Rapport

Projet TER - Daydream

Faculté des Sciences de Montpellier

Master 1 Imagine
Année 2021-2022

Élèves :
Pierre Hennecart (22107715)
Maxime Ginda (22114440)
Benjamin Villa (22110753)

Table des matières

1. Introduction 3
a. Motivations . . . 3
b. Sujet . . . 3
c. Synopsis . . . 3

2. Conception 4
a. Narration . . . 4
b. Gameplay et actions du joueur . . . 5
c. Direction artistique et sonore . . . 5
d. Articles de recherche . . . 7

3. Rapport d'activité 8
a. Organisation du travail . . . 8
b. Méthodes et outils . . . 8

4. Interface 11
a. Menu principal . . . 11
b. Menu pause . . . 11
c. Menu option . . . 12
d. Menu inventaire . . . 12
e. Algorithme affichage des cartes . . . 14
f. Algorithme choix des cartes à mettre dans son paquets . . . 15
g. Menu de fin de partie . . . 16
h. Affichage de la quête en cours . . . 16

5. Cinématiques 17
a. Sérialisation . . . 17
b. Actions . . . 18

6. Exploration 21
a. Déplacement du personnage . . . 21
b. TileMap & TilePalette . . . 22
c. NavMesh, NavAgent . . . 24
d. Interactions . . . 24

7. Combat 27
a. Lancement 27
b. Sérialisation des cartes 28
c. Attaques directes et Status 29
d. Courbe de Bézier 30
e. Feedback 32

8. Quêtes 35
a. Sérialisation 35
b. Gestion 35

9. Bilan du projet 37
a. Autocritique 37
b. Enseignements tirés 37
c. Perspectives 38

10. Conclusion 39

11. Annexes 40
a. Documentation 40
b. Bibliographie 40

1. Introduction

a. Motivations

Tous les trois joueurs depuis un grand nombre d'années, notre entrée en Master Informatique option IMAGINE s'est faite avec la profonde motivation de finir par travailler dans l'industrie du jeu vidéo, et nous avons tous les trois rapidement sympathisé. C'est ainsi la raison pour laquelle nous avons directement pris l'opportunité de développer un tel projet dans le cadre du TER, et avons commencé à réfléchir au sujet dès le premier semestre. Notre intérêt fut d'ailleurs aussi très orienté sur l'apprentissage de nouvelles compétences à travers, ici, l'usage d'un moteur de jeu tel que Unity, dont deux d'entre nous n'avaient encore aucune expérience. Puis par la même occasion, la possibilité de tenter une toute nouvelle forme d'organisation du travail en allant démarcher nous-même des artistes et dont l'intégration de leur travail devra se faire tout au long du cycle du développement du jeu.

b. Sujet

Nous avions donc dès le début de l'année prévu de réaliser un jeu dont nous pourrions nous servir dans notre portfolio, et voulions profiter de l'occasion du TER pour entamer la programmation d'un RPG, dans l'objectif final de publier le jeu. Nous nous sommes organisé très tôt dans l'objectif d'avoir la conception du jeu, et une équipe pouvant nous fournir des ressources graphiques et sonores. Nous avions pour idée de base de réaliser un Point&Click pour se concentrer sur la qualité de l'ambiance et le Juice. Cependant, après un premier entretien avec Mme Faraj, nous avons conclu que le TER nécessitait un plus grande complexité technique. Nous avons donc revu dès la première semaine nos objectifs, et avons poursuivi avec celui de développer un jeu vidéo de type aventure/jeu de rôle en 2D isométrique, avec un système de combat basé sur des cartes tout en restant orienté sur la narration et l'immersion du joueur.

c. Synopsis

Lorsque la nuit tombe, vous apparaissez dans un monde onirique créé par votre subconscient, qui ne semble toutefois pas si irréel. Une véritable société se présente sous vos yeux, et certains individus semblent être dotés d'autant de raison que vous, tandis que d'autres ont l'air de répondre uniquement à une puissante entité... Parcourez l'Oneiros, combattez les entités malveillantes à l'aide de cartes amassées lors de votre aventure, et percez les mystères qui composent ce monde.

2. Conception

a. Narration

Steve

C'est notre protagoniste qui dans le monde réel est infirmier à l'Hôpital Sainte Marie. Se retrouvant un soir à dormir sur place à cause d'une longue opération qui finit par le faire atterir dans Oneiros.

Jed Morar

Écrivain tourmenté qui s'est retrouvé par hasard à se réveiller une nuit dans l'Oneiros. C'est la première rencontre de Steve dans cet univers éthéré. Rencontre qui lui permettra de comprendre le lien indéfectible entre les rêves et la réalité.

Le Tyran

C'est une entité aux buts inconnus pour les habitants d'Oneiros. Il s'agit en fait du plus ancien rêveur, qui a été plongé dans un coma artificiel d'urgence suite à un traumatisme. Sa présence peut se faire sentir peu importe l'endroit où l'on se trouve car sa voix porte sur Oneiros tout entier. Il est le créateur de ce monde, qu'il utilise pour trouver la paix intérieure et pouvoir vivre de nouveau.

Prémices

Depuis longtemps déjà, le Tyran rêvait d'un monde idéal qu'il considérait comme un vrai paradis où lui et ses proches pourraient vivre dans l'harmonie des vices et de la luxure. Plongé dans ce monde imaginaire, Oneiros pris de plus en plus d'ampleur au point tel qu'il devint commun d'y voir s'immiscer des rêveurs étrangers à son expérience. Ils s'introduisaient dedans sans son autorisation, ce qui le rendait fou de rage et l'obligeant à les renvoyer chez eux pour préserver sa vie intime. Pour se faire, il disposait d'une méthode bien rôdée. Les kidnapper puis les lobotimiser afin de les transformer en Sans-paroles, soit des êtres esclaves et muets qui finiraient par errer pour l'éternité en Oneiros. Malheureusement pour les atteindre, il devait les empêcher de se réveiller dans le monde réel. Ce qu'il se passe concrètement est très direct : il tue les rêveurs dans leur sommeil et seul la mémoire qu'il avait d'eux demeure dans son rêve à lui. Il devient alors important de préserver à tout prix son anonymat dans l'intérêt de garder toute son intégrité, chose qui échappe évidemment à tout nouvel arrivant...

Intrigue

Au travers de ses différentes découvertes, Steve sera amené à jouer au même jeu que celui du Tyran en tentant tant bien que mal de déceler son identité dans la réalité afin de mettre fin à tout ce chaos en l'otant de la machine qui le gardait en vie. Chacune de ses étranges rencontres dans l'Oneiros lui permettra d'accumuler des indices qui pourront

potentiellement l'aider à déterminer la localisation concrète de ce dernier. Si tant est que les différents obstacles que le Tyran lui mettra sur son chemin ne l'amèneront pas lui aussi à finir emprisonné à jamais.

b. Gameplay et actions du joueur

Tout au long de l'exploration (temps réel) des différents environnements, le joueur se verra interrompu par des phases de combat (tour par tour).

Exploration

L'exploration pourra se dérouler soit dans le monde réel lorsque Steve est réveillé, soit dans Oneiros pendant ses songes. À tout instant il lui sera possible d'interragir avec les éléments composant le décor afin d'enquêter et récupérer parfois des objets clés pour l'intrigue. Cependant les cartes de combat étant uniquement effectives dans l'Oneiros, leur gestion se trouvera alors impossible à l'extérieur.

Combat

Pour se défendre face aux Sans-paroles, Steve n'aura à sa disposition que d'étranges cartes portant des images cryptiques. Chacune d'elles concentrant en fait en son sein un effet magique. Et lors d'un combat, le joueur et l'ennemi se verront chacun leur tour amené à en engager à condition de ne pas déjà subir un sort de statut. Seul la mort ou l'absence de carte pourra signer la défaîte du joueur, nécessitant alors de bien penser sa stratégie de jeu.

c. Direction artistique et sonore

Le monde des rêves n'a pas les mêmes normes que la réalité, il est l'invention totale du Tyran qui lui même souhaitait se départir de toute connexion au réel. Les styles visuels seront variés, pouvant passer d'une ville abandonnée comme à une jungle psychédélique ou une cité dans les nuages.

FIGURE 1 – Jungle psychédélique

L'inspiration graphique orientée sur la notion d'homonculus où chaque être en Oneiros se verra matérialisé sous l'apparence d'une créature caricaturant son essence propre.

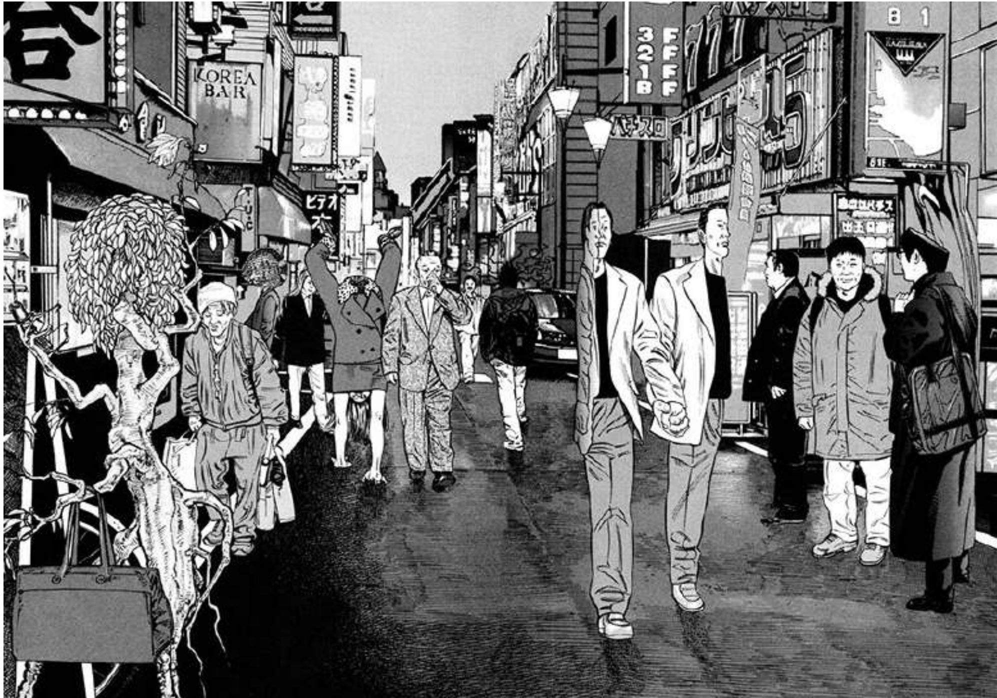

Figure 2 – Homoncules (Hideo Yamamoto)

Le jeu lui même se voyant modélisé dans une 2D isométrique en référence à un grand nombre de jeux d'aventure.

