

RAPPORT DE SOUTENANCE 2

# REMOVED FROM THE PAST

BY TIMESOFT DEV

EPITA SUP S2 - Promo 2026

Abel ROFFIAEN, Yusuf OZEN, Yann MESSE et Solène DINTINGER



Figure 1: Logo du jeu.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Partie Commune</b>	<b>3</b>
2.1	Répartition des tâches . . . . .	3
2.2	Avancement du projet . . . . .	4
2.3	Groupe du projet . . . . .	5
<b>3</b>	<b>Parties Individuelles</b>	<b>6</b>
3.1	Menu Principal . . . . .	6
3.1.1	Lancement du jeu (Yann) . . . . .	6
3.1.2	Musique (Abel) . . . . .	7
3.1.3	Graphismes (Solène) . . . . .	8
3.1.4	Boutons des panels (Yann) . . . . .	9
3.2	Sauvegarde (Yann) . . . . .	9
3.3	Carte . . . . .	11
3.3.1	Graphismes et placement (Solène et Yusuf) . . . . .	11
3.3.2	Placement carte extérieur (Yann) . . . . .	12
3.4	Interface du joueur . . . . .	13
3.4.1	Interface du joueur (Solène) . . . . .	13
3.4.2	Spécification de l'affichage (Yann) . . . . .	14
3.5	Inventaire (Yusuf) . . . . .	14
3.6	Système de combat (Solène) . . . . .	16
3.7	Ennemis (Solène) . . . . .	21
3.8	Niveau du joueur / Système d'amélioration (Yann) . . . . .	24
3.9	Multijoueur (Solène) . . . . .	25
3.10	Site web (Abel) . . . . .	26
<b>4</b>	<b>Conclusion</b>	<b>27</b>
<b>5</b>	<b>Sources</b>	<b>28</b>

# 1 Introduction

Presque deux mois suite à la première soutenance, ce rapport de deuxième soutenance nous permet de mettre à jour notre avancée et de faire revenir nos remarques et difficultés liées à nos nouveaux objectifs. Nous apporterons nos explications en se basant sur le CdC du projet.

Ce projet nous permet d'apprendre à travailler sur plusieurs projets en même temps. Effectivement, il est parfois difficile de trouver le temps d'alterner entre les devoirs de la semaine, les QCMs du lundi et le projet. De plus, le travail d'équipe apporte forcément des avantages comme de ne se préoccuper que de parties qui nous intéresseraient plus que d'autres et de montrer nos compétences. Cependant, se mettre d'accord sur le temps où une étape doit être terminée était plutôt difficile, surtout dû aux semaines où certains sont moins disponibles que d'autres, il a été plus compliqué de travailler tous ensemble que ça ne l'était avant la première soutenance.

Nous apprenons cependant en même temps à assumer nos responsabilités. En effet, il ne faut pas mettre le groupe entier en retard et cela permet d'apprendre à nous motivé pour pouvoir travailler dans les temps.

Nous nous habituons de plus en plus à la programmation en C# et surtout, à Unity. Nous le trouvons encore tous plutôt difficile à utiliser mais certains automatismes commencent à arriver, qui nous permettent d'avancer plus rapidement.

La base du jeu étant déjà prête, nous pouvions nous concentrer sur le technique comme la programmation et la création des objets sur Unity. Nous avons réussi à avancer dans les temps avec l'aide de tout le monde.

Nous avons pu profiter du temps libre durant nos vacances et des quelques jours avant la soutenance pour notamment nous concentrer sur le projet.

Notre jeu est un RPG reprenant des éléments principalement de "The Legend of Zelda" et "Pokémon". Notre personnage peut se déplacer sur grande carte dans une ville et à l'extérieur et pourra enclencher des combats avec différents ennemis se situant sur la carte. Notre personnage doit alors parcourir ce monde afin de trouver une potion qui sauvera sa soeur, dans le passé.

## 2 Partie Commune

### 2.1 Répartition des tâches

Tâche	Abel	Solène	Yusuf	Yann
Menu principal	Suppléant	Principal		
Commandes			Suppléant	Principal
Options	Principal		Suppléant	
Graphismes	Suppléant	Principal		
Carte		Suppléant		Principal
Apparition	Suppléant		Principal	
Interface joueur		Suppléant	Principal	
Histoire principale	Suppléant			Principal
Système de déplacement		Suppléant	Principal	
Interaction PNJ		Suppléant	Principal	
Récompenses	Principal		Suppléant	
Système de combat		Suppléant	Principal	
Ennemis		Principal		Suppléant
Niveau du joueur			Suppléant	Principal
Système d'amélioration	Principal			Suppléant
Multijoueur		Principal	Suppléant	
Sauvegarde	Suppléant			Principal
Quitter	Principal			Suppléant
Site Web	Principal			Suppléant

Figure 2: Tableau des répartitions des tâches.

Nous n'avons pas apporter de changement pour notre répartition des tâches, mais nous continuons de nous entraider sinon nous avons des soucis ou bloquons sur un passage.

Par exemple, nous nous occupons un peu tous de la carte car les objets et bâtiments sont longs à dessiner. Cela permet de réduire la tâche pour Solène et Yann, qui peuvent alors plus se concentrer sur le placement des objets sur la carte.

## 2.2 Avancement du projet

Tâche	A la soutenance 1	A la soutenance 2	A la soutenance 3
Menu principal	Avancé	Terminé	---
Commandes	Avancé	Terminé	---
Options	Commencé	Avancé	Terminé
Graphismes	Commencé	Avancé	Terminé
Carte	Avancé	Terminé	---
Apparition	----	Commencé	Terminé
Interface joueur	----	Avancé	Terminé
Histoire principale	Terminé	---	---
Système de déplacement	Terminé	---	---
Interaction PNJ	----	Avancé	Terminé
Récompenses	Commencé	Terminé	---
Système de combat	Commencé	Terminé	---
Ennemis	Commencé	Avancé	Terminé
Niveau du joueur	Commencé	Terminé	---
Système d'amélioration	Commencé	Terminé	---
Multijoueur	----	Commencé	Terminé
Sauvegarde	----	Avancé	Terminé
Quitter	Avancé	Terminé	---
Site Web	Commencé	Terminé	---

Figure 3: Tableau d'avancement des tâches.

