

REPUBLIQUE DU SENEGAL



MINISTERE DE L'ENSEIGNEMENT SUPERIEUR, DE LA RECHERCHE ET DE  
L'INNOVATION  
DIRECTION GENERALE DE L'ENSEIGNEMENT SUPERIEUR  
DIRECTION DE L'ENSEIGNEMENT SUPERIEUR PRIVE



**MEMOIRE DE FIN DE CYCLE**

**Pour l'obtention de la Licence Professionnelle en Informatique**

**Option : Programmation Développement**

Thème :

***Conception et développement d'une application  
Mobile de jeu vidéo : SPIRIT HUNTER***

Présenté et soutenu par : **Encadré par :**  
Mehdi TAGE EDDINE M. Bobo Diallo

**Titres et Grades/Fonctions :**  
Professeur d'Informatique en C#

**Année universitaire : 2022 – 2023**

## **Dédicaces**

Ce travail est dédié :

- ❖ A mes chers parents, pour l'éducation qu'ils m'ont prodiguée, avec tous les moyens et au prix de tous les sacrifices qu'ils ont consentis à mon égard, pour le sens du devoir qu'ils m'ont enseigné depuis l'enfance.
- ❖ A mes chères frère et sœurs, Hamoude, Ali et Zeinab qui m'ont toujours encouragé à l'idée de poursuivre mes études et de réaliser mes rêves.
- ❖ A ma fiancée Naomi, qui a toujours crus en moi et qui a toujours sut me relever lorsque je baissais les bras.
- ❖ A tous mes proches de la famille Tage Eddine, et plus particulièrement, ma cousine Assia et mes cousins Hussein, Noe et Zulfiqar.
- ❖ A mon meilleur ami Bryan, pour toutes les soirées passée à discuter de nos futurs projets.
- ❖ A mes chers amis Salah, Cherif, Moise, Lamine pour leur soutien.
- ❖ A tous mes camarades de classes de la première a la troisième année de Licence.
- ❖ A mes professeurs de programmation, pour m'avoir donné la logique et la vision de la programmation informatique.
- ❖ A toutes les personnes m'ayant aidé de près ou de loin, trouve ici l'expression de ma reconnaissance.

## **Remerciements**

- ❖ Tout d'abord, je remercie Dieu tout puissant, notre créateur de m'avoir donné la force, la volonté et le courage afin d'accomplir ce modeste travail.
- ❖ Ces remerciements vont au corps professoral et administratif de l'option Programmation & Développement de SUP'INFO, pour la richesse et la qualité de leur enseignement et qui déploient de grand effort pour assurer à leur étudiant une formation actualisée.
- ❖ Mes remerciements vont à mon encadreur Mr. Bobo Diallo, pour toute sa gentillesse, pour ses précieux conseils et pour sa patience avec moi, ainsi tous ceux qui m'ont aidée et soutenue dans mon travail.
- ❖ Je remercie également Mr Ousmane Cisse, pour toute sa collaboration, ses conseils et toutes les informations qui m'ont été transmises gracieusement de sa part.
- ❖ Finalement, je tiens à exprimer ma profonde gratitude à ma famille qui m'a toujours soutenue et à tous ceux qui m'ont aidé afin de réaliser ce mémoire. Ainsi que l'ensemble des enseignants qui ont contribué à ma formation.

## **Résumé**

Ce mémoire plonge dans l'univers passionnant de la conception et de la programmation de jeux vidéo. En se penchant sur les opportunités, les défis et les meilleures pratiques de l'industrie, il explore les frontières en constante expansion du jeu sur mobile.

Le projet de jeu mobile que nous avons développé représente une nouvelle ère dans le domaine du jeu vidéo, en apportant une perspective inédite grâce à la création de jeux de rôle (RPG) axés sur la culture africaine. Cette initiative marque un point de départ essentiel pour l'Afrique, lui permettant de laisser sa marque sur la scène mondiale du gaming en générant des expériences qui racontent l'histoire du continent et ses traditions.

À travers la programmation Orientée Objet avec le langage C# et la conception d'éléments graphiques 2D, ce projet démontre la symbiose entre l'art visuel et le code informatique. En unissant ces deux disciplines, il devient possible de créer des univers immersifs et interactifs qui captivent les joueurs.

Ce travail ne se limite pas à la création de jeu, il agit comme un moteur pour promouvoir les cultures africaines au sein de l'industrie du jeu vidéo. En contribuant à l'émergence de nouvelles entreprises spécialisées dans la conception de jeux vidéo africains, ce projet ouvre la voie à un futur où la richesse culturelle du continent prend vie à travers des mondes virtuels captivants.

## Liste des abréviations

**MVC** : Modèle-Vue-Contrôleur

**IA** : Intelligence Artificiel

**RPG** : Role-Playing Game (Jeu de Rôle)

**PNJ** : Personnage-Non-Jouable

**UI (IU)** : User Interface (Interface Utilisateur).

**UX** : User Experience (Expérience Utilisateur)

**FPS** : Frames Per Second (Images Par Seconde)

**IDE** : Integrated Development Environment (Environnement de Développement Intégré).

**TCAC** : Taux de Croissance Annuel Composé.

**PC** : Portable Computer (Ordinateur Portable)

**PV** : Points de Vie

**2D:** 2 Dimension. (Axe x, y)

**RAM:** Random Access Memory

## Glossaire

**IA** : L'intelligence Artificiel se réfère aux techniques et aux systèmes utilisés pour créer des comportements autonomes et réactifs pour les personnages non joueurs (PNJ) et d'autres éléments interactifs au sein du jeu.

**RPG** : Role-Playing Game (Jeu de Role). Un RPG est un genre de jeu vidéo dans lequel les joueurs assument le rôle de personnages fictifs et participent à des aventures au sein d'un monde imaginaire.

**PNJ** : Personnage-Non-Jouable : un PNJ est un personnage contrôlé par le jeu lui-même plutôt que par un joueur humain. Les PNJ sont présents pour fournir de l'interaction, des informations, des quêtes, des dialogues et d'autres éléments au joueur dans l'univers du jeu.

**C#** : Langage de programmation Orienté-Objet et fortement typé, il partage une syntaxe similaire à C/C++ et Java. Utilisé avec la plateforme .NET, il offre une gestion automatique de la mémoire grâce à son garbage collector. Il est utilisé pour développer des applications de bureau, web, mobiles et même des jeux avec Unity.

**GIT** : Git est un système de contrôle de version décentralisé largement utilisé pour la gestion du code source dans les projets de développement logiciel.

**Sprites** : Un Sprite est une image bidimensionnelle utilisée pour représenter un élément graphique dans un jeu ou une application.

**Framework** : Un Framework se réfère à un ensemble de fonctionnalités, d'outils pour le développement d'application. Unity est un Framework qui permet de créer des jeux vidéo en offrant des outils pour la conception 2D/3D.

**Application de jeu vidéo** : Logiciel interactif conçu pour le divertissement, l'amusement et l'apprentissage.

**Gameplay** : Un peu comme l'UX, le gameplay est la jouabilité. C'est l'ensemble des interactions, des mécaniques et des expériences interactives que les joueurs vivent lorsqu'ils jouent à un jeu vidéo.

**Gaming** : C'est l'activité de jouer à des jeux vidéo.

**Tile Palette** : La Palette de Tuile est un ensemble d'éléments graphiques 2D (appelés tuiles) qui sont utilisés pour créer des environnements, des cartes et des niveaux

## Liste de tableau et des figures

<b>Tableau 1 : Planning d'activité .....</b>	<b>7</b>
<b>Tableau 2 : Budgétisation.....</b>	<b>7</b>
<b>Figure 1 : Page par défaut d'un script lors de sa création.....</b>	<b>9</b>
<b>Figure 2 : Ajout des scripts dans Git .....</b>	<b>10</b>
<b>Figure 3 : Enregistrement des modifications dans Git.....</b>	<b>11</b>
<b>Figure 4 : Gameplay du jeu : The boy in the Savana.....</b>	<b>16</b>
<b>Figure 5 : Gameplay du jeu : Africa's Legend.....</b>	<b>16</b>
<b>Figure 6 : Créateur du jeu : Les Héros du Sahel .....</b>	<b>17</b>
<b>Figure 7 : Gameplay du jeu : Aurion .....</b>	<b>17</b>
<b>Figure 8 : Diagramme du chiffre d'affaires du marché mondial du jeu vidéo en 2021 .....</b>	<b>18</b>
<b>Figure 9 : Tableau des supports de jeux privilégiés dans le monde .....</b>	<b>20</b>
<b>Figure 10 : Revenu annuel des plateformes de jeux vidéo .....</b>	<b>22</b>
<b>Figure 11 : Interface Utilisateur du jeu : PUBG mobile .....</b>	<b>25</b>
<b>Figure 12 : Interface du logiciel UNITY Hub .....</b>	<b>30</b>
<b>Figure 13 : Installation d'une version de UNITY .....</b>	<b>30</b>
<b>Figure 14 : Création d'un projet 2D.....</b>	<b>31</b>
<b>Figure 15 : Interface d'un nouveau projet UNITY .....</b>	<b>31</b>
<b>Figure 16 : Installation du New Input System .....</b>	<b>32</b>
<b>Figure 17 : Interface de l'éditeur audio : Audacity .....</b>	<b>33</b>
<b>Figure 18 : Ouverture du GIT Bash après l'installation de Git .....</b>	<b>34</b>
<b>Figure 19 : Sprites de PNJ .....</b>	<b>34</b>
<b>Figure 20 : Découpe de chaque Sprite des PNJ.....</b>	<b>35</b>
<b>Figure 21 : Cr éation d'une TileMap Rectangulaire.....</b>	<b>35</b>
<b>Figure 22 : Application de la TileMap.....</b>	<b>36</b>
<b>Figure 23 : Cr éation d'une TilePalette .....</b>	<b>36</b>
<b>Figure 24 : Conception d'un village avec la TilePalette.....</b>	<b>37</b>
<b>Figure 25 : Cr éation du bouton d'attaque.....</b>	<b>38</b>
<b>Figure 26 : Positionnement Responsif du bouton d'attaque avec 'Anchor Preset' ....</b>	<b>38</b>

Figure 27 : Résultat de la position et de l'image source du bouton d'attaque ....	39
Figure 28 : Composants d'un objet maison.....	40
Figure 29 : Composants d'un élément UI.....	41
Figure 30 : Réalisation d'un couteau avec pixiart.com.....	42
Figure 31 : Sprite Editor.....	45
Figure 32 : Découpe des Sprites du joueur.....	45
Figure 33 : Découpe de l'effet d'attaque du joueur .....	46
Figure 34 : Ajout d'un Tag 'Player' a l'objet joueur.....	46
Figure 35 : Création d'un InputActions tactile (PlayerControlle) .....	47
Figure 36 : Configuration de l'InputActions.....	47
Figure 37 : Interface du Village de Tindaba .....	48
Figure 38 : Interface de la foret .....	48
Figure 39 : Interface du village voisin .....	49
Figure 40 : Interface du Grand lac.....	49
Figure 41 : Interface du labyrinthe .....	50
Figure 42 : Interface du territoire du Grand Saltigue .....	50
Figure 43 : Animation (Up & Down) du piège Lance .....	51
Figure 44 : Interface de l'Animator.....	52
Figure 45 : Transition de l'animation Idle a l'animation Up .....	52
Figure 46 : Script (LancePiege) de la Lance.....	53
Figure 47 : Méthode OnTriggerEnter2D du script de la Lance .....	54
Figure 48 : Coroutine ActivateCoroutine() du Script LancePiege .....	54
Figure 49 : Création d'un PNJ (Abdou).....	55
Figure 50 : Scripts du PNJ.....	56
Figure 51 : Script PnjControle du PNJ 'Abdou' .....	57
Figure 52 : Création de l'ennemi 'Slime' .....	58
Figure 53 : Animations de l'ennemi 'Slime' .....	59
Figure 54 : Script de contrôle de l'ennemi 'SlimeContolle' .....	61
Figure 55 : Script de Points de vie de l'ennemi Slime 'SlimeHp' .....	62
Figure 56 : Script d'attaque du Slime 'SlimeAttack.cs'	.....

Figure 57 : Script de contrôle du joueur ‘PlayerMove.cs’ .....	63
Figure 58 : Méthode DecreaseHealth du Script HpPlayer .....	65
Figure 59 : Méthode IncreaseHealth du Script HpPlayer .....	65
Figure 60 : Interface du Menu Principale .....	66
Figure 61 : Déplacement du joueur .....	67
Figure 62 : Récolte de pièces.....	67
Figure 63 : Interface de la boutique de Samba .....	68
Figure 64 : Dialogue avec le PNJ ‘Abdou’ .....	68
Figure 65 : Activation du bouclier de protection du joueur .....	69
Figure 66 : Ennemi ‘Diablotin’ vaincu .....	69
Figure 67 : Utilisation de la potion de Soin .....	70
Figure 68 : Sélection d’un ennemi pour verrouiller la cible .....	70

## **Sommaire**

Introduction générale .....	1
<b>Première partie : Présentation et Conception des jeux mobile.</b> .....	<b>3</b>
Chapitre 1 –Présentation générale .....	4
Chapitre 2 – Conception et Programmation des jeux mobiles .....	7
<b>Deuxième partie : Analyse des jeux Mobiles sur l’Afrique.</b> .....	<b>14</b>
Chapitre 3 – Les opportunités offertes par l’industrie des jeux mobile.....	15
Chapitre 4 – Les défis de l’industrie.....	20
Chapitre 5 – Meilleures pratiques pour la conception et la programmation de jeux vidéo mobiles .....	24
<b>Troisième partie : Réalisation du projet</b> .....	<b>28</b>
Chapitre 6 – Mise en place de l’environnement de développement pour une plateforme mobile.....	29
Chapitre 7 – Développement du jeu mobile en 2D .....	43
Conclusion .....	72
Webographie .....	73
Table des matières.....	74

## **Introduction Générale**

La programmation de jeux vidéo est un domaine passionnant qui implique de créer des jeux pour tous types d'appareils tels que les consoles, les ordinateurs ainsi que les appareils mobiles tels que les smartphones et les tablettes. Ils peuvent être des jeux simples tels que des puzzles ou des jeux plus complexes tels que des jeux de rôle (RPG) et des jeux de tir à la première personne.

Le développement de jeux mobiles implique plusieurs étapes, notamment la conception de l'interface utilisateur, la création des graphiques et des animations, l'écriture du code et les tests. Il est également nécessaire de prendre en compte les limitations matérielles des appareils mobiles tels que la taille de l'écran et les capacités de traitement.

Les jeux vidéo mobiles ont connu une croissance explosive ces dernières années et sont devenus l'une des formes de divertissement les plus populaires dans le monde. Cependant, cette croissance rapide a également créé des défis, tels que la concurrence féroce, les problèmes de monétisation et les exigences de plus en plus élevées des utilisateurs. Pour réussir dans ce secteur en constante évolution, il est essentiel de comprendre les défis et les opportunités auxquels les développeurs de jeux mobiles sont confrontés, ainsi que les meilleures pratiques pour la conception et la programmation de jeux mobiles réussis.

On peut aussi constater l'absence de jeux de rôle en 2D du style Zelda qui embrasse les cultures africaines. Bien qu'il existe de nombreux jeux avec des cultures Africaines comme Afro Samurai, Ayo: A Rain Tale, Sundiata Rebirth, on constate des problèmes liés à créer un héros africain qui marquera le monde comme les grands jeux dont The Legend of Zelda, Mario, Pokemon...

Dans la plupart des jeux africains disponibles, il y a certains problèmes qu'on remarque tel que :

- **Manque de diversité culturelle** : Certains jeux 2D RPG africains pourraient se concentrer uniquement sur une culture ou une région spécifique, négligeant ainsi la diversité des cultures africaines. Cela peut contribuer à la sous-représentation de certaines cultures et perpétuer des déséquilibres dans les représentations.

- **Appauvrissement des récits locaux** : Il est important de préserver et d'intégrer les récits locaux et les éléments culturels africains dans le jeu pour offrir une expérience authentique et enrichissante.
- **Mélange avec différentes cultures** : Lors de la création d'un jeu basé sur l'Afrique, il est important de trouver un équilibre entre l'incorporation d'éléments culturels africains et la possibilité d'attirer un public plus large. Il est nécessaire d'éviter un mélange incohérent ou superficiel des cultures, en veillant à représenter de manière respectueuse et précise les différentes cultures africaines.
- **Manque d'interaction et de jouabilité** : Pour créer une expérience de jeu captivante, il est crucial de concevoir des mécanismes de jeu interactifs et engageants. Cela peut inclure des quêtes intéressantes, des combats stimulants, des choix et des conséquences significatifs, ainsi que des interactions approfondies avec les personnages non-joueurs et l'environnement du jeu. Le manque d'interaction et de jouabilité peut rendre l'expérience du jeu monotone et peu attrayante pour les joueurs.