Figure 3 – Don't Starve (Jeu vidéo)

Tandis que la bande sonore, composée de thèmes anarchiques et répétitifs, viendra immerger encore davantage le joueur dans la folie du tyran.

d. Articles de recherche

Article 17 (Maxime)

L'article parle d'un "framework" permettant de modéliser des terrains complexes avec des surplombs, des caves ou des arches par exemple. Il permet également de généré des terrains rocheuses complexes avec des tas de roches sans aucune simulation physique exigeante en calcul. Le framework permet d'éditées et sculptées des terrains de manière interactive avec des outils de haut niveau.
Cet article ne nous a pas été utile pour notre projet car nous sommes partir sur un monde en 2D isométrique ne possédant par conséquent aucun relief réel.

Article 20 (Pierre)

L'article nous explique comment mettre en place de bonnes combinaisons audio-visuelles par rapport à des choix esthétiques. A savoir, mettre le bon son d'environnement au bon endroit dans notre jeu vidéo (comme par exemple un bruit d'oiseau dans une forêt). Le but d'appliquer les explications de cet article est d'avoir un plaisir audio-visuel plus important pour le joueur.
Nous n'avons pas utilisés cet article car nous n'avons pas implémenté de son d'environnement mais seulement des musiques qui ont été composées par des personnes tierces à l'université (mon petit frère en général, ou des musiques libres de droit).

Article 9 (Benjamin)

Cet article parle d'un algorithme optimisé d'animation de "sprite vidéo" afin d'animer des personnages réaliste. Cette technique sur des remplacements partiels répétés de la séquence vidéo permet de trouver un bons arrangement de frame pour l'animation du personnage. Cette technique est nottament utilisés pour animés des personnages animaux nottaments pour
Cependant l'utilisation de cet algorithme est uniquement résérvé pour des modèles 3D. Notre jeu utilisant de la 2D isomètrique, nous n'avons pas eu l'occasion d'utilisé les informations données dans cet article.

3. Rapport d'activité

a. Organisation du travail

Considérant la complexité qu'est le développement d'un jeu vidéo, surtout pour certains de nos profils encore débutants dans le domaine, nous avons à plusieurs reprises changé notre organisation pour au final se répartir comme suit :

— Maxime : Interface

— Pierre : Partie combat, Cinématiques, Quêtes, Mise en place de la carte isométrique

— Benjamin : Partie exploration

Par ailleurs, la gestion des différents artistes initialement prévue pour être hebdomadaire via des réunions s'est vue heurtée à des contraintes de disponibilitées. Nous avons donc réagit comme nous pouvions, nous avons notamment trouvé un moyen de générer des cartes nous-même, grâce à l'IA de Wombo Art, permettant de générer une image selon une liste de mots-clés et d'un style. Cependant, pour le reste nous avons dû chercher des tiles et éléments de décors par nous-mêmes, les ressources que nous avons obtenues de nos graphistes étants quelques sprites toutefois très bons.

b. Méthodes et outils

Trello

Le projet s'est vu partagé en plusieurs parties comme explicité précédemment, et c'est à travers l'usage d'un board Trello que nous avons pu les découper en de plus petites tâches.

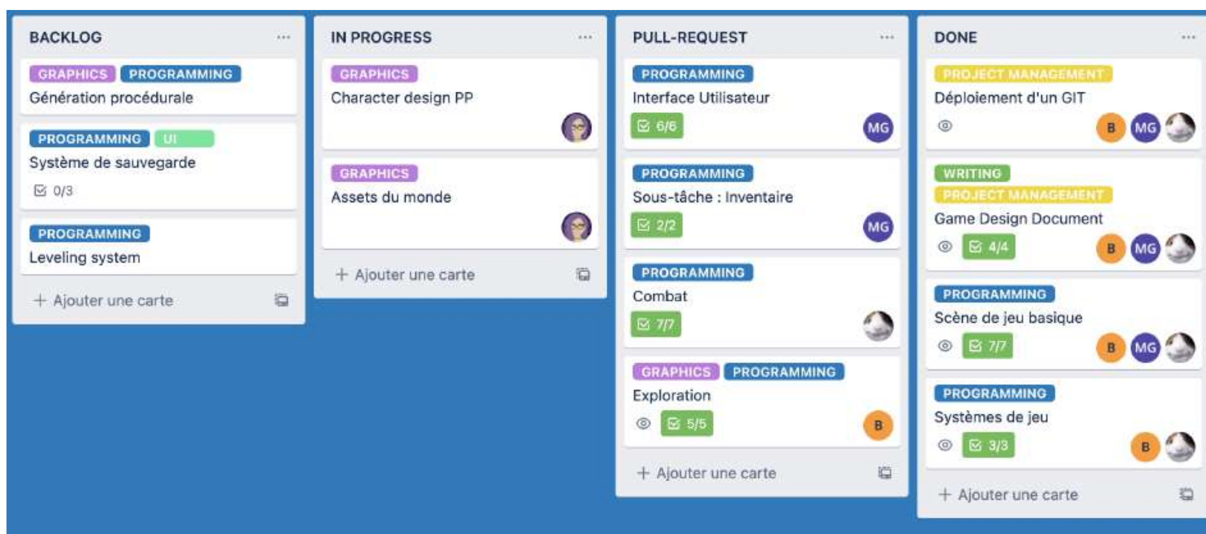

FIGURE 4 – Notre Trello

Réparti entre les tâches en attente (backlog), celles en cours (in progress), celles attendant un merge (pull request) et celles effectivement implémentées (done).

GitHub

Comme dans beaucoup d'autres projets, c'est avec un client Git, ici GitHub, que nous avons géré notre partie code. À partir d'une branche initiale (dev), chaque étape de développement se voyant séparée en des branches parallèles qui se retrouvaient merge par des pull requests sur dev dès la fin de leur implémentation.

main · Daydream / DayDreaming / Assets / · Go to file · Add file · ...

Pierrhum Merge pull request #33 from Pierrhum/FightMerge ... · 737b0b7 5 hours ago · History

..		
Animations	Merge branch 'Develop' into update-main-menu	yesterday
ColorfulButtons	First inventory version	last month
Externe	Up zone detecting	last month
Font	fix design hud	yesterday
GameContent	Merge pull request #33 from Pierrhum/FightMerge	5 hours ago
Import	Musique Menu	2 days ago
Libraries	NavMesh buggy	last month
Palettes	Interaction type teleport, in-house zone	8 hours ago
Prefabs	Status info pop up on hover	10 days ago
Resources	Boooooomerang	2 days ago
Scenes	Merge pull request #33 from Pierrhum/FightMerge	5 hours ago
Script	Merge pull request #33 from Pierrhum/FightMerge	5 hours ago
Sprite	Interaction type teleport, in-house zone	8 hours ago
TextMesh Pro	Merge pull request #31 from Pierrhum/Exploration	5 hours ago
TileMap	Player inputs & Collider + Map iso de test	3 months ago

FIGURE 5 – Dossier Assets

Unity

Afin de développer notre jeu, nous nous sommes servi du célèbre moteur de jeu Unity dans sa version 2021.2.11f1. Ce qui était pour nous la voie la plus pertinente afin de se concentrer directement sur le gameplay au vu de la multiplicité des éléments que nous souhaitions intégrer. D'ailleurs le fait que celui-ci soit multiplateforme fut un grand bénéfice au vu de la disparité des systèmes d'exploitation utilsés au sein de notre groupe. Tous les éléments composant notre projet sur ce logiciel se voyaient être intégrés directement à notre git.

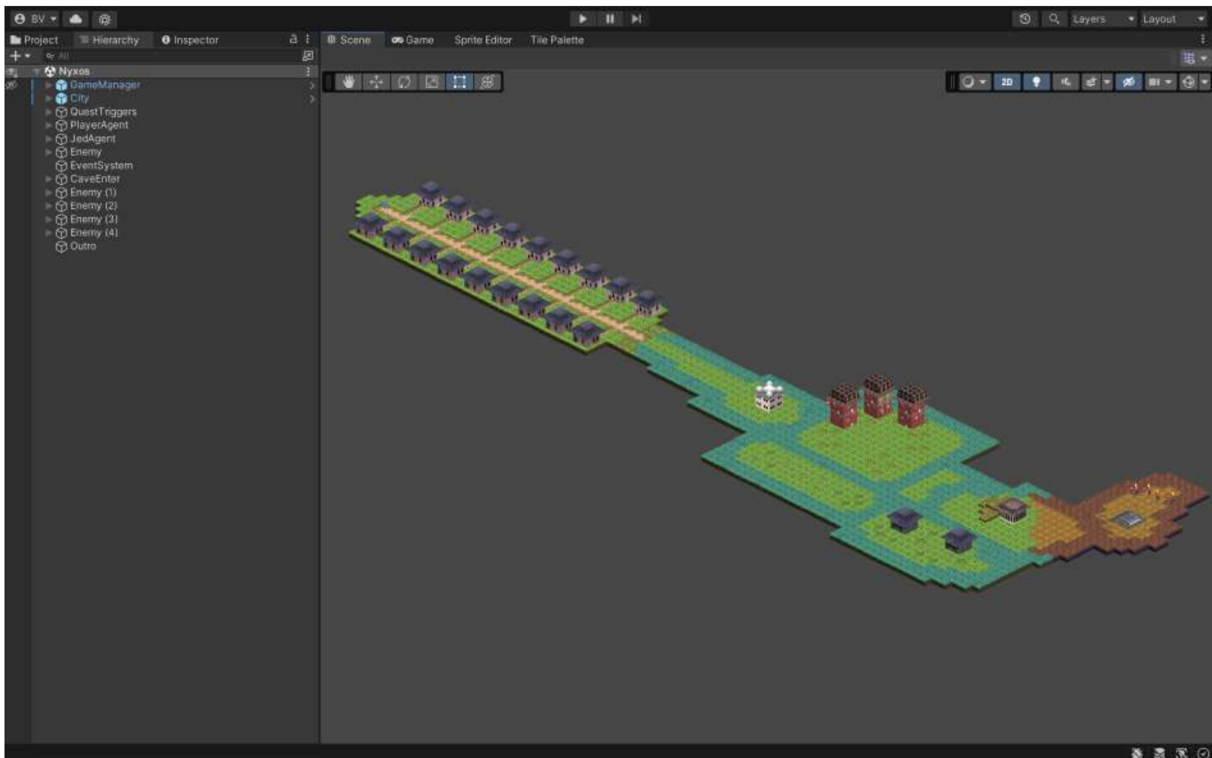

Figure 6 – Scène principale sur Unity

4. Interface

Pour les menus, nous avons utilisé les outils d'interface utilisateur de UNITY et notamment les "canvas", les "panels" ou bien encore les boutons par exemple.
Tous les menus sont adaptables en fonction de la résolution de l'écran choisie par l'utilisateur. Cela est rendu possible par le fait que le composant "Canvas Scaler" soit mis dans le mode "Scale with screen size" pour chaque canvas utilisé pour les menus.

a. Menu principal

Le menu principal offre trois possibilités au joueur. La possibilité de lancer le jeu à l'aide du bouton "play" (à gauche de l'écran), d'accéder au menu option à l'aide de l'icône "écrou" (en haut à droite de l'écran) et de quitter le jeu à l'aide du bouton "quit" (à droite de l'écran).
Celui-ci est animé (effet de brillance, l'arrière plan change de couleurs et le personnage se disparait par une transition de texture, puis devient un serpent) et possède une musique composée par une personne extérieure au projet qui est mise en boucle en attendant que le joueur lance ou quitte le jeu. Les animations ont été créées à l'aide de l' "animator" de UNITY et du Tool UIEffects.