Maintenant que nous avons pris connaissance avec Unity, nous avons pu créer plus de programmes et d'objets concrets que pour la première soutenance. Le jeu prend alors forme petit à petit.

Le menu principal étant presque déjà fini, nous n'avions qu'à apporter de petites modifications pour que celui-ci prenne entièrement forme. Nous avons alors pu nous concentrer pleinement aux 2 scènes de jeu principales, la ville et l'extérieur, ainsi que la scène de combat.

Nous avons tous pris nos repères avec Unity et nous pouvons maintenant avancer plus vite qu'à nos débuts.

Le programmation reste encore le point le moins clair pour nous. Il faut évidemment que nous cherchons par nous-mêmes des fonctions qui pourrait nous aider dans nos propres fonctions. Nous ne pouvons pas encore dire que nos codes sont parfaitement complexifié et écrit de la meilleure façon mais nous corrigons nos codes précédents en plus d'en écrire de nouveaux.

Maintenant que nous connaissons très bien nos objectifs et assez bien les méthodes que nous pouvons utiliser, nous alternons toujours entre les différentes tâches pour ne pas nous lasser et rester bloquer trop longtemps.

Nous faisons encore et toujours de notre mieux pour que le projet avance correctement et que nous puissions arriver à un résultat qui nous plaira à tous.

## 2.3 Groupe du projet

Notre groupe est composé de ses 4 membres originels : le chef de groupe Abel ROFFIAEN, Yann MESSE, Yusuf OZEN et Solène DINTINGER.

Aucun changement majeur n'a eu lieu dans notre groupe. Nous travaillons ensemble et aussi de notre propre côté régulièrement.

Nous utilisons l'option Collaborate de Unity pour travailler ensemble, cependant Collaborate a été maintenant entièrement remplacer par Plastic SCM. Les données de notre projet ont alors été automatiquement déplacé vers Plastic SCM sans que nous pouvions avoir notre mot à dire. Seulement, Plastic SCM étant payant pour un groupe de 4, nous ne souhaitons pas continuer comme cela. En attendant de trouver une autre alternative, nous utilisons alors le Git mis à disposition par l'école pour apporter nos modifications.

Ce changement a malheureusement été un grand contretemps dans notre avancée et nous cherchons encore d'autres solutions pour travailler ensemble sur le même projet.

D'un autre côté, pour un bon suivi de l'avancement de notre projet, nous organisons encore des réunions hebdomadaires en début de semaine. Nous parlons alors de notre avancée personnelle de la semaine précédente, de ce que nous avons besoin pour la suite et surtout de ce que nous allons faire la semaine suivante.

Ces réunions peuvent varié entre 30 minutes si tout c'est bien passé dans la semaine pour tout le monde à 1h30 si nous travaillons sur le coup ensemble pour régler les problèmes de chacun.

Nous discutons toujours sur l'application Discord sur notre serveur consacré au projet. Nous restons à l'affût de nouveaux messages pour pouvoir aider tout le monde et rester en contact régulièrement.

### 3 Parties Individuelles

#### 3.1 Menu Principal

##### 3.1.1 Lancement du jeu (Yann)

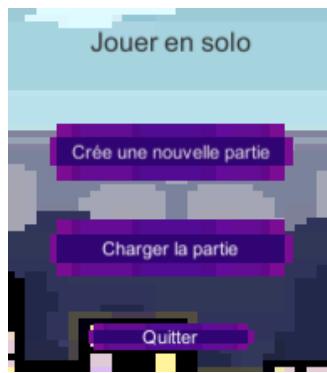


Figure 4  
Image de l'écran pour lancer le jeu

Tout d'abord, j'avais opté pour un système avec plusieurs sauvegardes avec un nom pour chaque sauvegarde. Cependant, cela c'est avéré très compliqué. J'ai donc décidé de commencer par faire une sauvegarde unique. On peut soit charger la dernière partie soit en créer une nouvelle.

Pour faire la sauvegarde, j'ai utilisé les PlayerPrefs qui sont dans le package initial de Unity Hub. Ils permettent de sauvegarder un tuple (nom,donnée). La donnée peut être un string, un entier ou un nombre décimal. Cependant, on ne peut pas protéger les données sauvegardées à l'aide des PlayerPrefs.

Quand on crée une nouvelle partie, on efface les anciennes données et initialise les nouvelles données principales telles que les coordonnées, la scène, mais aussi les statistiques initiales. Voici le script :

```
1  public void NewSave() //supprime la dernière sauvegarde et cree
2      une nouvelle sauvegarde
3  {
4      PlayerPrefs.DeleteAll();
5      PlayerPrefs.SetInt("SauvegardeExist", 1);
6      PlayerPrefs.SetString("Scene", "Game");
7      PlayerPrefs.SetFloat("X", 0);
8      PlayerPrefs.SetFloat("Y", 0);
9      PlayerPrefs.SetInt("Force", 5);
10     PlayerPrefs.SetInt("Vitesse", 5);
11     PlayerPrefs.SetInt("Agilite", 5);
12     PlayerPrefs.SetInt("Chance", 5);
13     PlayerPrefs.SetInt("Resistance", 5);
14     PlayerPrefs.SetInt("Level", 1);
15     PlayerPrefs.SetInt("Exp", 0);
16     SceneManager.LoadScene("Game");
17 }
```

Lorsque l'on lance le jeu avec le bouton charger la partie, je vérifie si une sauvegarde existe. Si elle existe alors on lance directement le jeu qui chargera automatiquement les données. S'il n'existe aucune sauvegarde alors un message d'erreur apparaît.

Voici le script permettant de vérifier si une sauvegarde existe et dans le cas contraire afficher le message d'erreur:

```

1 public void chargeGame() //lance la dernière sauvegarde ou alors
2         affiche une erreur si pas de sauvegarde
3     {
4         if (PlayerPrefs.HasKey("SauvegardeExist"))
5             SceneManager.LoadScene(PlayerPrefs.GetString(
6                 "Scene"));
7         else
8         {
9             errorWindow.SetActive(true);
10            myEventSystem=GameObject.Find("EventSystem");
11            myEventSystem.GetComponent<UnityEngine.Events.
12                EventSystem>().SetSelectedGameObject(GameObject.
13                Find("Quitter"));
14        }
15    }

```



Figure 5: Message d'erreur lorsque aucune sauvegarde n'existe

Un dernier avantage des PlayerPrefs est que l'on peut transférer des données entre différentes scènes en les sauvegardant.