---

Première partie : Présentation et Conception des jeux mobile.

---

## **Chapitre 1 : Présentation générale**

### **1.1 Présentation du thème**

À l'ère du numérique, les jeux mobiles ont conquis des millions de joueurs à travers le monde, transcendant les frontières géographiques et démographiques. Les smartphones, de plus en plus puissants et polyvalents, ont ouvert la voie à une nouvelle ère du jeu vidéo, où les joueurs peuvent accéder à une vaste gamme de jeux à tout moment et en tout lieu. Cette omniprésence des jeux mobiles dans nos vies a donné naissance à un domaine en constante évolution, marqué par des opportunités, des défis et des innovations sans fin.

### **1.2 Choix et intérêt du sujet**

Le choix de ce sujet, axé sur les jeux mobiles, repose sur une combinaison d'intérêt personnel et d'expérience. En tant que passionné de jeux vidéo, Nous avons été fasciné par l'évolution de l'industrie et l'impact qu'elle pourrait avoir sur la culture africaine.

Notre intérêt pour les jeux mobiles a été particulièrement aiguisé ces dernières années, alors qu'elles sont devenu un phénomène de masse, accessibles à un public diversifié et en constante expansion. Notre expérience personnelle en tant que joueur de jeux mobiles et nos connaissances dans les langages Orientés Objet, nous a également inspiré à en savoir plus sur ce domaine dynamique.

### **1.3 Problématique**

L'industrie du jeu vidéo en Afrique est en pleine croissance, mais elle fait face à plusieurs défis importants qui entravent son développement dans l'industrie du jeux vidéo tels que:

- Les infrastructures technologiques, telles que l'accès à Internet haut débit et les appareils compatibles avec les jeux vidéo, sont limitées.
- Certain pays connaissent des défis économiques, et de nombreux joueurs n'ont pas les moyens d'acheter des consoles de jeu coûteuses ce qui est une opportunité de se baser sur la plateforme mobile, qui est plus facile a se procurer.
- Les studios de développement de jeux vidéo en Afrique font souvent face à des ressources limitées en termes de financement, de talents techniques et de formation.

La question centrale qui guide cette recherche est la suivante : "Quels sont les défis et les opportunités pour le développement de jeux vidéo dans le contexte de l'industrie du gaming Africain ?"

Cette question constitue le socle de notre exploration approfondie de l'industrie du jeu vidéo en Afrique, un secteur en pleine expansion qui suscite un intérêt croissant tant au niveau régional qu'international.

#### **1.4 Etapes du processus :**

Une fois la conception terminée, vient la phase de développement où les programmeurs travaillent à la création du jeu en utilisant des langages de programmation et des outils adaptés aux plateformes mobiles. Cette étape inclut la programmation des mécaniques de jeu, l'optimisation des performances et la gestion des contrôles tactiles.

Le processus de test et de débogage est essentiel pour s'assurer du bon fonctionnement du jeu, identifier et corriger les éventuels bugs et améliorer l'expérience utilisateur.

Avant de passer au chapitre suivant sur la programmation des jeux vidéo, nous allons mener la planification du projet en illustrant les différentes étapes du processus de conception de notre jeu 2D. Le planning d'activité mettra en évidence les problèmes rencontrés lors de chaque étape ainsi que les solutions possibles pour les surmonter, et la budgétisation illustrera les ressources financières afin d'assurer la réussite globale du projet.

Activité	Ressource Humaines	Ressource Matériel	Résultats Attendus	Conditions Critiques
<b>Expression des besoins</b>	Moi-même	Ordinateurs, Cahiers de notes	Identification des fonctionnalités et des thèmes du jeu	Manque de compréhension claire des besoins
<b>Conception du monde du jeu</b>	Moi-même	Logiciels de conception, smartphone	Création des environnements, des personnages et des éléments du jeu	Difficulté à trouver le style artistique approprié
<b>Programmation du gameplay</b>	Moi-même	Ordinateurs, Unity Engine, smartphone	Mise en place des mécaniques de jeu, des combats et des quêtes	Problèmes de bugs ou de performances dans le jeu
<b>Création des ressources graphiques</b>	Moi-même	Logiciels de création graphique	Conception des sprites, des animations et des effets visuels	Délais de production ou contraintes artistiques

<b>Intégration sonore</b>	Moi-même	Logiciels de montage audio	Ajout de musiques et d'effets sonores	Problèmes de compatibilité ou de qualité audio
<b>Test et correction des bugs</b>	Moi-même, des amis	Ordinateurs, smartphone	Identification et résolution des problèmes et des erreurs	Bugs critiques non résolus ou difficultés d'équilibrage du jeu

**Tableau 1 : Planning d'activité.**

Activités	Dépenses	Montant
<b>Expression des besoins</b>	Formation UDEMY Effets Visuels	12.99\$ (7 890 FCFA) 4.99\$ (3 028 FCFA)
<b>TOTAL</b>		17.98\$ (10 829,47FCFA)

**Tableau 2 : Budgétisation.**

## **Chapitre 2 : Conception et programmation des jeux mobiles**

### **2.1 Méthodologie et approche adoptée**

La conception est essentielle au succès d'un jeu, car elle détermine l'expérience du joueur et son niveau d'engagement. En comprenant les principes fondamentaux de la conception de jeux mobiles et en relevant les défis spécifiques de cette plateforme, nous pourrons créer un jeu RPG 2D africain sur mobile offrant une expérience immersive et captivante pour les joueurs.

Nous explorerons les principes de base de la conception des jeux mobiles, tels que la jouabilité, l'accessibilité et l'engagement des joueurs. Nous aborderons également les défis spécifiques aux jeux mobiles, tels que les contraintes de l'écran tactile et les ressources limitées, en proposant des stratégies pour les surmonter. Enfin, nous présenterons les étapes du processus de conception des jeux mobiles, en soulignant l'importance de la documentation et du prototypage.

L'objectif de ce chapitre est de fournir un aperçu concis et pratique de la conception des jeux mobiles. En comprenant les principes fondamentaux et en relevant les défis spécifiques à cette plateforme, nous serons en mesure de créer un jeu captivant qui offrira une expérience mémorable aux joueurs.

#### **2.1.1 Principes de base :**

Dans le contexte de l'industrie des jeux mobiles, les principes de base de la conception des jeux revêtent une importance particulière. En effet, l'industrie connaît une croissance exponentielle, avec des millions de jeux disponibles sur les plateformes mobiles. Pour se démarquer et attirer les joueurs, il est essentiel de maîtriser ces principes fondamentaux. Voici les points clés que nous aborderons dans ce chapitre :

- Jouabilité optimale**

**Adaptation aux contrôles à l'écran tactile** : Utiliser des contrôles intuitifs et adaptés au format mobile, tels que des gestes tactiles ou des boutons virtuels, pour offrir une expérience de jeu fluide.

**Réactivité du jeu** : S'assurer que les actions du joueur ont une réponse rapide et immédiate, offrant ainsi une sensation de contrôle direct sur le personnage du jeu.

**Équilibre entre défi et satisfaction** : Proposer des défis intéressants et progressifs, tout en offrant des moments de réussite et de satisfaction pour maintenir l'intérêt des joueurs.

- **Expérience utilisateur (UX)**

**Interface utilisateur attrayante :** Créer une interface utilisateur visuellement attrayante, avec des éléments graphiques cohérents, une disposition intuitive des commandes et des indications claires pour guider les joueurs.

**Narration captivante :** Développer une histoire engageante qui se déroule tout au long du jeu, avec des personnages intéressants, des quêtes captivantes et des événements mémorables.

**Attention aux détails visuels et sonores :** Accorder une attention particulière à la qualité des graphismes, des animations et des effets sonores pour créer une immersion visuelle et auditive.

### **2.1.2 Défis spécifiques :**

L'un des défis majeurs de la conception de jeux mobiles est lié aux contraintes de l'écran tactile. Contrairement aux plateformes traditionnelles avec des contrôles physiques, les jeux mobiles doivent s'adapter à l'interaction tactile. Cela nécessite de repenser les mécanismes de jeu pour offrir une expérience de contrôle intuitive et réactive. Les concepteurs doivent trouver des solutions créatives pour des gestes tels que le tap, le swipe et le pinch-to-zoom, en tenant compte de la taille réduite de l'écran et des gestes couramment utilisés par les utilisateurs.

Un autre défi est lié aux ressources limitées des appareils mobiles. Les smartphones et les tablettes ont des limitations en termes de puissance de traitement, de mémoire et de durée de vie de la batterie. Les concepteurs doivent optimiser les performances du jeu en utilisant des techniques de programmation efficaces, en minimisant l'utilisation des ressources et en assurant une consommation d'énergie optimisée. Cela nécessite une planification minutieuse de l'utilisation des ressources, l'optimisation des graphismes et des effets sonores, ainsi que l'optimisation du code pour réduire la charge sur le processeur et la mémoire.

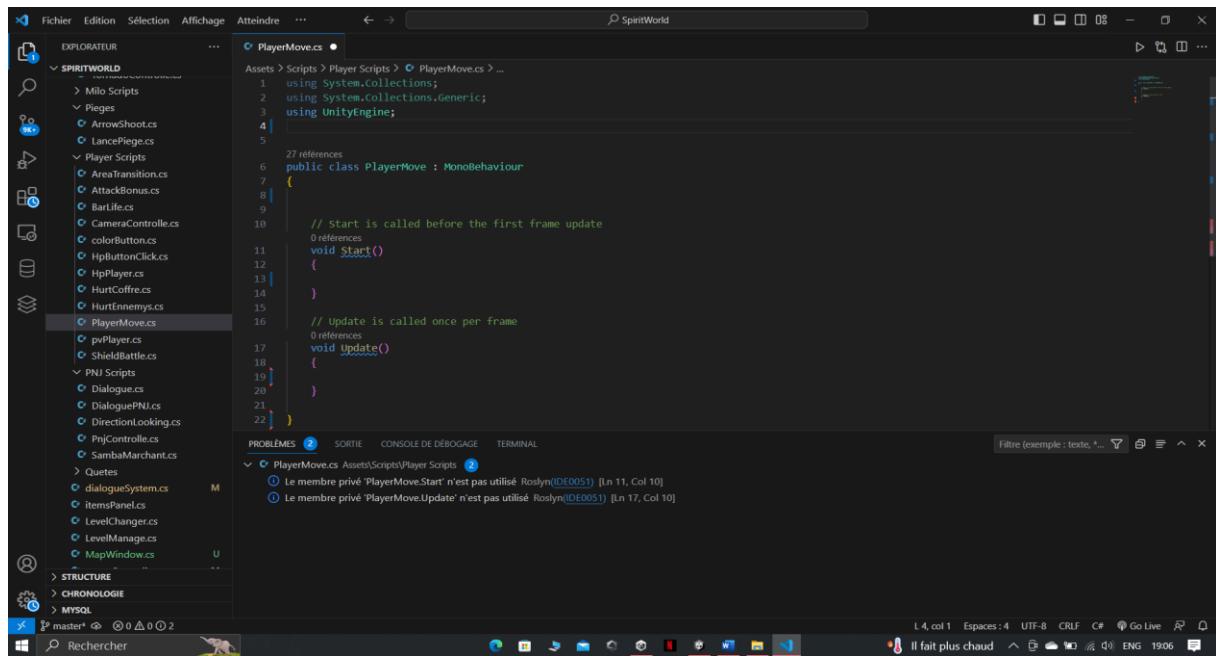
### **2.2 Langages de programmation orientée objet :**

Nous examinerons les langages de programmation orientée objet les plus couramment utilisés dans le développement de jeux mobiles. Nous étudierons leurs caractéristiques, leurs avantages et leurs inconvénients, ainsi que leur pertinence pour notre jeu. Nous nous concentrerons particulièrement sur l'utilisation du langage C# pour notre jeu 2D RPG basé sur l'Afrique, ainsi que sur l'utilisation de l'outil de contrôle de version Git.

Le langage C# est largement utilisé dans l'industrie du jeu vidéo, notamment pour le développement de jeux mobiles sur la plateforme Unity. C# offre une syntaxe élégante et

puissante, ainsi qu'une prise en charge complète de la programmation orientée objet. Il permet de créer des structures de données complexes, de définir des classes et d'établir des relations entre les objets du jeu.

Dans notre projet, nous avons choisi d'utiliser C# pour sa compatibilité avec Unity, qui est l'environnement de développement que nous utilisons. Nous exploitons les fonctionnalités de C# pour créer des scripts qui contrôlent le comportement des personnages, gèrent les interactions du joueur avec le monde du jeu, et implémentent les mécanismes de combat et de progression.



**Figure 1 : Page par défaut d'un script lors de sa création.**

Dans cette page de script nommée PlayerMove, nous effectuerons par la suite, le script de déplacement du joueur, avec ses attaques et ses animations.

Dans la méthode Start() toutes exécutions de code dans cette méthode sera appelée une seule fois uniquement lors du lancement du jeu, ce qui est pratique pour récupérer la références de nos objets ou initialiser des valeurs .

Dans la méthode Update(), toutes exécutions de code dans cette méthode sera appelée à chaque image rendue ou frame (généralement 60 FPS) tant que le script est attaché à un objet de la scène de jeu. C'est ici que nous allons gérer la logique de mise a jours continue tel que les mouvements des personnages, la détection de collision, les animations ainsi que le gameplay.

## 2.3 Technologies et outils de développement :

Dans cette section consacrée aux technologies et aux outils de développement, nous examinerons les éléments essentiels nécessaires pour créer notre jeu 2D RPG basé sur l'Afrique. Voici les technologies et outils clés que nous utiliserons :

- **Unity** : Unity est un moteur de jeu multiplateforme largement utilisé dans l'industrie du jeu vidéo. Il offre un large éventail de fonctionnalités pour le développement de jeux mobiles, y compris la gestion des graphismes, des collisions, des animations et des interactions. Unity fournit également un environnement de développement intégré (IDE) convivial qui facilite la création, le test et le déploiement de notre jeu.
- **Pixiart.com** : Pour concevoir les effets visuels et les éléments graphiques de notre jeu, nous utiliserons un site de graphisme et aussi des logiciels tel que Photoshop. Ces outils nous permettront de créer des personnages, des décors, des effets visuels et d'autres éléments visuels nécessaires pour donner vie à notre jeu.
- **Visual Studio Code** : Pour écrire et gérer notre code, nous utiliserons un environnement de développement intégré (IDE) tel que Visual Studio Code. Ces IDE offrent des fonctionnalités avancées pour la programmation, telles que l'autocomplétions du code, le débogage et la gestion des dépendances, ce qui facilite le processus de développement et garantit une programmation efficace.
- **Git** : Nous utilisons Git comme outil de contrôle de version pour notre projet. Git nous permet de gérer efficacement les différentes versions de notre code. Grâce à Git, nous sommes en mesure de suivre l'évolution de notre code, d'effectuer des sauvegardes régulières et de travailler de manière collaborative et transparente.

