

Projets de fin d'études collégiales

Faster than Pixels

Document de conception

Table des matières

Intro:	4
Engine :	4
Intro:	4
Patron de conception :	4
Données externes :	6
Structures de données:	6
Expressions régulières :	7
Algorithme:	7
Mathématique:	7
Jeu:	7
Intro :	7
Menu :	8
Menu option:	8
Cinématique de lancement:	8
Menu Identification :	8
Menu principal:	8
Menu de sélection de mode :	8
Menu de chargement :	9
Menu de fin de partie :	9
Menu profile du joueur :	9
Menu information :	9
Gameplay:	10
Environnement:	10
Les ennemis :	11
Modes de jeu :	12
Protéger une zone :	12
Attaquer une zone :	12
Éliminer une cible précise :	12
Rechercher des survivants:	12
La course spatiale :	12
Contrôles :	13
Cas d'usage :	13
Diagramme de classes :	13
Structure de données externes :	13
JSON:	13
BDD :	15
Gameflow :	16
Maquettes :	17
Menu principales :	17
Menu d'identification :	17

Menu d'option :	18
Menu de sélection de modes de jeu :	18
Interface en jeu :	19
Compendium (menu info) :	19
Menu de chargement :	20
Menu de fin de partie:	20
Menu profil du joueur:	21
Documentation doxygen :	21

Intro:

Le jeu et le moteur seront codés en C++ avec les librairies externes SFML et Box2D et une librairie interne et sera sur PC. La structure du projet visual studio sera la suivante : Engine (librairie), FasterThanPixels (jeu). La partie Engine sera compilée automatiquement en premier à la compilation de Jeu.

Engine :

Intro:

Engine aura son namespace avec toutes les fonctions nécessaires pour le jeu.

Nous nous inspirons du code source de différents moteurs comme UnrealEngine et GodotEngine pour faire la partie moteur. Engine contient tout le "Back-End" du jeu mais ne contient aucune logique liée à celui-ci. La partie Engine peut très bien être utilisée avec n'importe quel autre jeu en 2D après elle permet de simplifier au maximum la création du jeu et d'avoir une API qui facilite le tout.

Patron de conception :

Les patrons de conceptions sont codés dans Engine et sont appelés en héritage dans la partie jeu. Les modèles de code principaux seront Entity Component System (ECS) et Patron de conception (Factory Method Pattern) mais nous allons utiliser aussi d'autres modèles simplifiés comme les object pool ou l'abstract qui est en paire avec le factory pattern.

Pour l'ECS, nous avons donc créé dans Engine la base , il n'y a aucune logique lié à une entité, seulement la base du système ECS. Le but est de faire une Entité qui contient "n" composants puis des systèmes liés à des composants. Un composant n'a pas obligatoirement de système car il peut servir seulement pour stocker des données. Par exemple un composant d'objet ramassable, n'a pas besoin de système, mais quand le joueur va passer au-dessus de cette entité il va vérifier si cette entité possède le composant, si oui alors il peut le ramasser et le rajouter dans son component d'inventaire. D'un autre côté certains composants ont besoin de système, comme pour les entités qui bougent, il y a 3 composants et systèmes principaux . ceux-là se trouvent dans le jeu. Le premier le "Brain" qui est un composant qui est comme une IA pour les ennemis. Après cela nous avons un composant Mouvement_Actif qui reçoit des inputs, il peut recevoir des inputs du joueur ou d'une IA. Puis nous avons le Mouvement_Passif qui lui reçoit des inputs du moteur physique. Notre but est de limiter le nombre de systèmes actifs car ils nécessitent de faire des boucles dans les entités pour toutes les mètres à jour, ce qui est coûteux et de "multithreaded" chaque système pour que tout se fasse le plus rapidement possible. nous savons que certain système ont besoin d'autres systèmes au préalable pour fonctionner dans certain cas, de ce fait (par exemple le Actif qui attend soit l' input du joueur ou le input de l'IA), nous devons donc créer une boucle dans la gameLoop pour que tous ces systèmes marchent en harmonie.