### 3.1.2 Musique (Abel)

Comme expliqué durant la première soutenance, nous voulions créer une ambiance unique pour le jeu. J'ai donc continué la création des sons et musiques du jeu. Je dispose maintenant d'un panel d'une dizaine de bruitages différents (bruits de pas, bruits de respiration, bruits d'attaque, bruits de vent, etc.) et de trois musiques (musique de ville, musique de combat, musique de campagne).



Figure 6: Sons seulement lisible avec AdobeReader !

Les sons et musiques ont été produits grâce à FL Studio.

### 3.1.3 Graphismes (Solène)

Le menu principal étant déjà presque complet lors de la première soutenance, il ne restait pas grand chose à modifier. J'ai ici simplement modifier le logo du jeu que nous avions mis lors de la première soutenance. J'avais fait le choix de mettre la même icône que celle de l'icône du jeu. Seulement, le rendu n'était pas très satisfaisant. Nous avons alors opté pour une icône plus simpliste avec simplement du texte.

Le menu principal paraît alors plus simpliste et rend simplement mieux, ce qui est bien pour encourager l'utilisateur à jouer.



Figure 7: Premier logo du menu principal.



Figure 8: Logo final du menu principal.



Figure 9: Image menu principal.

### 3.1.4 Boutons des panels (Yann)

La première modification que j'ai faite dans le menu principale est d'unifier l'affichage des boutons. Désormais, ils ressemblent tous à ceci :



Figure 10: image bouton

De plus, nous avions un problème lorsque nous naviguions entre les panels, les boutons dans le panel que nous venions d'ouvrir n'étaient pas présélectionner. Pour remédier à cela, j'ai ajouter une ligne de code qui permet qu'à chaque fois que l'on ouvre un panel, l'EventSystem prend comme premier bouton le bouton du nouveau panel activé. Voici le script pour cela :

```
1 public void OptionButton() //active le panel d'option
2 {
3     optionWindow.SetActive(true);
4     myEventSystem = GameObject.Find("EventSystem");
5     myEventSystem.GetComponent<UnityEngine.Events.
6         EventSystem>().SetSelectedGameObject(GameObject.Find
7             ("Musique Volume"));
8 }
```

J'ai effectué ses modifications pour les boutons Histoire, Commandes, Options et les boutons quitter dans les Canvas.

## 3.2 Sauvegarde (Yann)

Comme expliqué précédemment, j'ai utilisé les PlayerPrefs pour faire la sauvegarde des données. L'avantage est la facilité de l'outil. Le désavantage est que la personne peut modifier ces données facilement car elle ne sont pas protégé. De plus, elle ne permettent que de sauvegarde des int, float ou string, chacun associé à un string qui est le nom de la valeur. On peut aussi vérifier si un nom est associé à une valeur et, évidemment, supprimer une valeur grâce à son nom.

Comme par exemple lorsque l'on charge une nouvelle partie :

```
1 public void NewSave() //supprime la dernière sauvegarde et cree
2     une nouvelle sauvegarde
3 {
4     PlayerPrefs.DeleteAll(); //ici on efface toute les
5         valeurs sauvegarde
6     PlayerPrefs.SetInt("SauvegardeExist", 1); // On
7         initialise un tuple avec comme nom SauvegardeExist
8         et comme valeur 1
9     PlayerPrefs.SetFloat("X", 0); // On initialise 2 float
10    coordonnees
11    PlayerPrefs.SetFloat("Y", 0);
12    PlayerPrefs.SetInt("Force", 5);
13    PlayerPrefs.SetInt("Vitesse", 5);
14    PlayerPrefs.SetInt("Agilite", 5);
15    PlayerPrefs.SetInt("Chance", 5);
16    PlayerPrefs.SetInt("Resistance", 5); //Les 5 lignes du
        dessus permettent de sauvegarder les statistiques
17    PlayerPrefs.SetInt("Level", 1); //Ici le niveau du joueur
18    PlayerPrefs.SetInt("Exp", 0); //Ici l'experience du
        joueur
19    SceneManager.LoadScene("Game");
20 }
```

Où que l'on récupère les dernières coordonées du joueur :

```
1
2 void Start()// On initialise les coordonnees du joueur
3 {
4     float X = PlayerPrefs.GetFloat("X");
5     float Y = PlayerPrefs.GetFloat("Y");
6     //Ici on recuperer les coordonnees qui ont ete
        enregister avant
7     rb.position = new Vector2(X, Y);
8 }
```

Ainsi on peut sauvegarder les coordonnées, statistiques, paramètres, niveau du joueur,... Et les récupérer très facilement d'une scène à l'autre. Ces fonctions sont utilisées dans tous les scripts ou les variables changent pour avoir les données en temps réel dans toutes les scènes et scripts.

### 3.3 Carte

#### 3.3.1 Graphismes et placement (Solène et Yusuf)

Premièrement, nous avons amélioré la carte de la première soutenance. Au niveau du plan, nous avons créé un schéma plus précis, qui représente les parties principales de la carte, sans les petits objets.

La plus grande différence est la création d'une nouvelle scène "Extérieur", qui se situe donc à l'extérieur de la ville, quand le joueur en sort. Dans cette zone, il n'y a aucun bâtiment. Nous pouvons simplement observer des ennemis, des lacs, des rochers et des feuilles.

