**Projet B1 Informatique**

-

**Développement Logiciel**

Premier Logiciel

-

Sujet 2

Brick Shooter

Réalisé par :

Gonzalez Marie

Alard Adrien

Suivi par :

M. Bejaoui

Sommaire

[1. Les choix du groupe 1](#_Toc42068073)

[1. Choix du sujet 1](#_Toc42068074)

[2. Choix du logiciel et du langage 1](#_Toc42068075)

[2. Tâches de chacun 2](#_Toc42068076)

[1. Gonzalez Marie (alias *Clawchette* sur *GitHub*) 2](#_Toc42068077)

[1. Apparition des ennemis 2](#_Toc42068078)

[2. Drop et buff des ennemis 4](#_Toc42068079)

[3. Sauvegarde 5](#_Toc42068080)

[2. Alard Adrien (alias *tankplasma* sur *GitHub*) 7](#_Toc42068081)

[1. Déplacement du personnage 7](#_Toc42068082)

[2. Tir du personnage 8](#_Toc42068083)

Lien GitHub : <https://github.com/Clawchette/ProjetDevLogiciel>

# 1. Les choix du groupe

## 1. Choix du sujet

Le deuxième sujet, le sujet consistant à réaliser un jeu de type brick shooter et celui que nous avons choisi pour ce projet, nous était le plus familier, il correspondait plus à des jeux auxquels nous avions eu l’occasion de jouer. Ce qui était demandé nous était donc plus clair et le sujet était, du coup, plus rassurant pour nous, contrairement au premier sujet.

Il s’agit aussi d’un sujet qui nous inspirait beaucoup, nous avons très vite trouvé des idées d’éléments supplémentaires à ajouter au jeu même si nous n’avons pas eu le temps de les réaliser et de les implémenter.

## 2. Choix du logiciel et du langage

Pour réaliser ce projet, nous avons choisi d’utiliser *Unity*, et plus particulièrement *Unity* version *2019.3.12f1*. En effet, nous étions déjà habitués à l’utilisation de ce logiciel car nous avons eu l’occasion de l’utiliser, que ce soit pour nos projets personnels ou lors des YDAYS Jeux Vidéos.

Les jeux vidéos étant un de nos domaines d’intérêts, nous désirions développer et améliorer nos compétences et nos connaissances de ce logiciel spécialisé dans la création et le développement de jeux vidéos.

Nous avons décidé de coder nos scripts en *C#* car c’est un langage que nous connaissions déjà et parce que les tutoriels et autres ressources aidant au développement pour *Unity* était majoritairement disponible dans ce langage de programmation.

Nous avons aussi utilisé [*GitHub for Unity*](https://unity.github.com/) que nous avons installé à partir de l’Asset Store dans notre projet.

# 2. Tâches de chacun

Nous avons commencé le projet en séparant les tâches générales à réaliser. Pour cela, nous avons repris les éléments demandés par le sujet et nous avons essayé de détailler les tâches à faire à partir de cela. Par la suite, nous avons simplement réaliser les scripts et les scènes nécessaires selon ce qui nous semblait être prioritaire en nous assurant de prévenir l’autre membre du groupe de ce que nous faisions pour ne pas perdre de temps.

## 1. Gonzalez Marie (alias [*Clawchette*](https://github.com/Clawchette)sur *GitHub*)

Après avoir mis en place le dépôt *GitHub* du projet avec *GitHub for Unity*, je me suis lancée dans la création de scripts permettant, entre autres :   
le déplacement des ennemis de types différents, leur apparition aléatoire (autant l’emplacement que le type est aléatoire), le déclenchement de la fin du jeu lorsqu’un ennemi atteint le joueur ou le sol, le drop de buff des ennemis spéciaux, la fonctionnalité des buffs lorsque le joueur les récupère, l’affichage et l’utilisation des crédits, la sauvegarde (et charge) de la partie, l’affichage et la fonctionnalité du menu pause, le passage d’une scène à l’autre (avec sauvegarde ou charge de partie si nécessaire, la conservation du gameManager d’une scène à l’autre et la fonctionnalité du menu équipement.