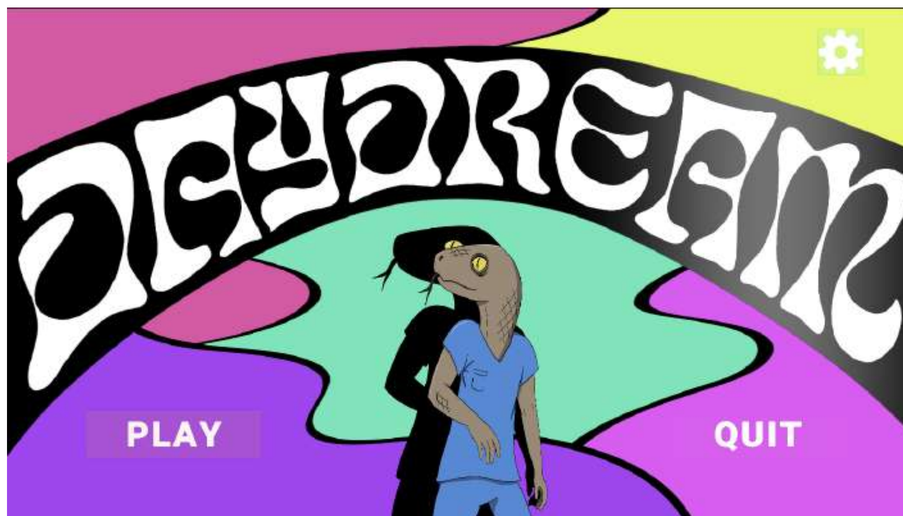

FIGURE 7 – Menu principal

b. Menu pause

Le menu pause possède trois boutons. Le premier permet de revenir au jeu, le second d'accéder aux options et le troisième de quitter le jeu. Il est accessible en appuyant sur la touche "Echap".
Lorsque le joueur accède au menu, la vitesse du temps dans le jeu passe à 0, il est donc impossible de déplacer le personnage et les personnages non joueurs resteront eux aussi statiques. De plus, toutes les autres informations de l'interface utilisateur sont masquées (H.U.D, menu inventaire, menu de paramètres si on reappuie sur echap).

Figure 8 – Menu pause

c. Menu option

Le menu option lui permet de choisir si le jeu est en plein écran, sa résolution et le volume du jeu. Le design n'est pas dans sa version finale et d'autres options pourront être ajoutées tel que le choix des touches.
Pour la résolution, Nous récupérons toutes les résolutions disponibles sur votre écran et l'ajoutons dans une liste déroulante.

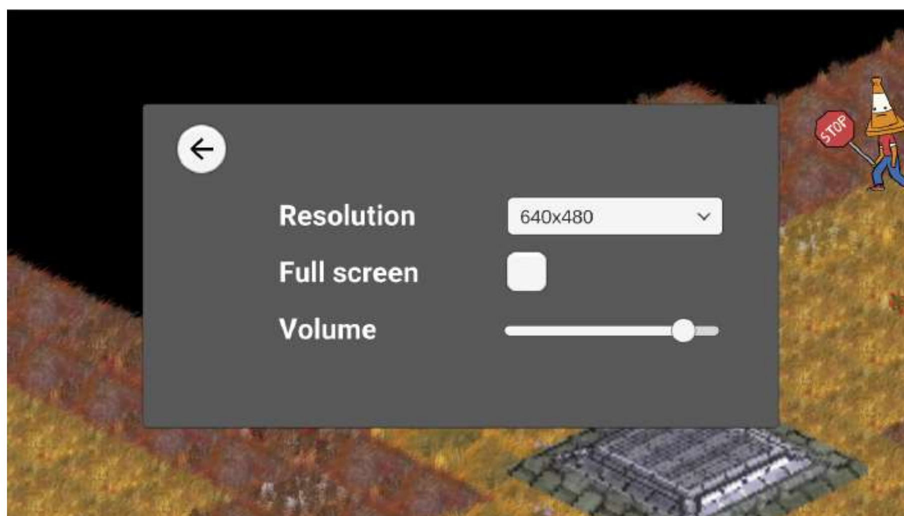

Figure 9 – Fenêtre de paramètres

d. Menu inventaire

Le menu d'inventaire est composé de deux fenêtres (représenté par des "panels" dans UNITY).

La première permet de visualiser les cartes possédées par le joueur et de les mettre ou non dans son paquet de cartes pour les combats à l'aide des boutons cochables.
L'inventaire de cartes peut posséder plusieurs pages (il possède cinq cartes par page). Le nombre de pages et le numéro de la page actuelle est affiché en bas à droite.

FIGURE 10 – Fenêtre d'inventaire

Un maximum de cinq cartes peut-être choisi par le joueur. S'il décide d'en choisir plus de cinq, une image apparaît pour le prévenir qu'il atteint la limite de cinq. Il est possible d'enlever l'image en cliquant dessus.

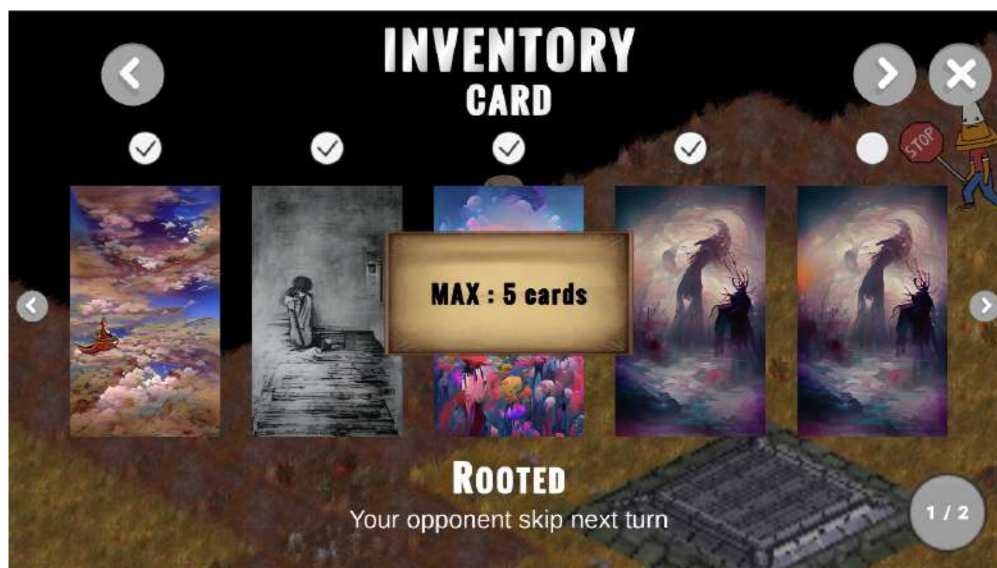

FIGURE 11 – Inventaire avec affichage de l'erreur nombres de cartes supérieur à 5

Lorsque l'on survole une carte, celle-ci est zoomée pour un meilleur aperçu du dessin. Lorsque l'on clique sur une carte alors apparaît en bas de l'écran son nom et la description de la dernière carte cliquée.

Figure 12 – Inventaire avec affichage du nom et de la description de la carte

La deuxième fenêtre de l'inventaire est l'inventaire contenant les objets possédés par le joueur. Cette fenêtre n'a pas encore été implémentée.

Figure 13 – Fenêtre objet de l'inventaire encore vide ici

e. Algorithme affichage des cartes

Voici l'algorithme permettant d'afficher les cartes dans l'inventaire en fonction de la page dans laquelle on est.

Fonction pour afficher les cartes dans l'inventaires

Variable :

numéro de la page : correspond à la page sur laquelle le joueur est dans l'inventaire. (5 cartes par pages)
bouton : tableau représentant les 5 boutons disponible sur l'inventaire.
checkbox : tableau représentant les 5 checkbox permettant de choisir les cartes dans l'inventaire
AlreadyChecked : liste représentant les checkbox qui était déjà coché auparavant

ALGO :

```
Pour i allant de 0 à 5 :
        checkbox ( i ) et bouton ( i ) sont activé
        Si i + numéro de la page * 5 < taille de la liste de carte du joueur :
                On affiche l'image de la carte rangé à la position i + numéro de la page * 5  sur le bouton( i )

                SI AlreadyChecked( i + numéro de la page * 5)  retourne vrai :
                        on coche la checkbox( i )
                Sinon on la décoche
        Sinon
                On désactive le bouton ( i )  et la checkbox ( i )
```

f. Algorithme choix des cartes à mettre dans son paquets

Voici l'algorithme permettant de choisir les cartes. Cette fonction est appelé à chaque fois que la valeur d'une "checkbox" est changé.

Fonction pour choisir les cartes dans l'inventaires

Variable :

numéro de la page : correspond à la page sur laquelle le joueur est dans l'inventaire. (5 cartes par pages)
id : id du checkbox (valeur passé en paramètre de la fonction
checkbox : tableau représentant les 5 checkbox permettant de choisir les cartes dans l'inventaire
AlreadyChecked : liste représentant les checkbox qui était déjà coché auparavant
nombre de cartes choisies : nombre de cartes déjà choisies.

ALGO :

```
Si checkbox ( id ) est coché :

        SI nombre de cartes choisies < 5 :
                SI la carte possédant l'id i + numéro de la page * 5 n'est pas déjà choisie :
                        on l'ajoute au cartes choisie
                        on ajoute son id dans AlreadyChecked
                        nombre de cartes choisies++
        Sinon :
                on décoche la checkbox ( id )
                on affiche le message "popup"
Sinon
        SI nombre de cartes choisies > 0 :
                SI la carte possédant l'id i + numéro de la page * 5 est déjà choisie :
                        on la retire des cartes choisie
                        on retire son id dans AlreadyChecked
                        nombre de cartes choisies--
```

g. Menu de fin de partie

Le menu de fin de partie est lancé quand le joueur n'a plus de vie. Une animation qui change la couleur du fond et une décomposition de notre personnage est faite avant d'afficher le menu.
Il est offre trois possibilités au joueur. La première est de relancer une partie, la seconde de retourner au menu principal et la troisième permet tout simplement de quitter le jeu.