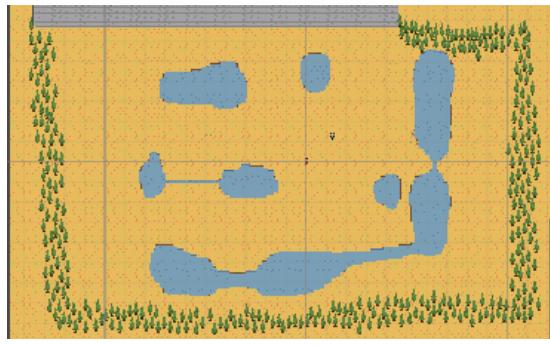


Figure 11: Image de l'extérieur

Ensuite, j'ai réalisé certains des objets dont nous avons besoin. Par exemple, les arbres de l'extérieur, les rochers, les feuilles, ... La création de design avec l'outil Krita devient de plus en plus facile vu le nombre de design dont nous avons besoin pour le jeu. Voici quelques exemples de ce que j'ai réalisé entre la première et la deuxième soutenance.



Figure 12:  
Sapin



Figure 13:  
Rocher 1

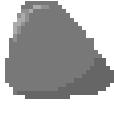


Figure 14:  
Rocher 2



Figure 15: Tas  
de feuilles

Pour ma partie, j'ai réalisé les bâtiments qui seront dans la carte comme les

magasins dans lequel le personnage pourra rentrer. Il y aura aussi des appartements, des maisons comme-ci dessous.

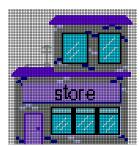


Figure 16: magasin

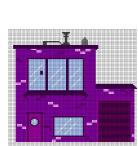


Figure 17: maison



Figure 18: poubelle



Figure 19: maison 2



Figure 20: maison

### 3.3.2 Placement carte extérieur (Yann)

J'ai placé des lacs et des arbres pour rendre la zone extérieure limitée de façon naturel. Ainsi, que la rendre plus intéressante à découvrir. Je compte encore ajouter les détails aux différentes zones qui n'ont pas encore été implémentés. Voir l'image 11.

## 3.4 Interface du joueur

### 3.4.1 Interface du joueur (Solène)

Pour commencer l'interface du joueur, j'ai affiché les statistiques du personnages comme son nom, son niveau, sa barre de vie et sa barre d'expérience dans le coin supérieur gauche.

A droite, nous pouvons aussi trouver une icône représentant les paramètres. Ces informations servent au joueur à se repérer et à connaître ses informations dans le jeu directement sans avoir à ouvrir une nouvelle fenêtre.

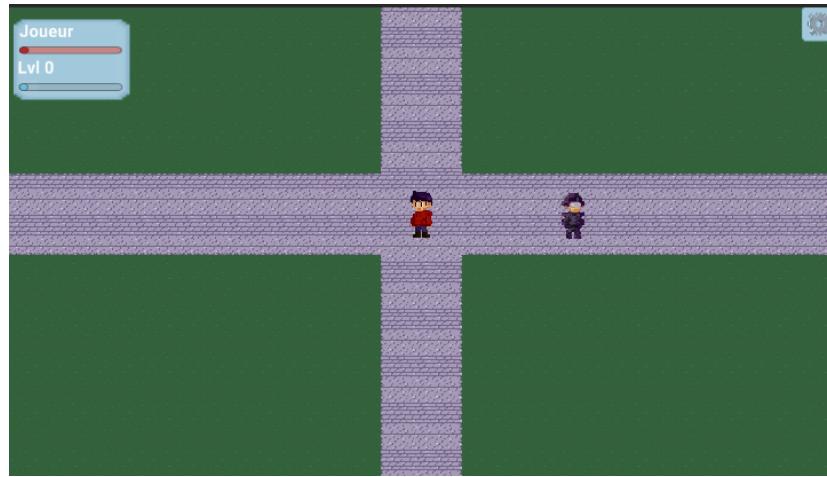


Figure 21: Interface du joueur.

Ces informations sont créées sur un Canvas avec l'option ScreenSpace - Camera. Cette option permet d'assigner la caméra principale au Canvas, qui permet alors au Canvas de suivre la caméra en permanence.

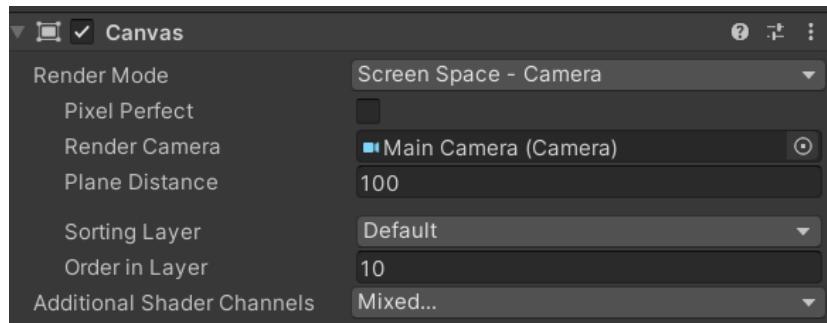


Figure 22: Option du Canvas caméra.

### 3.4.2 Spécification de l'affichage (Yann)

Les valeurs sont afficher en temps réel en haut à droite. Pour cela j'ai récupéré les données de sauvegarde qui sont les données actuelles et les ai mis en valeur de ce qui est affiché. Voici le script :

```
1  public void Update()
2  {
3      nameText.text = playerPrefab.unitName;
4      levelText.text = "Lvl " + (PlayerPrefs.GetInt("Level")).
6          ToString();
5      hpSlider.value = playerPrefab.currentHP;
6      xpSlider.value = PlayerPrefs.GetInt("Exp");
7  }
```

## 3.5 Inventaire (Yusuf)

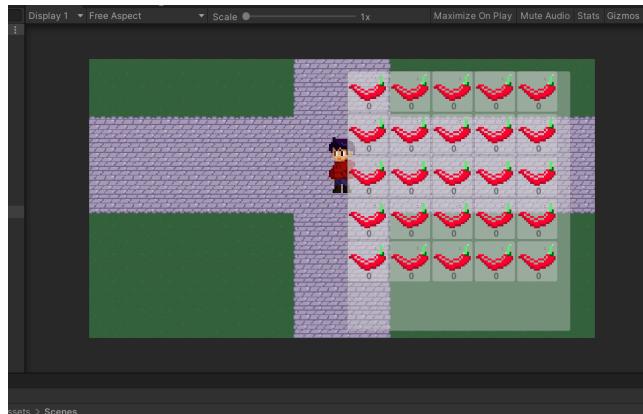


Figure 23: Inventaire

L'inventaire continuera de changer de design avec le temps, ce n'est donc pas la version finale. Le personnage pourra ramasser les objets qu'il trouve sur la carte. Quand on appuie sur la touche E, le joueur pourra voir son inventaire, et pour fermer l'inventaire il devra appuyer à nouveau sur E.