J’ai aussi réalisé les prefabs et le placement dans les scènes des éléments du menu pause, du gameManager, des quatres types d’ennemis et des buffs des ennemis.

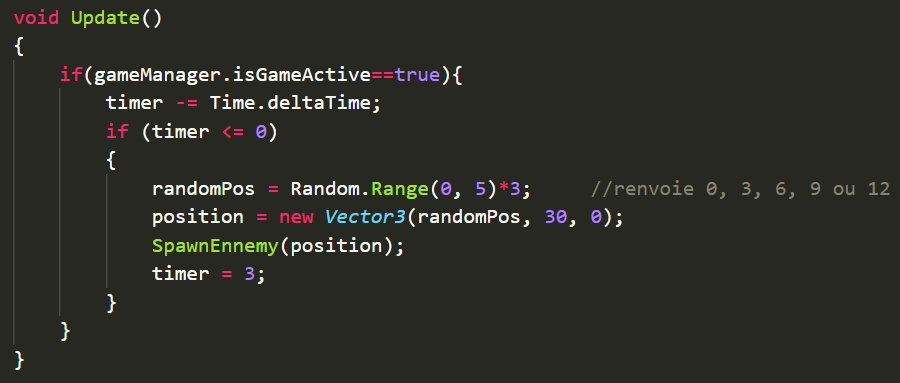
Ensuite, je me suis concentrée sur l’optimisation et le débogage du jeu.

### 1. Apparition des ennemis

Cette fonctionnalité est réalisée par le script *SpawnEnnemies.cs* et utilise les prefabs *DefilementAugm*, *DeplacementAugm*, *VitTirDimin* et *Neutre*. Ce script est sur l’objet *gameManager*.

Tout d’abord, la fonction Update() contient un minuteur de trois secondes1 qui fonctionne tant que le jeu est actif (soit tant que la variable *isGameActive* du script *GameManager.cs* présent sur l’objet *gameManager* et égal à *true*).

A l’intérieur de ce minuteur se trouve un ligne contenant la fonction *Random.Range* qui permet, ici, de renvoyer, au hasard, l’une des cinq positions possible sur l’axe x pour un ennemi, soit 0, 3, 6, 9 ou 12.



Contenu de la fonction *Update()* de *SpawnEnnemies.cs*

Cette valeur est ensuite utilisée par la fonction *SpawnEnnemy(Vector3 position)* qui contient aussi la fonction *Random.Range*. Ici, elle renvoit un nombre au hasard entre 1 et 100 exclu qui est ensuite analysée par une condition *if*.

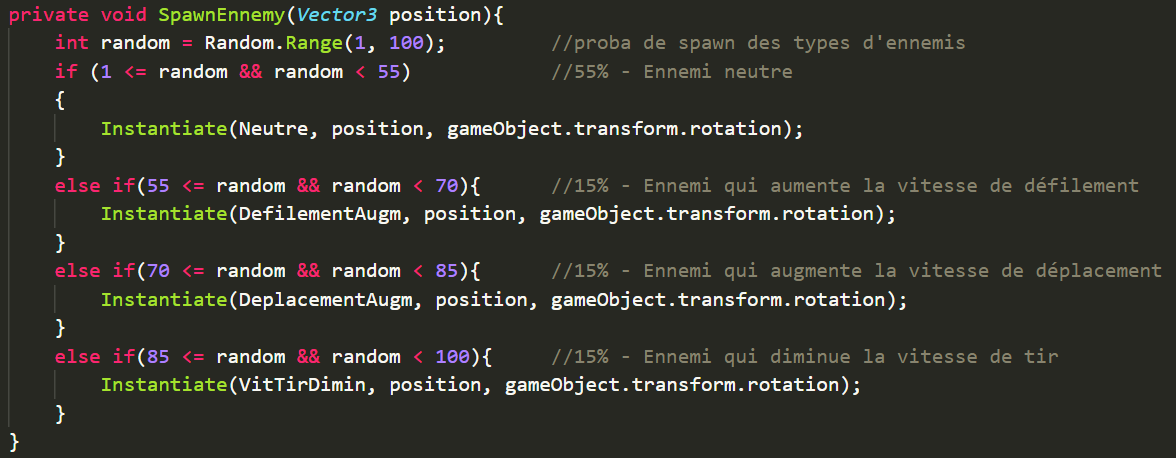
Si ce nombre est strictement inférieur à 55, la fonction *Instantiate* fait apparaître un prefab *Neutre* à la position aléatoire donnée, donnant à l’ennemi neutre une chance d’apparaître de 55%.