Pour le factory pattern, nous sommes partie du constat que nous devrons traiter un grand nombre d'instances de classe à chaque fois , et que nous devrons tous les mettre à jour en même temps. De ce fait, nous avons choisi cette conception avec une structure qui permet d'avoir un nombre n de scènes , qui chacunes gère leur propre factory.

Le jeu en lui-même est contenu dans le FTP_GameManager qui a 2 fonctions, initialiser le jeu et la gameLoop. Le GameManager, quand il lance sa fonction d'initialisation, initialise un AssetManager et un SceneManager. Le premier charge tous les assets disponibles dans un dossier. Il les reconnaît grâce à des fichiers JSON qui sont créés pour chaque élément pour savoir quel asset est utile à quoi. L'avantage d'utiliser un système comme cela est que nous pouvons à n'importe quel moment recharger tous les assets du jeu mais aussi permettre à n'importe qui de rajouter ses propres assets (un premier pas vers une API modding). L'AssetManager contient donc toutes les textures, polices et sont disponible pour tout le GameManager.

Après cela nous avons le SceneManager qui est aussi dans le GameManager. Celui-ci gère toutes les scènes (niveaux) du jeu. On utilise un simple string pour sauvegarder le nom de la scène actuelle dans le GameManager et on cherche dans le SceneManager la scène actuelle. Au changement de ce string, 2 événements importants se passent : "S_End_Scene" qui est une fonction qui vient de la scène précédente, si elle a besoin de sauvegarder des données avant d'être fermée par exemple. Puis nous avons la fonction "S_Begin_Play" qui est appelée au début de la scène automatiquement . Cela permet d'avoir un gestionnaire de scène efficace .

Après cela, chaque scène a son propre EntityManager, car chaque scène est différente et n'a pas besoin du même nombre d'entités. Cela permet de nous assurer que nous n' utilisons pas d'entité d'autre scène dans la scène actuelle. Chaque EntityManager contient toutes les entités qui seront rendues dans la scène actuelle. De plus, dans celui-ci nous utilisons le "memory pooling". Quand la scène se lance, nous allouons n entités dans l'EntityManager, ces entités sont vides, quand le jeu a besoin de rendre une entité il n'a qu'à faire une demande pour avoir accès à une entité libre. Quand le jeu a fini avec une entité, il n'a qu'à la remettre dans liste d'entités utilisables. Cela nous permet d'éviter d'allouer de la place et en enlever dans la mémoire et d'avoir une taille définie pour avoir de

meilleure performance en jeu. Pour finir nous utilisons un petit peu le patron de conception d'abstract grâce à notre structure de dossier, qui génère les bases du code dans Engine puis qui est utilisé dans le Jeu en temps qu' Enfant. Par exemple, dans nos fonctions si nous voulons que celle-ci est comme argument le joueur, nous ne mettons pas la classe joueur comme argument mais la classe Entité, car la classe Joueur est un enfant de Entité. Ce qui permet que cette fonction devienne générique est peut être utilisé avec n' importe quelle autre entité.

Pour conclure, nous avons utilisé ces méthodes car, selon nous, elles sont les plus optimisées pour un jeu qui possède un grand nombre d'entités à traiter avec un nombre de niveaux élevés. Chaque Manager a son propre champ d'action et de manœuvre ce qui réduit les problèmes que les Managers peuvent avoir s' ils ont accès à tout. De plus, on pense que cette structure de données est la plus performante pour ce que nous voulons faire.

Données externes :

Chaque asset aura un fichier JSON lié à celui-ci avec divers paramètres pour les charger dans le moteur. Cela nous permet de pouvoir mettre ces assets dans des endroits précis pour les retrouver après dans le code. Ça permet aussi de pouvoir avoir les images ,polices et audio dans des dossiers différents bien rangés et tous les JSON à la racine . Cela facilite l'ajout d'asset dans le jeu que se soit pour nous ou bien même un moddeur.

Base de données :

Une base de données PostgreSQL sera utilisée. Nous allons sauvegarder divers donner dessus , Le joueur se connectera au lancement du jeu et nous allons mettre à jour la bdd tout au long de ces parties. La partie Engine va gérer toutes les interactions avec la BDD, le jeu n'aura qu'à appeler les fonctions dans le moteur avec les données à envoyer.

