|  |
| --- |
| SPACE INVADERS |

Mauro Santos

SI-C4a

3/29/2021

Table des matières

[Analyse préliminaire 3](#_Toc67920927)

[Introduction 3](#_Toc67920928)

[Organisation 4](#_Toc67920929)

[Objectifs 4](#_Toc67920930)

[Planification initiale 4](#_Toc67920931)

[Analyse / Conception 5](#_Toc67920932)

[Concept 5](#_Toc67920933)

[Stratégie de test 8](#_Toc67920934)

[Risques techniques 8](#_Toc67920935)

[Dossier de conception 8](#_Toc67920936)

[Réalisation 9](#_Toc67920937)

[Liste des documents fournis 9](#_Toc67920938)

[Assets utilisés dans le projet 10](#_Toc67920939)

[Description des tests effectués 11](#_Toc67920940)

[Dossier d’impementation 12](#_Toc67920941)

[Conclusions 21](#_Toc67920942)

[Annexes 22](#_Toc67920943)

[Sources – Bibliographie 22](#_Toc67920944)

[Manuel d'Utilisation 22](#_Toc67920945)

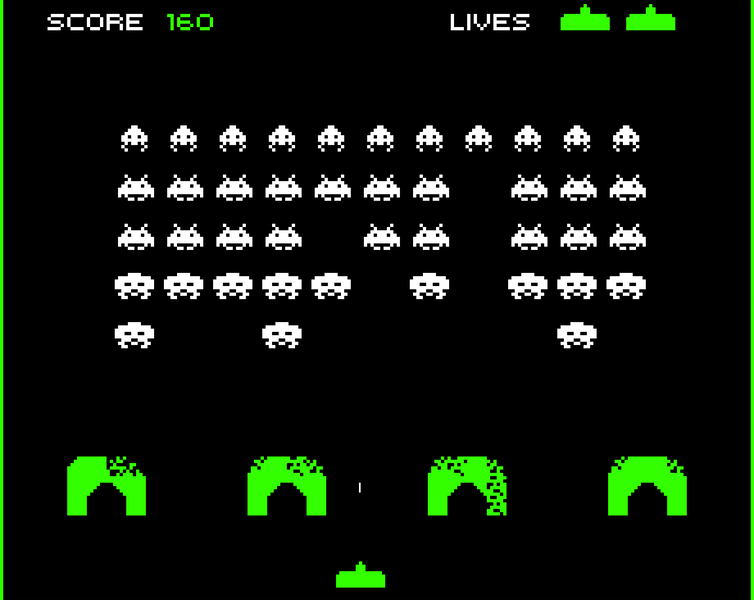
# Analyse préliminaire

## Introduction

Le projet Space Invaders est un jeu fait sur le framework Unity et programmé en C#.

Ceci a été fait pour ma formation entant qu’informaticien dans le CPNV, pour le cours de Pre TPI. L’objectif final est d’avoir un projet contant toutes les fonctionnalités demandées par mon « project manager ». Comme dit précédemment, le projet sera basé sur le jeu d’arcade Space Invader, où nous jouons entant qu’un vaisseau spatial et notre but est de tuer le plus d’aliens possible.

Ce projet me permettra de m’entrainer pour le projet TPI, pas seulement en termes sur avec le Framework unity mais aussi avec la façon de travail de M. Ithurbide.



## Organisation

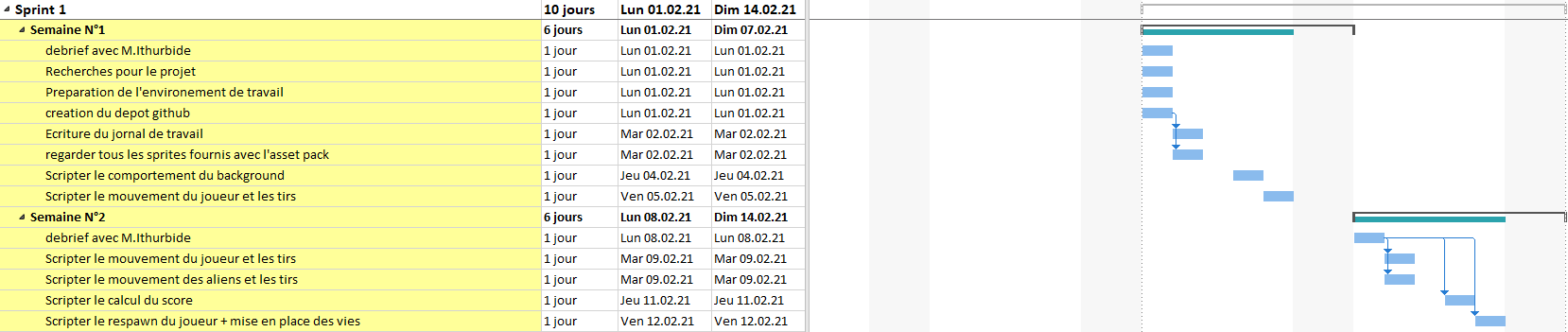
|  |  |  |
| --- | --- | --- |
| Éleve | Mauro Santos | [Mauro-alexandre.costa-dos-santos@cpnv.ch](mailto:Mauro-alexandre.costa-dos-santos@cpnv.ch) |
| Chef de projet | Julien Ithurbide | [Julien.ITHURBIDE@cpnv.ch](mailto:Julien.ITHURBIDE@cpnv.ch) |
| Expert | Loic Viret | [Loic.VIRET@cpnv.ch](mailto:Loic.VIRET@cpnv.ch) |

