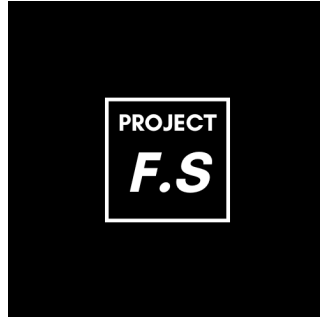


Rapport de soutenance



PROJECT F.S

MTGA

Martin Molines
Thomas Ma-Paw-Youn
Gauthier Ruellan
Aurélien Soula

Janvier 2025

Table des matières

1	Introduction	1
1.1	Spécifications du projet	1
1.2	Rappel des tâches	2
1.3	État de l’art	2
2	Avancement du projet	3
2.1	Premier Prototype	3
2.1.1	Mise en place du joueur et gestion des inputs	3
2.1.2	Intelligence Artificielle	4
2.1.3	Prototype de map	8
2.1.4	Prototype de système de tir	11
2.2	Site Web	16
2.2.1	Choix du thème	16
2.2.2	Mise en place des pages	18
2.3	Scénario	24
3	Conclusion	25

1 Introduction

Ce rapport présente l'état de développement du jeu *Project F.S*, un *FPS* (jeu de tir à la première personne) réalisé par l'entreprise MTGA lors de l'année scolaire 2024-2025 à l'EPITA.

La suite de cette section rappellera les spécifications de notre projet ainsi que l'état de l'art, la section suivante présentera les réalisations de manière thématique, et enfin la conclusion offrira un récapitulatif général de l'état actuel du projet et ouvrira sur l'état souhaité dans les semaines et mois à venir.

1.1 Spécifications du projet

Project F.S est un jeu vidéo de type *Fast FPS/Movement shooter*. Celui-ci sera principalement axé sur le gameplay et plus particulièrement sur les déplacements du personnage joueur dans l'environnement. L'histoire se déroule dans le cadre futuriste d'un laboratoire, plus particulièrement dans des chambres de tests — tests auxquels le joueur est forcé de participer.

Le protagoniste enchaînera les chambres de tests dans lesquelles il devra vaincre de nombreux ennemis avec les armes fournies par le laboratoire. Au fur et à mesure de sa progression, le joueur rencontrera des ennemis de plus en plus puissants et débloquera de nouvelles capacités.

Les graphismes se rapprocheront du style des premières consoles de jeux 3D (Play Station 1, Play Station 2, Nintendo 64) qualifiés de "Low-Poly", c'est-à-dire des modèles 3D utilisant peu de polygones. Les décors ainsi que les couleurs seront également simplistes sans être répétitifs.

1.2 Rappel des tâches

Tâches	Martin	Thomas	Gauthier	Aurélien
Scénario	R		S	
Musique et sons		R		S
Multijoueur		S	R	
IA	S			R
Graphismes/Textures		R	S	
Site Web	S			R
Mécaniques de Gameplay	R			S
Moteur Physique			R	S

Comparé à la première version du cahier des charges, nous avons séparé la section "Gameplay" en deux tâches distinctes pour une plus grande précision.

1.3 État de l'art

Une de nos plus grandes inspirations pour *Project F.S* est l'un des jeux les plus connus dans le domaine du FPS : *Doom*. Le premier titre de la série a été publié en décembre 1993 et la licence est aujourd'hui détenue par l'éditeur Activision. *Doom* est une référence dans le genre des FPS dits "Fast-FPS" ; en effet, il a tellement influencé l'industrie que les jeux ayant repris ce type de gameplay sont souvent qualifiés de "*Doom-like*". *Project F.S* ne sera pas épargné : notre jeu se veut rapide, fluide et jouissif, tout comme l'a été *Doom*.

L'inspiration la plus visible quand il s'agit de gameplay est le célèbre ancien concurrent de *Call of Duty* : *Titanfall*. Ce dernier, publié en 2014 par le studio *Respawn Entertainment* et édité par *Electronic Arts*, a pendant très longtemps essayé de faire de l'ombre au géant du FPS *Call of Duty*. En effet, dans *Titanfall*, c'est aussi la verticalité des combats qui est mise en avant. Le grappin est un élément essentiel dans la liberté de mouvement du joueur et dans sa capacité à s'élever dans les airs.

Enfin, une inspiration qui est cette fois visible dans la direction artistique de *Project F.S* est le très célèbre jeu de réflexion : *Portal*. Ce jeu, développé et édité par le monument du jeu vidéo *Valve Software* en 2008, raconte l'histoire d'une prisonnière, forcée par une intelligence artificielle maléfique à effectuer des tests avec un engin permettant de créer des portails "magiques" qui se relient. L'idée des "salles de tests" cubiques et le concept de "rat de

laboratoire" sont repris dans l'agencement des cartes de *Project F.S* et dans l'histoire du jeu.