Sinon, si ce nombre est strictement inférieur à 70, la fonction *Instantiate* fait apparaître un prefab *DefilementAugm* à la position aléatoire donnée.

Sinon, si ce nombre est strictement inférieur à 85, la fonction *Instantiate* fait apparaître un prefab *DeplacementAugm* à la position aléatoire donnée.

Sinon, si ce nombre est strictement inférieur à 100, la fonction *Instantiate* fait apparaître un prefab *VitTirDimin* à la position aléatoire donnée.

Ainsi, les ennemis spéciaux auront, quant à eux, 15% de chance d’apparaître.



Contenu de la fonction *SpawnEnnemy(Vector3 position)* de *SpawnEnnemies.cs*

### 2. Drop et buff des ennemis

Ces fonctionnalitées sont réalisées par les scripts suivants :

Pour le drop des buffs :

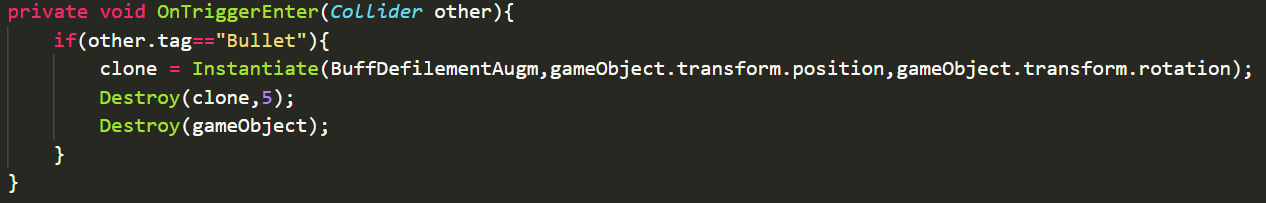
* *DropDefilement.cs* sur le prefab *DefilementAugm* et qui utilise le prefab *BuffDefilementAugm*.
* *DropDeplacement.cs* sur le prefab *DeplacementAugm* et qui utilise le prefab *BuffDeplacementAugm*.
* *DropTir.cs* sur le prefab *VitTirDimin* et qui utilise le prefab *BuffVitTirDimin*.

Pour les buffs eux-mêmes :

* BuffDefilement.cs sur le prefab *BuffDefilementAugm.*
* BuffDeplacement.cs sur le prefab *BuffDeplacementAugm.*
* BuffTir.cs sur le prefab *BuffVitTirDimin.*

Les scripts permettant de drop le buff fonctionnent tous de la même façon. Ils possèdent une fonction *OnTriggerEnter(Collider other)* avec une condition *if* vérifiant si l’objet avec lequel l’ennemi est entré en contact contient le tag “Bullet” et est donc une munition du joueur.

A l’intérieur de cette condition se trouve une ligne qui fait apparaître le buff correspondant à l’ennemi détruit à son emplacement. Il y a aussi une ligne détruisant le buff au bout de cinq secondes, soit une fois qu’il n’est plus visible à l’écran, si le joueur ne le récupère pas. Finalement, il y a une ligne détruisant l’ennemi qui a été touché par la munition du joueur.