Et voici les codes associés à l'inventaire :

```
1  public class Inventaire : MonoBehaviour
2 {
3     bool activation = false; // inventaire fermer.
4     GameObject P;
5     public int [] slot;
6     void Start()
7     {
8         GetComponent<Canvas> ().enabled = false; // L'affichage
9             de l'inventaire est inactif au depart.
10            P = transform.GetChild (0).gameObject;
11            slot = new int [P.transform.childCount];
12        }
13
14    void Update()
15    {
16        if (Input.GetKeyDown(KeyCode.E)) // Si on appuie sur la
17            touche E.
18        {
19            activation = !activation; // Lorsqu'on appuie une
20                premiere fois sur E, l'inventaire est ouvert.
21                Puis lorsqu'on appuie une deuxieme sur la touche
22                    E, l'inventaire est ferme.
23            GetComponent<Canvas> ().enabled = activation;
24        }
25    }
26    public void UpdateTXT(int nrslot, string txt)
27    {
28        P.transform.GetChild(nrslot).GetChild (1).GetComponent<
29            Text> ().text = txt;
30    }
31 }
```

### 3.6 Système de combat (Solène)

Beaucoup de changements ont été amenés sur le système de combat sur tous les points. Premièrement, le système même a été modifié. Pour rappel, le système de combat est un combat au tour par tour, chacun pouvant attaquer une fois à son tour. Il est possible soit d'attaquer, soit de prendre la fuite. Les informations du joueur et de l'ennemi sont tous les deux affiché au dessus de leur image.

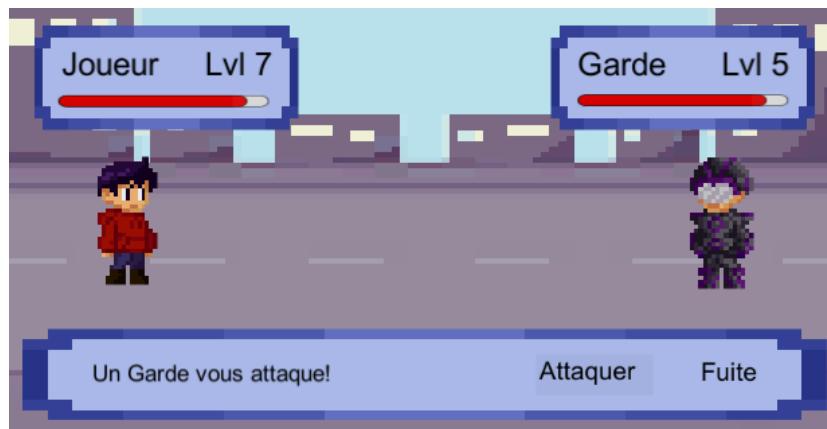


Figure 24: Scène de combat.

Le bouton Fuite a d'ailleurs été implémenté. Il repose uniquement sur le hasard, le joueur a tout simplement une chance sur 2 de réussir à fuir, et donc de quitter le combat.

```

1     IEnumerator FuiteSuccess()
2     {
3         dialogueText.text = "Tu prends la fuite..";
4         //quitter la scene
5         yield return new WaitForSeconds(2f);
6         SceneManager.LoadScene("Game");
7     }
8
9     IEnumerator FuiteFail()
10    {
11        dialogueText.text = "Tu n'as pas reussit a prendre la
12            fuite.";
13        yield return new WaitForSeconds(2f);
14        state = BattleState.ENEMYTURN;
15        StartCoroutine(EnemyTurn());
16    }
17
18    public void OnFuiteButton()
19    {
20        //lance un aleatoire pour sortir
21        bool quitter = false;
22        int value = UnityEngine.Random.Range(1, 3);
23        if (value == 1)
24            quitter = true;
25        if (quitter)
26        {
27            StartCoroutine(FuiteSuccess());
28        }
29        else
30        {
31            StartCoroutine(FuiteFail());
32        }
33    }

```

Maintenant, il n'y a plus une seule attaque pour le joueur, mais il peut choisir entre 4 attaques différentes. La dernière attaque, elle, est différente selon le type d'arme que possède le joueur. Ici, par exemple, le joueur possède une hache, d'où l'attaque "Bucheron" sur l'image ci-dessous.

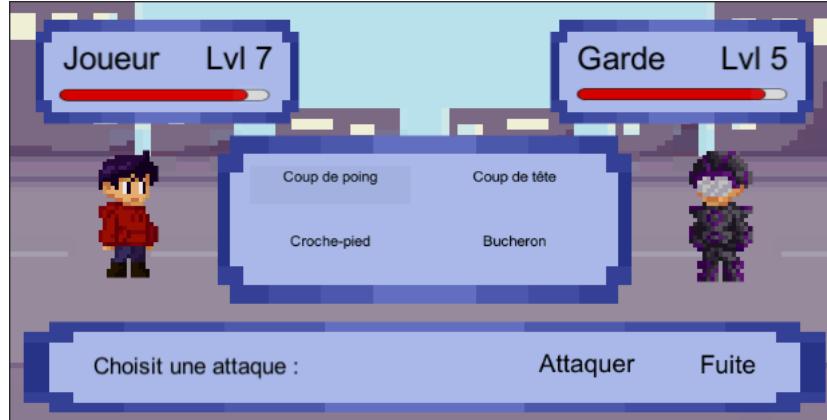


Figure 25: Choix entre les 4 attaques du combat.

Les différents boutons sont accessibles avec les flèches du clavier, comme ceux du menu principal. Le problème qui s'est posé est celui de changer la sélection des boutons d'attaque et fuite aux 4 boutons quand la page des 4 attaques s'ouvre. Pour cela, ces lignes de codes ont été ajoutés au fichier "BattleSystem" afin de changer le paramètre FirstSelected dans l'EventSystem associé à la scène.

```
1 GameObject myEventSystem = GameObject.Find("EventSystem");
2 myEventSystem.GetComponent<UnityEngine.EventSystems.EventSystem>()
    >().SetSelectedGameObject(GameObject.Find("AttackButton"));
```

Les boutons des 4 attaques, d'attaque et de fuite sont également reliés grâce à l'option Explicit dans les options des boutons. Grâce à cela, selon le premier bouton sélectionné, il est possible de se déplacer entre certains boutons, mais d'autres sont inaccessibles.