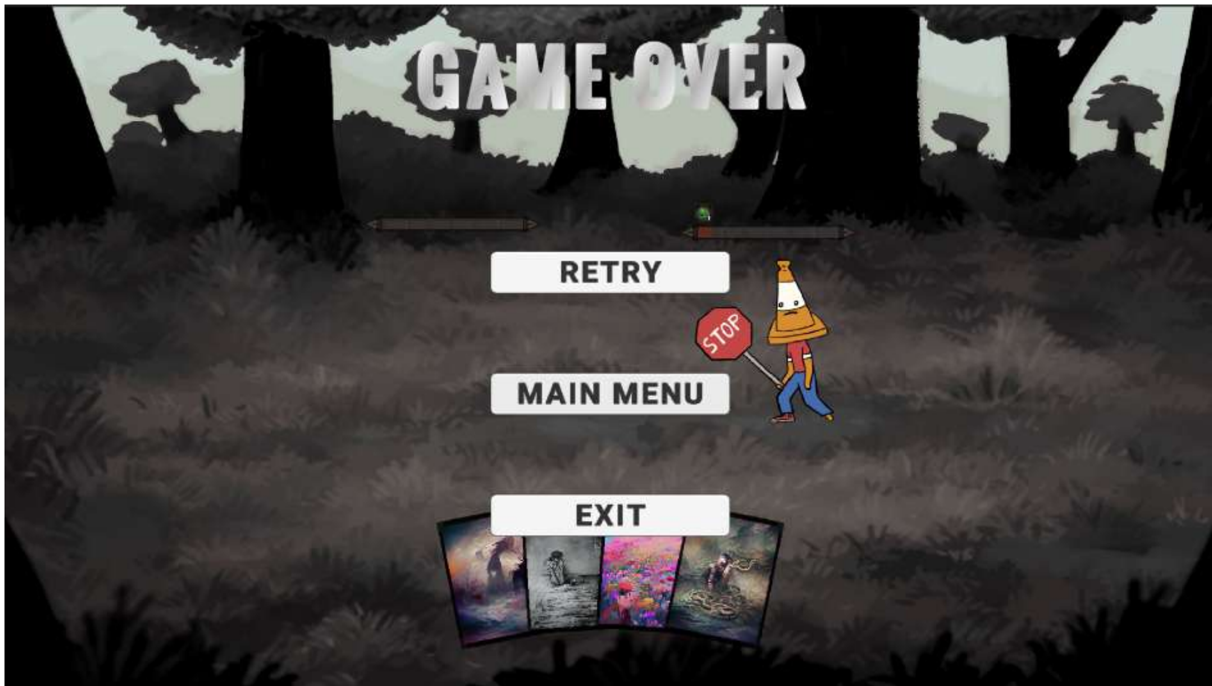

FIGURE 14 – Menu de fin de partie

h. Affichage de la quête en cours

La quête en cours que doit réaliser le joueur est affiché en haut à droite de l'écran. Son affichage disparait quand un autre menu est ouvert. Le texte est mis à jour à chaque fois qu'une quête est terminé et que par conséquent la fonction "next quest" est appelé.

FIGURE 15 – Affichage de la quête

5. Cinématiques

Daydream étant un jeu d'aventure, nous avions besoin de narrer l'histoire. Bien que le déroulement de cette histoire se passe par le biais de quêtes, nous avons également implémenté un système de gestion de cinématiques génériques permettant de mettre le jeu en pause et d'effectuer une séquence d'actions sérialisables dans un Game Object.

a. Sérialisation

En effet, le script *Cinematic.cs* effectuera une séquence d'actions, qui sont chacune paramétrables grâce à la classe C# *ActionCinematic*. Cette classe, qui n'hérite pas du *MonoBehaviour* d'Unity (permettant de déposer un script sur un GameObject), est toutefois définie par l'étiquette [*System.Serializable*] permettant d'instancier cet objet directement depuis l'éditeur.

Pour gérer la séquence d'actions de la cinématique, la fonction *Cinematic.Play()* se doit d'être une **Coroutine**. Une Coroutine permet de marquer des temps de pause dans son exécution. C'est grâce à cela que l'on va attendre la fin d'une action pour passer à la suivante, car l'exécution des actions passent également par des Coroutines, définissant pour chacune une séquence qui leur sont propres.

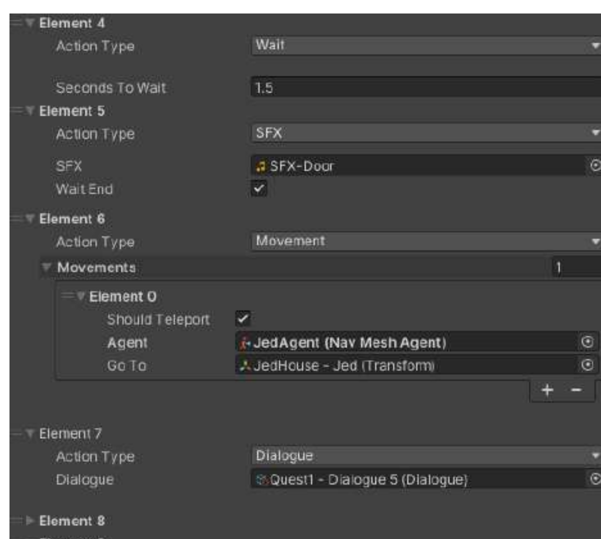

FIGURE 16 – Instanciation des objets *ActionCinematic* depuis l'éditeur

Pour permettre une paramétrisation ergonomique des actions, nous avons intégré le Tool MyBox, permettant d'avoir accès à des extensions pour customiser l'éditeur sans avoir à passer par la redéfinition de la méthode *Editor.OnInspectorGui()*. Nous avons seulement utilisé l'étiquette [*ConditionalField*], permettant de montrer/cacher des champs dans l'éditeur, basé sur la valeur d'un autre champ. Cette fonctionnalité permet proposer au développeur responsable de l'intégration, de sélectionner une valeur d'une enum (*ActionCinematic.ActionType*) qui affichera ensuite les paramètres concernés par le type d'action sélectionné.À l'heure actuelle, 8 actions différentes sont utilisables, et il est très facile d'en ajouter de nouvelles si besoin, tout le côté gestion étant déjà mis en place.

Vous trouverez ci-dessous les détails du fonctionnement de ces actions :

b. Actions

Dialogue

La prémière action a avoir été implémentée a été le dialogue. La première étape pour créer un dialogue est d'en créer l'asset qui contiendra les données nécessaires. Pour cela, on a définit un **ScriptableObject** qui pourra être créé depuis l'éditeur en effectuant un clique droit dans l'explorateur de fichier d'Unity, puis aller dans *Create > QuestSystem > Dialogue*. Tout ceci est possible grâce à l'étiquette :
[CreateAssetMenu(fileName = "New Dialogue", menuName = "QuestSystem/Dialogue")]
dans *Dialogue.cs*.

Le **ScriptableObject** est une liste de *Talk*, qui contiennent chacun un *texte* à afficher, un ou deux *sprites* à définir (deux dans le cas où les deux personnages parlent en même temps par exemple) et le *mode d'affichage* de ces sprites (*Left, Right, Both, None*). *None* permet notamment d'écarter la présence d'un personnage pour laisser le développeur s'adresser au joueur, comme quand il vient de débloquer un nouvelle aptitude et qu'il est nécessaire de lui expliquer comment l'utiliser.

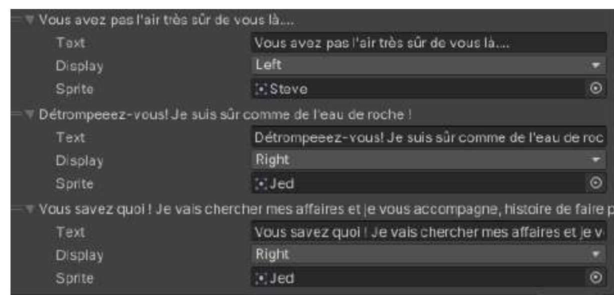

FIGURE 17 – ScriptableObject : Dialogue

Avec cet asset de donnée, l'affichage sera facilité : un simple parcours de *Talk* affichant dans une UI les données adéquates. Nous avons également implémenté un **affichage lettre par lettre** dans cet UI, qui lors d'un clique sur la boîte de dialogue (qui est un bouton) d'une pression du bouton entrée, se **complétera** si le texte n'était pas entièrement affiché, ou passera au **Talk suivant** si l'affichage était complet.

FIGURE 18 – Déroulement d'un dialogue

Dernier point concernant les dialogues : il est possible de les appeler en dehors d'une cinématique, lors d'un contact avec un collider **trigger** par exemple.

Mouvement

L'action Mouvement prend quant à elle une liste d'objets *Mouvement* (classe sérialisable), dont chaque entité contient un booléen **ShouldTeleport**, un **NavAgent** et un **Transform**.

La séquence de l'action est plutôt simple ensuite, il suffit de parcourir cette liste, de téléporter ceux qui ont besoin (utile dans le cas d'une apparition d'un personnage), ou de déplacer les autres de leur position point à une destination. Il faut cependant faire attention au cas du joueur. Si l'agent concerné par le mouvement est celui du joueur, il va falloir synchroniser la position du NavAgent (qui est le parent du joueur) avec la position du joueur. La raison pour laquelle le NavAgent est parent du joueur est qu'avec le cas de l'isométrique, nous avons besoin d'effectuer une rotation de l'agent de 90° en X.

Pour indiquer où l'agent en question doit se rendre, il faut simplement créer un point dans la scène et référencer ce point dans l'objet Movement.

Jouer une musique

Cette action permet de jouer une musique en référençant un script **Musique**, qui contient des extraits audio *begin/loop/end*.

Arrêter une musique

Nécessite d'avoir l'action Jouer une musique avant d'être appelée, et permet d'arrêter la musique qui était jouée, en attendant la fin de l'extrait *end* si le booléen **FadeOut** est vrai, avant de rejouer la musique principale.

Jouer un SFX

Dernière action sonore implémentée, elle permet de jouer un SFX renseigné, en attendant la fin de ce dernier si le booléen **WaitEnd** est vrai.

Attendre

Cette action permet simplement d'attendre selon un **flottant** indiqué (en secondes).

Donner des récompenses

Pour l'instant, cette action permet de donner uniquement un type de récompense au joueur : des **cartes**. Cependant, elle a été mise en place de façon à en offrir d'autres, comme de l'expérience par exemple.

Afficher un item de quête

Dans l'optique de vérification que tout type d'action est facile à implémenter avec le système mis au point, nous avons réalisé une action interactive plutôt sympathique : l'affichage d'un objet de quête. En renseignant uniquement un **QuestItem** (qui est un script contenant une image dotée d'un bouton), l'appel de cette action fera apparaitre celui-ci à l'écran, attendant que le joueur clique dessus pour le faire disparaitre et passer à l'action suivante.

FIGURE 19 – Affichage d'un item de quête

6. Exploration

La majeure partie du temps, le joueur sera amené à explorer les environnements d'Oneiros comme ceux du monde réel. C'est pourquoi nous avons mis une attention toute particulière dans les décors afin de retranscrire au mieux l'ambiance pesante du triste piège dans lequel Steve était tombé. Dans notre cas nous avons cela dit seulement la première zone d'Oneiros qui est une ville dévastée et à l'abandon.

a. Déplacement du personnage

Le sprite de Steve est animé sur les 8 différentes directions via le système d'Animator mis en lien avec les entrées clavier associées au déplacement soient haut, bas, gauche et droite. Nous utilisons le new input system d'Unity dans ce projet.