La configuration du joueur sera sauvegardée localement dans un fichier JSON. Cela facilite l'interaction que le joueur a avec le jeu. Avec notre expérience de joueurs assidus, il arrive que certaines configurations du jeu dans le menu des options fait que le jeu ne peut plus être lancé. Pour résoudre ce problème on a 2 choix, réinstaller le jeu ou essayer de trouver le fichier de sauvegarde de la configuration si il n'était pas crypté ou sur le cloud de steam. On a choisi de le mettre en JSON à côté du .exe pour simplifier cela.

Structures de données:

Nous utilisons les vecteur très fréquemment car nous stockons un nombre n de données , par exemple presque chaque Manager a un ou plusieurs vecteurs pour sauvegarder tous leurs enfants.

Les cartes sont généralement utilisées pour avoir une référence à un élément d' un vecteur en utilisant un tag ou un moyen de le retrouver très rapidement.

Nous avons créé la structure de données de graph pour le mode de jeu "Action rapide" celui ci permet d'avoir un graph qui nous permet de savoir après le mode de jeu actuel, lequel est

le plus cohérent de prendre après, additionné à une valeur aléatoire (baser sur un seed) si il y a plusieurs branches possibles pour aller vers un autre modes de jeu.

Expressions régulières :

Nous utilisons le principe d'expression régulière pour vérifier que le mail suit bien le modèle “***@***.***”

Algorithme:

Les ennemis utilisent un algorithme de Steering modifié pour nos besoins et une state machine. Notre base de steering est basée sur celle ci :

<https://gamedevelopment.tutsplus.com/series/understanding-steering-behaviors--gamedev-1-2732>

Pour le reste des algorithmes, nous allons essayer d'utiliser le plus possible tout ce qui apporte de l'aléatoire pour donner une rejouabilité maximum.

Le jeu en général, va utiliser des systèmes qui ont des sous systèmes, par exemple GameManager et le système principal qui contient AssetManager qui est un système qui gère les assets, et SceneManager qui gère les scènes. Chaque Scène à un manager de Entity pour gérer tout ce qui apparaît à l'écran. La Gameloop va être bridée à 30 images par secondes, si votre système est plus rapide vous aurez un plus haut framerate mais la physique et le rendu sera limité à 30 images par seconde.

Mathématique:

Pour notre jeu nous aurons besoin de formule mathématique liée au vecteur, comme calculer la distance entre deux points, l'angle, normaliser un vecteur.

Nous centralisons toutes les fonctions mathématiques dans un fichier “MathF” qui est disponible dans Engine, cela permet un accès dans tout le projet de ces fonctions.

Jeu:

FasterThanPixels aura toute la logique du jeu .

Intro :

La plupart du contenu de Jeu est l'héritage de classe créée dans Engine pour pouvoir se concentrer sur la partie logique et le contenu du jeu.

Menu :

Pour les menus, nous allons utiliser une state machine.

Voici une liste des différentes pages présent en jeu :

- Cinématique lancement du jeux
- Menu d'authentification
- Menu principale
- Menu mode entraînement
- Menu mode action rapide
- Menu mode
- Menu de chargement
- Menu de jeu
- Menu d option
- Menu de pause
- Menu de fin de partie
- Menu profile du joueur
- Menu information avec différentes explications du jeu et de ces fonctionnalités (tuto)

Menu option:

Différentes options sont disponibles dans le menu de paramétrage pour les volumes sonores et les paramètres vidéo (résolutions et plein écran), mais aussi les dialogues et les éléments en jeu. Une playlist de musique sera dispo en fond sonore, nous savons que nos pilotes adore la musique de leur époque (1980).

Cinématique de lancement:

La cinématique de lancement représente 2 images , la première est le logo du jeu, le second le nom de la compagnie. elle ne doit pas durer plus de 5 sec . Elle lance le menu d'identification une fois fini.

Menu Identification :

Ce menu permet de s'identifier à son profil de joueur pour sauvegarder et charger nos stats. si le joueur a un compte il va charger les données, s' il n'en a pas il va créer un compte et charger les stats(qui seront par défaut à 0 partout) une fois connecté il va dans le menu principal.