Figure 26: Navigation boutons explicit 1.



Figure 27: Navigation boutons explicit 2.

En plus de la première soutenance, l'ennemi choisit également de lancer une des attaques qu'il possède. Il n'y a pas d'IA à ce niveau là, l'attaque est choisie au hasard parmi les 4 attaques. Sa dernière attaque change elle aussi selon l'arme qu'il possède. Les armes changent selon le type d'ennemi.

Déclaration des variables dans le fichier Unit :

```
1 public string[] attacks = new string[] { "Coup de poing", "Coup
2     de tete", "Croche-pied" };
3 public int[] damageAttacks = new int[] { 3, 5, 10 };
4 public string[] armes = new string[] { "Epee", "Hache", "Baton",
5     "Sabre laser", "Mains" };
6 public string[] attaqueArmes = new string[] { "Slay the ennemis"
7     , "Bucheron", "Tape tape", "Shlash", "Critical hit" };
8 public int[] damageAttaquesArmes = new int[] { 5, 5, 2, 5, 4 };
```

Choix des boutons et de l'attaque dans le fichier BattleSystem :

```
1 public void OnFirstAttack()
2 {
3     dialogueText.text = "Joueur attaque " + playerUnit.
4         attacks[0];
5     //touche ennemi
6     isDead = enemyUnit.TakeDamage(playerUnit.damageAttacks
7         [0]);
8     attackWindow.SetActive(false);
9     StartCoroutine(PlayerAttack());
10 }
11
12 public void OnSecondAttack()
13 {
14     dialogueText.text = "Joueur attaque " + playerUnit.
15         attacks[1];
16     //touche ennemi
17     isDead = enemyUnit.TakeDamage(playerUnit.damageAttacks
18         [1]);
19     attackWindow.SetActive(false);
20     StartCoroutine(PlayerAttack());
21 }
22
23 public void OnThirdAttack()
24 {
25     dialogueText.text = "Joueur attaque " + playerUnit.
26         attacks[2];
27     //touche ennemi
28     isDead = enemyUnit.TakeDamage(playerUnit.damageAttacks
29         [2]);
30     attackWindow.SetActive(false);
31     StartCoroutine(PlayerAttack());
32 }
33
34 public void OnFourthAttack()
35 {
36     dialogueText.text = "Joueur attaque " + playerUnit.
37         attaqueArmes[playerUnit.indexArme];
38     //touche ennemi
39     isDead = enemyUnit.TakeDamage(playerUnit.
40         damageAttaquesArmes[playerUnit.indexArme]);
41     attackWindow.SetActive(false);
42     StartCoroutine(PlayerAttack());
43 }
```

```

37     void ChooseAttack()
38     {
39         attackWindow.SetActive(true);
40         GameObject myEventSystem = GameObject.Find("EventSystem")
41             );
42         myEventSystem.GetComponent<UnityEngine.Events.
43             EventSystem>().SetSelectedGameObject(GameObject.Find
44             ("FirstAttackButton"));
45         dialogueText.text = "Choisit une attaque :";
46
47         attaque1.text = unit.attacks[0];
48         attaque2.text = unit.attacks[1];
49         attaque3.text = unit.attacks[2];
50         attaque4.text = playerUnit.attaqueArmes[playerUnit.
51             indexArme];
52     }

```

Pour finir, le système de combat se lance maintenant quand le joueur rentre en contact avec l'ennemi ou inversement. Pour celà, les 2 personnages possèdent un box collider. Celui-ci dépasse un peu de leur sprite pour qu'ils puissent rentrer en contact. Dès que les 2 se touchent, une animation de fondu en noir se lance pour indiquer un changement de scène, celui-ci vers la scène de combat.

Le joueur et l'ennemi sont alors placé sur la scène. Un problème qu'il reste à régler est que l'ennemi qui se place sur la scène n'est pas forcément le même que celui que l'on vient de toucher.

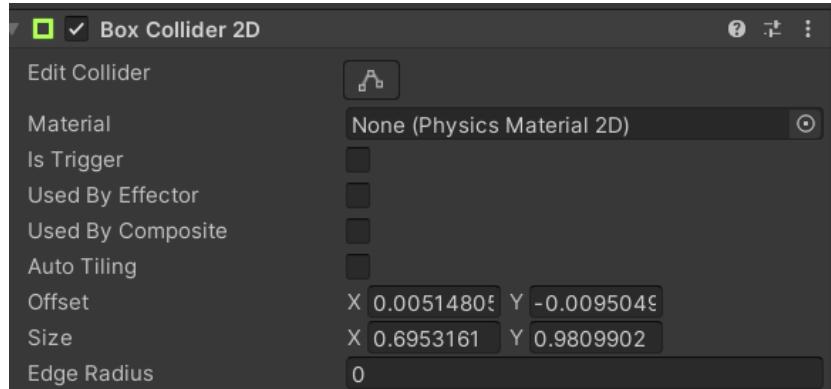


Figure 28: Image box collider.

Comme précision, pour le fondu en noir, cela se fait simplement grâce à une animation. Une image noir devient de plus en plus transparente avec le temps. Celle-ci est alors actionnée en transparent à noir quand le joueur quitte une scène et de noir à transparent quand le joueur entre une scène.

Pour changer de scène, rien de plus simple. Une seule ligne de commande suffit grâce au module UnityEngine.SceneManagement.

```

1  using UnityEngine.SceneManagement;
2  IEnumerator ToGame()
3  {
4      animat.SetBool("Fade", true);
5      yield return new WaitUntil(() => black.color.a == 1);
6      SceneManager.LoadScene("Game");
7  }

```

### 3.7 Ennemis (Solène)

Premièrement, j'ai pu continuer les animations des personnages. Seuls celles du joueur, du garde et du premier infecté ont été faite mais le système est le même pour tous. J'ai d'abord créer 12 images différentes, 3 pour chaque sens.