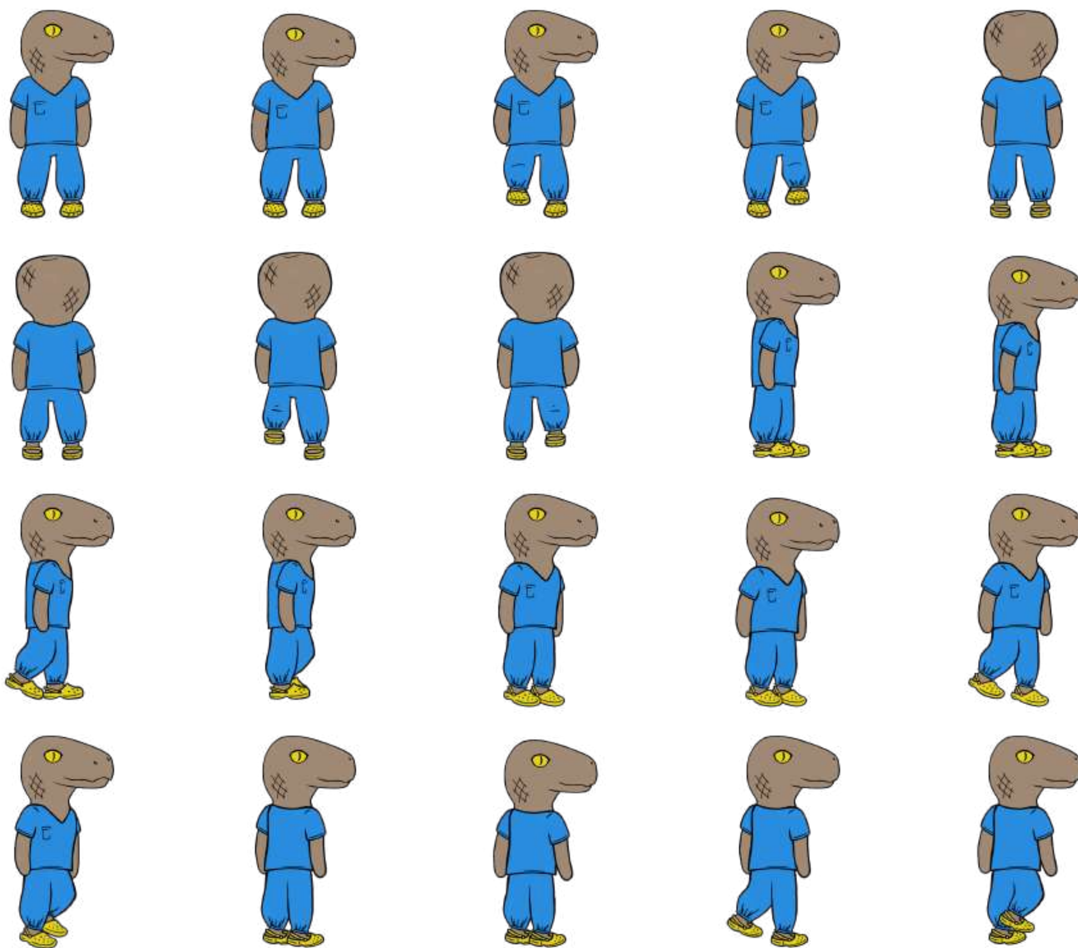

FIGURE 20 – Spritesheet des animations de Steve

b. TileMap & TilePalette

Au sein de Unity, la modélisation de la carte de jeu nécessite l'usage de Tilemaps qui permettent de dessiner directement des sprites dans la scène de jeu afin de concevoir tout notre monde. Ces sprites ont d'ailleurs été récupérés sur une des différentes banques de ressources libre de droit qu'il existe sur le web. La difficulté venait cependant du fait que dans notre cas ce n'était pas de simple sprites 2D dont il était question mais bien de sprites en 2D isométrique.

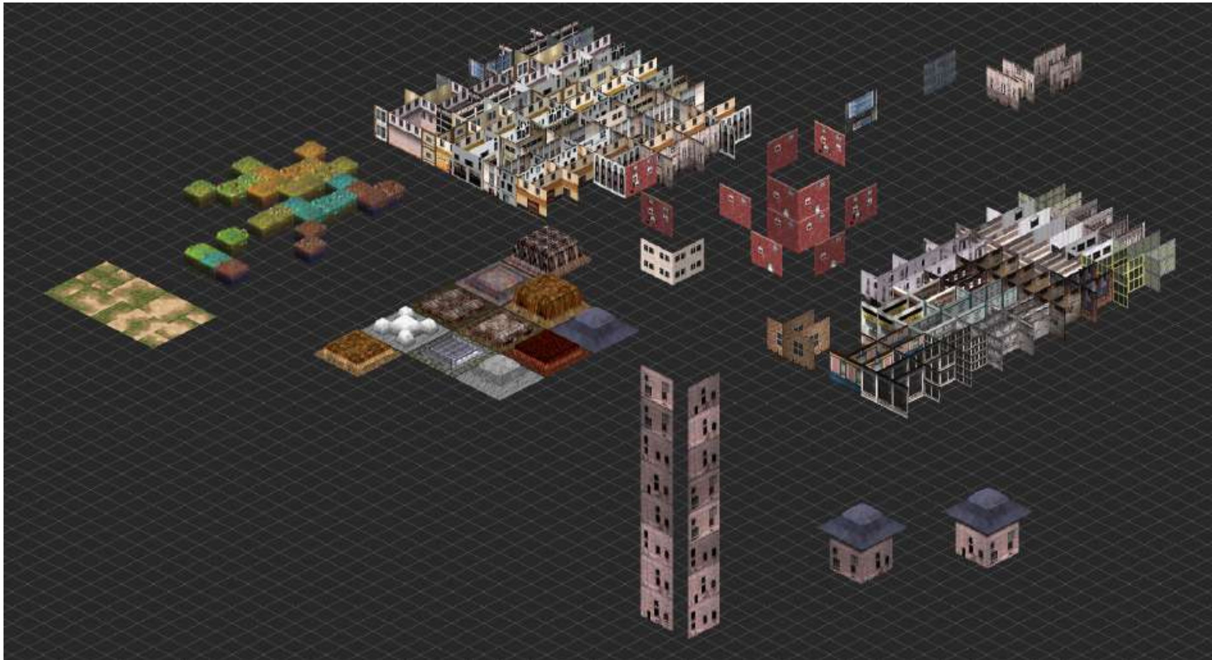

Figure 21 – La palette de sprites utilisée pour la ville

Au sein d'une même Tilemap, nous avons décidé de construire les éléments de la ville avec différents niveaux de layering nous permettant alors de mieux gérer les hauteurs et la profondeur.

- Ground : Le sol
- Layer 0 : Les élements au rez-de-chaussée comme les premiers murs
- Layer 1 : Premier étage des bâtiments
- Layer 2 : Toit des bâtiments

Figure 22 – Steve caché en partie derrière un immeuble grâce au layering

On notera toutefois que ce pavage régulier est permis grâce aux outils fournis par le Sprite Editor qui nous a offert la possibilité de découper précisément chacun de nos sprites au sein de leurs spritesheets respectives tout en offrant la possibilité d'éditer manuellement chaque sprite avec ses propres propriétés.

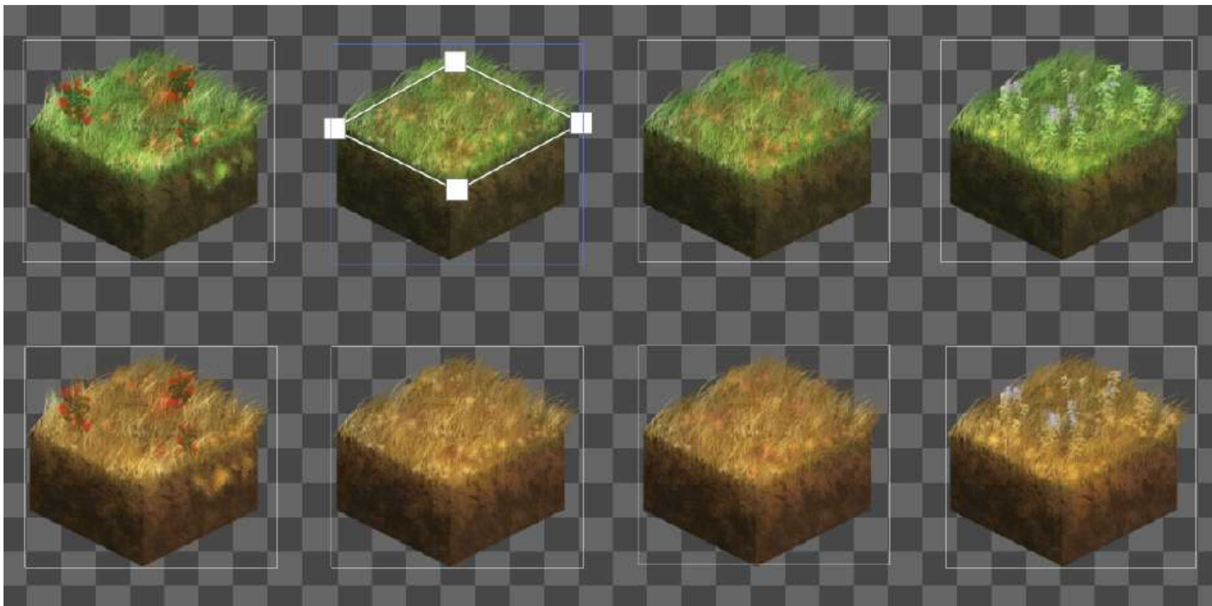

Figure 23 – Modification manuelle d'un sprite

c. NavMesh, NavAgent

Tout ceci structurant l'essentiel des méthodes utilisées pour représenter les diverses zones de la ville. Tandis que la navigation en son sein impliquera de générer des collisions aux bordures de la carte dont le CompositeCollider associé au TilemapCollider permettra de border finement les textures afin de délimiter la zone de jeu. L'IA liée au déplacement des différents personnage quant à elle est épaulée par un NavAgent nécessitant la présence d'un NavMesh qui est calculé justement grâce aux colliders précédemment cités.