Menu principal:

Ce menu donne accès à d'autres sous menu : le menu de sélection de mode de jeu, les options, quitter le jeu, le menu du compendium et le menu du profil du joueur.

Menu de sélection de mode :

Ce menu permet de choisir entre les modes de jeux disponibles, le joueur a le choix entre entraînement ou Action Rapide, une fois le premier mode sélectionné, il doit choisir la

difficulté entre les 3 choix disponibles (Facile, Moyen et Difficile) et entrer une seed s' il le veut ou en générer une aléatoire (génération de la map). Une liste de seed sera disponible sur le côté pour tester un ordre spécifique avec un nombre spécifique de gamemodes.(les best générés par nous, les meilleures de la communauté)

Menu de chargement :

Ce menu permet au jeu de charger tous les éléments, il sera sur un thread différent que le jeu pour qu'il puisse charger. Il y aura une barre de progression, elle sera de minimum 5 sec, si le joueur prend plus de temps pour charger le jeu alors elle sera plus longue.

Menu de fin de partie :

Ce menu récapitule la partie du joueur, elle affiche les différents modes de jeu joués avec les statistiques de chacun ainsi que les statistiques globales de fin de partie. Il envoie à la base de données les nouvelles informations du joueur.

Menu profil du joueur :

Sur ce menu, le joueur pourra voir son profil extrait de la BDD et voir ces statistiques liées au modes de jeu qu'il a joué , ses seed, ses succès

Menu information :

Le menu aura principalement des images et du texte, il représente la bible du jeu, comment le jeu marche, mais aussi ce que représente chaque mode de jeu et éléments du jeu.

Gameplay:

Environnement:

Chaque niveau sera généré procéduralement à partir d'un seed fourni par le joueur ou random.

La taille de la map sera fixe (taille à tester)

un nombre aléatoire d'éléments d'environnement vont apparaître à des positions aléatoires (x planètes apparaissent à n endroits sans se superposer).

Un nombre aléatoire d'astéroïdes va être généré. Les astéroïdes vont apparaître à des positions aléatoires avec un vecteur de poussée d'origine.

Les astéroïdes doivent avoir un énumérateur pour catégoriser leur taille, dépendamment de cette valeur, si celui-ci prend des dégâts soit il se divise en plus petits morceaux ou explose.

Il faudrait centraliser les variables les plus importantes pour l'équilibrage du jeu qu'il se fasse dans un nombre réduit de fichier.

Quand le joueur arrive à un bord, il sera téléporté à l'autre bord de la map pour un effet infini de la map.

Les bonus du joueur seront placés dans des coffres, qui seront générés de plusieurs manières :

- Aléatoire au début de la partie.
- Un pourcentage de chance après avoir tué un ennemi.
- Après avoir réussi un objectif.

Les ennemis :

Des vaisseaux adverses seront présents pour vous empêcher d'accomplir vos objectifs. Il existe plusieurs type d'ennemis :

- Les intercepteurs :
Stats : bouclier : 2, vie : 1, vitesse : rapide, arme : laser.
Comportement : Ils prennent en chasse le/la pilote quand il /elle passe à portée et tire dessus quand le vaisseau du pilote est dans un angle de 10° devant lui et tire toutes les secondes environ.
- Les bombardiers :
Stats : bouclier : 4, vie : 2, vitesse : lente, armes : tourelle laser, mines magnétiques.
Comportement : Ils restent à distance du pilote et posent des mines afin de réduire le champ d'action du pilote. Quand le vaisseau est détruit il peut soit lâcher 2 à 3 mines soit créer une explosion qui endommage toute entité prise dans son rayon.
- Les chasseurs :
Stats : bouclier : 3, vie : 2, vitesse : rapide, arme : laser.
Comportement : Ils ont le même comportement que les Intercepteurs mais tirent 50% plus vite que ces derniers.
- Les porte-vaisseau :
Stats : Bouclier : 6, Coque : 4, vitesse : lente, armes : 2 tourelles laser, intercepteurs.
Comportement : Ils tournent autour de la zone de combat et tirent à distance. Ils produisent 5 intercepteurs et quand un de leurs intercepteurs est détruit ils en produisent un autre au bout de 10 secondes. Quand ils sont détruits, ils produisent entre 3 et 5 intercepteurs
- Tourelles fixes :
Stats : Coque : 10, Arme : laser
Comportement : Elles sont statiques et tirent des lasers en rafales vers le vaisseau du pilote, 3 tirs toutes les 1.5 secondes.