Avec ces images, j'ai alors pu créer 4 animations, représentants les 4 sens. Il suffit alors de glisser les images les une à coté des autres. Pour ne pas que l'animation se fasse trop rapidement, il a fallut changer le nombre d'image par secondes.



Figure 29: Animation.

Pour lier les animations ensemble, on utilise alors un Animator. Dans celui-ci, il suffit de mettre les paramètres en 2D afin de pouvoir placer les 4 animations qui s'enclenche selon certaines variables. Ces variables sont horizontal, vertical et speed. Les 2 premières varient entre 1 et -1. Pour l'horizontal (resp. vertical), la valeur 1 est associé au haut (resp. droite) et la valeur -1 est associé au bas (resp. gauche).

Le paramètre speed sert de condition pour actionner l'animation. Le paramètre est positif pour la valeur 1 et négatif pour la valeur -1. Il est de la même valeur que la valeur absolue de la vitesse. C'est grâce à elle que l'animation s'arrête lorsque le joueur ne bouge plus.

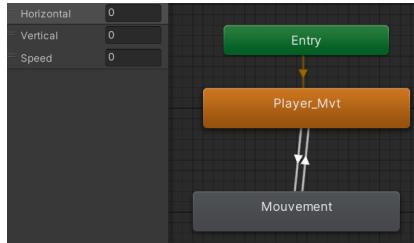


Figure 30: Liaisons des animations.

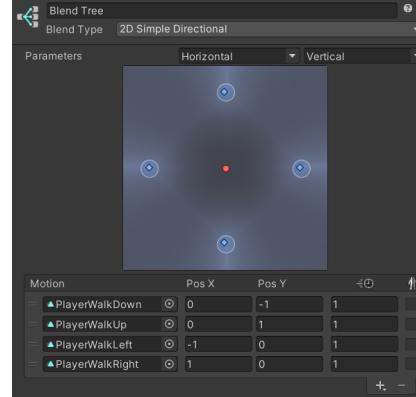


Figure 31: Directions.

Du côté de l'intelligence artificielle, celui-ci est implémenté avec la patrouille des ennemis. Chaque ennemi fait des patrouilles, pour le moment seulement sur l'axe horizontal, d'un point à un autre. Ces points sont créés avec des gameobjects sans images mais qui contiennent un box collider. Ce composant sert à savoir quand l'ennemi et le point entrent en collision, l'ennemi possédant aussi un box collider.

Un problème s'est cependant posé. Le joueur lui aussi possède un box collider. Pour régler cela, j'ai alors placé les points de patrouilles sur des objets existants dans la scène. Comme cela, il n'est pas étrange si le joueur se retrouve bloquer par le point de patrouille car il est placer sur un objet sur lequel nous ne pouvons pas passer de toute façon. Les points de patrouilles sont représentés en vert clair sur l'image ci-dessous.

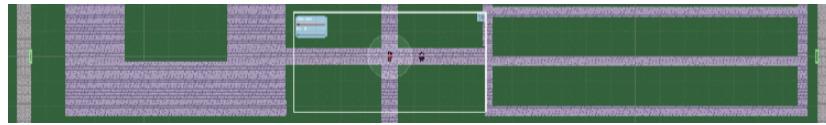


Figure 32: Points de patrouille.

Pour le script, tout ce passe dans le fichier AIPatrol. La fonction Start sert à initialiser. Ici, on définit l'animation et on dit que l'ennemi patrouille dès le début.

```

1 void Start()
2 {
3     mustPatrol = true;
4     anim = GetComponent<Animator>();
5 }
```

La fonction Update se lance continuellement, c'est donc ici que l'on va se servir de la variable mustPatrol de la fonction Start. Tant qu'elle est vrai, l'ennemi continue de se déplacer automatiquement à gauche ou à droite selon sa valeur de vitesse.

```

1      // Update is called once per frame
2      void Update()
3      {
4          if (mustPatrol)
5          {
6              if (moveSpeed > 0)
7                  movement.x = 1;
8              else
9                  movement.x = -1;
10             movement.y = 0;
11             anim.SetFloat("horizontal", movement.x);
12             anim.SetFloat("vertical", movement.y);
13             anim.SetFloat("speed", movement.sqrMagnitude);
14             Patrol();
15         }
16     }
```

C'est ici que les variables qui permettent à l'ennemi de se mettre à jour et d'avancer correctement. La fonction FixedUpdate sert à régler les problèmes qui peuvent apparaître avec le temps.

```

1      private void FixedUpdate()
2      {
3          if (mustPatrol)
4          {
5              mustTurn = Physics2D.OverlapCircle(groundCheckPos.
6                                              position, 0.1f, groundLayer);
7          }
8          rb.MovePosition(rb.position + movement * moveSpeed *
9                          Time.fixedDeltaTime);
}
```

La fonction Patrol s'active dans Update. Celle-ci est divisée en 2 parties. Dans la première, nous disons que s'il faut tourner ou si le box collider de l'ennemi touche un point de patrouille, la valeur de speed devient négative et l'image est inversée. Dans la deuxième, nous disons que si le box collider de l'ennemi touche celui du joueur, on lance la transition vers la scène de combat.

```

1     void Patrol()
2     {
3         if (mustTurn || bodyCollider.IsTouching(leftCollider) ||
4             bodyCollider.IsTouching(rightCollider))
5         {
6             Flip();
7         }
8         if (bodyCollider.IsTouching(playerCollider))
9         {
10            StartCoroutine(ToCombat());
11        }
12    }

```

La valeur de la vitesse devient bien négative si le coté et inversé, le faisant changer de direction. On arrête puis remet la variable mustPatrol afin de le personnage ait le temps de tourner. On modifie en même temps le sens de l'image.

```

1     void Flip()
2     {
3         mustPatrol = false;
4         transform.localScale = new Vector2(transform.localScale.
5             x * -1, transform.localScale.y);
6         moveSpeed *= -1;
7         mustPatrol = true;
8     }

```

Dans le futur, pour améliorer l'IA, il est prévu que l'ennemi puisse détecter quand le joueur se situe assez proche de lui et décide de se mettre à sa poursuite.