## Objectifs

Voici les différents objectifs qui sont cités dans le cahier des charges

1. Le jeu comporte au moins 5 niveaux
2. Il existe des options permettant de changer d’arme
3. Le jeu peut être mise en pause
4. Le jeu comporte un menu des scores
5. Le jeu comporte des sons originaux
6. Le menu permet de couper le son
7. une version deux joueurs est utilisable

## Planification initiale



# Analyse / Conception

## Concept

Voici toutes les taches à réaliser dans le projet :

### Trouver des assets pour le jeu

Pour commencer le projet je vais aller sur le magasin de Unity et je chercherais des « assets »pour le jeu. La seule contrainte c’est que il faut que le pack que prends soit adapté à un jeu de ce type et qui il aient tous les « spirites » que j’aurais besoin pour le jeu

### Créer un fond déroulant à l’infini

Je voulais faire un fond qui se déroule à l’infini à fin de donner la sensation qu’on se balade dans l’espace.

### Créer un vaisseau pour le joueur

Créer un vaisseau contrôlable par le joueur, contenant quelques fonctionnalités de base comme le mouvement gauche droite et le tir.

### Créer un vaisseau ennemi

Créer un vaisseau des ennemis, contenant des fonctionnalités de base comme le mouvement gauche droite et le tir. Quand un de ces vaisseaux arrive à un des bords de l’écran, tous les ennemis avancent et changent de sens. Les ennemis tirent avec un timing aléatoire

Une fois que tous les ennemis ont été détruits, le jeu chargera un nouveau niveau avec des nouveaux ennemis

### Ajouter le comptage de points

Les points seront la façon de comparer la performance entre chaque « run ». Il y a 2 façons de gagner des points :

En détruisant un ennemi, que donne 100 points au joueur.

En détruisant un projectile ennemi, qui ajoute 50 points.

Pour favoriser une gameplay plus précise, un système de combo sera implémenté, ceci augmentera à chaque fois que un ennemi est éliminé. Ceci sera remis à 0 si le joueur ratte sa balle ou se fait éliminer.

Les avantages de faire un combo sont que à la place de gagner 100 points par ennemi éliminé, le joueur gagnera 100\*Combo.

### Ajouter les vie du joueur

Le système de vies est simple, le joueur commencé avec 3 vies. à Chaque fois qu’il se fait toucher par un ennemi, il utilise une vie pour retourner en jeu après un petit délais.

Une fois que le joueur perds sa dernière vie, la partie se termine, montrant l’écran de Game Over

### Création du HUD

Le HUD (ou heads up display) est l’interface qui est par-dessus du jeu et sera utiliser pour afficher le nombre de vies du joueur, son score et le combo actuel (pour faire une référence à guitar hero, ceci s’affichera que si le combo est plus grand que 4)

### Création du menu principal

Le menu principal est la première chose que l’utilisateur voit lors qu’il lance le jeu, Ceci est aussi affiché après chaque partie. Dans ce menu nous pouvons : lancer le jeu, ouvrir le menu de scores, aller dans le menu options(nous permettant de changer le volume du jeu et le type de tir) et quitter le jeu

### Création du menu pause

Le menu de pause est le menu qui sera affiché lors que le joueur met le jeu en pause . Dans ce menu nous pouvons : Retourner dans le jeu, Recommencer une partie, aller dans le menu options(nous permettant de changer seulement le volume du jeu) et quitter la perte en cours

### Ajouter les 5 différents niveaux

Dans cette version du jeu, 5 niveaux différents seront implémentés, ces niveaux seront joués en boucle, c’est-à-dire, que une fois que joueur a terminé le 5eme niveau, il passera de nouveau au premier.

Voici la logique derrière chacun des niveaux est :

* 1 ligne d’ennemis « calmes »
* 2 lignes d’ennemis « calmes »
* 1 lignes d’ennemis « agressifs »
* 2 lignes d’ennemis « agressifs »
* 3 lignes d’ennemis « agressifs »

### Tableau de scores

Le tableau de scores affiche le top 8 des anciens scores comme dans une arcade. Ceci est affiché quand le jouer appuie sur le bouton scoreboard du menu ou après une partie. Si le joueur a eu la un score qui est dans le top 8, il aura l’option de sauvegarder son score avec un nom de 3 caractères

### Ajouter l’état de jeu « Game over »

Comme dit dans le points précédents, l’état de jeu « Game over » contiendra un écran Game over qui servira en tant que transition avant l’affichage du scoreboard, une fois que je joueur quitte se scoreboard, il retournera sur le menu principale du jeu.