Les bonus :

- Clef anglaise : Réparation de la coque du vaisseau (40% en mode facile, 30% en mode moyen et 20% en mode difficile).
- Éclair : Augmentation temporaire de la cadence de tir simple de 50% (20 secondes en facile, 15 secondes en moyen et 10 en difficile).
- Perforation : Les tirs font légèrement plus de dégâts (x1.2 environs) et traversent la première cible qu'ils touchent (20 secondes en facile, 15 secondes en moyen et 10 en difficile).
- Boosters : Augmente temporairement la vitesse du vaisseau (x1.25) (20 secondes en facile, 15 secondes en moyen et 10 en difficile).

Modes de jeu :

Pour les modes de jeux, nous allons essayer de les rendre le plus random possible pour éviter que le joueur ressente une lassitude d'avoir tout le temps les mêmes endroits.

Être le dernier survivant, la zone à la première génération le centre du cercle sera le joueur puis pour chaque rappétissement de la zone il aura un offset random.

Protéger une zone :

Facile :

- Vague 1 : 10 intercepteurs.
- Vague 2 : 10 intercepteurs, 2 Bombardiers.
- Vague 3 : 10 intercepteurs, 4 Bombardiers, 3 Chasseurs.
- Vague 4 : 10 intercepteurs, 6 Bombardiers, 6 Chasseurs, 1 Porte-Vaisseaux.
- Vague 5 : 10 intercepteurs, 8 Bombardiers, 9 Chasseurs, 2 Porte-Vaisseaux.

Moyen :

- Vague 6 : Vague 5 + Vague 1
- Vague 7 : Vague 5 + Vague 2
- Vague 8 : Vague 5 + Vague 3
- Vague 9 : Vague 5 + Vague 4
- Vague 10 : Vague 5 + Vague 5

Difficile :

- Vague 11 : Vague 10 + Vague 1
- Etc ...

Attaquer une zone :

La structure à attaquer est sélectionnée de manière aléatoire placée de façon tout aussi aléatoire sur la carte. Le joueur doit détruire le cœur énergétique de la structure en 10 minutes en facile, 7 minutes et 30 secondes en moyen et 5 minutes en difficile.

Éliminer une cible précise :

Un nombre aléatoire, selon la difficulté de la partie, d'ennemis sera généré à un endroit et tenteront de quitter la zone de jeu à l'arrivée du joueur. Parmi ces ennemis, la cible prioritaire sera d'une couleur différente.

- Facile : entre 5 et 10 vaisseaux, au moins 2 chasseurs et 2 bombardiers
- Moyen : entre 10 et 15 vaisseaux, au moins 3 chasseurs, 3 bombardiers et 1 porte-vaisseau.
- Difficile : entre 15 et 20 vaisseaux, au moins 5 chasseurs, 5 bombardiers et 2 porte-vaisseaux.

Rechercher des survivants:

Les épaves des vaisseaux alliés seront placés de façon random par chunk pour avoir une consistance mais qu'ils ne soient pas tous au même endroit ou trop dispersés. Les survivants à sauver seront groupés par capsules, entre 5 et 10 par capsule. Il faut sauver au moins 80% des survivants avant de quitter la zone de mission.

La course spatiale :

Nous allons générer une map, puis faire apparaître des points random puis orienter le checkpoint par rapport au précédent avec une petit random pour que ce ne soit pas parfait et qu'il y est un petit angle léger.

Facile : 5 checkpoints, 2 tours.

Moyen : 10 checkpoints , 2 tours.

Difficile : 12 checkpoints, 3 tours.