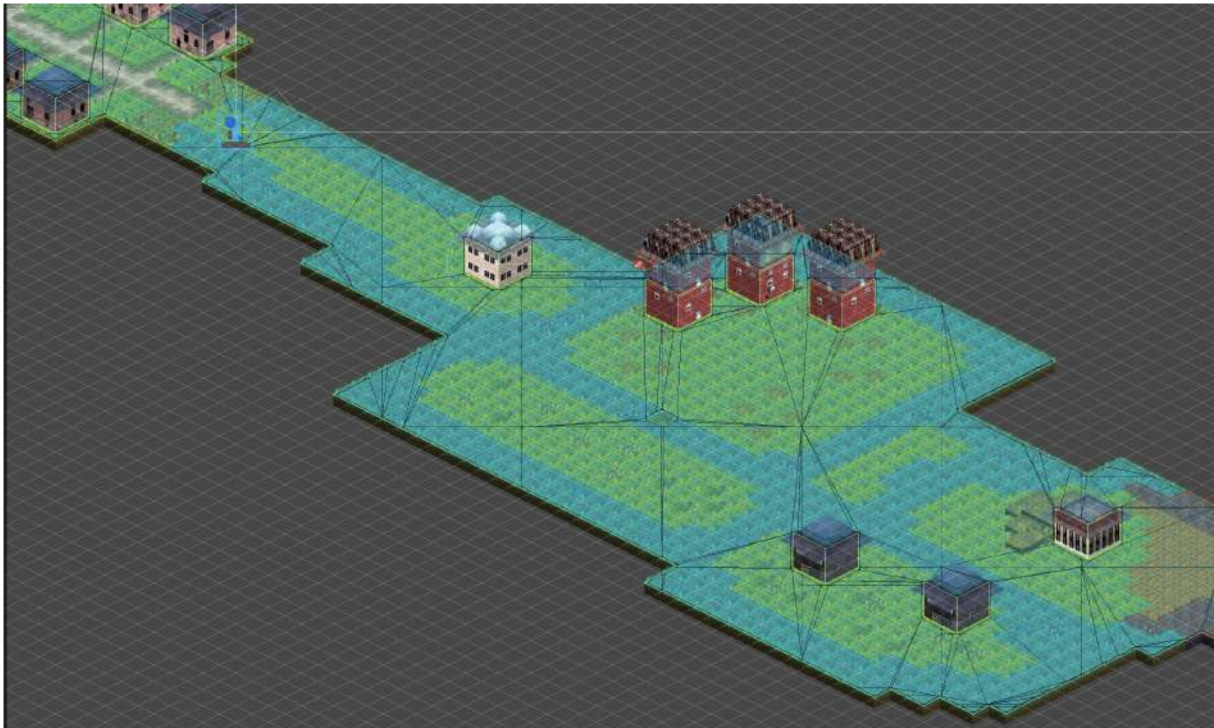

Figure 24 – NavMesh de la ville

d. Interactions

Chacune des zones quant à elles possèdent des points d'interactions déclenchés par des BoxColliders, permettant à sa guise de définir des comportements spécifique lors de leurs activations. En général, il sera question ici d'enclencher un dialogue, de récupérer un objet ou d'entrer dans une zone spéciale comme l'intérieur d'une maison abandonnée.

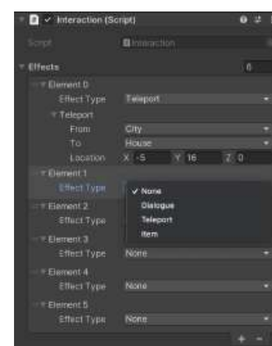

Figure 25 – Système de liste d'effets pour une interaction

Figure 26 – Une interaction est possible avec la maison

Figure 27 – Dialogue suite à une interaction

La carte étant par endroit peuplée d'individus errant dans les alentours avec parfois un niveau d'hostilité assez élevé, en particulier lorsque Steve sera confronté à des Sans-paroles.

Figure 28 – Steve poursuivi par des Sans-paroles

7. Combat

Le système de combat est une fonctionnalité très travaillée. Après les retours obtenus à **Into The Game**, les avis étaient unanimes, nous devions nous **concentrer** sur cet élément de gameplay, qui plaisait beaucoup aux testeurs. Au détriment de la *génération procédurale* initialement prévue, qui n'était plus si pertinente par rapport aux objectifs que nous avions, j'ai passé le temps qu'il me restait à perfectionner cet élément de Gameplay.

Je tiens à préciser que beaucoup d'effets visuels et d'animations ont été rendues possibles grâce à l'intégration d'un Tool très utile que nous avons trouvé sur GitHub : UIEffects.

a. Lancement

Commençons par le commencement : la lancement du Menu :

Lorsque le **collider** d'un ennemi entre en contact avec celui du joueur, le jeu passe en mode combat, ce qui *arrête* toutes les entités de bouger, et qui fait apparaitre le menu en fondu. Bien que ce ne soit pas exactement un fondu : à l'aide d'un *UIDissolve*, le menu va transiter vers son image de fond à l'aide d'une **texture de transition**.

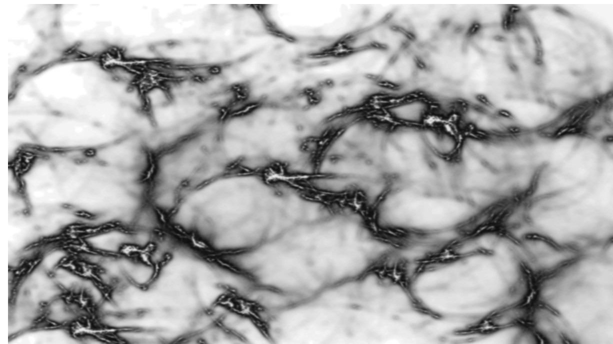

Figure 29 – Texture de transition

Comme indiqué précédemment, le Tool ne gère pas tout, ou du moins nous n'avons pas trouvé de moyen direct d'*animer*. Nous avons donc mis en place notre propre moyen d'animation, grâce à une **interpolation linéaire du facteur de transition** du *UIDissolve* (et des autres *UIEffects* par ailleurs) à nouveau grâce aux **Coroutines**.

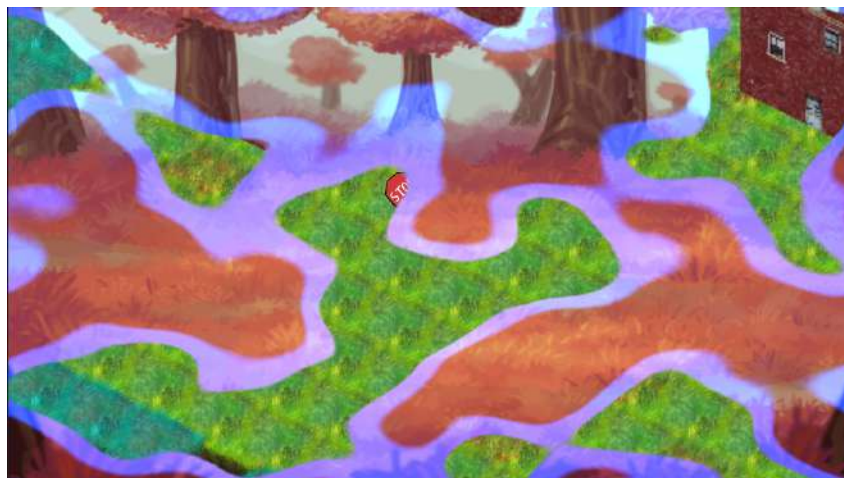

Figure 30 – Menu de combat en cours de transition

De ce fait, l'animation de la *transition* du menu peut se réaliser. Une fois le fond affiché, nous affichons également les *autres images* nécessaires au combat (les cartes, les sprites des deux combattants, les barres de vie et l'indication du tour) à l'aide d'un simple Fade réalisé à la main interpolant l'*alpha* de chaque images de 0 à 1, toujours par des **Coroutines**.

FIGURE 31 – Menu de combat

Une fois le menu de combat chargé, c'est au tour du joueur de **commencer** par défaut. Chaque carte possède des EventTrigger, qui réagiront aux événements de la souris, et renverront des **callback** propres à Unity dans le script *UICard*.

Les évenements OnEnter, OnExit, OnDrag, OnDrop sont implémentés :

- **OnEnter** : Ecarte la/les cartes autour de celle survolée. Pour plus d'informations, voir la Figure 35.
- **OnExit** : Réinitialise la position des cartes écartées.
- **OnDrag** : Déplace la carte sélectionné à l'endroit où la souris se trouve sur l'écran tant que le bouton de la souris est pressé
- **OnDrop** : Si le bouton de la souris est relâché, on va vérifier où la carte se trouve sur l'écran. Si le joueur a relâché sa carte sur la zone de combat, elle se consomme par un *UIDissolve* et applique son effet. En revanche, si elle se trouve à l'extérieur de cette zone, elle se replace *linéairement* à la position qui lui était attribuée dans la main du joueur.

b. Sérialisation des cartes

Tout comme le reste des données, la *sérialisation* des cartes passe par un *ScriptableObject*. Le *ScriptableObject*, *CardAsset.cs*, contient le nom, la description l'image et la rareté de la carte. De plus, si la carte est **unique**, elle doit renseigner un **identifiant** concernant à quelle courbe de Bezier son animation est rattachée, plus d'informations sur la Figure 36 dans la partie **Courbe de Bézier**.

Cependant, il ne s'agit pas d'un *ScriptableObject* basique. En effet, un *ScriptableObject* est censé ne contenir que des données, être un asset. Mais ce script propose également une fonction permettant de gérer ses effets, ce qui permet de programmer directement depuis le *ScriptableObject*, les effets qui seront appliqué par une certaine carte.

Pour cela, on va effectuer un *switch* sur le nom de l'asset, c'est à dire le nom du fichier engendré par ce script. Selon le nom du fichier, il va appliquer certains effets. Les effets sont très simple à implémenter, en général entre **une et 3 lignes** suffisent à créer une carte, ce qui est très **pratique** pour **implémenter** de nouvelles cartes imaginées par un futur *Game Designer*.

```
// Debuff
case "FlyingCarpet":
    fighter.status.Clear();
    break;

// Cure 20% of your health
case "Resting":
    fighter.Heal((int)(fighter.MaxHP * 0.2f));
    break;

// Deal 5% damage, then 10%, and 15%
case "Loneliness":
    opponent.Hurt((int)(opponent.MaxHP * 0.05f));
    AddStatus(opponent, new Status(this, opponent, Status.Type.Hurt, (int)(opponent.MaxHP * 0.10f)), Turn + 1);
    AddStatus(opponent, new Status(this, opponent, Status.Type.Hurt, (int)(opponent.MaxHP * 0.15f)), Turn + 2);
    break;

// Skip next turn
case "Rooted":
    AddStatus(opponent, new Status(this, opponent, Status.Type.Skip), Turn + 1);
    break;
```

FIGURE 32 – Code source présentant l'implémentation de quelques effets de cartes

c. Attaques directes et Status

On peut remarquer sur la Figure ci-dessus (32) plusieurs *patterns* possibles lors de la création des différents effets.

Tout d'abord, nous avons deux entités en jeu : "**fighter**" et "**opponent**". Il s'agit des deux combattants prenant part au combat. fighter est l'entitée invoquant la carte, et opponent est l'opposant.