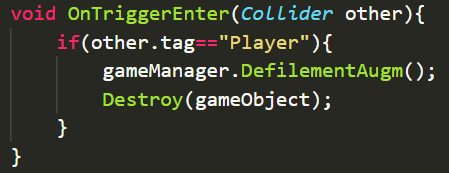


Contenu de la fonction *OnTriggerEnter(Collider other)* de *DropDefilement.cs*

Les scripts permettant la fonctionnalité des buffs fonctionnent eux aussi de la même façon. Ils possèdent une fonction *OnTriggerEnter(Collider other)* avec une condition *if* vérifiant si l’objet avec lequel le buff est entré en contact contient le tag “Player” et est donc le joueur.

A l’intérieur de cette condition se trouve une ligne lançant une fonction permettant de modifier la variable à affecter (la fonction *DefilementAugm()* sur *GameManager.cs* sur l’objet *gameManager* augmente la vitesse de défilement des ennemis, la fonction *DeplacementAugm()* sur *player\_move.cs* sur l’objet *player* augmente la vitesse de déplacement du joueur et la fonction *VitTirDimin()* sur *player\_shot.cs* sur l’objet *player* diminue la vitesse de tir du joueur).

Ensuite, une ligne détruit le buff utilisé.

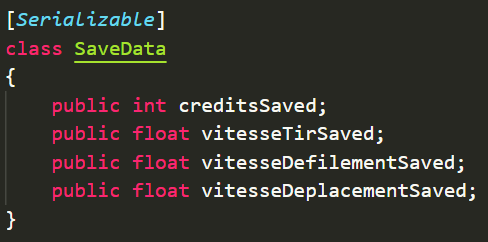


Contenu de la fonction *OnTriggerEnter(Collider other)* de *Buffdefilement.cs*

### 3. Sauvegarde

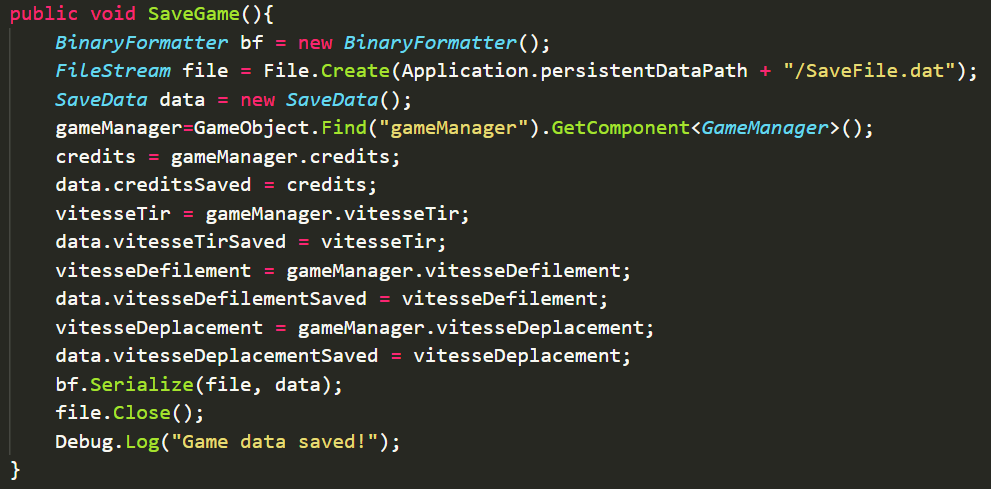
Cette fonctionnalitée est réalisée par le script *Save.cs* qui est sur l’objet *gameManager*. Ce script est séparé en trois parties : la création de la classe *SaveData*, la fonction *SaveGame()* et la fonction *LoadGame()*.