Entrainement :

Facile :

1000 astéroïdes apparaissent dans le niveau, le/la pilote à 60 sec pour détruire le plus d'astéroïdes possible. L'inertie angulaire est désactivée. Chaque grand astéroïde offre 15 points, les moyens offrent 10 points et les petits offrent 5 points.

Moyen :

1000 astéroïdes apparaissent dans le niveau, le/la pilote à 60 sec pour détruire le plus d'astéroïdes possible. Des intercepteurs et des bombardiers sont présents pour gêner le/la pilote. L'inertie angulaire est activée. Chaque grand astéroïde offre 20 points, les moyens offrent 15 points et les petits offrent 10 points.

Difficile :

1000 astéroïdes apparaissent dans le niveau, le/la pilote à 60 sec pour détruire le plus d'astéroïdes possible. Des intercepteurs, des bombardiers, des chasseurs et des porte-vaisseaux sont présents pour gêner le/la pilote. L'inertie angulaire est activée. Chaque grand astéroïde offre 25 points, les moyens offrent 20 points et les petits offrent 15 points.

Contrôles :

La cartographie des touches sera la suivante :

W : Avancer

S : Reculer

D : Pivoter à droite

A : Pivoter à gauche

Q : Dash à gauche

E : Dash à droite

Shift : Turbo

Espace : Tirer

Ctrl : Mines

F : Missiles

1 : Réparations

2 : Laser Perforants

3 : Surfréquençage

Diagramme :

Cas d'usage :

Dans le dossier C61

Diagramme de classes :

Nous savons qu'il manque encore quelques fonctions, variables et classes à ajouter dans nos futurs sprint.c'est une liste aussi exhaustive que possible car nous construisons un moteur à côté qui doit être capable de faire fonctionner le jeu. Par exemple, les classes liées au GUI comme Button,Slider,Input sont en cours de création et de test et nous n'avons pas eu le temps de les désigner pour le sprint 0.

Dans le dossier C61

Structure de données externes :

JSON:

Textures :

```
  "name": "logoratatoskr",
  "url": "/img/Cinematique.png",
  "type": "Texture",
  "collider": [
    {
      "x": 0,
      "y": 0
    },
    {
      "x": 0,
      "y": 0
    }
  ]
}
```

Font:

```
  "url": "./font/pilotcommand1_2lasersuperital.ttf",
  "name": "FontIntro",
  "type": "Font"
}
```

Audio:

```
{  
    "name": "logoratatoskr",  
    "url": "/audio/Cinematique.wav",  
    "type": "Audio",  
    "repeat": false,  
    "tag": "IntroSound",  
    "sous-type": "Music"  
}
```

BDD :

Il y aura une première table qui servira à stocker les informations relatives au joueur/ à la joueuse telles que :

- ID(PK INT)
- Pseudonyme(VARCHAR 16)
- Mot de passe chiffré
- Progression en expérience(FLOAT)
- Mail (TEXT)
- Nombre total d'intercepteurs détruits (INT)
- Nombre total de bombardiers détruits (INT)
- Nombre total de chasseurs détruits (INT)
- Nombre total de porte-vaisseaux détruits (INT)
- Nombre de missions terminées (INT)
- Nombre de naufragés sauvés (INT)

Une seconde table sera présente pour stocker :

- ID(PK INT)
- Seed(TEXT)
- Le meilleur Score(FLOAT)
- Pseudonyme du détenteur du meilleur score (TEXT)

Une troisième table contiendra les succès:

- ID(PK INT)
- Nom(TEXT)
- Description(TEXT)

Une quatrième table contiendra les succès des joueurs :

- ID (PK INT)
- ID du joueur (FK INT)
- ID du succès (FK INT)
- Date d'obtention (DATE)

Gameflow :



Maquettes :