### Ajouter les sons du jeu

Le jeu contiendra 4 types de sons différents, ceux si joueront quand :

* Le joueur se fait toucher.
* Un ennemi se fait toucher.
* Un projectile est instancié
* Le joueur interagit avec le menu

Le volume des sons peut-être modifié dans les options du jeu

### Mode Multijoueur

Le jeu donnera la possibilité de jouer avec un ami en local, pour que ceci soit fonctionnel, il faut avoir une manette (soit Xbox ou PlayStation).

La gameplay ne changera pas beaucoup, le jeu s’arrête toujours quand le deux joueurs perdent toutes leurs vies( les vies sont partagés entre les 2 joueurs).Le système de combo est aussi partagé par les 2 joueurs, si le deuxième joueur rate sa balle, les joueurs perdront leur combo.

### Différents modes de tir

Le jeu offre 3 modes de tir aux joueurs :

* Tir normal (balle de taille normal, 60 tirs par minute)
* Tir « machinegun » (balles de taille petite, 300 tirs par minute)
* Tir « big bullet » (balles de grande taille, 45 tirs par minute)

Ces modes de tir peuvent être changés en allant dans les options avant de lancer une partie.

## Stratégie de test

Pour la stratégie de test, nous sommes parties sur une façon plus manuel en utilisant icescrum. Ceux-ci ont été validés par M.Ithurbide à la fin de chaque sprint, à chaque fois nous avons checké les test des taches effectués pendant le sprint. Ceux.ci ont été fait dans ma machine personnelle dans l’éditeur de unity.

## Risques techniques

Le plus grand risque de ce projet a été la gestion du temps, puis que il était quand même un peu grand en niveaux des fonctionnalités. Ma façon de travail n’a pas beaucoup aidé non plus puisque, je déteste faire de la documentation et du coup je me mets a fond dans la solution et je laisse la documentation pour le dernier moment.

## Dossier de conception

Pour réaliser ce projet j’ai utilisé deux ordinateurs :

* Ordinateur portable
  + OS :Windows 10 Home
  + CPU : AMD Ryzen7 4700u with Radeon Graphics
  + Ram :16GB
  + GPU :  « AMD Radeon Graphics »
* Ordinateur fixe
  + OS :Windows 10 Pro
  + CPU : Intel Core i9 9900k
  + Ram :32GB
  + GPU :  Nvidia RTX2080 Super

Pour ce projet, j’ai utilisé le Framework Unity ,la raison pour laquelle je suis partie sur Unity et non pas un autre Framework connu (comme Unreal engine, Godot) c’est simplement parce que j’ai profité du fait que j’aie déjà des connaissances avec cet outil me permettant de partir directement dans sur le projet sans devoir perdre du temps. Unity me permet de gérer les collisions entre objets, créer des « prefabs »(des objets crées par moi que avec une configuration et script spécifiques),ajouter un script à plusieurs objets, ajouter plusieurs objets ,etc.

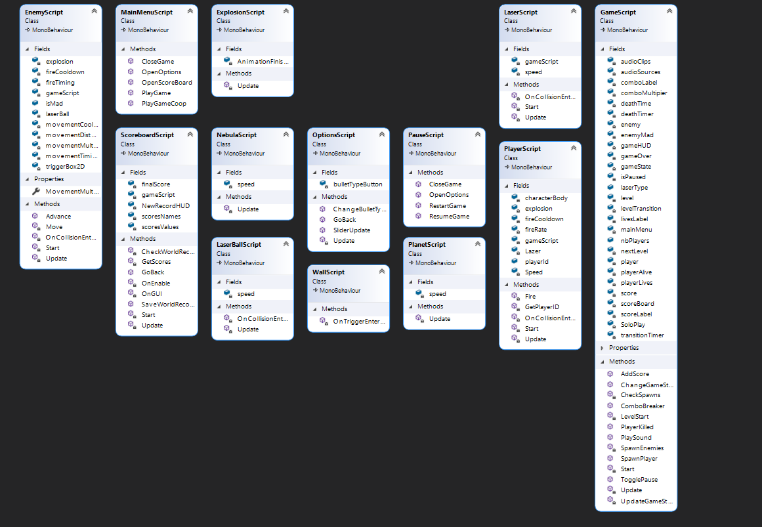
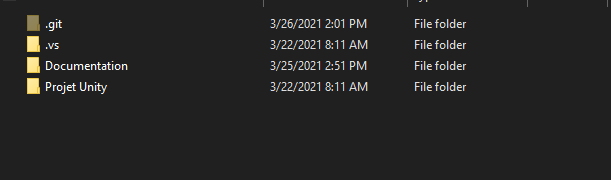


Figure Diagramme de Classes

# Réalisation

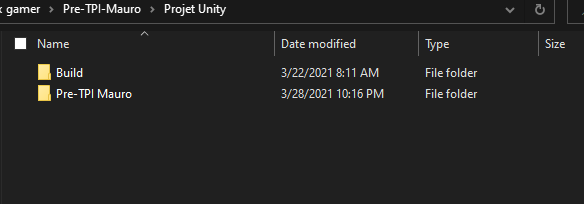
## Liste des documents fournis



Voici les dossiers que vous retrouverez dans le dépôt GitHub.