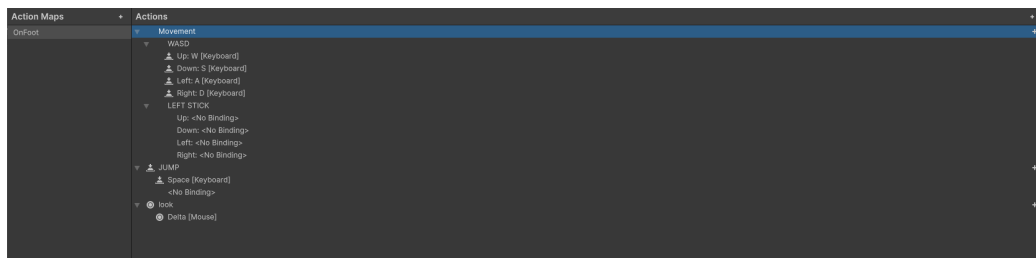
2 Avancement du projet

2.1 Premier Prototype

Ici seront décrits les divers éléments que nous avons intégrés dans le prototype ainsi que les diverses pistes d'améliorations que nous avons trouvées.

2.1.1 Mise en place du joueur et gestion des inputs

Nous avons donc commencé (en particulier Gauthier) par la mise en place d'un personnage joueur (qui n'est pour le moment qu'une capsule). C'est avec cette dite capsule que nous avons testé et utilisé le Unity Input System. Il s'agit d'un outil intégré à Unity qui permet de prendre en charge différents périphériques et de facilement les paramétrer pour les implémenter dans un projet. Nous nous en sommes donc servi pour faire les mouvements de base du personnage joueur, c'est-à-dire marcher.



Pour faire fonctionner le tout, nous avons aussi dû écrire plusieurs scripts C#. Le premier est l'InputManager qui est chargé de détecter les inputs du joueur et de les utiliser dans les fonctions adéquates, comme se déplacer ou sauter, par exemple.

```

public class InputManager : MonoBehaviour
{
    private PlayerInput playerInput;
    private PlayerInput.OnFootActions onFoot;

    private PlayerMotor motor;

    private Look look;

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Awake()
    {
        playerInput = new PlayerInput();
        onFoot = playerInput.OnFoot;
        motor = GetComponent<PlayerMotor>();
        onFoot.JUMP.performed += ctx => motor.Jump();
        look = GetComponent<Look>();
    }
}

```

Toutes les modifications de mouvement sont elles faites dans un script à part entière qui reçoit les mouvements de l'InputManager et les exécute en conséquence. La plus basique de ces exécutions est donc la marche, qui consiste simplement à déplacer le joueur sur les axes x et z.

```

public void ProcessMove(Vector2 input)
{
    Vector3 MoveDirection = Vector3.zero;
    MoveDirection.x = input.x;
    MoveDirection.z = input.y;
    controller.Move(motion: transform.TransformDirection(MoveDirection) * speed * Time.deltaTime);

    |
    playervelocity.y += gravity * Time.deltaTime;
    if (isgrounded && playervelocity.y < 0)
    {
        playervelocity.y = -2f;
    }
    controller.Move(motion: playervelocity * Time.deltaTime);
}

```

2.1.2 Intelligence Artificielle

Pour implémenter des ennemis "intelligents", nous avons commencé par consulter un tutoriel sur l'implémentation de l'IA pour un jeu en 3 dimensions sur Unity, nous apprenant à utiliser les fonctions pré-implémentées de mouvement des IA. Cette vidéo utilisait trois états pour les ennemis : en

cours de patrouille, de poursuite et d'attaque, l'ennemi changeant d'état selon la position du joueur. Nous n'avons pour le moment implémenté que les deux premiers.

```
void Awake()
{
    player = GameObject.FindGameObjectWithTag("Player").transform;
    agent = GetComponent<NavMeshAgent>();
}

// Update is called once per frame
Event function  Aurelien Soula
void Update()
{
    playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);

    if (!playerInSightRange) Patrolling();
    else ChasePlayer();
}
```

Codes des fonctions Awake, permettant d'initialiser les attributs "player" (pour localiser le joueur) et "agent" (pour accéder à son *NavMesh agent*) de l'ennemi, et Update, la fonction qui actualise son action en permanence