Voici comment Enregistrer la version de nos scripts de notre projet Unity dans GIT :

```
MINGW64/d/Workspace_Unity/SpiritWorld
Author: Tage <mehditage10@outlook.fr>
Date: Tue Jun 27 18:19:49 2023 +0000

version 1.0.3
commit 878102cfaf0862fc9ed0201ef3b7d5b68c652f
Author: Tage <mehditage10@outlook.fr>
Date: Sun Jun 25 21:55:07 2023 +0000

Version 1.0.1
commit c254f40e27f8e1c1cfca39c47491c3268c815e6e
Author: Tage <mehditage10@outlook.fr>
Date: Sat Jun 24 23:11:22 2023 +0000

Version 1.0.0
HeidelBLAPTOP-2F90GH0H MINGW64 /d/workspace_Unity/spiritworld [master]
$ git add .
warning: in the working copy of 'Assets/Animations/Ennemy/BossFinal/LanceAttack.anim', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/BossFinal/Necromancer.controller', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Animation/DiableLoin/controller', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Images/Acteurs/Cyclops Sprite Sheet.png.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Images/FX/Fire_Incendary_96x32.png.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Scripting/Internal/AssemblyDefinition.cs.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Library/Bees/13000bees.dap.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'ProjectSettings/TagManager.asset', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'ProjectSettings/TextureImporter.asset', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclops.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope.controller.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/cycle.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/hit.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/hitGround.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/hitGround.animation.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/idle.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/move.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/rockThrow.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/rockThrow.animation.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Animations/Ennemy/Cyclope/walk.animation', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Images/Acteurs/Cyclops Sprite Sheet 1.png.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Prefabs/Objects/Rock.prefab', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Prefabs/Objects/Rock.prefab.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Scripts/Ennemy Scripts/Cyclope/CyclopController.cs.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Scripts/Ennemy Scripts/Necroman/Necroman.cs.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Scripts/Ennemy Scripts/Necroman/Necromancer.cs.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Scripts/Ennemy Scripts/RayAttack.cs.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Scripts/RockShoot.cs.meta', LF will be replaced by CRLF the next time Git touches it
HeidelBLAPTOP-2F90GH0H MINGW64 /d/workspace_Unity/spiritworld [master]
```

Figure 2 : Ajout des scripts dans Git.

```
MINGW64/d/Workspace_Unity/SpiritWorld
warning: in the working copy of 'Assets/Tiles/waterTile/[A]Water_pipo_997.asset', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Tiles/waterTile/[A]Water_pipo_998.asset', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Tiles/waterTile/[A]Water_pipo_998.asset.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Tiles/waterTile/[A]Water_pipo_999.asset', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Assets/Tiles/waterTile/[A]Water_pipo_999.asset.meta', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'Library/InputUserSettings.json', LF will be replaced by CRLF the next time Git touches it
HeidelBLAPTOP-2F90GH0H MINGW64 /d/workspace_Unity/Spiritworld [master]
$ git commit -m "version 1.0.7"
Auto packing the repository in background for optimum performance.
See 'git help gc' for more housekeeping.
Compressing objects: 100% (12691/12691), done.
Counting objects: 100%, 12691/12691, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12515/12515), done.
Writing objects: 100% (12691/12691), done.
Total 12691 (delta 0), reused 0 (delta 0), pack-reused 0
Removing duplicate objects: 100% (256/256), done.
[master 9abb227] version 1.0.7
 9203 files changed, 738573 insertions(+), 77690 deletions(-)
create mode 10064 Assets/Animations/Ennemy/BossFinal/Apparait.anim
create mode 10064 Assets/Animations/Ennemy/BossFinal/BossFinal/disparait.anim.meta
create mode 10064 Assets/Animations/Ennemy/BossFinal/disparait.anim
create mode 10064 Assets/Animations/Ennemy/BossFinal/Disparait.anim.meta
create mode 10064 Assets/Animations/Ennemy/Little_Slime.meta
create mode 10064 Assets/Animations/Ennemy/Little_Slime/SlimeLittle.controller
create mode 10064 Assets/Animations/Ennemy/Little_Slime/SlimeLittle.controller.meta
create mode 10064 Assets/Animations/Ennemy/Little_Slime/attack.anim
create mode 10064 Assets/Animations/Ennemy/Little_Slime/attack.animation.meta
create mode 10064 Assets/Animations/Ennemy/Little_Slime/die.anim
create mode 10064 Assets/Animations/Ennemy/Little_Slime/die.animation.meta
create mode 10064 Assets/Animations/Ennemy/Little_Slime/hit.anim
create mode 10064 Assets/Animations/Ennemy/Little_Slime/hit.animation.meta
create mode 10064 Assets/Animations/Ennemy/Little_Slime/move.anim
create mode 10064 Assets/Animations/Ennemy/Little_Slime/move.animation.meta
create mode 10064 Assets/Animations/Ennemy/Slime.meta
create mode 10064 Assets/Animations/Ennemy/Slime/slime.controller
create mode 10064 Assets/Animations/Ennemy/Slime/slime.controller.meta
create mode 10064 Assets/Animations/Ennemy/Slime/attack.anim
create mode 10064 Assets/Animations/Ennemy/Slime/attack.animation.meta
create mode 10064 Assets/Animations/Ennemy/Slime/die.anim
create mode 10064 Assets/Animations/Ennemy/Slime/die.animation.meta
create mode 10064 Assets/Animations/Ennemy/Slime/hit.anim
create mode 10064 Assets/Animations/Ennemy/Slime/hit.animation.meta
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/dead.anim
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/dead.animation.meta
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/hit.anim
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/hit.animation.meta
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/biteDown.anim
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/biteLeft.anim
create mode 10064 Assets/Animations/Ennemy/SpiderBoss/biteRight.anim
HeidelBLAPTOP-2F90GH0H MINGW64 /d/workspace_Unity/Spiritworld [master]
```

Figure 3 : Enregistrement des modifications dans Git.

Ici, nous nous déplaçons dans le répertoire de notre projets grâce à au mot clé « cd », puis avec « git add. » nous ajoutons les modifications de notre projet à l'index Git ce qui va ajouter tous les scripts du projet dans Git. Ensuite pour enregistrer la version de notre projet, nous

utilisons « **commit -m ‘version 1.0.7’** » par exemple et cela va enregistrer tous les fichiers du projet. Nous vérifions si tout s'est bien passé et nous allons vérifier si notre commit s'est bien créer grâce à la commande « `git log` » et nous verrons une liste de tous les commit effectuée dans notre projet.

L'utilisation de C# et Git dans notre projet de développement de jeux mobiles nous permet de bénéficier d'un langage de programmation solide et d'un système de contrôle de version fiable. Cela garantit une gestion efficace du code, une collaboration fluide et la possibilité de revenir en arrière en cas de besoin.

- **Audacity** : Pour ajouter des effets sonores, nous utiliserons Audacity qui est un logiciel d'édition audio gratuit et open-source largement utilisé dans l'industrie du jeu vidéo. Nous enregistrons des bruits pour les appliquer à notre jeu.

En utilisant ces technologies et ces outils de développement, nous serons en mesure de créer notre jeu 2D RPG basé sur l'Afrique de manière efficace et professionnelle. Chaque outil joue un rôle clé dans le processus de développement, de la conception graphique à la programmation en passant par l'animation et la gestion du code.

## **2.4 Défis techniques de la programmation de jeux :**

Lors du développement des jeux mobiles, nous sommes confrontés à plusieurs défis techniques qui nécessitent une approche soigneuse et des solutions adaptées. L'un de ces défis est l'optimisation des performances pour garantir que les jeux fonctionnent de manière fluide et réactive sur différents appareils mobiles. Cela peut impliquer l'utilisation de techniques telles que la gestion efficace des ressources, la réduction de la charge de calcul et l'optimisation des algorithmes.

Un autre défi majeur est la gestion de la mémoire, car les appareils mobiles ont des ressources limitées. Il est essentiel de s'assurer que les jeux utilisent la mémoire de manière efficace, en évitant les fuites et les gaspillages de mémoire. Cela peut être réalisé en utilisant des techniques telles que la gestion des allocations et des libérations de mémoire, la mise en cache des ressources et la réutilisation des objets.

Un autre aspect important est la gestion des contrôles tactiles, car les jeux mobiles sont souvent contrôlés via l'écran tactile. Nous devons concevoir des contrôles intuitifs et réactifs qui offrent une expérience de jeu agréable. Cela peut inclure l'utilisation de gestes tactiles, de boutons

virtuels ou de joysticks virtuels, en veillant à ce qu'ils répondent de manière précise et fluide aux actions des joueurs.

Enfin, l'adaptation à différentes tailles d'écran est également un défi technique à prendre en compte. Les appareils mobiles ont des résolutions et des ratios d'aspect variés, ce qui nécessite de concevoir des jeux de manière à s'adapter automatiquement à différentes configurations d'écran. Cela peut impliquer l'utilisation de techniques de mise en page adaptative, d'ancrages flexibles et de redimensionnement dynamique des éléments graphiques pour assurer une expérience visuelle cohérente sur tous les appareils.

## **2.5 Bonnes pratiques en matière de programmation de jeux :**

Pour garantir un développement efficace et de haute qualité de notre jeu 2D RPG basé sur l'Afrique, il est essentiel de suivre certaines bonnes pratiques en matière de programmation de jeux mobiles. L'une de ces pratiques est la structuration du code. Il est recommandé d'organiser notre code en utilisant une architecture modulaire, en découplant le jeu en classes, fonctions et scripts bien définis. Cela facilite la lisibilité, la maintenance et l'extensibilité du code.

L'utilisation de patterns de conception est également une bonne pratique courante dans le développement de jeux. Nous utiliserons ici l'architecture MVC:

- Le pattern MVC (Modèle-Vue-Contrôleur) peut être utilisé pour séparer la logique de jeu, la présentation et l'interaction utilisateur. Cela permet de maintenir une structure claire et facilite la gestion des différents aspects du jeu.

Enfin, la collaboration et le partage de connaissances sont des pratiques précieuses. Même si nous travaillons seul, il est bénéfique de participer à des communautés en ligne, de suivre des tutoriels, de lire des documentations ou de regarder des vidéos Tutoriel pour acquérir de nouvelles compétences et rester à jour avec les dernières avancées dans le domaine de la programmation de jeux mobiles. Cela nous permettra d'améliorer notre expertise et de bénéficier des idées et des conseils d'autres développeurs.

---

*Deuxième partie : Analyse des jeux Mobiles sur  
l'Afrique*

---

## **Chapitre 3 – Les opportunités offertes par l'industrie des jeux mobiles**

### **3.1 Introduction**

Créer un jeu vidéo n'a jamais été aussi simple et accessible qu'aujourd'hui, que l'on soit développeur, designer... ou que l'on n'ait aucune compétence technique dans ces domaines. Aujourd'hui plus que jamais, nous pouvons saisir l'opportunité de devenir libre et indépendant grâce à la création de jeux vidéo, et plus particulièrement de jeux vidéo mobiles.

Depuis 2009, année durant laquelle apparaissent l'AppStore et le PlayStore (à l'origine appelé Android Store), le marché des jeux vidéo mobiles a progressé à très grande vitesse, et bien plus rapidement que d'autres industries du divertissement telles que le cinéma, la musique et les livres !

### **3.2 Perspectives de croissance de l'industrie des jeux mobiles**

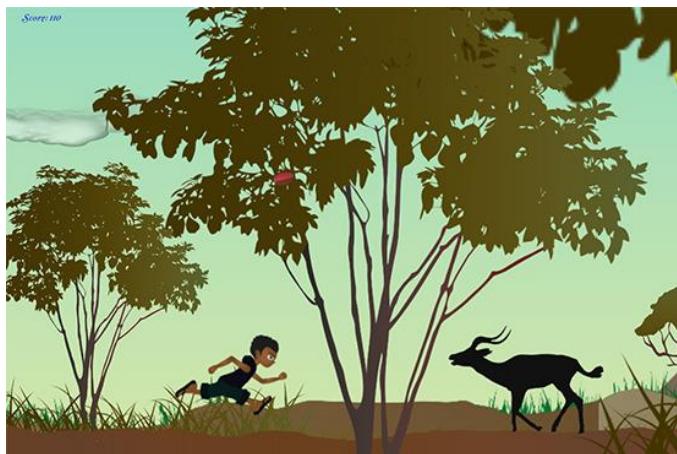
L'adoption du numérique continuera de s'accélérer dans les prochaines années. D'ailleurs, il existe aujourd'hui divers jeux qui sont de plus en plus accessibles via des appareils mobiles en Afrique. Et les jeux de bingo sont sans aucun doute les plus populaires auprès de la population africaine. Des sites comme [onlinebingocanada.co](http://onlinebingocanada.co) gagnent en popularité.

L'Afrique possède cinq des marchés mobiles les plus dynamiques et le début de l'accord de libre-échange conduira les gouvernements à identifier les moyens de faciliter la circulation et l'interaction des entreprises à travers le continent.

Des nations de l'éducation : Cela reste un point crucial, mais nous sommes susceptibles de voir une scission. Le premier est centré sur la poursuite de certaines des excellentes entreprises lancées en 2020. Le [nombre croissant de jeunes](#) conduira au déploiement de modèles éducatifs plus panafricains.

### **Voici une liste de divers jeux Africains :**

- The boy in the Savana :



**Figure 4 : Gameplay du jeu : The boy in the Savana**

Le 25 Décembre 2015 le studio Togolais LimPio propose un jeu mobile disponible sur le PlayStore où le joueur est soumis à des épreuves qui sont entre autres des animaux sauvages à éviter.

- Africa's Legend



**Figure 5 : Gameplay du jeu : Africa's Legend**

Avec Africa's Legends, sorti au Ghana au mois de novembre sur mobile, le studio voit encore plus loin. Pour ce puzzle game dans la veine de Bejeweled ou Candy Crush, les développeurs ont conçu plusieurs personnages typiquement africains. Les personnages

s'affrontent dans un univers mythologique. Il faut alors aligner sur l'écran différentes positions de combat pour triompher de ses adversaires.

- **Les Héros du Sahel**



**Figure 6 : Créateur du jeu : Les Héros du Sahel.**

Mahaman Sani Housseyn Issa est tombé dans le jeu vidéo quand il était tout petit. C'est « un mordu », comme il se plaît à le dire, en confiant qu'il tient son surnom, Mog, de l'univers du célèbre « Final Fantasy ».

Avec la sortie du jeu « Les Héros du Sahel », téléchargeable gratuitement sur la plateforme Mediafire, il a réalisé un rêve d'enfance. C'est donc avec le souci de parler aux joueurs nigériens que Mog donne vie à « Shamsou, le guerrier du soleil », personnage principal des « Héros du Sahel ». Shamsou porte une tunique aux couleurs du drapeau national, il est entouré de figures des contes traditionnels nigériens, il frappe ses ennemis avec l'énergie solaire, très importante pour l'avenir du Niger.

- Aurion : l'héritage des Kari-Odan**



Figure 7 : Gameplay du jeu : Aurion.

*Aurion: Legacy of the Kori-Odan* est un jeu vidéo de rôle d'action développé par le développeur camerounais Kiro'o Games. Le joueur contrôle Enzo Kori-Odan, le prince de Zama, qui est trahi par le frère de sa fiancée Erine. Basé sur la mythologie africaine, le couple visite d'autres pays pour demander de l'aide pour vaincre l'usurpateur.

### 3.3 Modèles économiques courants

#### Aperçu du marché :

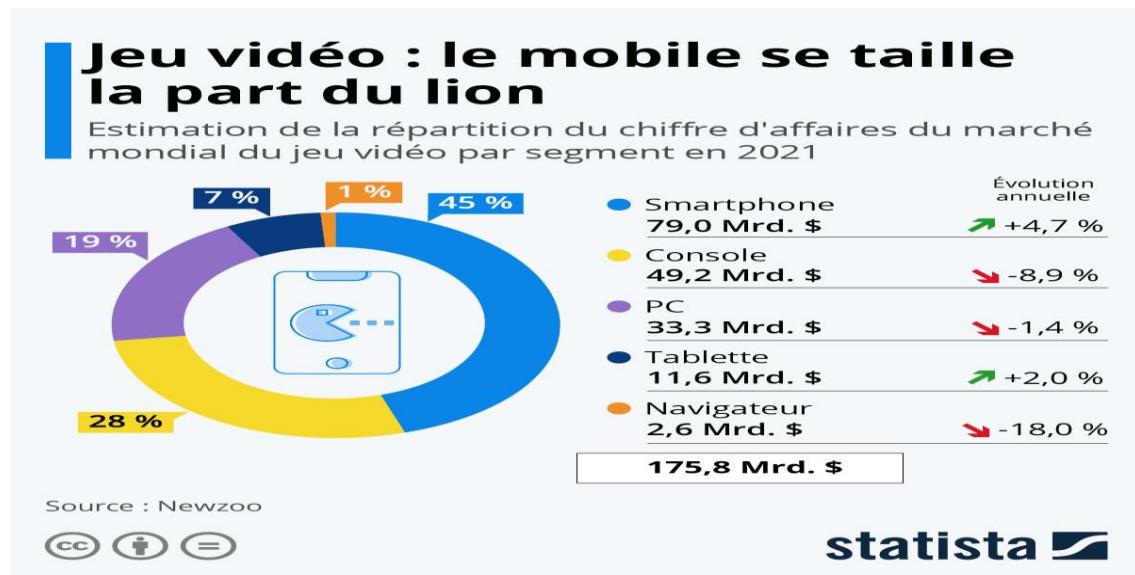


Figure 8 : Diagramme du chiffre d'affaires du marché mondial du jeu vidéo en 2021

L'industrie du jeu devrait enregistrer un TCAC de 12 % au cours de la période de prévision (2021 - 2026). La région africaine est l'une des régions du monde où la population de jeunes est en augmentation. D'ici 2050, les jeunes africains, c'est-à-dire ceux âgés de 0 à 24 ans, connaîtront une augmentation d'environ 50 %. L'Afrique devrait avoir le nombre le plus important de jeunes. La jeunesse africaine est essentielle pour l'avenir du continent en matière de développement de jeux.

Le marché des jeux vidéo en Afrique du Sud a connu une forte croissance ces dernières années. On s'attend à ce qu'il y ait plus de 11 millions de joueurs en Afrique du Sud. Les joueurs du pays se tournent vers les jeux sociaux, le modèle de revenus gratuit avec des achats en aval dans le jeu. Il est devenu de plus en plus populaire dans le pays.

En raison de ces facteurs, le marché devrait connaître une croissance au cours de la période de prévision. Des pays comme le Kenya, le Nigéria et l'Ouganda sont les autres principaux développeurs de jeux de la région africaine. L'industrie du jeu rapporte également des millions de dollars aux pays chaque année. C'est pour toutes les catégories de jeux vidéo : mobiles, PC, Xbox et PS.