Dans le dossier « documentation » vous trouverez tous fichiers de documentation, comme la planification initiale, le journal de travail, la documentation de projet, etc…

Dans le dossier « projet unity » vous trouverez les dossiers suivants :

2

Le dossier « Build » contenant le build final de jeu.

Le dossier « Pre-TPI Mauro » qui contient le projet unity.

## Assets utilisés dans le projet

### Space shooter free

Pour les visuels du jeu, je suis parti sur un pack que j’ai trouvé sur l’asset store d’Unity.

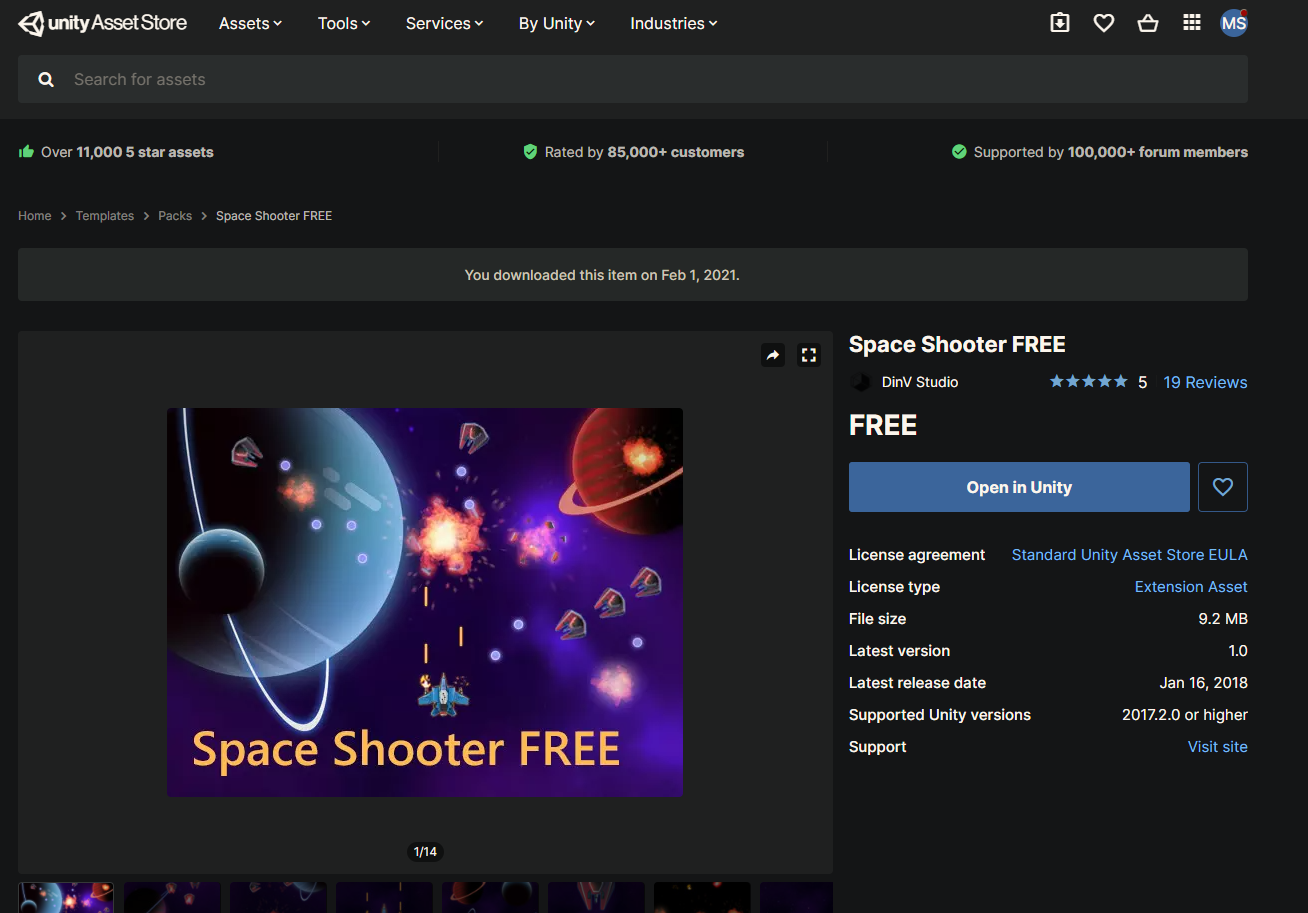


Figure https://assetstore.unity.com/packages/templates/packs/space-shooter-free-107260 visité le 29/03/2021

Ceci avait contenaient tous les éléments que j’avais besoin, comme un vaisseau pour le joueur, des vaisseaux ennemis, des projectiles et un fond. Quand j’ai importé ce pack, j’ai seulement pris les « sprites » (c’est-à-dire les images du jeu), parce ceci contenait une scène unity complètement programmée.