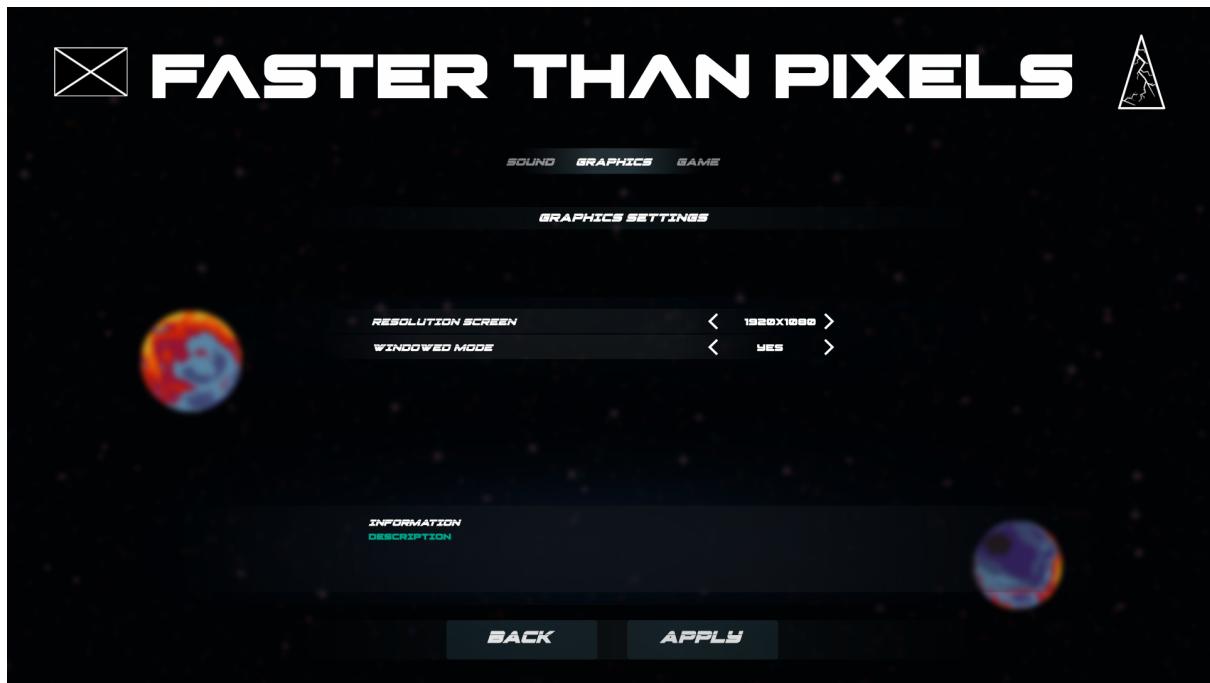
Menu principales :



Menu d'identification :



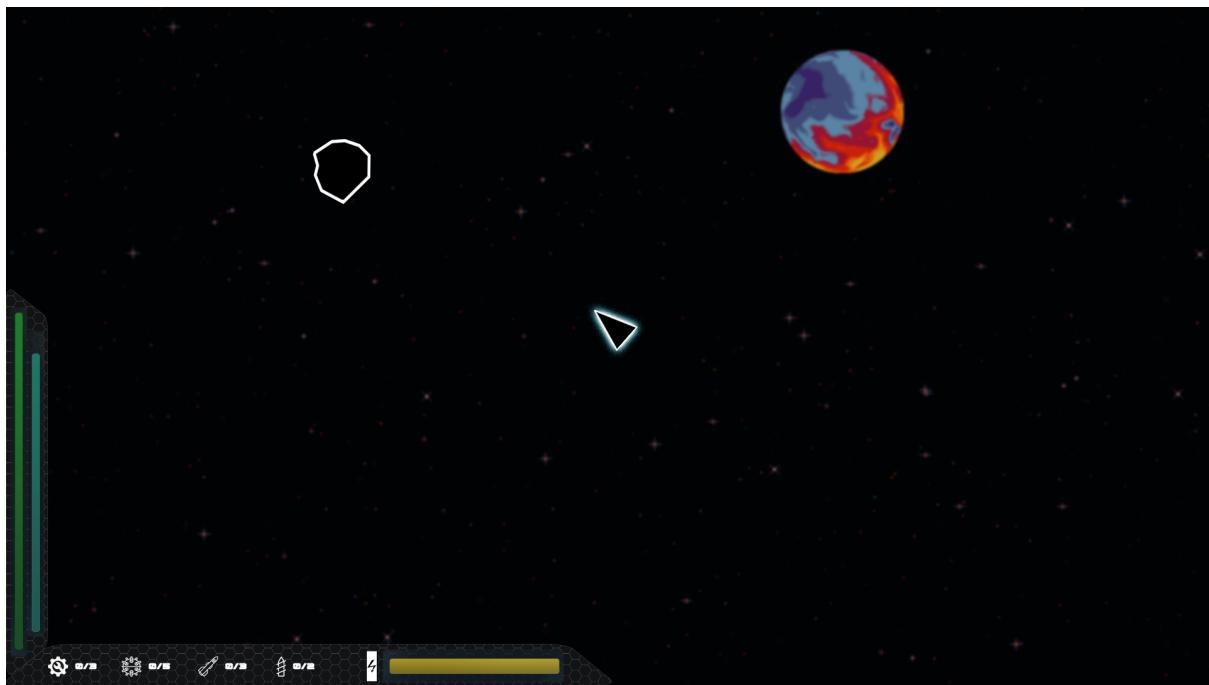
Menu d'option :