### **Portée du rapport :**

L'industrie africaine du jeu est influencée par de nombreux facteurs, notamment le revenu disponible, la population jeune et les lois entourant les jeux de hasard en ligne. L'Afrique du Sud est l'un des plus grands marchés de jeux vidéo en Afrique. Le marché a dépassé le marché des films et de la musique dans le pays. Aux fins du présent rapport, le jeu est défini comme la pratique de jeux électroniques par le biais de divers moyens, tels que des ordinateurs, des téléphones portables, des consoles ou d'autres supports. Il y a une prévalence croissante des connexions Internet à haut débit, en particulier dans les économies émergentes, ce qui a rendu les jeux en ligne pratiques pour un plus grand nombre de personnes ces dernières années.

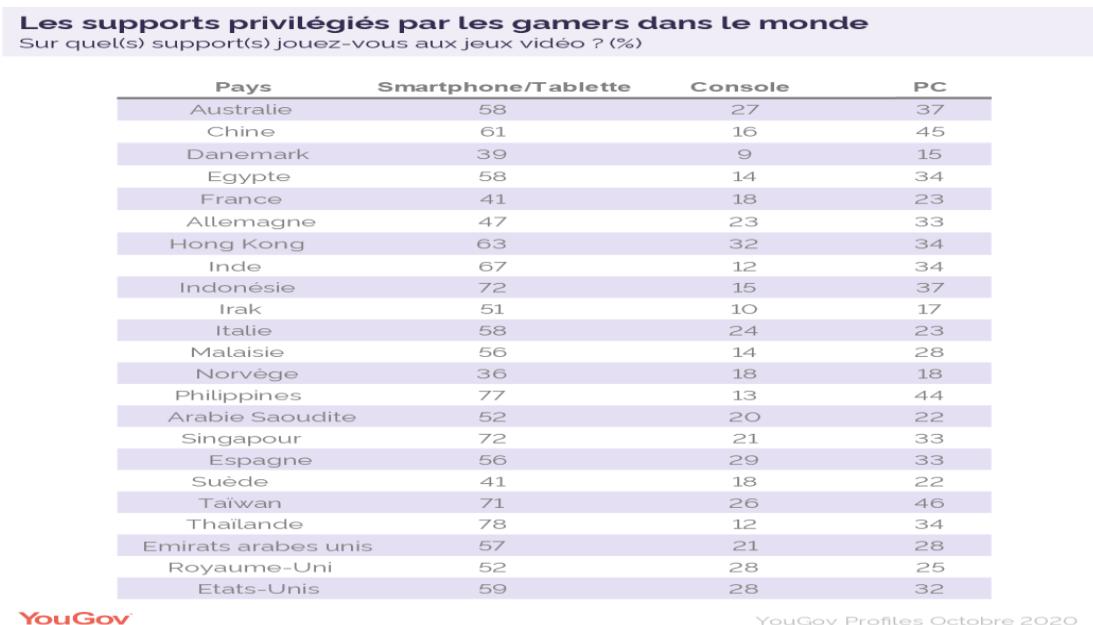
La portée de l'étude est actuellement centrée sur l'Afrique. L'étude suit les paramètres clés du marché, les influenceurs de croissance sous-jacents et les principaux fournisseurs opérant dans l'industrie. L'étude suit également l'impact de COVID-19 sur l'ensemble du marché du jeu en Afrique et ses performances. Des estimations détaillées pour les cinq prochaines années sont incluses, ainsi que l'étude des principaux acteurs et de leurs stratégies.

## **Chapitre 4 – Les défis de l'industrie**

#### **4.1 Introduction :**

En pleine croissance, l'industrie florissante du jeu vidéo mobile séduit aujourd'hui des millions de gamers à travers le monde. Le premier jeu sur mobile **Snake** de Nokia a laissé place à de nouveaux jeux plus complexes et disposant de plus de fonctionnalités tels que **Candy Crush Saga**, **Call of Duty** ou encore **Fortnite**.

Le dernier livre blanc YouGov « **Gaming & E-sport : la nouvelle ère** » explore le marché mondial du jeu vidéo en 2020 et offre un focus sur les profils des gamers du monde entier. Menée dans 24 pays, l'étude s'intéresse également aux joueurs de jeux vidéo sur mobile – smartphone ou tablette – qui représentent dans chacun de ces marchés la plus grande audience de gamers.



**Figure 9 : Tableau des supports de jeux privilégiés dans le monde.**

Partout dans le monde, les joueurs sur mobile sont bien plus nombreux que les joueurs sur PC et consoles. Ce constat est particulièrement visible en Asie du Sud-Est : 4 personnes sur 5 jouent aux jeux vidéo sur mobile en Thaïlande (78%) et aux Philippines (77%). En France, 41% jouent sur mobile, 23% sur PC et 18% sur console.

Le jeu vidéo mobile a atteint une audience mondiale massive grâce à sa facilité d'accès et sa portabilité. Cependant, derrière le succès apparent, plusieurs défis majeurs se posent :

**Modèle économique et monétisation :** L'un des défis clés concerne la monétisation des jeux mobiles. De nombreux jeux adoptent le modèle "freemium", où le jeu est gratuit, mais propose

des achats intégrés pour déverrouiller des fonctionnalités, des objets ou accélérer la progression. Trouver le bon équilibre entre des achats attrayants et une expérience équitable pour les joueurs sans dépenser beaucoup d'argent reste un défi.

**Qualité et diversité :** L'App Store et Google Play Store regorgent de milliers de jeux, allant des chefs-d'œuvre créatifs aux clones de basse qualité. Les développeurs doivent relever le défi de se démarquer dans ce marché saturé en offrant des expériences originales, de haute qualité et variées pour attirer et fidéliser les joueurs.

**Rétention et engagement des joueurs :** Les joueurs mobiles ont tendance à sauter d'un jeu à l'autre rapidement. Garder les joueurs engagés et intéressés sur le long terme est un défi majeur, nécessitant des mises à jour régulières, de nouvelles fonctionnalités et une compréhension approfondie du comportement des joueurs.

#### **4.2 Concurrence intense dans l'industrie des jeux mobiles**

Au cours des deux dernières années, le marché des jeux pour smartphones a connu un essor sans précédent. De nombreuses sociétés se sont positionnées sur ce secteur particulièrement lucratif, au point que l'on peut se demander aujourd'hui s'il y a encore de la place pour la concurrence sur ce marché.

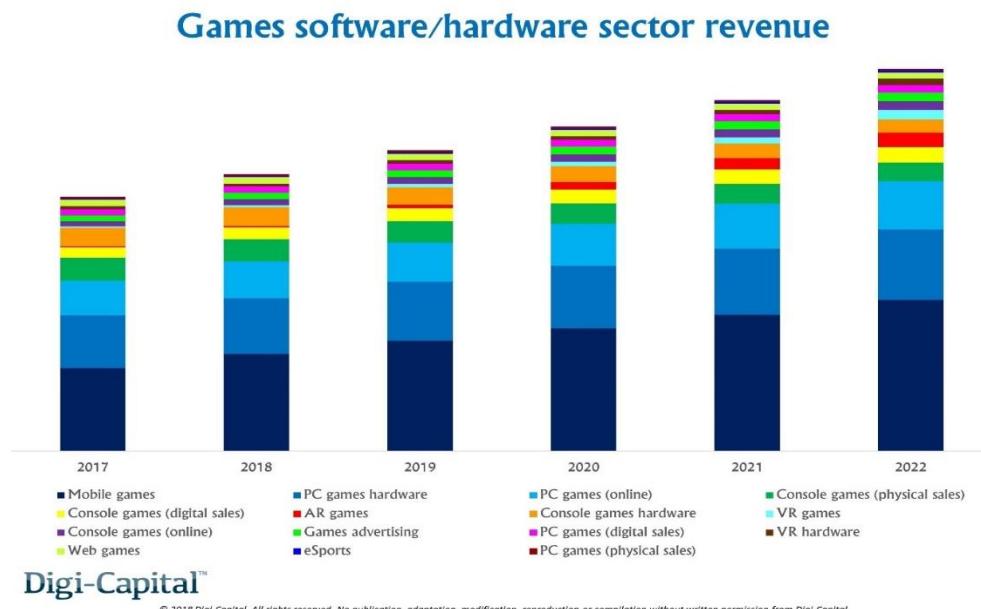
Certes, nous sommes moins confinés mais le jeu vidéo ne s'arrête pas de progresser. Le chiffre d'affaires du marché du gaming aurait progressé 1,4% en 2021, 180 milliards de dollars de recette.

Le jeu mobile aurait rapporté à lui seul 93 milliards de dollars de revenu, c'est 52% du marché du jeu vidéo. Avec une augmentation de 4,4% de ce segment de marché, il aurait dépassé celui du gaming sur consoles de salon et PC. Ce marché devrait en effet enregistrer un taux de croissance annuel de 11% entre 2019 et 2024. Le Google Play Store devrait générer des revenus de 28,2 milliards de dollars, tandis que les magasins d'applications Android tiers rapporteront 21,3 milliards de dollars.

- **Encore de la place pour la concurrence**

Tout le marché asiatique apparaît aujourd’hui bloqué dans ce domaine, théâtre de l’affrontement entre les géants des jeux pour téléphones mobiles. En Chine, Alibaba et Tencent écrasent la concurrence, tout comme le conglomérat Netmarble en Corée du Sud. La forte demande en jeux et la place moindre occupée par les grands noms du secteur donnent la possibilité à de petits éditeurs de tirer leur épingle du jeu. En pleine croissance, le domaine du jeu mobile possède donc toujours un potentiel économique certain.

Selon le cabinet **Digi-Capital** qui a mené l’analyse, environ 75% de ce chiffre sera généré par les ventes de logiciels, ce qui englobe les ventes physiques, dématérialisées mais aussi tous les DLC et microtransactions. Dans les années à venir, ce sont donc le mobile et le PC qui seront les forces motrices pour l’industrie, et qui continueront à dominer le marché. À titre d’exemple, les performances du marché du jeu mobile sont telles qu’on estime que sa contribution au chiffre d’affaires global de l’industrie du jeu vidéo pourrait être de 60 milliards de dollars en 2018, et de 95 milliards en 2022.



**Figure 10 : Revenu annuel des plateformes de jeux vidéo.**

### 4.3 Monétisation équilibrée et éthique des jeux

À l'époque de Super Mario Bros., The Legend of Zelda et Donkey Kong, il n'y avait rien de plus simple : on payait pour s'approprier un jeu et on pouvait jouer jusqu'à ce qu'on perde l'intérêt... ou la cassette. Mais l'univers des jeux vidéo a subi une profonde transformation depuis l'arrivée d'Internet et des écrans, ce qui a permis aux concepteurs de jeux vidéo de revoir leurs stratégies commerciales.

- Petit lexique de la monétisation

La monétisation dans les jeux vidéo peut prendre plusieurs formes :

- **Microtransaction ou micropaiement** : achat de priviléges, comme progresser plus rapidement dans le jeu ou ajouter des attributs à son personnage, contre une petite somme (un ou deux dollars).
- **Loot boxe (coffre à butin)** : coffre payant contenant des récompenses aléatoires, qui peuvent bonifier l'expérience de jeu, comme améliorer les caractéristiques de son personnage, ou être complètement futiles, comme changer son aspect seulement.

Les modes de paiement en ligne sont variés : par carte de crédit, par carte prépayée (ex. : App Store, Google Play, PlayStation, Xbox) ou par compte bancaire (s'il est lié à un compte PayPal) par exemple.

- Le problème avec les *loot boxes*

Toutes les formes de monétisation peuvent devenir problématiques si elles entraînent des dépenses excessives. Elles peuvent également devenir une source de conflit majeure dans les familles lorsque l'ado utilise la carte de crédit de ses parents pour effectuer ce type de transactions.

Cela dit, le problème spécifique aux coffres à butin, c'est qu'ils s'apparentent beaucoup aux jeux de hasard et d'argent : dans les deux cas, on est appelé à investir dans quelque chose dont le résultat est laissé au hasard. Cela active le « circuit de la récompense » dans le cerveau et augmente la sécrétion de dopamine. Avec le temps, le joueur s'habitue à cette sensation de plaisir, ce qui l'amène à vouloir jouer plus souvent et plus longtemps pour la retrouver. Pour celui-ci, les jeux vidéo peuvent devenir une source de dépendance ainsi que de problèmes financiers.

- **Equilibrage de la monétisation des jeux**

Il faut savoir qu'aujourd'hui de plus en plus de jeux s'approprient gratuitement mais avec des Saisons Pass ou il faut souscrire à un pass de récompense en payant une somme assez raisonnable (entre 4.99 EUR. À 7.99 EUR.). Dans Clash Of Clans qui est un jeu gratuit de stratégie en ligne sur mobile, les joueurs peuvent souscrire à un Pass de Saison ou à chaque défi effectuer, le joueur aura des récompenses précieuses comme un skin pour le Roi des Barbares ou la Reines des Archers ainsi que d'autre héros du jeu, des boosts pour accélérer le temps d'amélioration des bâtiments, et même une réduction des prix que peuvent coûter les bâtiments à améliorer dans le jeu. Cela ne pénalise pas les joueurs qui ne payent pas le Saison Pass car les récompenses du Pass ont été réfléchis à ce qu'il n'y ait pas trop de différence entre les joueurs qui jouent gratuitement et ceux qui payent le Pass.

## **Chapitre 5 : Meilleures pratiques pour la conception et la programmation de jeux vidéo mobiles**

### **5.1 Introduction :**

Dans ce chapitre, nous allons explorer les différentes pratiques pour la conception et programmation d'un jeu mobile. Les appareils mobiles sont devenus des plateformes de jeu incontournables, offrant aux joueurs une opportunité unique de pouvoir s'immerger dans un univers virtuel ou qu'ils soient. Cependant, derrière chaque jeu addictif, se cache des heures de travail acharné, de créativité et d'ingéniosité. Dans cette quête pour créer des expériences de jeu engageantes et mémorables, il est essentiel de suivre les meilleures pratiques qui nous guideront vers le chemin du succès. De la conceptualisation du jeu à son optimisation pour les performances mobiles, en passant par la création d'une interface intuitive et l'élaboration de mécanismes de monétisation équilibrés, chaque étape du processus est cruciale pour façonner un jeu qui saura capturer l'attention et les cœurs des joueurs. En combinant l'art de la créativité avec la science de la programmation, nous serons en mesure de créer des expériences qui captivent et divertiront les joueurs du monde entier tout en leur enseignant une nouvelle diversité de jeu basé sur l'Afrique.

## 5.2 Les principaux éléments d'un jeu mobile réussi

Bien évidemment, tout commence par trouver une idée de jeu convaincante. Il nous faut définir un concept clair et original qui définit le genre, les mécanismes de jeu (Gameplay), le public cible et l'histoire. Les scénarios basés sur des énigmes faciles présentent assurément un certain intérêt. Mais n'y allons pas par quatre chemins, les casse-têtes de ce type sont facilement et rapidement résolus par un grand nombre de joueurs.

Une fois ce concept mis en place, le gameplay doit captiver les joueurs avec des mécaniques intuitives, mais offrant également des défis progressifs pour maintenir l'engagement. La simplicité est incontestablement l'un des secrets de la réussite des jeux les plus sollicités. Flappy Bird, dont le gameplay ne se limite qu'à la seule action de « taper sur l'écran », a connu un succès mondial. On peut également parler de Fruit Ninja. Même si les graphismes du jeu ne sont pas si simples, c'est le principe du jeu qui est extrêmement simple et facile à prendre en main.

Créer un bon gameplay est essentiel pour le succès d'un jeu mobile. Voici les étapes sur lesquelles nous allons nous baser pour notre jeu :

- **Mécaniques de jeu claires** : Définir dès le départ les mécaniques de jeu essentielles. S'assurer qu'elles sont faciles à comprendre, mais offrent suffisamment de profondeur pour maintenir l'intérêt des joueurs.
- **Objectifs et récompenses** : Établir des objectifs clairs pour les joueurs à atteindre. Les récompenses pour avoir accompli ces objectifs renforcent le sentiment de progression et d'accomplissement.
- **Réactivité et fluidité** : Les contrôles doivent être réactifs et intuitifs. Les joueurs doivent sentir qu'ils ont un contrôle total sur l'action du jeu.

Un élément des plus importants est une interface utilisateur (UI) intuitive. Nous devons rester sur de la simplicité, opter pour un design épuré et minimaliste, tout en évitant la surcharge visuelle pour que les éléments essentiels ressortent.

Pour ce faire nous allons nous baser sur l'interface de PUBG : Play Unknow Battle Ground pour le déplacement du joueur grâce à un joystick virtuel, ainsi que des boutons pour la carte du jeu et pour les actions du joueurs (Attaque, Dialogue, Interactions.).