8-Bit Style Sound Effects

Pour les sons du jeu, je suis reparti sur l’asset store d’Unity.

J’ai testé plusieurs packs de son mais pour finir j’ai choisi celui-là

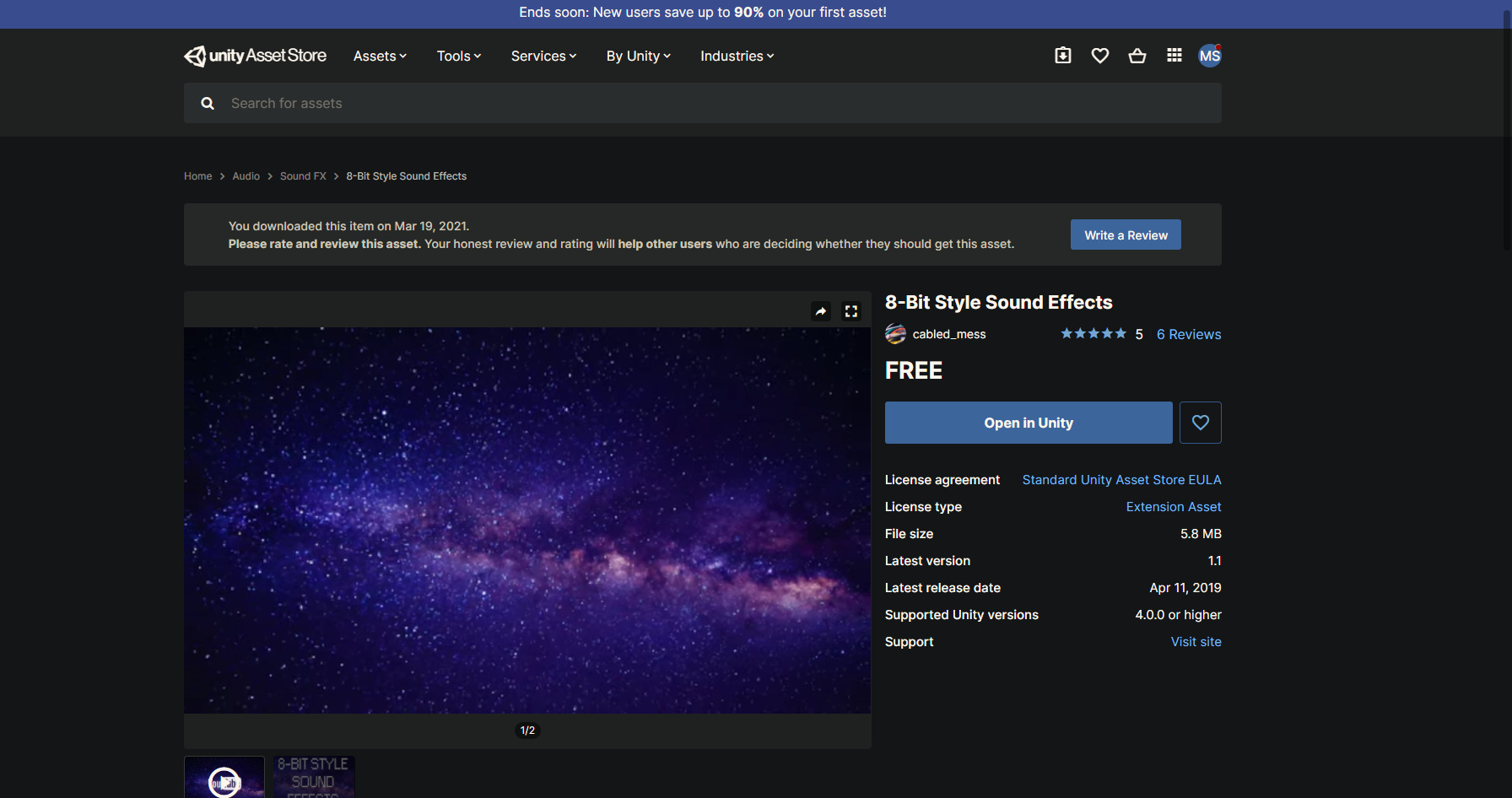
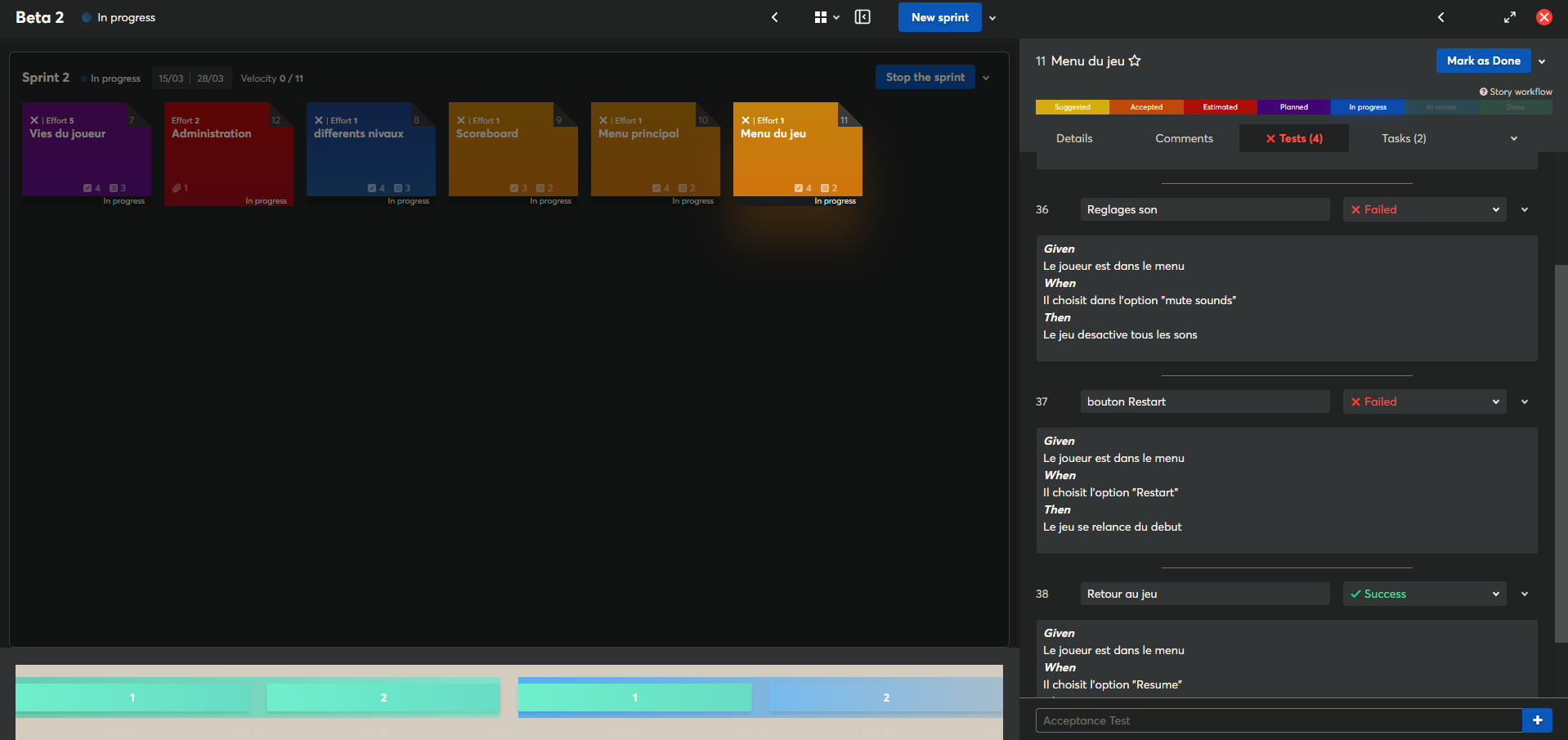