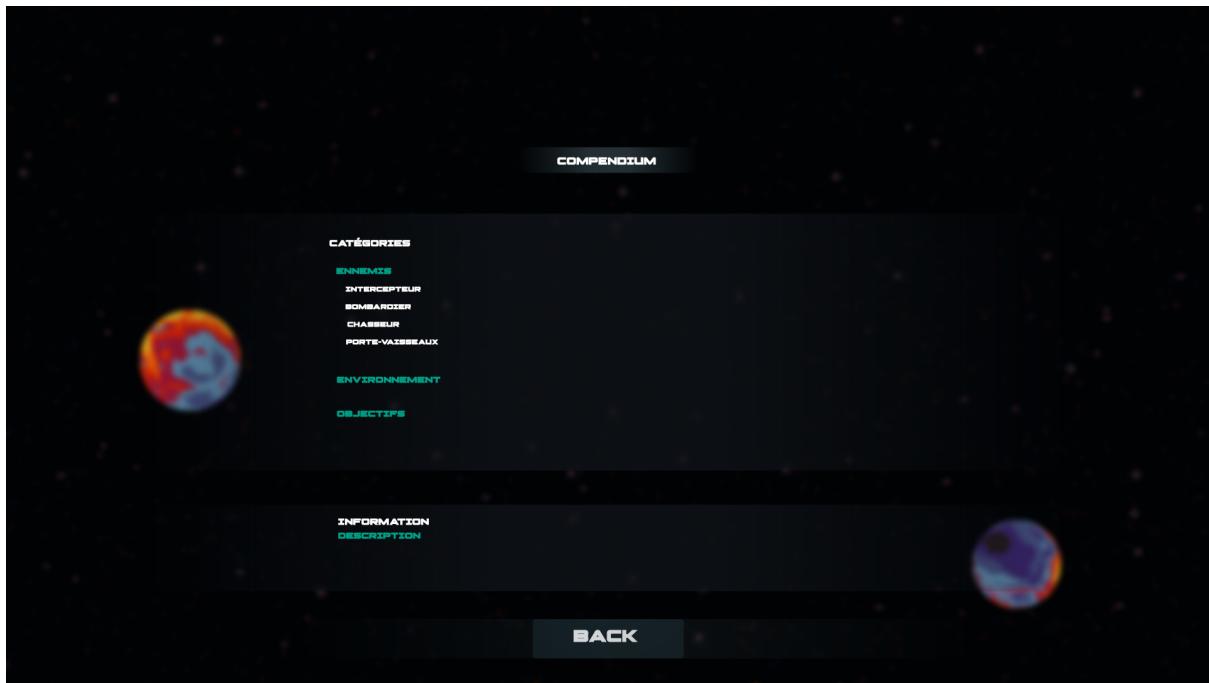
Menu de sélection de modes de jeu :



Interface en jeu :



Compendium (menu info) :



Menu de chargement :



Menu de fin de partie:



Menu profil du joueur:



Documentation doxygen :

Dans le dossier C61