Pour l'état de patrouille, l'ennemi prend un point au hasard dans sa distance de recherche, vérifie si ce point est atteignable et si oui, s'y dirige grâce à son *NavMesh agent*. Une fois le point atteint, l'ennemi cherche un nouveau point de la même façon.

```

private void Patrolling()
{
    if (!walkPointSet) SearchWalkPoint();

    if (walkPointSet)
    {
        agent.SetDestination(walkPoint);

        Vector3 distanceToWalkPoint = transform.position - walkPoint;

        if (distanceToWalkPoint.magnitude < 1f)
        {
            walkPointSet = false;
        }
    }
}

Frequently called 1 usage Aurelien Soula
private void SearchWalkPoint()
{
    float randomZ = Random.Range(-walkPointRange, walkPointRange);
    float randomX = Random.Range(-walkPointRange, walkPointRange);

    walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);

    if (Physics.Raycast(origin: walkPoint, direction: -transform.up, maxDistance: 2f, whatIsGround)) walkPointSet = true;
}

```

Code de la fonction de patrouille et de sa fonction auxiliaire
SearchWalkPoint

La fonction de chasse, elle, est beaucoup plus simple. Il nous suffit de prendre la position du joueur comme destination et le *NavMesh agent* se charge du reste.

```

private void ChasePlayer()
{
    agent.SetDestination(player.position);
}

```

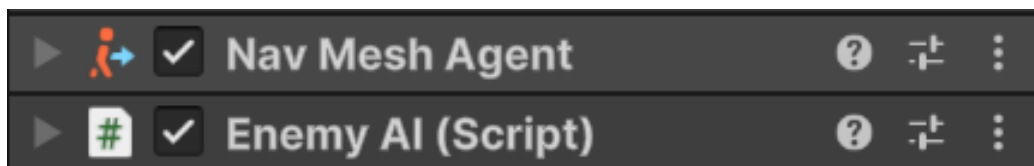
Code de la fonction de chasse

Les ennemis possèdent également une fonction TakeDamage, qui est appelée lorsqu'ils sont touchés par les attaques du personnage joueur et qui décrémente simplement l'attribut "Health" de l'ennemi considéré.


```
public void TakeDamage(int damage)
{
    Health -= damage;
}
```

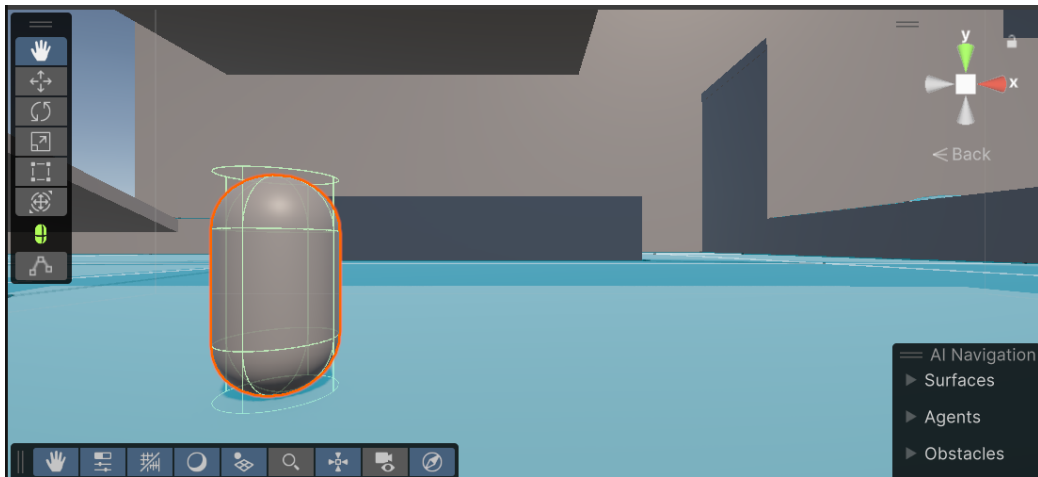
Code de la fonction TakeDamage

Une fois le script fini, nous l'avons ajouté à notre prototype d'ennemi. Nous lui avons également ajouté le fameux *NavMesh agent* qui gère ses déplacements.



Pour que ce script fonctionne correctement, nous avons également eu à ajouter des "layers", plus précisément le layer *whatIsPlayer* sur lequel se trouve le protagoniste et le layer *whatIsGround* où sont placés les objets sur lesquels les ennemis peuvent marcher.

Pour finir, nous avons ajouté une *NavMesh surface* pour que le *NavMesh agent* sache quelles surfaces utiliser pour calculer le chemin à prendre.