Figure <https://assetstore.unity.com/packages/audio/sound-fx/8-bit-style-sound-effects-68228> visité le 29/3/2021

## Description des tests effectués

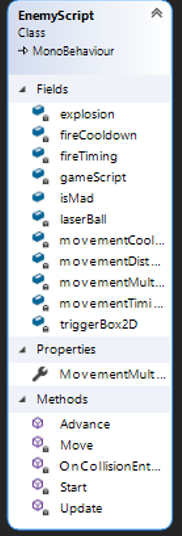
*Comme dit dans l’analyse, les tests on été effectués manuellement avec l’aide d’[icescrum](https://icescrum.cpnv.ch/p/PRETPIMAUR/" \l "/project)*

**

# Dossier d’impementation

### EnemyScript Class Reference

Inheritance diagram for EnemyScript:



### Public Member Functions

* void **Advance** ()

### Public Attributes

* bool **isMad** = false

### Properties

* int **MovementMultiplier** [get]

### Private Member Functions

* void **Start** ()
* void **Update** ()
* void **OnCollisionEnter2D** (Collision2D collision)
* void **Move** ()

### Private Attributes

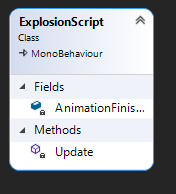
* float **fireTiming**
* float **fireCooldown**
* GameObject **laserBall**
* GameObject **explosion**
* **GameScript** **gameScript**
* BoxCollider2D **triggerBox2D**
* int **movementMultiplier** = -1
* float **movementTiming** = 2
* float **movementCooldown** = 0
* float **movementDistance** = 0.1f

The documentation for this class was generated from the following file:

* Assets/Scripts/EnemyScript.cs

### ExplosionScript Class Reference

Inheritance diagram for ExplosionScript:



### Private Member Functions

* void **Update** ()