Voici un aperçu du gameplay de PUBG :



**Figure 11 : Interface Utilisateur du jeu : PUBG mobile.**

Dans ce gameplay nous allons juste nous inspirer de la disposition du Joystick Virtuel, la carte et le bouton d'attaque qui représente une balle de fusil à droite, ainsi que le menu pause représenté par un icône Paramètre qui est situé juste à côté de la carte en haut à droite.

### 5.3 Considérations ergonomiques et de jouabilité du gameplay

Les considérations ergonomiques et de jouabilité jouent un rôle crucial dans la création d'une expérience de gameplay fluide et engageante pour les joueurs de jeux mobiles. Pour offrir une expérience optimale, les contrôles doivent être intuitifs et réactifs, facilitant la prise en main sans entraîner de fatigue excessive.

Pour maintenir un sentiment de progression, nous introduirons de nouvelles mécaniques de manière progressive, permettant aux joueurs d'apprendre et de s'améliorer naturellement. Un équilibrage attentif est essentiel pour éviter des pics de difficulté frustrants, tout en offrant un défi suffisant pour maintenir l'intérêt.

Pour offrir une meilleure ergonomie et une jouabilité optimale, il est important de prendre en compte deux points tel que :

- **Script bien structuré**

Les scripts programmer jouent un rôle crucial dans la création d'une expérience ergonomique et de jouabilité optimale. Il est donc important de travailler avec un langage de programmation orientée objet, dans notre cas ce sera le langages C#.

Un code bien structuré est essentiel pour le développement de jeux, car il améliore la lisibilité, la maintenabilité et les performances du jeu. Il est donc important de connaître les bonnes pratiques pour obtenir un code bien structuré en divisant les codes classes et fonctions. Chaque classe aura une responsabilité claire et définie en encapsulant notre code par les fonctions et en utilisant une architecture MVC (Modèle-Vue-Contrôleur).

- **Interface Utilisateur (UI)**

Pour offrir une expérience de jeu optimale et agréable, il est essentiel de mettre en place une interface utilisateur (UI) soigneusement conçue.

Tout d'abord, la disposition et l'organisation des éléments sur l'écran jouent un rôle crucial. Une disposition intuitive et logique permet aux joueurs de naviguer facilement dans l'interface sans confusion. Les éléments essentiels doivent être placés de manière stratégique et mis en évidence de manière à attirer l'attention.

L'utilisation d'icônes et de symboles universellement reconnus simplifie la compréhension des actions et des informations. Cela permet aux joueurs de réagir rapidement et efficacement à l'interface.

Enfin, si le jeu est destiné à être joué sur différents appareils mobiles, il est crucial d'assurer une adaptabilité optimale de l'interface tout en maintenant une expérience cohérente pour les joueurs, indépendamment du dispositif utilisé et en utilisant des composants responsifs.

---

*Troisième partie : Réalisation du projet*

---

## **Chapitre 6 – Mise en place de l'environnement de développement pour une plateforme mobile**

### **6.1 Introduction**

Dans le monde contemporain du jeu vidéo, les plateformes mobiles ont acquis une position prédominante en offrant aux joueurs un accès facile à des expériences interactives où qu'ils soient. Le développement de jeux pour ces dispositifs requiert une approche spécifique, tenant compte des particularités des écrans tactiles, de la variété des configurations matérielles et des contraintes de performances inhérentes aux appareils mobiles. Ce chapitre se consacre à la mise en place de l'environnement de développement nécessaire à la création d'un jeu vidéo optimisé pour une plateforme mobile.

L'objectif fondamental de ce chapitre est de présenter en détail les étapes cruciales liées à la configuration d'un environnement de développement adapté aux spécificités des appareils mobiles. Nous aborderons les outils logiciels essentiels, les langages de programmation couramment utilisés et les techniques de conception optimisées pour garantir une expérience fluide et captivante aux utilisateurs de ces dispositifs.

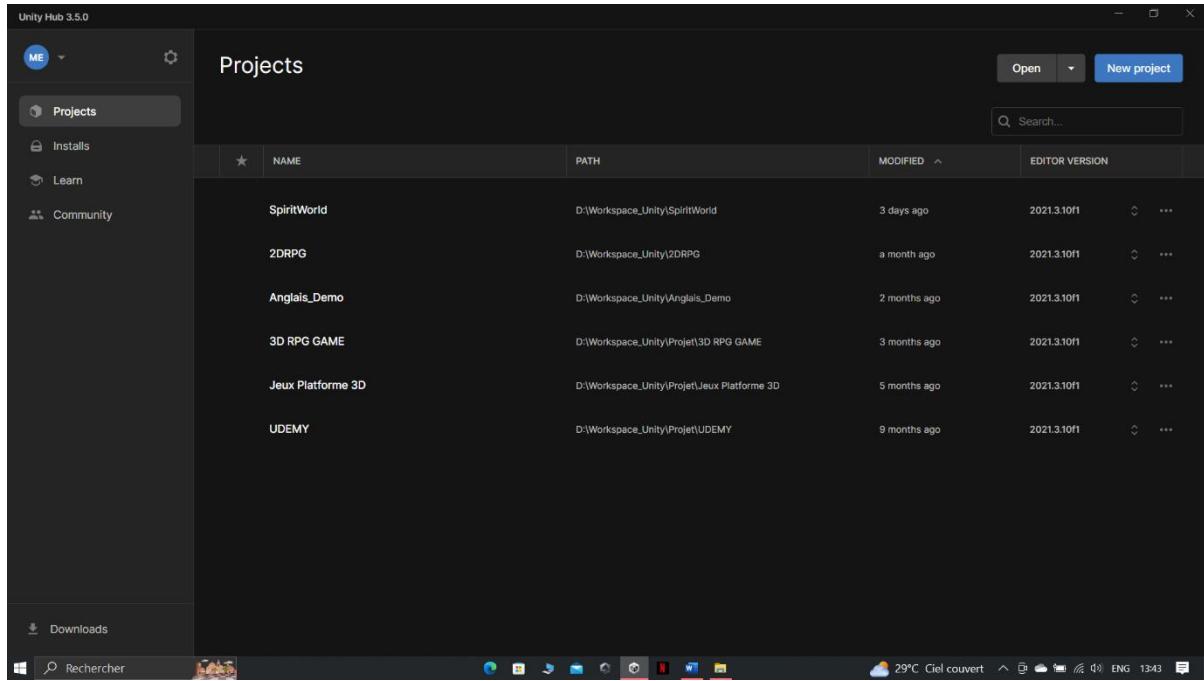
### **6.2 Installation et configuration des outils de développement**

#### **• UNITY**

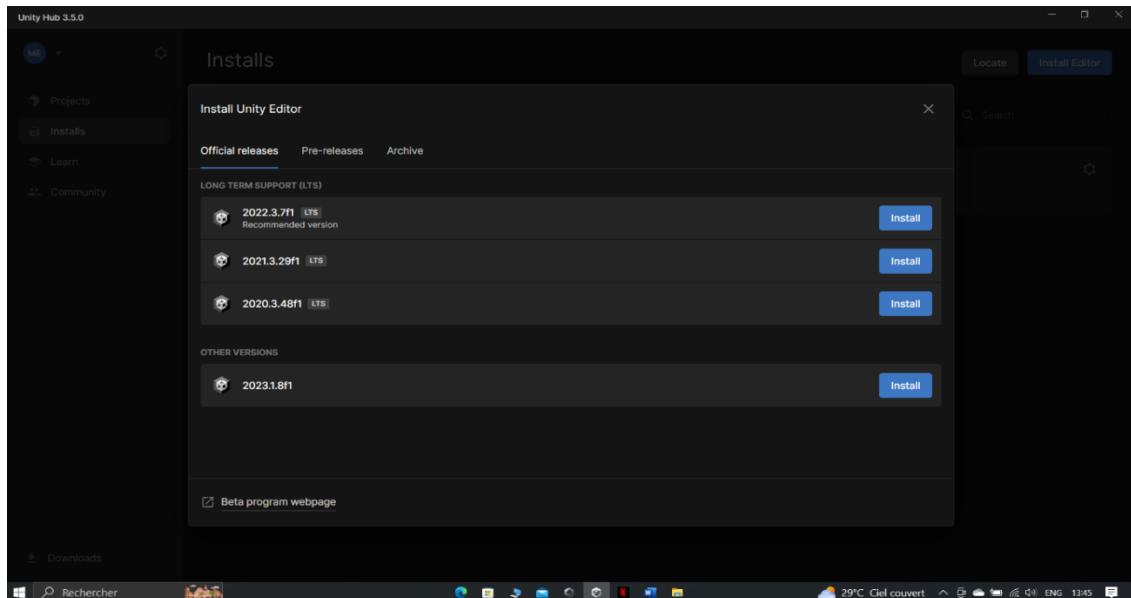
Pour la création de notre jeu vidéo, le premier logiciel à télécharger et mettre en place est UNITY, une plateforme de développement de jeux vidéo populaire et très puissant pour tout type d'appareil. Il permet aux développeurs de créer des jeux en 2D, 3D, en réalité virtuelle (VR) et en réalité augmentée (AR) pour une variété de plateformes, notamment PC, consoles, mobiles et le web. Unity offre un large éventail d'outils et de fonctionnalités pour la conception, la programmation, l'animation, les effets visuels, le son et bien plus encore.

Voici le lien : [unity.com/download](http://unity.com/download).

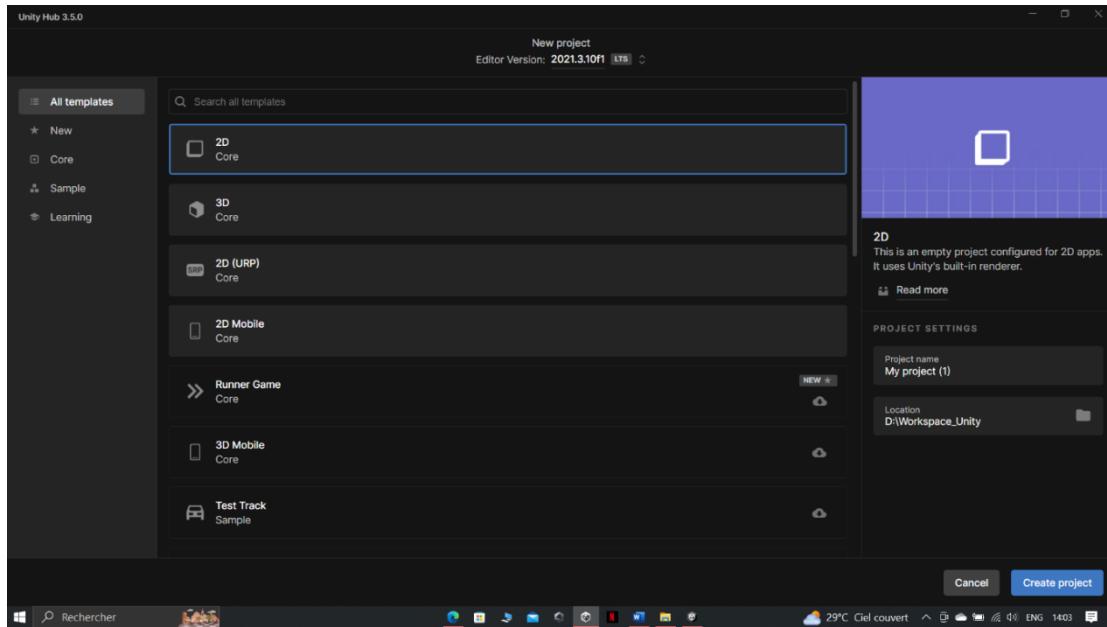
Après l'avoir téléchargé, Unity nous offre le Unity Hub, une application pour répertorier tous nos projets et facilite le démarrage de chaque projet.

**Figure 12 : Interface du logiciel UNITY Hub**

Unity Hub nous offre aussi la possibilité d'installer différentes versions de Unity et d'utiliser chaque version en même temps. En général, une version plus ou moins antérieure serait plus coûteuse qu'une version récente car les développeurs n'y sont pas encore au point sur les dernières versions, il faut toujours attendre quelque mois pour télécharger une version récente après sa sortie. Dans notre cas, nous utiliserons la version 2021.3.10f1 (LTS) qui est déjà installée mais d'autres versions peuvent toujours être installées comme ci-dessous :

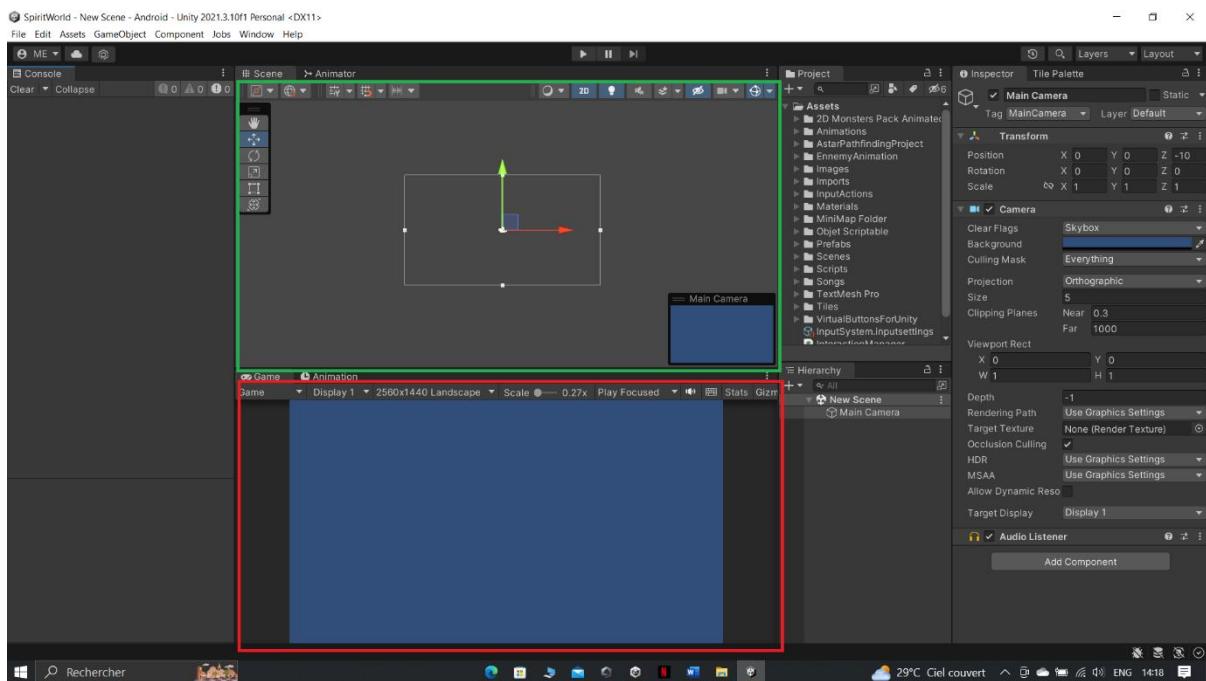
**Figure 13 : Installation d'une version de UNITY**

A présent, il est temps de créer notre projet et de l'ouvrir. Pour notre cas nous voulons créer un nouveau projet en 2D :



**Figure 14 : Crédit d'un projet 2D.**

Nous donnerons comme nom ‘SpiritWorld’ pour notre projet et nous choisissons le modèle 2D et c'est tout. Il nous suffit d'appuyer sur ‘Create Project’ et il va nous charger notre scène en 2D :

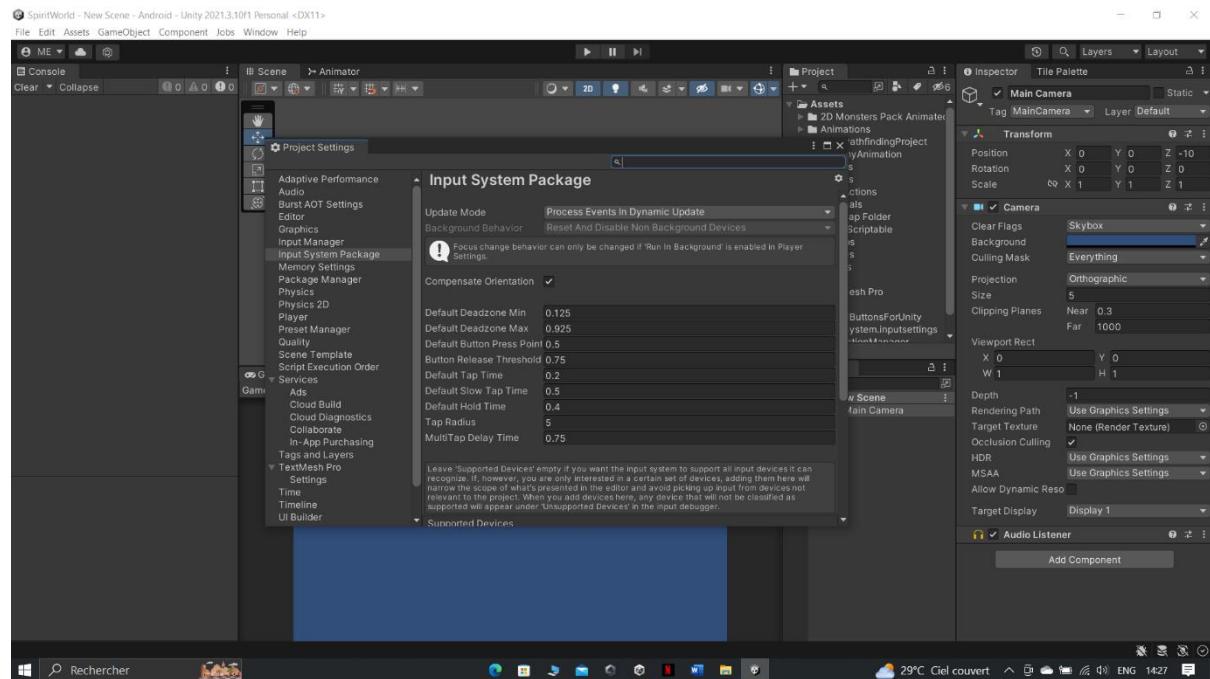


**Figure 15 : Interface d'un nouveau projet UNITY**

Et voilà notre scène pour la création de notre jeu RPG (Role Playing Game). Comme nous le voyons, nous avons deux fenêtres, celle en vert c'est la scène ou nous allons travailler, et celle en rouge c'est ce que la caméra voit autrement dit, le rendu de ce que la caméra détecte dans la scène.