### 3.8 Niveau du joueur / Système d'amélioration (Yann)

J'ai décidé de réunir ces 2 parties car elles sont très liées. Voici le script pour monter en niveau le joueur à la fin du combat :

```

1 void EndBattle()
2     {
3         //choix message victoire ou defaite et si victoire,
4             faire monter de niveau le joueur et ameliore les
5                 stats
6         if (state == BattleState.WON)
7         {
8             int level = PlayerPrefs.GetInt("Level");
9             int exp = PlayerPrefs.GetInt("Exp");
10            exp += 100;
11            int capLevel = 100 * Carre(2, level);
12            if (exp < capLevel)
13                PlayerPrefs.SetInt("Exp", exp);
14            else
15            {

```

```

14         exp -= capLevel;
15         PlayerPrefs.SetInt("Exp", exp);
16         level++;
17         PlayerPrefs.SetInt("Level", level);
18         //Je fait monter de niveau le joueur avec l'exp
19         et une verification via palier
20         PlayerPrefs.SetInt("Force", PlayerPrefs.GetInt("Force",
21             level)+1);
22         PlayerPrefs.SetInt("Vitesse", PlayerPrefs.GetInt("Vitesse",
23             level)+1);
24         PlayerPrefs.SetInt("Resistance", PlayerPrefs.Get
25             Int("Resistance", level)+1);
26         PlayerPrefs.SetInt("Agilite", PlayerPrefs.GetInt("Agilite",
27             level)+1);
28         PlayerPrefs.SetInt("Chance", PlayerPrefs.GetInt("Chance",
29             level)+1);
30         // J'augmente toutes les statistiques de combats
31         du joueur
32     }
33     dialogueText.text = "Victoire!";
34 }
35 else if (state == BattleState.LOST)
36     dialogueText.text = "Tu as perdu..";
37 StartCoroutine(ToGame());
38 }
```

Dans ce script, à chaque augmentation de niveau, toutes les statistiques du joueurs augmentent de 1. Pour l'instant, l'expérience gagnée est fixe. Cependant, cela évoluera pour la fin du jeu. Les statistiques sont initiées lors de la création du personnage (voir partie 3.1.1, Lancement du jeu). Les statistiques sont sauvegardées à l'aide des PlayerPrefs comme expliquer dans la partie sauvegarde.

### 3.9 Multijoueur (Solène)

Comme je ne connaissais rien au multijoueur ou à son fonctionnement, j'ai d'abord du faire des recherches et regarder des vidéos pour savoir comment cela pouvait fonctionner.

Le multijoueur se jouerait à 2 maximum et se lancerait avec le localhost. Rien n'est sûr pour le moment mais ceci me semble être la solution la plus réalisable et la plus pratique pour les joueurs également.

Pour le moment, j'ai réalisé une nouvelle scène pour la lancement du jeu en multijoueur. J'utilise alors un Network Manager dans cette scène qui permet de créer une partie en Host mais aussi de rejoindre une partie. Un personnage apparaît alors à chaque fois que quelqu'un se connecte. Pour le moment, ils ont les mêmes sprites, mais nous verrons plus tard s'il est possible de modifier les sprites pour les 2 joueurs.

Il a donc fallut également ajouter un Network Management HUD qui permet d'ajouter les boutons pour accéder aux différentes options de jeu et également ajouter un Network Identity au joueur afin que celui-ci se fasse reconnaître par le Network Management.

Le jeu n'est cependant pas encore exécutable en multijoueur.

### 3.10 Site web (Abel)

Pour créer le site internet du jeu, j'ai utilisé le site de création de site Wix.com. J'ai mis un certain temps pour m'adapter au site mais je suis parvenu à un bon résultat après quelques dizaines d'heures. Le site comprend plusieurs pages dont une page d'accueil qui introduit le projet, un blog qui au fil des post retrace l'histoire de la création du jeu, une page qui présente les membres du groupe et enfin, une page qui donne les contacts de notre groupe. Le site web de notre projet est disponible à l'URL suivante :

<https://roffiaenabel.wixsite.com/removedfromthepast>.

De plus, j'ai aussi créé une adresse e-mail spécialement pour le projet :

removedfromthepast@gmail.com

## 4 Conclusion

Pour conclure, notre projet avance à bon rythme, et ce, malgré les difficultés à concentrer nos efforts dessus à cause des cours. Depuis la première soutenance, nous avons très largement avancé sur les programmes. Le jeu est presque fonctionnel et il ne reste plus à implémenter l'histoire avec les PNJ (Personnages Non-Joueurs).

Nous avons pris un léger retard sur quelques tâches mais il ne s'agit en rien d'un retard irratractable. Par exemple dans la carte il suffit de modifier certains sprites ou de faire des tâches identiques sur plusieurs personnages.

Avancement de fin de soutenance :

- vert : dans les temps
- jaune : peu de retard
- bleu : en avance

Tâche	A la soutenance 1	A la soutenance 2	A la soutenance 3
Menu principal	Avancé	Terminé	---
Commandes	Avancé	Terminé	---
Options	Commencé	Avancé	Terminé
Graphismes	Commencé	Avancé	Terminé
Carte	Avancé	Terminé	---
Apparition	---	Commencé	Terminé
Interface joueur	---	Avancé	Terminé
Histoire principale	Terminé	---	---
Système de déplacement	Terminé	---	---
Interaction PNJ	---	Avancé	Terminé
Récompenses	Commencé	Terminé	---
Système de combat	Commencé	Terminé	---
Ennemis	Commencé	Avancé	Terminé
Niveau du joueur	Commencé	Terminé	---
Système d'amélioration	Commencé	Terminé	---
Multijoueur	---	Commencé	Terminé
Sauvegarde	---	Avance	Terminé
Quitter	Avancé	Terminé	---
Site Web	Commencé	Terminé	---

Figure 33: Avancée à la deuxième soutenance

## 5 Sources

Documentation EventSystem  
Résolution probleme fonction Find  
Documentation GetActiveScene  
Unity Forum: Résolution probleme création object  
RemovedFromThePast Website  
Exemple multijoueur  
Exemple Patrol  
Animation  
Récupérer une variable d'un autre script  
Changer de scène