### Private Attributes

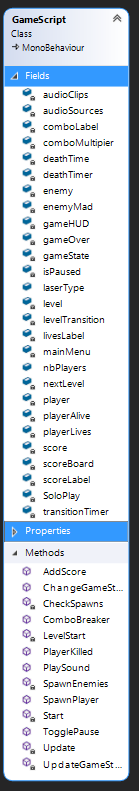
* bool **AnimationFinished** = false

The documentation for this class was generated from the following file:

* Assets/Scripts/ExplosionScript.cs

### GameScript Class Reference

Inheritance diagram for GameScript:



### Public Member Functions

* void **ChangeGameState** (int stateToChange)
* void **AddScore** (int value)
* void **ComboBreaker** ()
* void **PlayerKilled** ()
* void **SpawnPlayer** (int playerId)
* void **TogglePause** ()
* void **PlaySound** (int soundID)

### Properties

* int **GameState** [get]
* bool **IsPaused** [get]
* int **Score** [get]
* bool **NextLevel** [set]
* bool **SoloPlay1** [get]

### Private Member Functions

* void **Start** ()
* void **Update** ()
* void **UpdateGameState** ()
* void **LevelStart** ()
* void **SpawnEnemies** (int rows)
* void **CheckSpawns** ()

### Private Attributes

* int **gameState** = 1

*1: Main menu 2: Game 3: Gameover sequence*

* GameObject **mainMenu**
* GameObject **gameHUD**
* GameObject **scoreBoard**
* GameObject **gameOver**
* GameObject **levelTransition**
* float **transitionTimer**
* AudioClip[] **audioClips**
* AudioSource[] **audioSources**
* GameObject **scoreLabel**
* GameObject **comboLabel**
* GameObject **player**
* GameObject **livesLabel**
* GameObject **enemy**
* int **score**
* int **comboMultipier**
* int **playerLives**
* float **deathTime** =0f
* float **deathTimer** =0
* bool **playerAlive** =false
* bool **isPaused** = false
* bool **nextLevel** = false
* int **level**
* bool **enemyMad**
* bool **SoloPlay** =false

### Member Function Documentation

#### void GameScript.ChangeGameState (int *stateToChange*)

1: Main menu 2: Game 3: Gameover sequence

#### void GameScript.PlaySound (int *soundID*)

##### Parameters

|  |  |
| --- | --- |
| *soundID* | 0-Lazer sound,1-Menu sound,2-player Death, 3- Enemy death |

#### void GameScript.UpdateGameState ()[private]

1: Main menu 2: Game 3: Gameover sequence

### Member Data Documentation

#### int GameScript.gameState = 1[private]

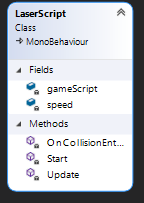
1: Main menu 2: Game 3: Gameover sequence

#### The documentation for this class was generated from the following file:

* Assets/Scripts/GameScript.cs

### LaserScript Class Reference

Inheritance diagram for LaserScript:



### Private Member Functions

* void **Start** ()
* void **Update** ()
* void **OnCollisionEnter2D** (Collision2D collision)

### Private Attributes

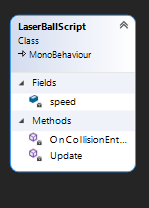
* float **speed** = 20
* **GameScript** **gameScript**

The documentation for this class was generated from the following file:

* Assets/Scripts/LaserScript.cs

### LazerBallScript Class Reference

Inheritance diagram for LazerBallScript:



### Private Member Functions

* void **Update** ()
* void **OnCollisionEnter2D** (Collision2D collision)

### Private Attributes

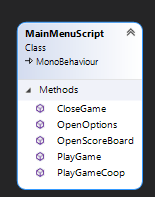
* float **speed** = 20

The documentation for this class was generated from the following file:

* Assets/Scripts/LazerBallScript.cs

### MainMenuScript Class Reference

Inheritance diagram for MainMenuScript:



### Public Member Functions

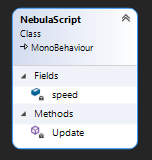
* void **PlayGame** ()
* void **OpenOptions** ()
* void **OpenScoreBoard** ()
* void **CloseGame** ()

The documentation for this class was generated from the following file:

* Assets/Scripts/MainMenuScript.cs

### NebulaScript Class Reference

Inheritance diagram for NebulaScript:



### Private Member Functions

* void **Update** ()

### Private Attributes

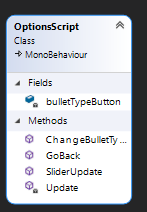
* float **speed**

The documentation for this class was generated from the following file:

* Assets/Scripts/NebulaScript.cs

### OptionsScript Class Reference

Inheritance diagram for OptionsScript:



### Public Member Functions

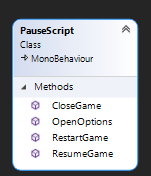
* void **SliderUpdate** ()
* void **GoBack** ()

The documentation for this class was generated from the following file:

* Assets/Scripts/OptionsScript.cs

### PauseScript Class Reference

Inheritance diagram for PauseScript:



### Public Member Functions

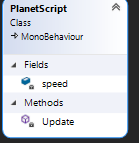
* void **ResumeGame** ()
* void **RestartGame** ()
* void **OpenOptions** ()
* void **CloseGame** ()