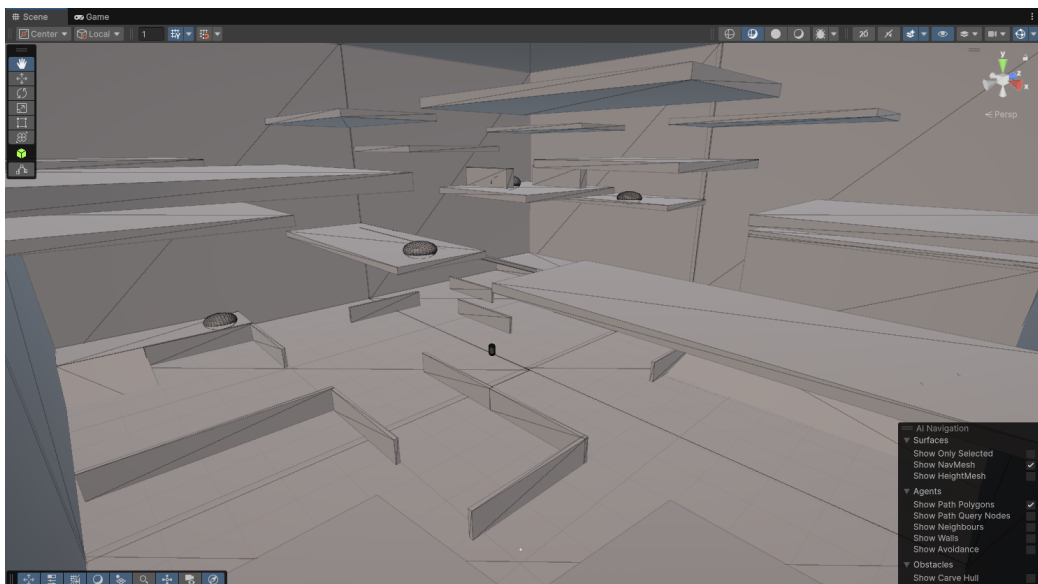


En bleu, les zones sur lesquels la capsule ennemie peut se déplacer

Nous avons donc implémenté une IA pour un ennemi prototype qui se déplace vers le joueur quand celui-ci entre dans son champ de vision. Cette partie a été gérée en majorité par Aurélien.

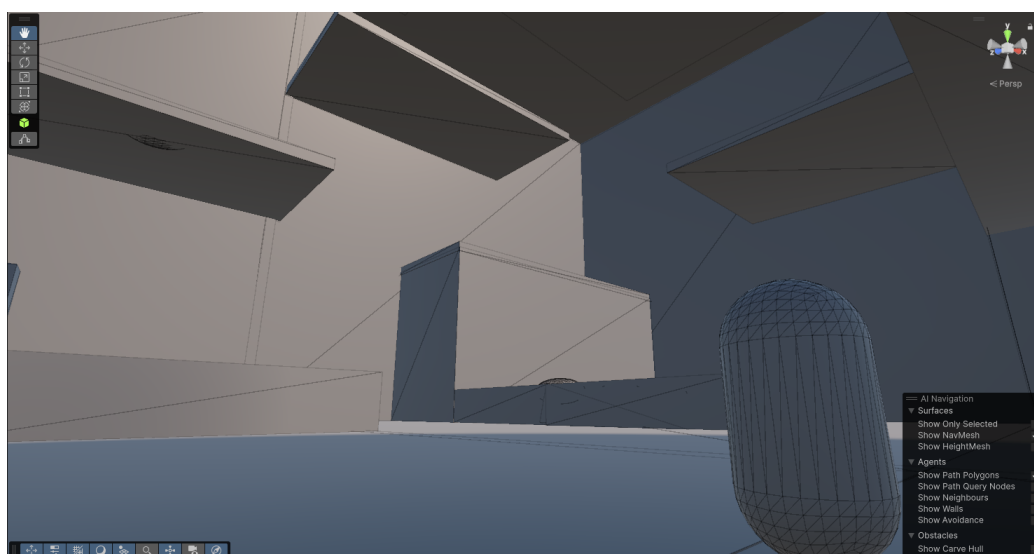
2.1.3 Prototype de map

Nous avons établi une première maquette de la première carte de Project F.S :