Pour avoir un travail propre et bien structuré, nous créerons plusieurs dossiers dans la section ‘Project’ qui nous aideront à répertorier les scripts, les animations, images (Sprites), audios ainsi que les prefabs (objet préfabriqué).

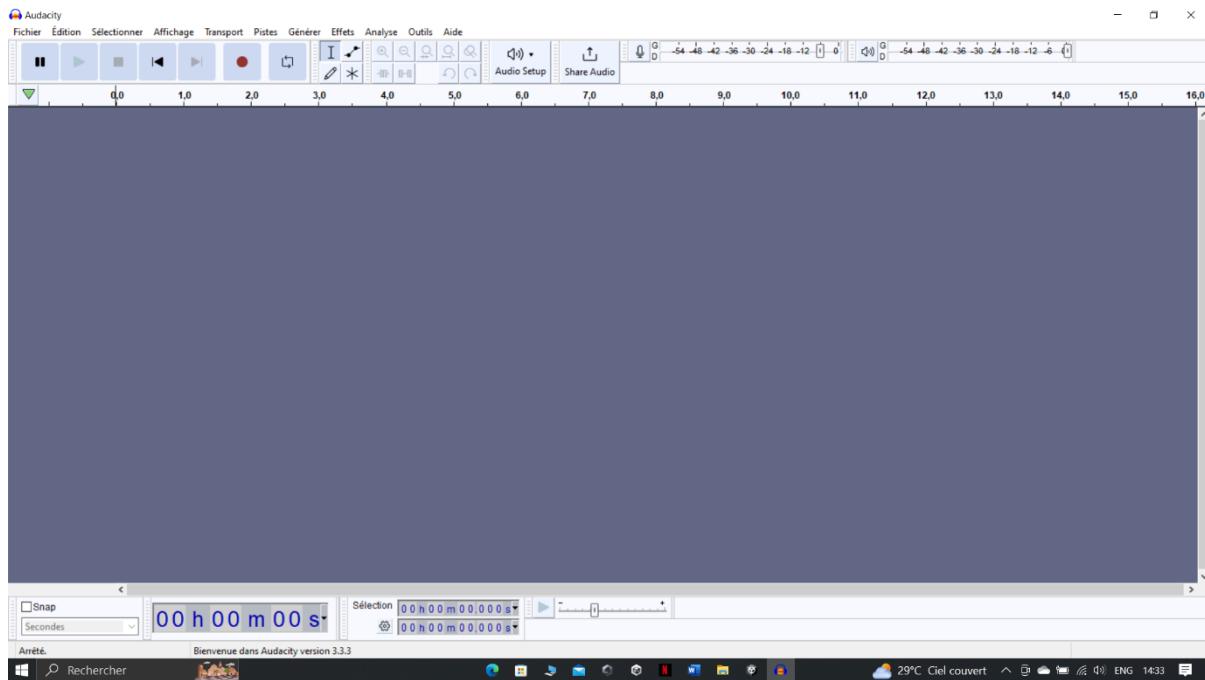
Un dernier point pour finaliser la configuration d’Unity est le nouveau système d’Input de Unity qui nous facilitera énormément le codage pour les déplacements avec l’écran tactile. Par défaut il n’est pas installé, il nous faudra donc le faire manuellement en allant sur Edit, puis Project Settings puis installer le new Input System :



**Figure 16 : Installation du New Input System**

- **AUDACITY**

Maintenant que Unity est prêt, il nous faut installer d’autres outils essentiels pour le développement de notre jeu. Nous allons maintenant télécharger Audacity, un logiciel d’enregistrement et de modification audio pour les bruitages du jeu.



**Figure 17 : Interface de l'éditeur audio : Audacity**

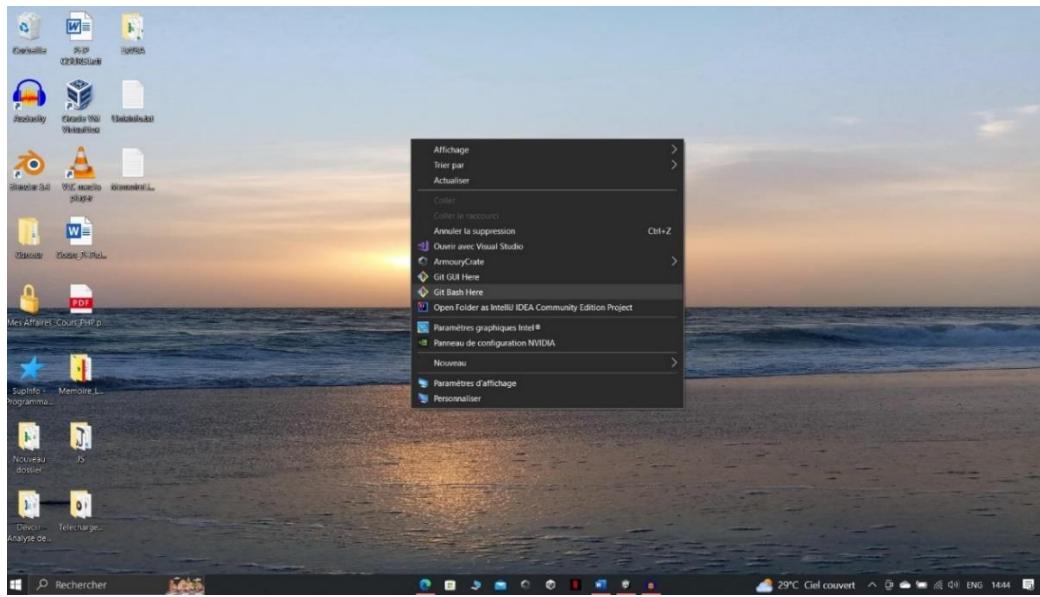
Avec ce logiciel, nous pouvons créer des bruits en les enregistrant et nous pouvons aussi importer des audios pour les modifier par la suite pour faire des bruits de monstre ou pour améliorer la qualité des effets sonores ou pour la musique de fond.

- **GIT & Visual Studio Code**

**Git** est un système de contrôle de version distribué largement utilisé pour le suivi des modifications apportées à des projets informatiques tels que le code source de logiciels ou les scripts de jeux.

**Visual Studio Code** est sans aucun doute l'un des meilleurs environnements de développement intégré (IDE) et il est largement utilisé pour la programmation et le développement de logiciels. Il est également bien intégré avec Git, ce qui en fait un choix populaire pour le développement de jeux Unity.

Pour avoir Git il nous suffit de l'installer à partir du lien suivant : [Git - Downloading Package \(git-scm.com\)](https://git-scm.com/). Après installation de l'exécutable, nous pourrons utiliser GIT avec 'GIT Bash' en allant sur le bureau du PC, faire un clic droit et ouvrir 'Git Bash'.



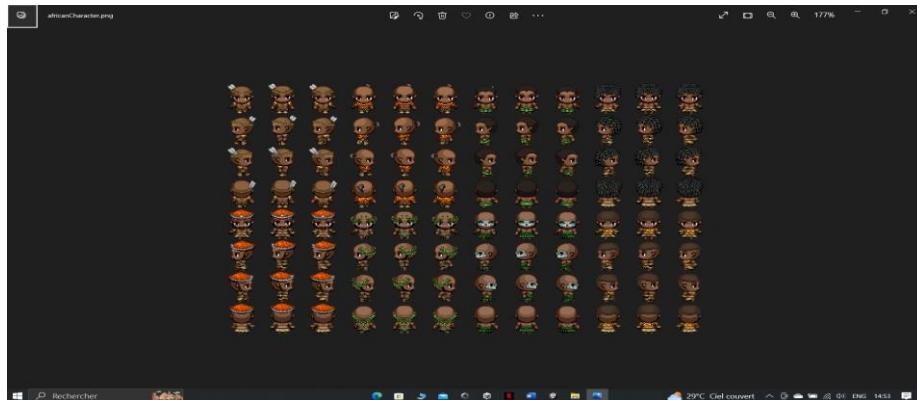
**Figure 18 : Ouverture du GIT Bash après l'installation de Git**

Ainsi, tout est prêt pour que développer notre jeu 2D. En ayant tous les outils à disposition, il nous sera plus facile de commencer sans problèmes.

### **6.3 Importation des Sprites pour l'environnement 2D**

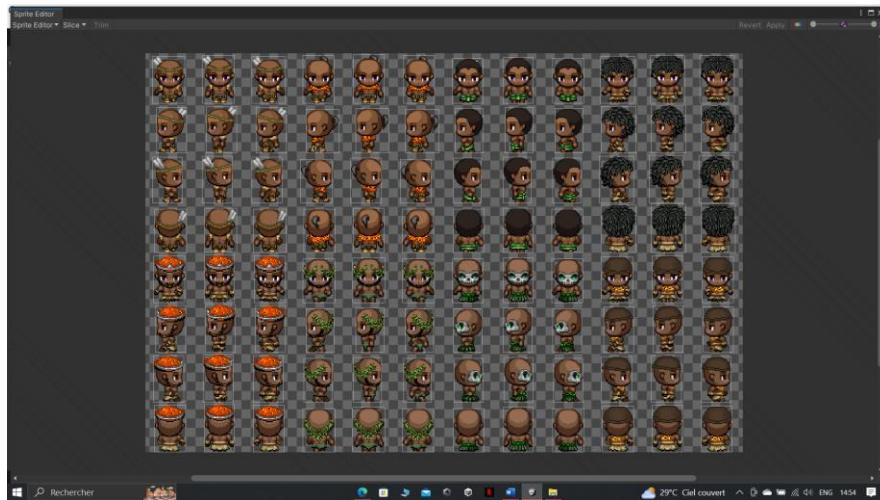
Après avoir préparé tous les outils nécessaires pour la conception du jeu, il nous faut maintenant des images ou des sprites. Les sprites sont le plus important pour développer le jeu car ils représentent les personnages avec les animations, les environnements (l'eau, la terre et tous les éléments décoratifs), les objets tels que les bâtiments, les objets, l'interface utilisateur ainsi que les effets spéciaux.

Pour ce faire nous allons télécharger des images avec des sprites sur Internet, puis nous les diviserons sur Unity grâce au ‘Sprite Editor’ pour en faire des objets et les animera.



**Figure 19 : Sprites de PNJ**

Voici une image de personnages PNJ (Personnage non jouable) qui serviront pour rendre le jeu plus sympa en discutant avec eux ou pour avoir des quêtes etc... Nous allons maintenant importer cette image sur UNITY pour découper chaque Sprite de chaque personnage.

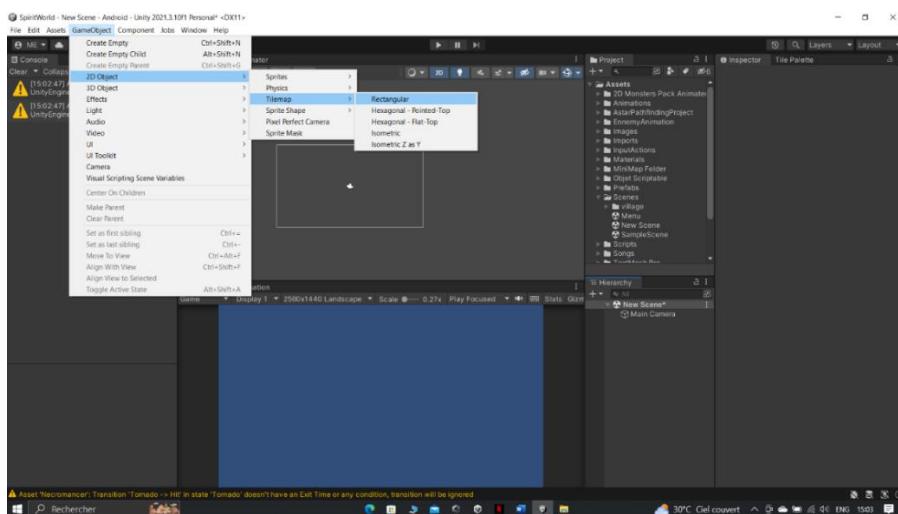


**Figure 20 : Découpe de chaque Sprite des PNJ.**

Nous répéterons la même chose pour tous les objets, les bâtiments, personnages, ennemis et tous les sprites que nous utiliserons.

Maintenant nous avons une liste de chaque Sprites de chaque personnage et nous pouvons glisser déposer un sprite sur la scène pour en faire un objet et travailler avec.

A présent il nous faut une grille qui nous servira de palette (TileMap) pour dessiner notre environnement avec des sprites que nous aurons téléchargée.



**Figure 21 : Crédit d'une TileMap Rectangulaire**

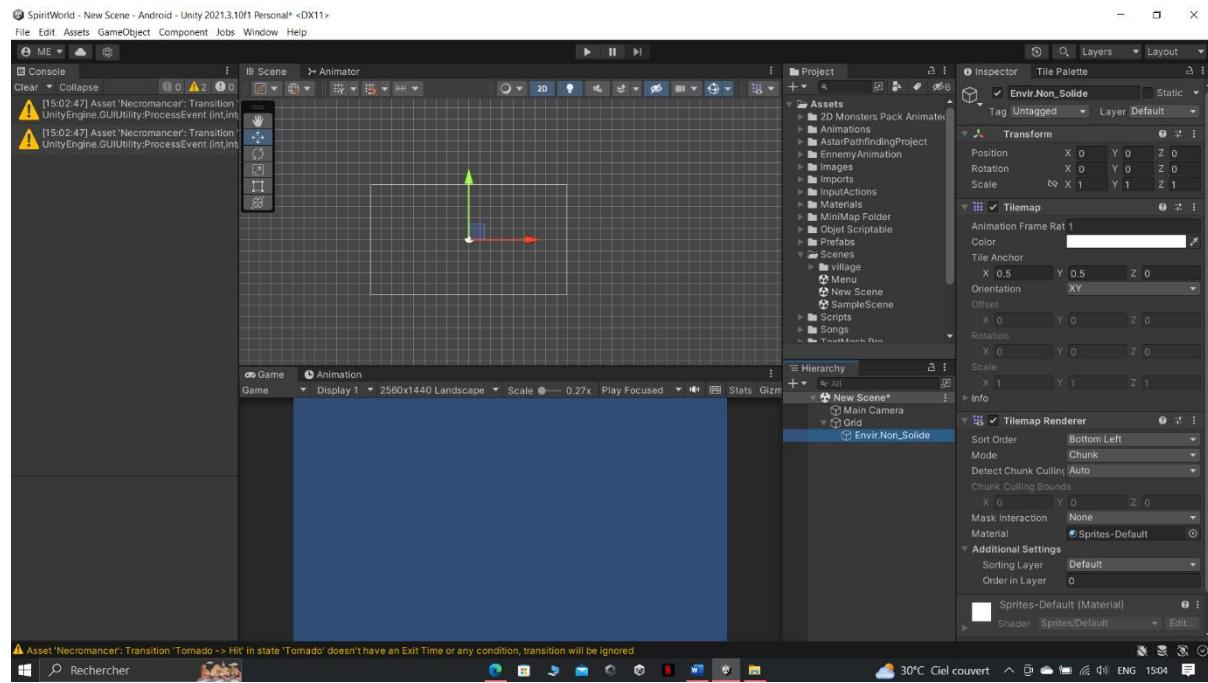


Figure 22 : Application de la TileMap

La scène contient plusieurs cases qui nous serviront à dessiner notre environnement non solide.