The documentation for this class was generated from the following file:

* Assets/Scripts/PauseScript.cs

### PlanetScript Class Reference

Inheritance diagram for PlanetScript:



### Private Member Functions

* void **Update** ()

### Private Attributes

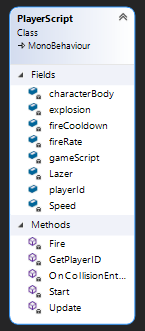
* float **speed**

The documentation for this class was generated from the following file:

* Assets/Scripts/PlanetScript.cs

### PlayerScript Class Reference

Inheritance diagram for PlayerScript:



### Private Member Functions

* void **Start** ()
* void **Update** ()
* void **Fire** ()
* void **OnCollisionEnter2D** (Collision2D collision)

### Private Attributes

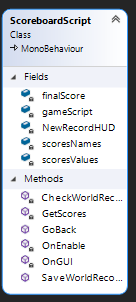
* int **fireRate** = 1
* float **fireCooldown** =1
* GameObject **Lazer**
* Rigidbody2D **characterBody**
* **GameScript** **gameScript**
* float **Speed** = 15
* GameObject **explosion**

The documentation for this class was generated from the following file:

* Assets/Scripts/PlayerScript.cs

### ScoreboardScript Class Reference

Inheritance diagram for ScoreboardScript:



### Public Member Functions

* void **SaveWorldRecord** ()
* void **GoBack** ()

### Private Member Functions

* void **Start** ()
* void **OnEnable** ()
* void **Update** ()
* void **CheckWorldRecord** ()
* void **OnGUI** ()
* void **GetScores** ()

### Private Attributes

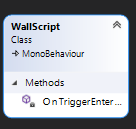
* GameObject **NewRecordHUD**
* **GameScript** **gameScript**
* List< int > **scoresValues**
* List< string > **scoresNames**
* int **finalScore**

The documentation for this class was generated from the following file:

* Assets/Scripts/ScoreboardScript.cs

### WallScript Class Reference

Inheritance diagram for WallScript:



### Private Member Functions

* void **OnTriggerEnter2D** (Collider2D collision)

The documentation for this class was generated from the following file:

* Assets/Scripts/WallScript.cs

# Conclusions

Dans la globalité le projet c’est bien passé, j’ai pu implémenter tous les objectifs établis au début du projet. Dans la durée de ce projet j’ai pu me rendre compte que j’arrive à rapidement implémenter une premier version des fonctionnalités mais que je perds beaucoup de temps pour faire les détails. Je me suis aussi rendu compte à la fin que la documentation était un peu en retard, ça aurait été une meilleure idée de faire un peu plus au long du projet à la place de tout faire à la fin.

Le point où j’ai eu le plus de problèmes a été l’implémentation des menus, principalement parce que c’était la première fois que j’ai fait un menu sur unity. Heureusement ceci est bien documenté me permettant de vite comprendre son fonctionnement.

# Annexes

## Sources – Bibliographie

Documentation Unity <https://docs.unity3d.com/ScriptReference/>

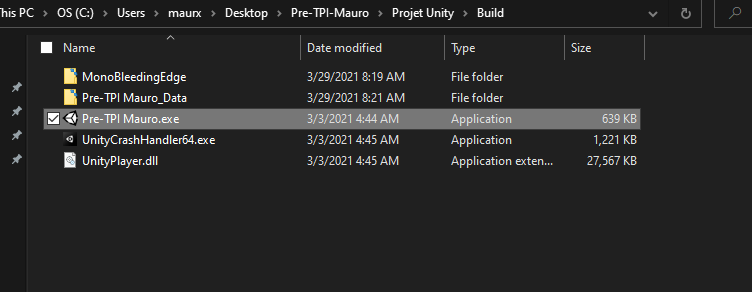
Space Shooter Free <https://assetstore.unity.com/packages/templates/packs/space-shooter-free-107260>

8-Bit Style Sound Effects <https://assetstore.unity.com/packages/audio/sound-fx/8-bit-style-sound-effects-68228>

Tous les sites ont été consultés le 29/3/2021

## Manuel d'Utilisation

Pour lancer le jeu, vous devez lancer l’exécutable « Pre-TPI Mauro.exe » qui se trouve dans le dossier « …\Pre-TPI-Mauro\Projet Unity\Build »



.Dans le menu principale vous avez les différentes options(les noms dissent ce que chaque bouton fait).Si vous faites jouer vous aller commencer le jeu avec 3 vies et votre but est d’éliminer les vagues infinies d’ennemies Les touches au clavier sont les flèches pour déplacer le vaisseau et espace pour tirer. Co-op est presque la même chose mais il aura 2 joueurs à la place d’un seul, Pour ce mode de jeu, vous avez besoin d’une manette Xbox ou PlayStation. Le bouton scoreboard permets d’afficher le top 8 meilleurs scores sur votre machine. Options permets de changer le mode de tir et le volume du jeu. Le bouton Quit permet de fermer le jeu.