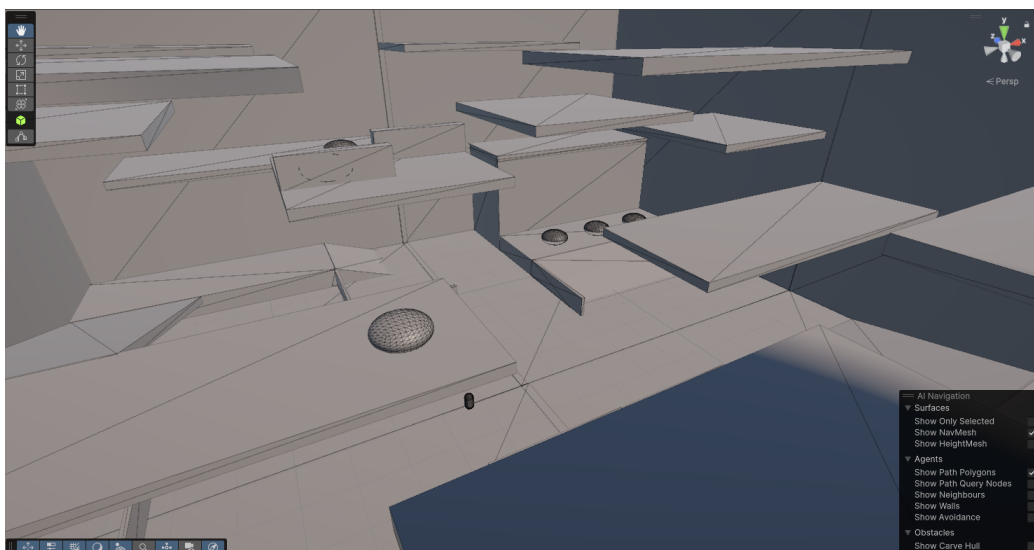


Comme nous pouvons le voir sur cette capture d'écran, le level design se dispose sur plusieurs niveaux. En effet, le joueur (ici représenté par la petite capsule au milieu de l'image) pourra se déplacer où bon lui semble grâce à ses différentes capacités de mobilité. La carte a donc été pensée de manière à fluidifier les mouvements du joueur en lui permettant de s'élever facilement vers les étages supérieurs.

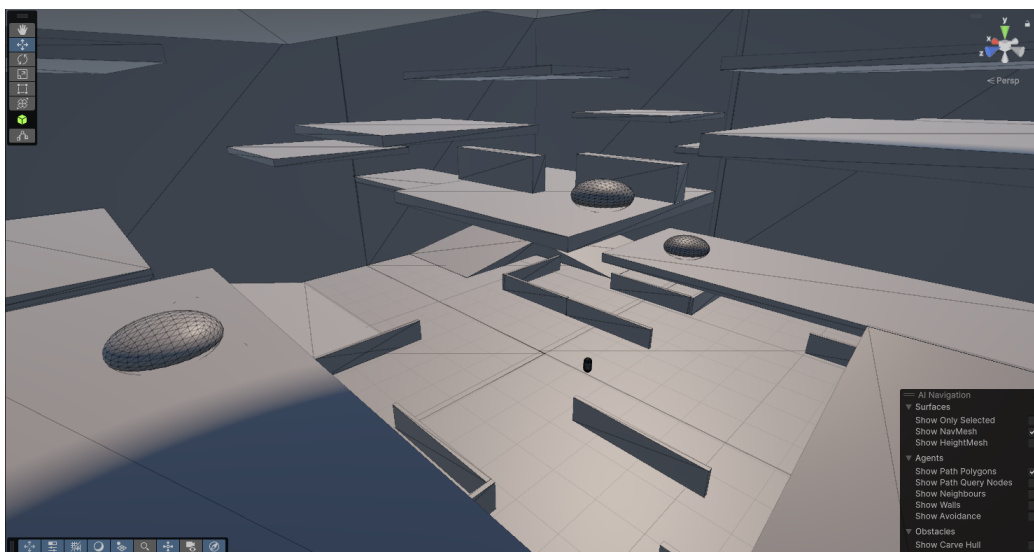
Pour vous donner un ordre de grandeur :



Le joueur se situe donc dans une énorme pièce cubique, constituée de murs lui permettant de se mettre à couvert en cas de difficulté, et de différentes plateformes lui permettant de prendre beaucoup de hauteur. Certains endroits de la carte sont pensés comme étant relativement "plus sûrs", et dans lesquels des "kits de soins" seront mis à disposition pour permettre au joueur de regagner rapidement des points de vie et de reprendre ensuite le combat. D'autres de ces espaces sont des points de réapparition pour le mode multijoueur, ce qui permettra aux joueurs de ne pas tout le temps réapparaître au même endroit et donc d'empêcher une mort instantanée lors de la réapparition.



Les formes sphériques que nous pouvons apercevoir représentent des "tremplins" et serviront à propulser le joueur pour lui donner encore plus de liberté de mouvement.