Ensuite, élément important du Gameplay : deux types d'effets peuvent être appliqués : soit un **effet immédiat**, soit un "**status**". Un effet immédiat est simplement appliqué dès lors que la carte est jouée. Toutefois, ce que j'ai nommé "status" est un effet qui est *ajouté au joueur* mais qui sera appliqué dans un certain nombre de tour. La fonction AddStatus va ajouter un Status dans un **Dictionnaire** d'un combattant.

Le **Dictionnaire** en question a pour *clés* des **entiers** (signifiant un tour) et pour *valeurs* des **Status**. Grâce au Dictionnaire, on peut *accumuler* plusieurs Status dans un même tour, mais également accéder aux différents Status *associé* à un tour en reseignant la bonne clé. Ainsi, lorsqu'un tour commence pour un combattant, on va vérifier s'il a des Status pour le tour courant, et s'il en a, on les applique un par un.

Cet fonctionnalité permet de **varier** les cartes facilement, en plus d'ajouter un côté startégie aux cartes que le joueur utilisera.

d. Courbe de Bézier

Pour l'affichage des cartes, nous nous sommes inspiré de *Slay The Spire*, c'est à dire un affichage plutôt incurvé, comme si l'on jouait nous même avec les cartes dans nos mains. Pour ce faire, nous avons implémenté une **courbe de Bézier**.

Cette courbe de Bézier a été implémentée à l'aide du cours de **Modélisation 3D** que nous avons eu, dans laquelle nous avions eu l'occasion de réaliser celle par les polynômes de Bernstein et par l'algorithme de Casteljau, il s'agit de celui de **Casteljau** dans notre code.

A l'aide de *points de contrôles* nous allons pouvoir définir une courbe, pour laquelle nous aurons accès aux *positions* des points, mais égalements à leurs *normales*, qui nous serviront à obtenir l'**angle de rotation** nécessaire à la carte pour un point de cette courbe.

Pour obtenir cette **normale**, référons-nous au schéma 33. Il va falloir calculer le vecteur $\vec{u} \wedge \vec{v}$, pour cela commençons par déterminer $\vec{u}$ et $\vec{v}$. $\vec{u}$ est le vecteur *séparant un point de son voisin* : on va donc soustraire la position du point avec celle de son voisin et le *normaliser*. $\vec{v}$ est le vecteur *forward*, celui qui pointe vers nous. En effectuant le produit vectoriel $\vec{u} \wedge \vec{v}$ avec ces 2 vecteurs, nous obtenons les lignes rouges de la Figure 34.

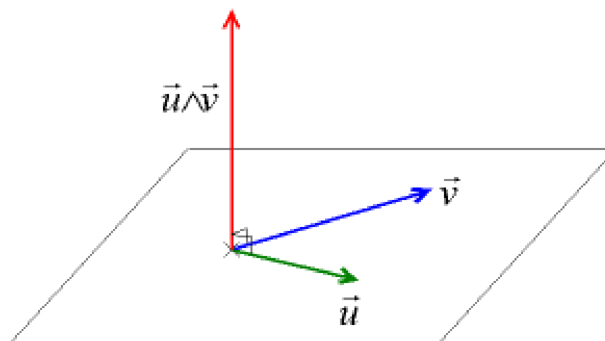

FIGURE 33 – Schéma du produit vectoriel

Pour avoir une meilleure visualisation de cette courbe, nous nous sommes servis du OnDrawGizmos() d'Unity, permettant d'afficher des formes et des lignes dans le viewport à des positions données en temps réel.

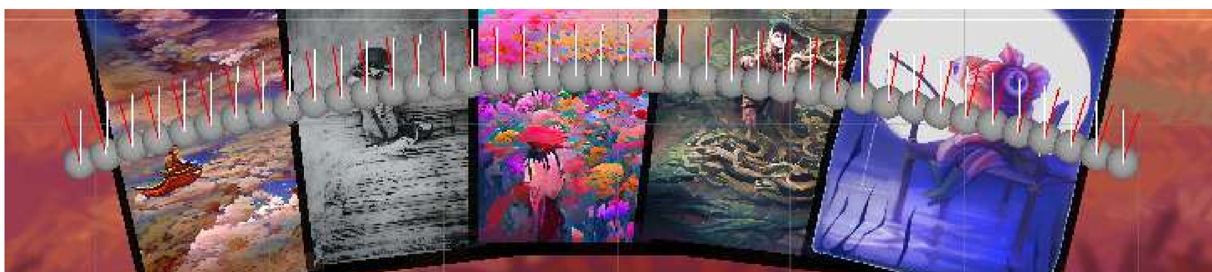

FIGURE 34 – Courbe de Bézier de la main du joueur

Ainsi, nous obtenons donc la **normale du point**, qui est censé être l'indicateur de l'orientation de la carte. Seulement, il nous reste à calculer l'*angle* à obtenir à partir de

cette normale. Unity propose des méthodes déjà toute faites pour cela, par **Vector2.Angle** qui nous permettra d'obtenir l'angle entre deux vecteurs en réalisant le *produit scalaire* de ces deux vecteurs en suivant la formule $\vec{u} \cdot \vec{v} = \|u\| \, \|v\| \, cos(\theta)$ dont l'angle à obtenir est θ.

Une fois *les positions* et *les angles* de chaques points initialisés et enregistrés, il est possible de positionner chaque carte sur un point et d'effectuer la rotation nécessaire. Seulement le programme va encore plus loin. En effet, nous avons mis au point une **interpolation linéaire** de la position et de la rotation des cartes de la main dans certain cas. Lorsque le joueur survole une carte, la/les cartes adjacentes se *déplaceront sur cette courbe à l'opposé de la carte* pour laisser celle survolée plus espacée, afin dans un premier lieu d'apprécier le design de la carte, mais aussi pour donner une impression de sélectionner cette carte de sa main.

FIGURE 35 – Effet de survol d'une carte

Toutefois, il aurait été dommage de ne pas plus **exploiter le potentiel** de ces courbes. Nous avons donc utilisé une courbe qui servira pour un type de *Feedback* d'attaque. Lorsque le joueur utilise une carte de rareté *Unique*, cette dernière lancera une **animation spéciale**, suivant cette courbe point par point.

À l'heure actuelle, seulement une carte unique a été implémentée, ce qui suffit pour prouver la fonctionnalité de l'*animation spéciale*, il s'agit d'un **Boomerang**. Comme le montre la Figure 36, cette courbe peut être très *modulable*. Notamment, ce parcours de courbe peut être effectué **un sens et dans l'autre**, ce qui fait que le joueur peut lancer le boomerang de son côté vers l'ennemi, et vice-versa.

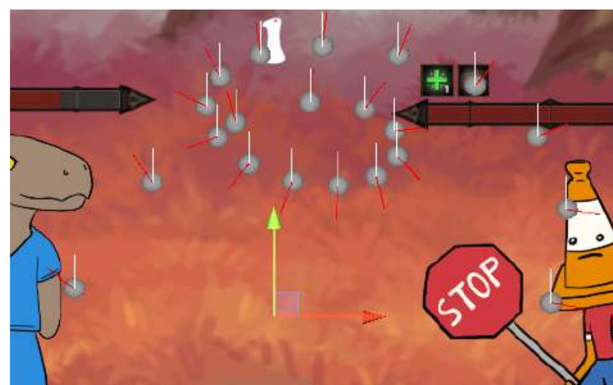

FIGURE 36 – Courbe de Bézier servant pour un Feedback

e. Feedback

La *courbe de Bézier* n'est pas le seul médiateur de Feedback, nous avons aussi exploité le Tool *UIEffect*, ou en général les *Sprites* de l'UI de combat pour renvoyer des Feedback aux attaques du joueur et de l'ennemi.

Dégâts

Lorsqu'un des deux combattant subit des dégâts, le sprite associé devient **rouge**, et **clignotte** pendant une demi-seconde avant de revenir à l'état normal. C'est une méthode plutôt classique mais efficace pour ce type de feedback.

FIGURE 37 – Ennemi subissant des dégats

Soin

Pour représenter le soin, j'ai essayé d'intégrer des *particules* sur l'UI. Seulement le mode de base d'Unity ne permet pas cela et requiert d'intégrer URP, ce qui causait beaucoup de problèmes par la suite avec notre UI. Le soin est donc finalement représenté avec un **gradient** appliqué au sprite, qui augmente de bas en haut vers le vert pendant 1 seconde.

FIGURE 38 – Joueur recevant un soin

Etourdissement

L'étourdissement est lui un effet **Sepia** multiplié au Sprite concerné. Il est appliqué de façon linéaire et *disparait* également de la même manière lorsque le combattant est libéré de son entrave.

FIGURE 39 – Ennemi passant son tour

Mort

Lorsque les points de vie du joueur atteignent 0 et donc qu'il meurt, l'écran de GameOver apparaît, mais pas seulement. À l'aide d'un **HsvModifier** et d'un **Dissolve**, le fond d'écran du menu de combat *s'assombrit*, pendant que le sprite du joueur *se dissout et disparait*. Ensuite, un écran de *GameOver apparait*, proposant au joueur de **réessayer** (ce qui rechargera le niveau), de **retourner** au menu principal ou bien de **quitter** le jeu.

FIGURE 40 – Animation du GameOver

Status

Lorsqu'un Status est appliqué, le joueur peut voir que légèrement au dessus de la barre de vie, une **icône propre au type de status** est apparue, ainsi que le *nombre de tours restant*. De plus si l'utilisateur survole ce status, une **boîte indiquant l'effet** qui sera appliqué (avec le détail du dommage ou du soin si une valeur entre en jeu), ainsi que la *carte responsable de l'effet*.

FIGURE 41 – Feedback du Status

Cartes

Lorsque le joueur *survole* une carte, d'autres feedback que le fait d'écarter les cartes adjacentes entrent en jeu. Premièrement, une *boîte décrivant l'effet* de la carte apparait. Ensuite, la carte se met à *émettre une petite lueur* réalisée grâce à un **UIShiny**, pour faire comprendre au joueur qu'il doit intéragir avec. Enfin, il y a aussi un cas spécial : les cartes de rareté **Unique**, qui *brille continuellement* sans même qu'on les survoles. Toutefois, c'est le seul type de rareté autre que commun d'implémenté : les cartes rares et légendaires n'ont pas encore de propriétés qui leur sont propres.

Figure 42 – Affichage de la description de la carte

8. Quêtes

Très liées à la partie **Cinématique**, les Quêtes représentent l'avancement dans le jeu, et fonctionnent selon le principe d'une file.

a. Sérialisation

Dans ce projet, sérialisation rime avec ScriptableObject, et cela en va de même avec les quêtes. En suivant le chemin *Create > QuestSystem > Quest*, nous pouvons instancier un nouvel asset **Quest**.

