[] boardScore = leaderBoardString.Split(separateur);
            leaderBoardString = "";
            for (int i = boardScore.Length-1; i >= 0; i--)
            {
                leaderBoardString += boardScore[i] + "\n";
            }
            reader.Close();
            showEndGameGraphicalElements = false;
            LeaderBoardisVisible = true;
        }

    }
}
```