Thomas est le contributeur principal à la map.

2.1.4 Prototype de système de tir

Afin de mettre en place notre système de tir, réalisé par Aurélien, nous avons, pour cette fonctionnalité également, été aidés par un tutoriel vidéo. Cependant, nous avons effectué plusieurs changements pour que l'implémentation de cet aspect crucial de notre jeu soit conforme à notre vision.

Nous avons créé la nouvelle classe GunSystem dans laquelle nous avons commencé par implémenter le même système de gestion d'input que notre modèle, mais, ayant déjà un "input system", nous savions que cette fonction allait être temporaire.






```

private void InputGet()
{
    if (allowButtonHold)
    {
        shooting = Input.GetMouseButton(0);
    }
    else
    {
        shooting = Input.GetMouseButtonDown(0);
    }

    if (readyToFire && shooting)
    {
        bulletsShot = bulletsPerShot;
        Shoot();
    }
}

```

Des modifications ont plus tard été apportées à notre "input system" pour prendre en compte l'action de tirer, remplaçant cette fonction.

Actions		+
▼	Movement	+,-
▶	WASD	
▶	LEFT STICK	
▼	 JUMP	+,-
	 Space [Keyboard]	
	<No Binding>	
▶	 look	+,-
▼	 Shoot	+,-
	 Left Button [Mouse]	

```
gun = GetComponentInChildren<GunSystem>();  
onFoot.Shoot.performed += StartShooting;  
onFoot.Shoot.canceled += StopShooting;
```

nouvelles lignes de la fonction Awake de InputManager.cs

```
private void StartShooting(InputAction.CallbackContext context)  
{  
    gun.shooting = true;  
}  
  
1 usage  Aurelien Soula  
private void StopShooting(InputAction.CallbackContext context)  
{  
    gun.shooting = false;  
}
```

Fonctions ajoutées à InputManager.cs

Nous avons également implémenté la fonction Shoot, qui utilise la fonction Physics.Raycast pour trouver le point d'impact. Elle gère à la fois le retour visuel et le fait d'infliger des dégâts aux ennemis.


```

public void Shoot()
{
    readyToFire = false;

    if (Physics.Raycast(origin: cam.transform.position, direction: cam.transform.forward, out RayHit, range))
    {
        Debug.Log(RayHit.collider.name);

        if (RayHit.collider.CompareTag("Enemy"))
        {
            RayHit.collider.GetComponent<EnemyAI>().TakeDamage(damage);
        }
    }

    Instantiate(bulletHoleGraphic, RayHit.point, rotation: Quaternion.Euler(0, 180, 0));
    Instantiate(muzzleFlash, attackPoint.position, Quaternion.identity);

    bulletsShot--;

    Invoke(nameof(ResetShot), timeBetweenShooting);

    if (bulletsShot > 0)
    {
        Invoke(methodName: "Shoot", timeBetweenShots);
    }
}

```

```

private void ResetShot()
{
    readyToFire = true;
}

```

Code de la fonction Shoot et de sa fonction auxiliaire ResetShot. Ce code prévoit de tirer plusieurs balles en un appui sur la touche mais nous n'avons pas encore implémenté d'armes utilisant cette option

Après des difficultés pour trouver des assets graphiques permettant un bon retour visuel, nous sommes finalement arrivés à une première version de

notre système de tir.



Prototype du système de tir, avec un cube en guise d'arme

2.2 Site Web

Cette partie a été majoritairement réalisée par Aurélien.

2.2.1 Choix du thème

En vue de réaliser le site web, nous avons tout d'abord suivi le tutoriel *GitHub* portant sur *GitHub Pages* pour prendre en main cet outil. Nous avons, entre autres, appris à utiliser le fichier `_config.yml`. Une fois ces bases assimilées, nous nous sommes lancés à créer la *repository* qui contiendra notre site web.

Nous avons ensuite commencé à chercher un thème approprié pour ce dernier parmi ceux proposés par *GitHub Pages*. Le premier thème nous ayant attiré l'œil a été le thème *Modernist*. En lisant la documentation pour ce thème, nous avons appris comment modifier son code HTML et CSS. Après avoir effectué quelques changements au HTML, surtout au niveau des boutons en haut de page, nous avons abouti à une première version de la page d'accueil.



Première version du site web, celle-ci utilise le thème *Modernist*

En le testant sur nos idées, nous avons trouvé que ce thème faisait trop "blog". Nous sommes donc partis à la recherche d'un nouveau thème. C'est alors que nous avons découvert le thème *Midnight*. La disposition de ce thème correspondait bien à ce que nous recherchions, et ses couleurs étaient en accord avec la charte graphique de notre jeu. Nous avons donc édité le `_config.yml` du projet pour changer le thème ainsi que le fichier `_layouts/default.html` pour modifier le HTML par défaut, encore une fois principalement pour toucher aux boutons du site.