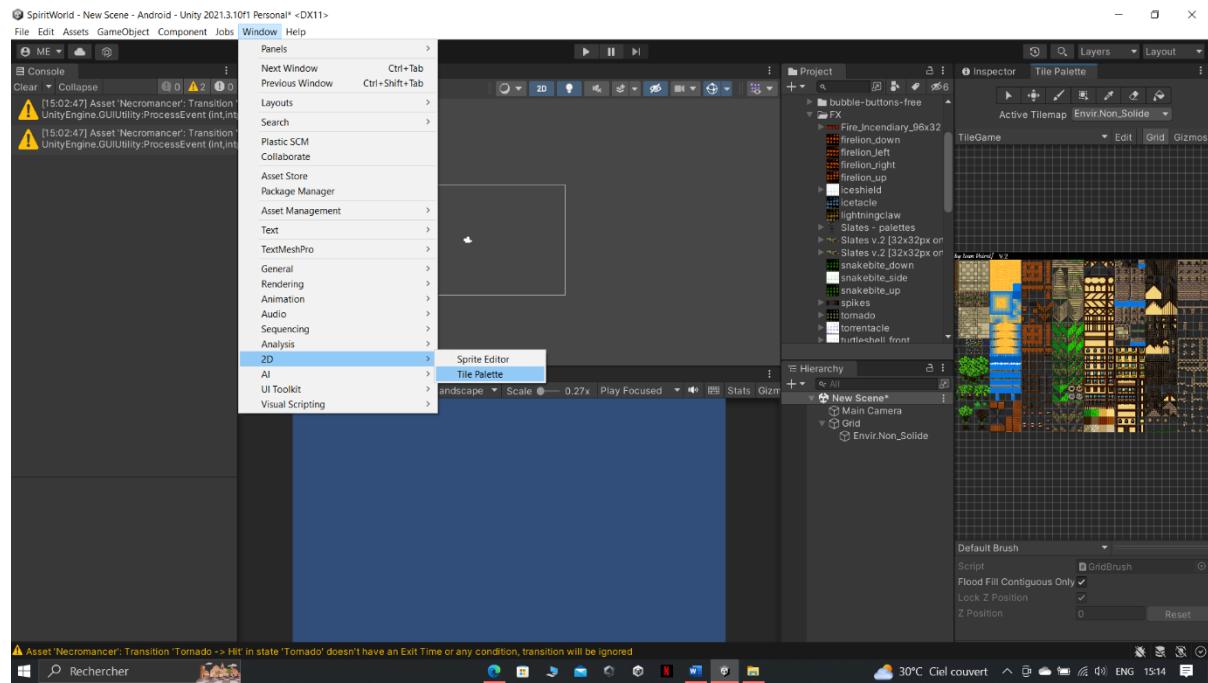
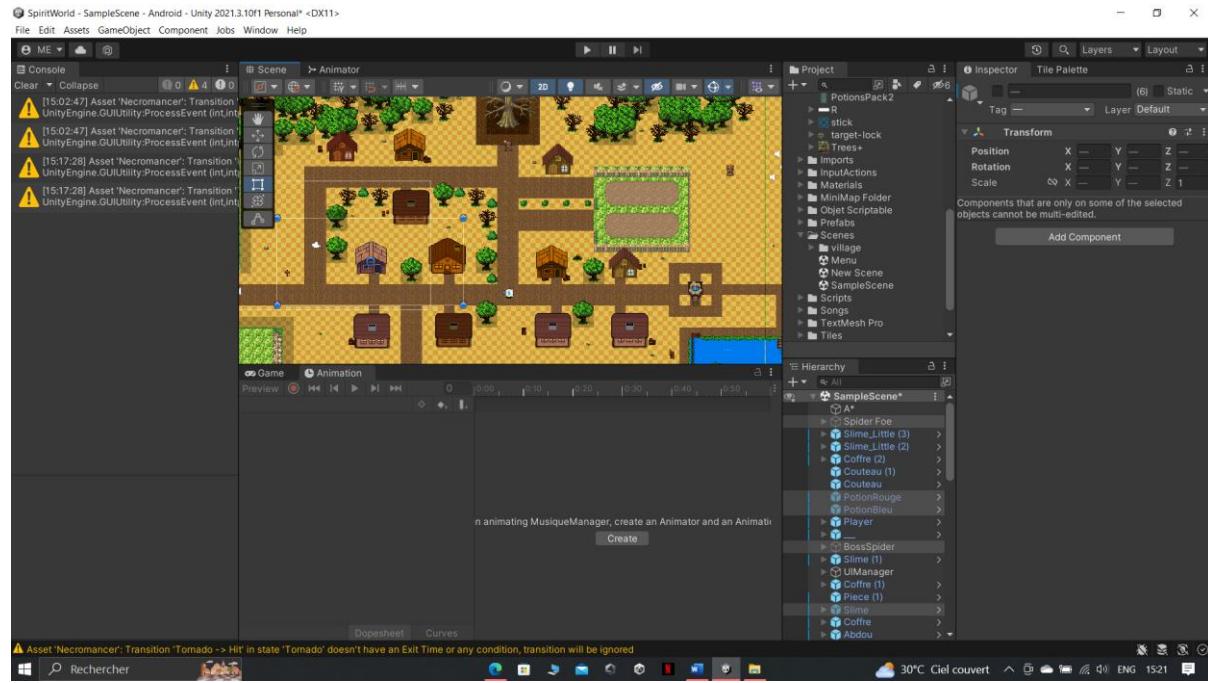


Figure 23 : Crédation d'une TilePalette

Maintenant que nous avons configurer nos sprites pour l'environnement, nous allons créer une Tile Palette (Palette de tuile) dans laquelle nous allons glisser notre environnement configurer et enfin il ne nous suffira qu'à choisir une case et de dessiner sur la scène grâce à la grille que nous avons créé précédemment.



**Figure 24 : Conception d'un village avec la TilePalette**

Et voilà, nous avons créé un village avec des bâtiments que nous avons déjà configurer des potager, un lac etc...

Un dernier point à effectuer est la configuration des éléments UI pour les boutons, le Joystick pour le déplacement du joueur, la carte du jeu, le bouton du menu etc...

Pour ce faire on fait la même chose pour la configuration des sprites et grâce à UNITY nous pouvons créer des boutons pour l'attaque par exemple :

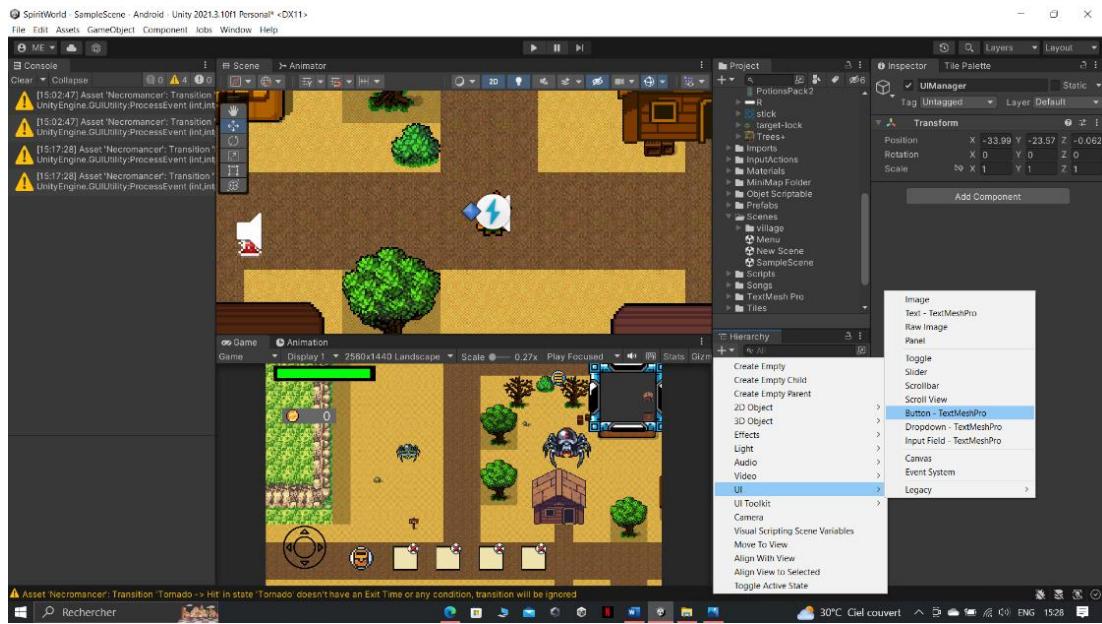


Figure 25 : Crédit du bouton d'attaque

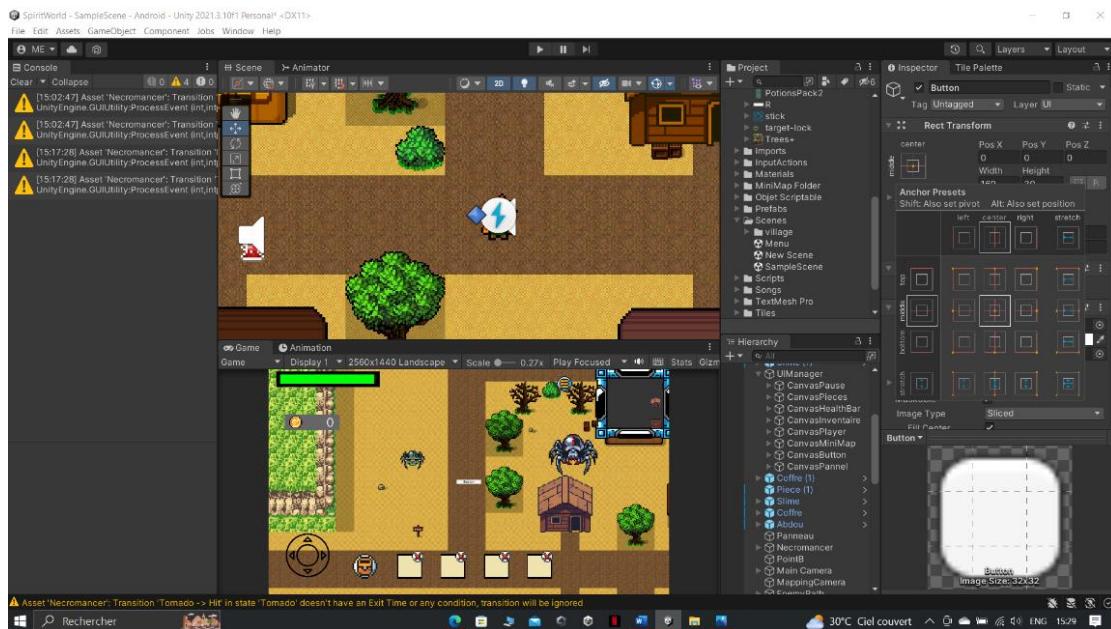
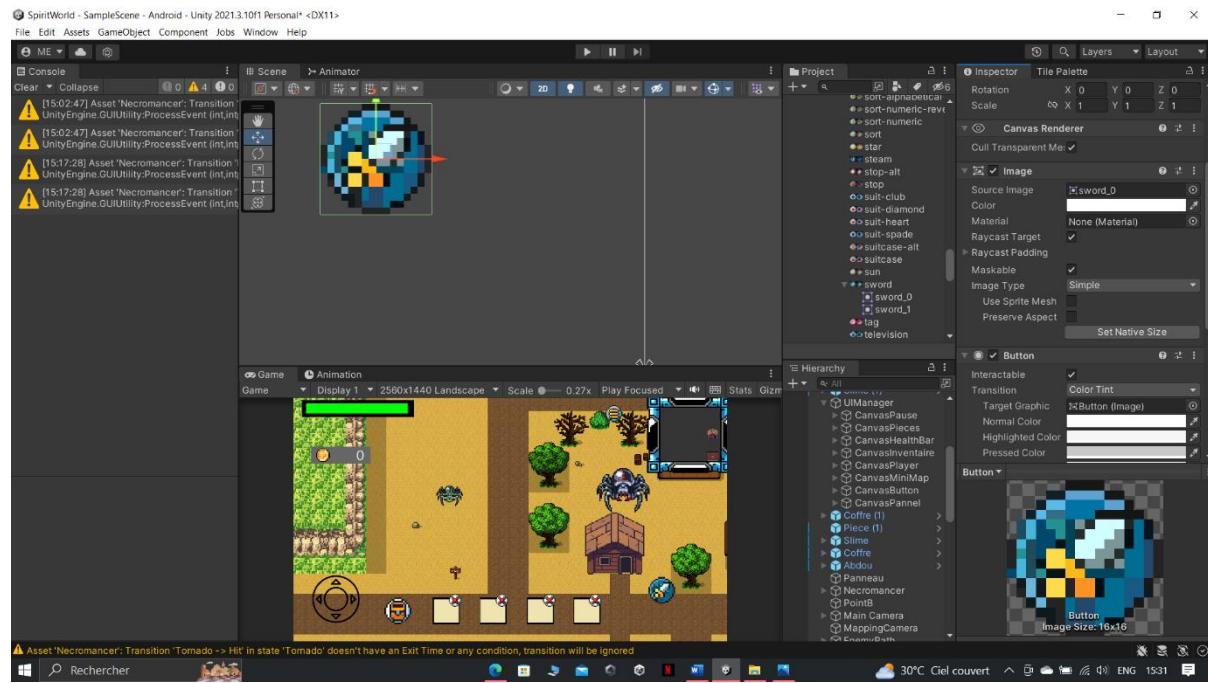


Figure 26 : Positionnement Responsif du bouton d'attaque avec 'Anchor Preset'



**Figure 27 : Résultat de la position et de l'image source du bouton d'attaque**

Nous avons ici créé un bouton, ensuite nous l'avons positionné responsivement et enfin nous avons remplacé l'image Source par notre Sprite Sword pixélisée. C'est exactement la même procédure pour les images tels que la barre de vie, la carte du jeu (Map), l'inventaire, les pièces à récolter et tout élément UI.

## 6.4 Architecture du jeu

La conception d'un jeu ou tout type d'application nécessite de réfléchir à une architecture responsive de façon à ce que le jeu puisse s'exécuter sur tout type d'appareil mobile de toute les tailles d'écrans, et a une architecture durable pour notre environnement. Il est donc important de connaître les bonnes pratiques pour faire en sorte que le jeu puisse avoir une clarté au niveau du gameplay et que les éléments UI puissent s'adapter à n'importe quelle taille d'écran.

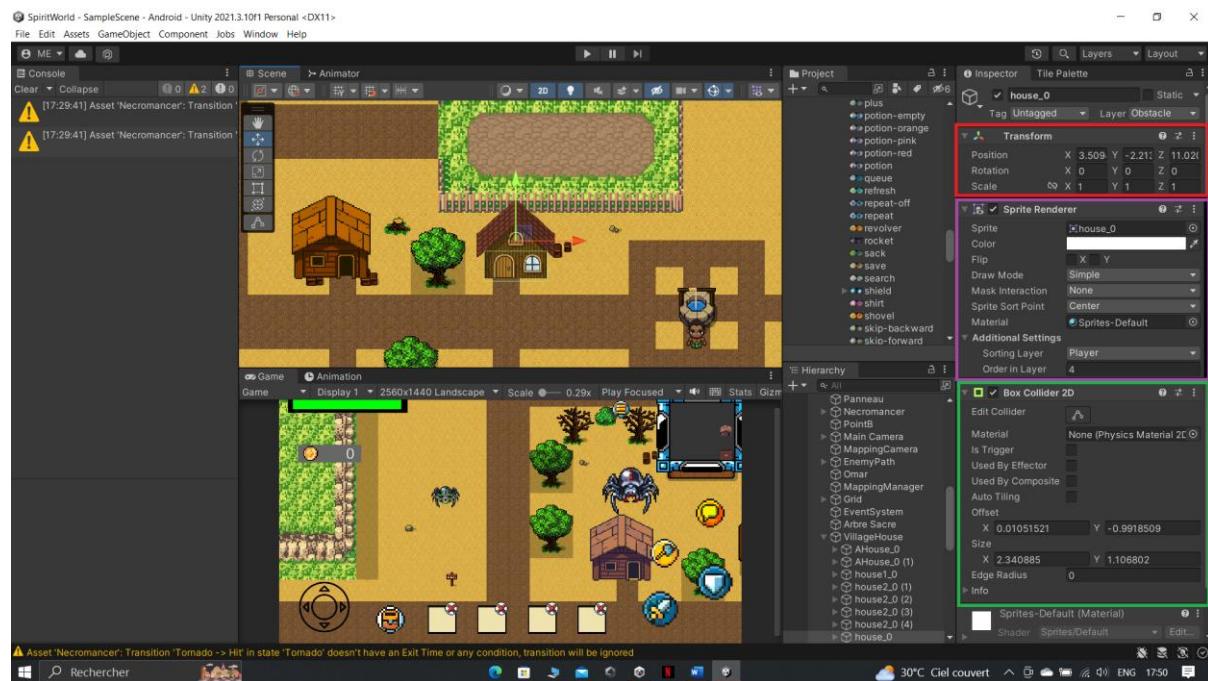
Dans le cas de notre jeu, tous les éléments de l'interface utilisateur seront positionnée grâce au ‘Rect Transform’, un composant qui est utilisé sur les éléments de l'interface utilisateur tels que les images, boutons, texte etc...