La classe SaveData est précédée par l’indication [Serializable] et sera utilisée pour stocker les variables à sauvegarder dans un seul objet. Il contient les variables *creditsSaved*, *vitesseTirSaved*, *vitesseDefilementSaved* et *vitesseDeplacementSaved*.



Contenu de la classe *SaveData* du script *Save.cs*

La fonction *SaveGame()* qui, comme l’indique son nom, permet de sauvegarder la partie, créé un fichier *SaveFile.dat* et créé aussi une instance de la classe *SaveData* nommée *data*. Les variables à sauvegarder, c’est-à-dire le nombre de crédits, la vitesse de tir, la vitesse de défilement et la vitesse de déplacement, sont mises dans data qui est elle-même formatée et serialized dans le fichier de sauvegarde avant de fermer ce dernier.



Contenu de la fonction *SaveGame()* du script *Save.cs*

La fonction *LoadGame()* reprend les étapes de *SaveGame()* dans le sens inverse, mais d’abord elle vérifie si le fichier de sauvegarde *SaveFile.dat* existe, l’ouvre si c’est le cas, puis vérifie si le fichier contient du texte. Si c’est le cas, le fichier de sauvegarde est fermé après que l’instance *data* de la classe *SaveData* ait été récupérée et deserialized, et les variables qu’elle contenait sont mises dans les variables du jeu qui correspondent.



Contenu de la fonction *LoadGame()* du script *Save.cs*

## 2. Alard Adrien (alias [*tankplasma*](https://github.com/tankplasma)sur *GitHub*)

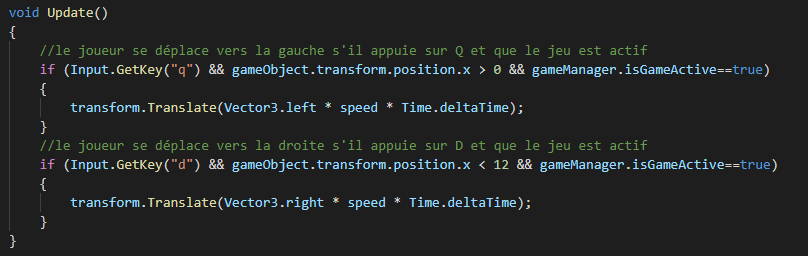
Après avoir clonée le *GitHub* sur mon bureau et avoir découvert le fonctionnement de *GitHub for Unity*, j’ai tout de suite commencé par le déplacement du personnage et réfléchis à la façon par laquelle il devait bouger selon le gameplay et mes capacités, je me suis aussi occupé du tir du personnage tout en gardant les variables changeables pour les bonus/malus comme la vitesse de tir ou la vitesse de déplacement. J’ai eu aussi la tâche de m'occuper des différentes scènes comme :

* Le menu (*menue\_scene.unity*) qui est la scène principale de transition entre les autres, pour lancer une partie, quitter le jeu et aller dans le menu des améliorations
* Améliorations (*amelioration.unity*) qui est donc la scène pour changer les statistiques décrites au dessus (vitesse de déplacement, vitesse de tir, vitesse de défilement des ennemis)
* About (*About.unity*) qui renseigne le joueur sur le but du jeu, comment jouer et les touches afin de pouvoir jouer
* Lose (*lose.unity*) qui est tout bêtement l'écran qui affiche le score du joueur et ses crédits totaux avec l’information de sa défaite.
* Quitter (*player\_quit.unity*) qui, quand le joueur quitte la partie, l’informe du nombre de crédit qu'il a.
* Intro (*intro.unity*) qui lance une cinématique au lancement du jeu

### 1. Déplacement du personnage

Le personnage se déplace avec les touches Q et D qui sont détectées par la fonction *GetKey()* dans le script *player\_move.cs***.**

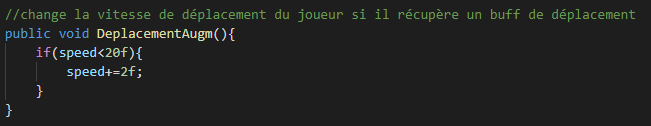
Le personnage qui est représenté par le prefab *player* se déplace grâce à la fonction *transform.Translate()*, translate signifiant un déplacement.