Nouvelle version du site web, avec le thème *Midnight*

2.2.2 Mise en place des pages

Ayant le squelette du site ainsi qu'une ébauche de page d'accueil, nous nous sommes lancés dans la réalisation des autres pages. Mais avant de créer les pages en elles-mêmes, nous voulions faire en sorte que les boutons envoient réellement vers les autres pages.

Cependant, nous avons rencontré des difficultés pour y parvenir. En effet, nous n'arrivions pas à ce que le fichier markdown auquel nous cherchions à accéder soit correctement interprété par *GitHub Pages*. Nous nous sommes rendu compte que c'était parce qu'il manquait l'en-tête au fichier. Cependant, une fois l'en-tête ajoutée, nous arrivions sur une erreur 404 en tentant d'aller sur la page qui était en train d'être ajoutée. Lors de ces péripéties, nous avons configuré la variable `baseurl` dans le fichier de configuration pour un maintien plus propre des liens à travers le site. Mais ce n'est pas l'action qui a réglé les problèmes d'accès, en effet c'était quelque chose de beaucoup plus simple.

Quand on ajoute à un fichier une en-tête markdown, le fichier n'est plus accessible à l'adresse `nom.github.io/repository/fichier.md` mais à l'adresse `nom.github.io/repository/fichier` à la place, et c'est cette particularité qui s'est révélée coûteuse en temps.

Une fois les liens corrigés pour prendre en compte ce changement, nous avons pu réellement commencer à travailler sur le contenu des autres pages, à savoir la page de présentation du projet, celle détaillant les ressources utilisées et celle comportant un lien de téléchargement du jeu ainsi que des rapports. Nous avons commencé par cette dernière. Cette page n'est pour l'instant pas très remplie, le jeu n'étant pas encore téléchargeable, mais pour la réaliser, nous avons dû acquérir les connaissances pour réaliser des sections en markdown pour séparer la zone de téléchargement du jeu de celle des rapports.

```
# Téléchargement
Cette page inclura un lien de téléchargement du jeu ainsi que des différents rapports rédigés au cours de l'année.

## Téléchargement du rapport
Cette section inclura les liens de téléchargement vers les rapports rédigés pour les multiples soutenances de l'année.

## Téléchargement du jeu
Cette section inclura un lien de téléchargement du jeu une fois celui-ci finalisé, ainsi qu'un lien de téléchargement vers une version allégée du projet.
```

Code markdown

Téléchargement

Cette page inclura un lien de téléchargement du jeu ainsi que des différents rapports rédigés au cours de l'année.

Téléchargement du rapport

Cette section inclura les liens de téléchargement vers les rapports rédigés pour les multiples soutenances de l'année.

Téléchargement du jeu

Cette section inclura un lien de téléchargement du jeu une fois celui-ci finalisé, ainsi qu'un lien de téléchargement vers une version allégée du projet.

Résultat visible sur le site

Ensuite, nous avons fait la page de présentation du projet, à laquelle nous avons souhaité intégrer une présentation des membres. Nous nous sommes en conséquence renseignés sur la façon d'ajouter des images grâce à du HTML ainsi que des citations en markdown pour rendre cette partie plus vivante.

```

### Soula Aurélien

> Passionné de jeux vidéo du plus loin que je me souvienne,
> j'ai joué à presque tous les styles de jeux vidéo existants, des RPGs aux jeux de combat en passant par les MOBA, les
jeux de plateformes, les jeux de courses, les _party games_, les _Rogue Like_... J'ai ainsi accumulé de l'expérience aussi
bien sur des jeux solos que multijoueurs voire massivement multijoueurs. A défaut de pouvoir m'investir dans une pratique
sportive compétitive, je me suis investi dans la compétition esportive sur Pokémon VGC.
>
> Je me suis vite intéressé à l'aspect programmation qu'il y a derrière chaque jeu vidéo. Je me suis donc essayé à RPG
maker - logiciel facilitant la création de _Role Playing Games_, généralement en deux dimensions - ce qui m'a donné un
avant-goût de la programmation. Au fur et à mesure des années, j'ai pris la décision d'entreprendre des études dans
l'informatique au sens large sans restreindre les débouchés de ma formation à l'industrie vidéoludique. L'an dernier, j'ai
contribué à améliorer un site web, ce qui a été pour moi une très bonne occasion d'apprendre les bases des langages HTML et
JavaScript. Je vais maintenant m'appuyer sur ces connaissances pour, malgré ma non-expertise du genre des _Fast FPS_ (type
de jeu retenu par notre groupe), participer activement au développement du jeu et de son site web et développer de nouveaux
savoirs en chemin.