Le *ScriptableObject* en question possède 5 champs :

- **Name** : Indique le nom de la quête, à titre indicatif
- **Description** : Indique la description de la quête, à titre indicatif
- **Next** : La quête à assigner au joueur une fois qu'il aura complété l'actuelle. C'est une sorte de référence vers l'élément suivant de la file de quêtes du jeu. Toutefois, si la quête suivante est null, la fin du jeu est détectée (mais n'est bien sûre pas implémentée pour l'heure).
- **Trigger** : un objet **QuestGiver**, qui gérera la validation de la quête et déclenchera la fin de celle-ci. *QuestGiver* est un MonoBehaviour, ce qui signifie qu'il ne peut pas être référencé dans un *ScriptableObject*. La résolution de ce problème est détaillé dans la partie **Gestion**.
- **Cinematic** : la cinématique déclenchée à la fin de la quête. Nous avons déjà vu comment fonctionnait le système de cinématique. Cependant, il s'agit également d'un MonoBehaviour.

b. Gestion

Pour l'heure, seul un type de résolution de quête a été implémenté : **entrer dans un collider**, mais d'autres ont été pensées : *intéragir avec un personnage non-joueur* (PNJ), *éliminer un certain nombre d'ennemis*, etc... Mais par soucis de nécessité d'avancement des autres sytèmes du jeu (combat, intéraction environnement), l'implémentation de ces éléments de résolutions ont dus être mis en pause.

Le fonctionnement est simple, comme décrit plus haut : une quête référence la suivante et ainsi de suite jusqu'à la dernière. Cependant comment vérifier la fin de celle-ci ? Nous passons par un script **QuestGiver**, attaché à un GameObject dans une scène. Comme seul le cas du trigger est actuellement dans le jeu, nous le plaçons pour l'instant uniquement sur des trigger, mais il est possible de gérer facilement un cas "parler" avec un PNJ si l'on rattache le script sur une entitée du genre et en renseignant le type d'interaction à *TALK* au lieu de *ENTER_ COLLIDER* par exemple.

En outre, ce script possède lui aussi un asset issu du *ScriptableObject*. Si ce dernier est le même que celui possédé par le joueur et que la condition est remplie, on assigne la quête courante du joueur comme étant celle de *Next*.

Cependant, avant d'assigner cette nouvelle quête au joueur on déclenche la fin de la quête, qui est synonyme de cinématique. En effet, le GameObject contenant le script *QuestGiver* doit également contenir un script **Cinématique**, sérialisé comme souhaité, pour que tout se déroule comme prévu.

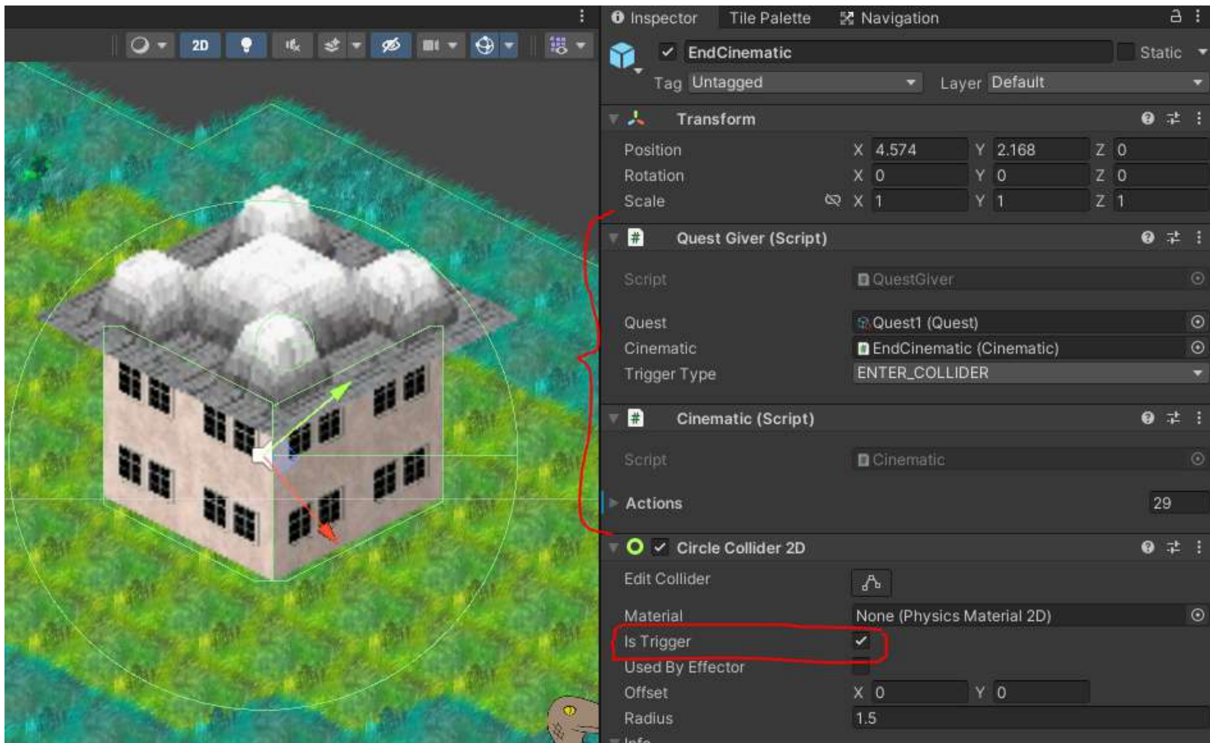

FIGURE 43 – Création typique d'un GameObject contenant les scripts **QuestGiver** et **Cinématique**

9. Bilan du projet

a. Autocritique

Nous avons sous-estimé l'ampleur de ce projet en terme de temps considérant que la charge de travail que nous avions à côté sur les cinqs autres projets majeurs du semestre ne nous semblait pas tant que ça critique. Ce fut très logiquement la raison principale de nos difficultés à avancer au rythme auquel nous nous étions fixé. Mais également d'un point de vue technique de par nos lacunes à l'égard de l'outil et des différents paradigmes en cause au sein de la programmation d'un jeu, qui diffère assez largement de nos précédents projets d'applications. Cela fini par nous pousser à revoir nos exigences à la baisse et couper une partie du contenu espéré afin de terminer dans les temps. Ceci dit malgré le fait que nous n'avons pas pu implémenter l'intégralité des fonctionnalités, nous restons toutefois satisfait du résultat final.

b. Enseignements tirés

Les difficultés que nous avons rencontrées tout au long du développement de notre jeu nous ont permis de tirer des leçons et de gagner de l'expérience.

Tout d'abord, Nous nous sommes rendu compte que pour un débutant, Unity est un moteur de jeu d'une complexité conséquente, et qu'il nécessitait de suivre beaucoup de tutoriels et de très fréquemment se plonger dans sa documentation pour le maitriser. Le temps d'apprentissage de cet outil fut ainsi bien plus long que prévu, ce qui a entraîné du retard sur notre projet. Nous avons cependant réagit à temps et revu nos objectifs en réattribuant nos tâches.

Mais grâce à cette expérience, nous savons déjà que les futurs travaux avec une technologie de cette envergure devra passer par de longues sessions d'entraînement sur l'outil afin de gagner en maîtrise. D'ailleurs il sera de bon ton de penser le découpage des tâches avec en tête la partie apprentissage auquelle la personne en besoin ne pourra s'y couper.

Nous avons par ailleurs pu trouver des liens avec certains des enseignements suivis auprès de Mme Faraj notamment en Moteur de Jeu et en Modélisation/Programmation 3D qui permet de mieux se représenter ce qu'est Unity et obtenir la concrétisation de toute la théorie vue en cours.

Également, nous nous sommes rendu compte de l'importance d'une bonne organisation au sein d'un tel projet. Sans quoi très rapidement des problèmes d'envergures allaient survenir et causer des retards malvenus. Il est nécéssaire de donner des rôles clairs à chacun et répartir les tâches équitablement afin d'être efficace. Il existe d'autant plus de très bons outils pour gérer toute cette facette du développement comme pour nous avec Trello qui propose une solution très pertinente à l'égard de la répartition et la classification des tâches.

Enfin nous nous sommes rendu compte qu'un jeu vidéo représentait tellement plus de

choses que de simples éléments à imbriquer les uns les autres. Au contraire, c'était là toute la difficulté de réussir à faire fonctionner tout harmonieusement sans que pour autant cela n'impacte les performances.

c. Perspectives

Le développement du jeu peut aisément continuer au delà puisque celui-ci a été concu de manière à ce que tout le contenu puisse être facilement intégré (personnes, cartes de jeu, environnement, musique, quêtes, etc..). Il est donc facile pour une personne tierce du projet de pouvoir remodeler le gameplay et la direction artistique comme il le souhaite. Nous prévoyons déjà d'intégrer dans les temps à venir un réel scénario composés de différents actes, actuellement en crous d'écriture, et potentiellement publier le résultat final sur des plateformes telles que Steam ou Itch.io.

10. Conclusion

Initialement, nous comptions implémenter une multitude de systèmes au sein du jeu afin de permettre de gérer toute l'étendue des idées que nous avions imaginées à la base. Effectivement nous pouvons compter sur la présence d'un système de quêtes, d'inventaire et d'interaction avec l'environnement ainsi qu'une exploration complète et un système de combat poussé mais nous n'avons pas réussi dans les délais à avoir un système de sauvegarde ou d'évolution du personnage. Les environnements ont d'ailleurs finalement été dessinés à la main plutôt que via une génération procédurale pour les mêmes raisons.

Nous disposons donc du premier acte de notre jeu où Steve se voit découvrir Oneiros et ses environs avec la possibilité d'interagir et combattre ce qui l'entoure jusqu'à atteindre la fin de la première quête du jeu qui cloturera la partie. Laissant donc la possibilité de poursuivre l'histoire avec d'autres actes, environnements et personnages par la suite. Les systèmes de jeu étant pour la plupart déjà présent.

Finalement, nous avons pu réaliser l'étendue du travail nécessaire pour produire un jeu de bout en bout. Cette expérience demeurera très enrichissante pour nous au vu de toutes les connaissances acquises qui pourront peut-être dans le futur nous approcher plus encore de la publication d'un jeu complet.

11. Annexes

a. Documentation

Voici la différence documentation utilisé pour ce projet :
- La chaine youtube de **TUTO UNITY FR** nous a permis de monter en compétence sur l'outil UNITY.
Lien : https ://www.youtube.com/c/TUTOUNITYFR
- La documentation officielle de **Unity**.
Lien : http ://docs.unity3d.com/
- Tuto sur les **Les Coroutines (Unity 3D)** **Lien :** https ://www.youtube.com/watch ?v=1YnbTilK6js&t=
Le site de Questions-Réponses **stackoverflow**.
Lien : http ://stackoverflow.com/

b. Bibliographie

Les articles utilisés dans la partie "Article de recherche" sont :

- Arches : a Framework for Modeling Complex Terrains par A. Peytavie, E. Galin, J. Grosjean et S. Merillou.
- A good sound in the right place : Exploring the effects of auditory-visual combinations on aesthetic preference par Ronghua Wanga et Jingwei Zhao.
- Controlled Animation of Video Sprites par Arno Schödl et Irfan A. Essa.

Les Tools utilisés sont :
- UIEffect
- MyBox