Le personnage se déplace uniquement si la fonction *isGameActive* du *gameManager* est lancé pour éviter tout problème sur les autres scènes.

Les deux variables sur lesquelles sont basées *transform.translate* sont la direction (droite pour la touche D, gauche pour la touche Q) et le temps (*time.deltaTime*), *speed* est une variable qui permet de contrôler la vitesse du personnage.

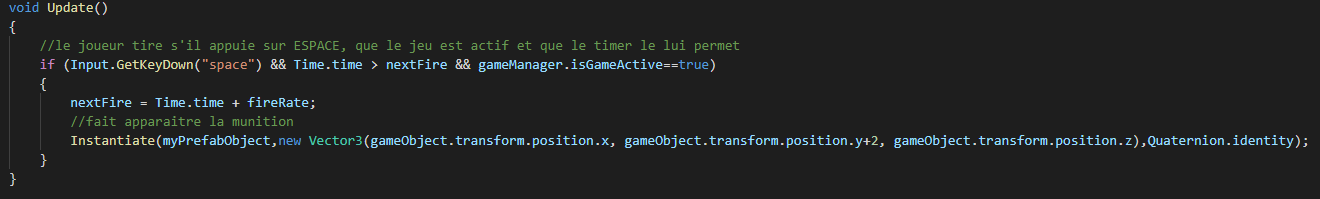




La première approche pour faire bouger le personnage était tout simplement qu’il se "téléporte" de droite à gauche mais la notion de vitesse de déplacement aurait été oubliée.

### 2. Tir du personnage

Deux scripts sont utilisés pour le tir du personnage : le script *player\_shot.cs* et *bullet\_animation.cs*. Le script *player\_shot.cs*est utilisé par *player.*

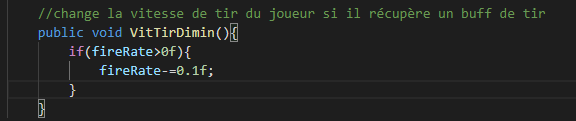
****

La barre d’espace est la touche qui nous permet de tirer : la fonction n’est pas *GetKey()* mais *GetKeyDown()*. La différence est que *GetKey()* capture la touche tant qu'elle est appuyée alors que*GetKeyDown()* capture la touche seulement une fois quand elle est appuyée et il faudra appuyer dessus une nouvelle fois pour que la fonction se relance.

La variable*nextFire* est une limite qui est liée au minuteur pour que le joueur ne puisse pas tirer trop rapidement entre deux tirs. Si *Time.time*, qui est une variable de Unity qui mesure le nombre de secondes passées après le lancement du script, est plus grand que *nextFire* qui est donc une variable donnée égale à 0, alors le joueur peut tirer. Cela donne un intervalle d’une seconde avant le prochain tir. Quand la fonction se lance, *nextFire*se met à jour pour avoir le temps du *Time.time*. Ce qui donne encore une seconde et ainsi de suite.

La fonction *Instantiate()*est une fonction qui permet de faire apparaître un prefab. Il prend les options *myPrefabObject* qui est la balle nommée*bullet*, la position de l’apparition de l’objet grâce à *gameObject* qui est l'entité sur laquelle le script est mis, soit *player,*avec *transform.position*qui est sa position selon les axes X, Y, et Z.

J’ai rajouter +2 à l’axe Y pour éviter de le faire apparaître à l'intérieur de *player* car il disparaîtrait instantanément à cause de la collision.



La fonction *VitTirDimin()*sert à diminuer la vitesse de tir si on ramasse un buff qui le permet.

Le script *bullet\_animation.cs* est rattaché au prefab *bullet*, donc il se lance à chaque fois que *bullet* est créée.