```

Code markdown/HTML

Soula Aurélien



Passionné de jeux vidéo du plus loin que je me souviene, j'ai joué à presque tous les styles de jeux vidéo existants, des RPGs aux jeux de combat en passant par les MOBA, les jeux de plateformes, les jeux de courses, les party games, les Rogue Like... J'ai ainsi accumulé de l'expérience aussi bien sur des jeux solos que multijoueurs voire massivement multijoueurs. A défaut de pouvoir m'investir dans une pratique sportive compétitive, je me suis investi dans la compétition esportive sur Pokémon VGC.

Je me suis vite intéressé à l'aspect programmation qu'il y a derrière chaque jeu vidéo. Je me suis donc essayé à RPG maker - logiciel facilitant la création de Role Playing Games, généralement en deux dimensions - ce qui m'a donné un avant-goût de la programmation. Au fur et à mesure des années, j'ai pris la décision d'entreprendre des études dans l'informatique au sens large sans restreindre les débouchés de ma formation à l'industrie vidéoludique. L'an dernier, j'ai contribué à améliorer un site web, ce qui a été pour moi une très bonne occasion d'apprendre les bases des langages HTML et JavaScript. Je vais maintenant m'appuyer sur ces connaissances pour, malgré ma non-expertise du genre des Fast FPS (type de jeu retenu par notre groupe), participer activement au développement du jeu et de son site web et développer de nouveaux savoirs en chemin.

Résultat visible sur le site

Pour la page sur les ressources utilisées, nous avons appris comment intégrer des hyperliens et des listes en markdown pour rendre la présentation des ressources la plus propre possible.

```
## Ressources générales pour le code du jeu
* [Introduction à Unity par Game Maker's Toolkit](https://www.youtube.com/watch?v=xtQMytORBmM)
* [Guide sur le mouvement dans un jeu à la première personne par Natty GameDev](https://www.youtube.com/watch?v=rJqP5FesxLk)
```

Code markdown

Ressources générales pour le code du jeu

- » Introduction à Unity par *Game Maker's Toolkit*
- » Guide sur le mouvement dans un jeu à la première personne par *Natty GameDev*

Résultat visible sur le site

Une fois la charte graphique exacte du jeu établie, la couleur des hyperliens pourrait être amenée à changer.

2.3 Scénario

Le scénario de Project F.S se déroule dans un futur sombre, où le mystère et la tension dominant. Le joueur est plongé dans un laboratoire secret, conçu pour tester l'efficacité de nouvelles armes dans des situations extrêmes. Ce cadre oppressant aide à immerger le joueur dans un environnement où chaque mouvement peut être fatal, tout en justifiant des mécaniques de gameplay axées sur la rapidité, la stratégie et l'utilisation d'armes.

L'histoire commence avec le réveil brutal du protagoniste dans une cellule froide et impersonnelle. Une voix artificielle lui annonce qu'il a été choisi pour participer à des tests visant à évaluer l'efficacité des armes et des technologies de destruction. Ce n'est pas simplement un test de survie, mais une série d'épreuves pour voir comment ces armes se comportent dans des conditions extrêmes, sans se soucier du sort du cobaye.

Le joueur est ensuite envoyé dans une série de "chambres de tests", conçues pour mettre à l'épreuve les capacités de ses outils à réagir face à des menaces mortelles. Chaque salle contient des armes sophistiquées, et le joueur devra les utiliser pour se défendre ou accomplir des objectifs spécifiques. L'architecture des salles est simple et froide, renforçant l'idée que ce laboratoire est un endroit clinique et inhumain. L'objectif principal de ces tests est de voir jusqu'où les armes peuvent aller, sans aucune considération pour la vie humaine.

Ce scénario a été écrit en majeure partie par Martin.

3 Conclusion

En conclusion, comme prévu, nous avons réalisé un prototype fonctionnel du jeu avec un moteur physique, une implémentation basique de l'IA, un système de tir et une première "map", ainsi que posé les bases pour le site web accompagnant le jeu.

D'ici la prochaine soutenance, nous comptons commencer à implémenter le multijoueur, peaufiner la physique, améliorer l'IA et introduire plus de mécaniques de jeu et d'interactions. Nous allons continuer à travailler pour faire de *Project F.S* un jeu abouti.