Il faut savoir que les objets utilisent ‘Transform’ pour se positionner dans la scène. Nous allons voir dans un premier temps la différence entre la position des objets et des éléments UI pour une architecture responsive et adaptée pour toute les tailles d'écrans, puis enfin nous verrons

comment créer une architecture durable pour l'environnement du jeu afin d'avoir un rendu clair et net.

- **Architecture Responsive**

Tous les objets et éléments UI disposent d'un composant de positionnement dans la scène. Prenons le cas d'une maison qui est un objet dans notre scène et découvrons ses composants :



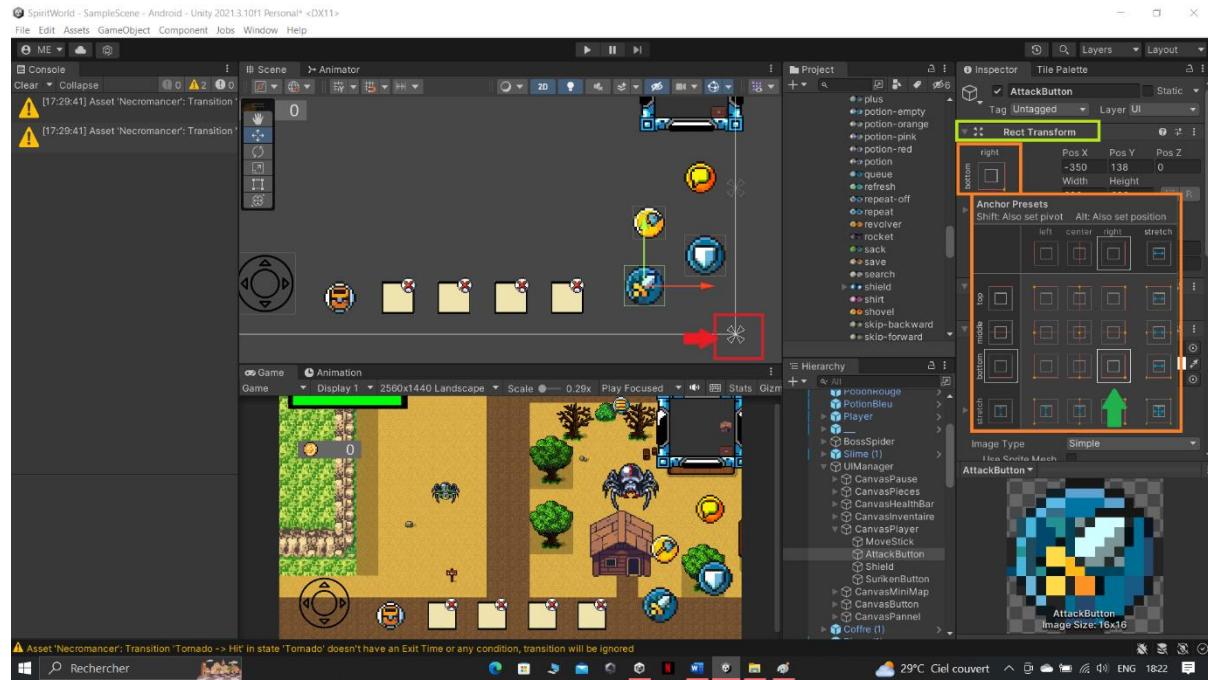
**Figure 28 : Composants d'un objet maison**

La partie en rouge et en violet sont les composants Transform et Sprite Renderer. C'est deux composants sont créés par défaut sur tous les objets de la scène. Transform est le composant qui nous servira à positionner, orienter ou redimensionner les objets. Sprite Renderer est le composant qui sert à afficher l'image de l'objet dans la scène et d'agir sur le Sprite, la couleur pour qu'il soit visible dans la scène.

La case en vert représente le composant 'Box Collider 2D' que nous avons ajouté pour que l'objet ait une collision 2D avec le joueur ou avec des projectiles et des ennemis.

Avec le Transform, nous n'avons pas à nous soucier de la responsivité car elle est automatique, c'est-à-dire que la position s'adapte sur tous les écrans.

Passons aux éléments UI qui doivent être responsive manuellement avec le Rect Transform. Nous allons nous baser sur le bouton d'attaque pour mieux comprendre comment faire en sorte que sa position soit toujours la même sur toute les taille d'écran :



**Figure 29 : Composants d'un élément UI.**

Avec le Rect Transform, les éléments s'affichent par défaut au milieu de la zone UI qui est différentes de celle de la scène car les éléments UI ne sont pas dans la scène mais dans notre interface utilisateur (Canvas) autrement dit, l'écran de notre téléphone ou tablette. Nous allons établir une disposition responsive à l'aide de ‘Anchor Preset’ qui est la petite case orange, elle représente l'ancrage qui permet de spécifier rapidement comment un élément doit être ancré dans l'interface utilisateur. En cliquant dessus on a accès au menu de l'Anchor Preset, il nous permet de positionner exactement l'élément sur tous les appareils mobile ou tablette en appuyant sur SHIFT + ALT puis on choisit la position souhaitée. Cette pratique est essentielle pour concevoir un jeu adapté pour tous les appareils de toutes les tailles d'écrans.

- **Architecture Durable ou Environnementale**

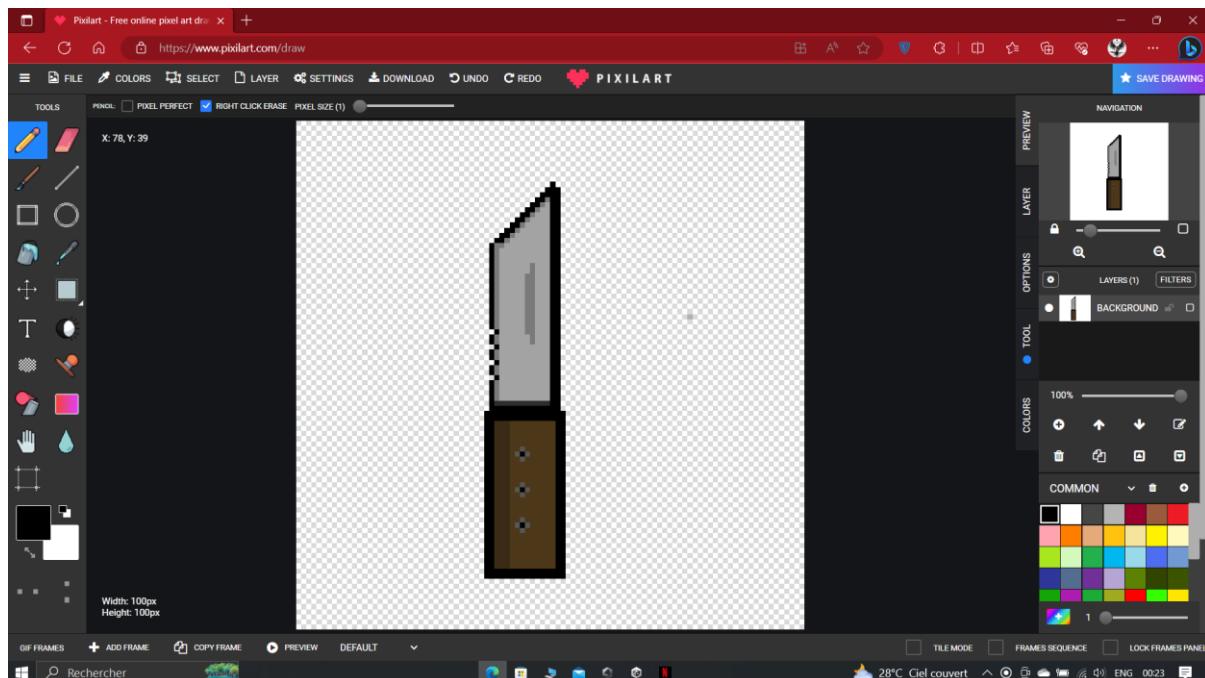
Lorsque l'on parle "d'architecture environnementale" pour un jeu, on fait généralement référence à la conception et à l'organisation d'environnements et de paysages de manière à ce qu'ils soient visuellement cohérents, fonctionnels et qu'ils soutiennent la narration et le gameplay du jeu. Sur Unity, la mise en place d'une telle architecture se décompose en plusieurs étapes.

### Planification & Conception :

- Narration et Gameplay : L'environnement devrait aider à raconter l'histoire du jeu. La conception initiale doit donc tenir compte des éléments narratifs et des exigences de gameplay.
- Esquisses et Croquis : Commencer par esquisser les grands traits de l'environnement, en envisageant différents points de vue, zones d'intérêt et obstacles potentiels.

### Création d'Assets 2D :

- **Sprites** : Créer des sprites pour les éléments du décor, les personnages, les objets, ou pour le cas de notre jeu, nous allons les télécharger, nous pourrions cependant créer des objets faciles avec pixiart.com, un site de dessin en pixel :



**Figure 30 : Réalisation d'un couteau avec pixiart.com**

Comme nous le voyons, nous souhaitons que notre joueur puisse jeter des couteaux en les ramassant, pour cela il faut déjà avoir un couteau ! Sur pixiart.com, nous avons dessiné un couteau que nous allons importer sur UNITY sous un format .png pour une meilleure optimisation. Les fichiers .png sont très bien adapté pour la netteté de l'image sur UNITY.

## **Chapitre 7 – Développement du jeu mobile en 2D**

### **7.1 Introduction**

Après-en, nous avons étudié les différents principes de base pour la conception d'un jeu vidéo, nous avons configuré tous les outils nécessaires pour le développement de notre application et nous pouvons enfin débuter sur la conception.

Dans ce chapitre nous allons développer notre jeu RPG mobile africain. Nous commencerons par développer notre synopsis pour avoir un meilleur aperçu et une bonne architecture pour l'environnement du jeu et le système du Gameplay. Nous allons continuer en créant notre personnage principal et notre environnement de jeu. Nous poursuivrons par l'animation de l'environnement du jeu ainsi que des personnages (joueur, PNJ, ennemis). Pour finir nous créerons nos toutes les entités autonomes et enfin nous allons mettre en place tous les scripts nécessaires pour le développement du jeu en les créant et les programmant avec le langage C#.

#### **7.1.1 Synopsis**

Après avoir réfléchi à une idée de jeu RPG (Role-Playing-Game), il est nécessaire d'avoir un Synopsis pour travailler selon les conditions de l'histoire du jeu et offrir un maximiser le développement de l'environnement en s'appuyant des évènements de l'histoire.

- **Synopsis**

L'histoire du jeu se passe dans une région de la Casamance Tindaba. L'harmonie de l'écosystème est menacée lorsque l'arbre sacré, qui est la source vitale d'énergie pour tout le village, est corrompu par un esprit maléfique ancestral. Les forces maléfiques absorbent l'énergie vitale de l'arbre et répandent le chaos dans la région, invoquant ainsi des monstres qui s'en prennent aux habitants des villages voisins dans la forêt. Nous incarnons **Baldé Mendy**, un jeune fils de Saltigué, qui est choisi pour entreprendre une quête épique visant à sauver l'arbre sacré et rétablir l'équilibre de la région.

## 7.2 Création du personnage et de l'environnement RPG 2D

Etant dans le domaine de la programmation informatique, nous allons télécharger les sprites d'un personnage 2D qui se déplace en 8 directions :

### Déplacements :

- Bas & haut : 
- Gauche & droite : 
- Bas & haut à droite : 
- Bas & haut à gauche : 

### Attaques :

- Bas & haut à gauche : 
- Bas & haut à droite : 

### Effet de l'épée d'attaque :

- Bas & haut à gauche : 
- Bas & haut à droite : 

Comme nous l'avions vue précédemment lors de la configuration des PNJ (page 51) nous allons diviser chaque sprite du joueur grâce au Sprite Editor pour créer des animations :

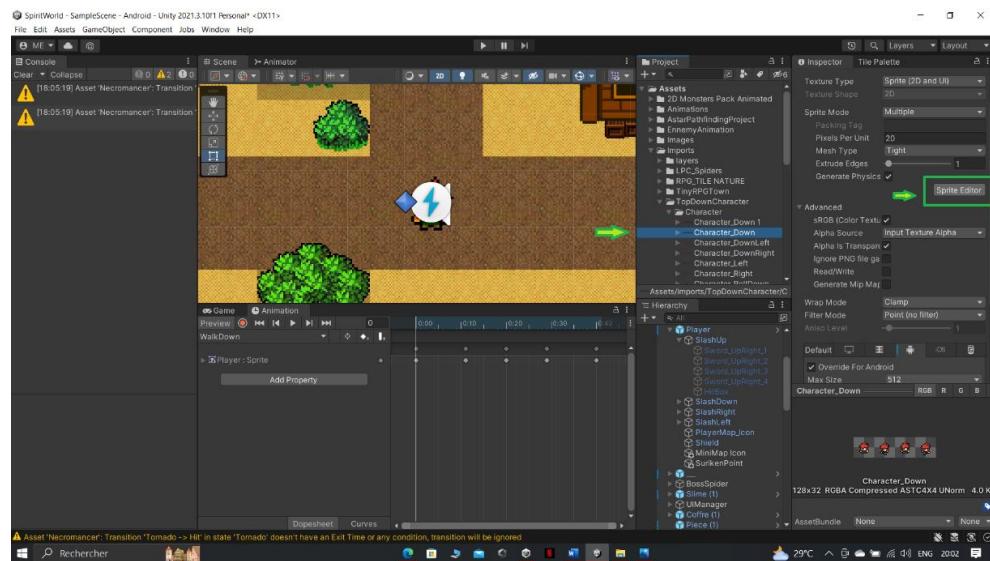


Figure 31 : Sprite Editor

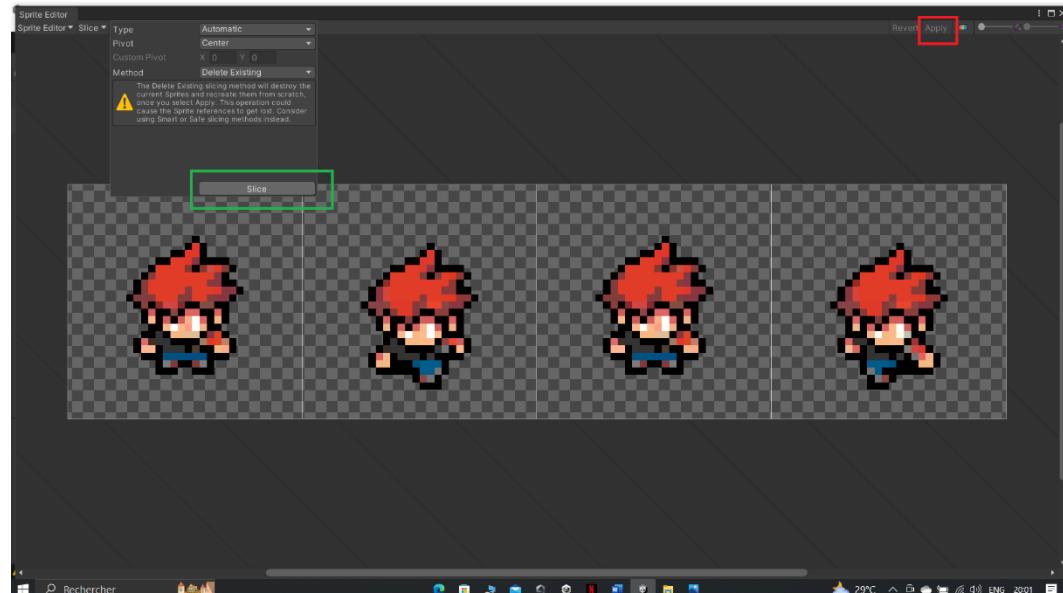


Figure 32 : Découpe des Sprites du joueur

Avec le **Sprite Editor**, nous pouvons diviser chaque Sprite pour en créer une animation. Dans le **Sprite Editor**, on choisit comme type de découpe automatique de ce fait qu'Unity va détecter chaque image du joueur et la découper pour en faire une frame (image), après la découpe on clique sur **Apply** et c'est tout. Nous effectuerons la même procédure pour toutes les sprites de déplacement et pour l'attaque.

L'effet de l'attaque aussi doit être découpée pour en faire une animation montrant l'effet d'une épée lorsque le joueur joue l'animation d'attaque :

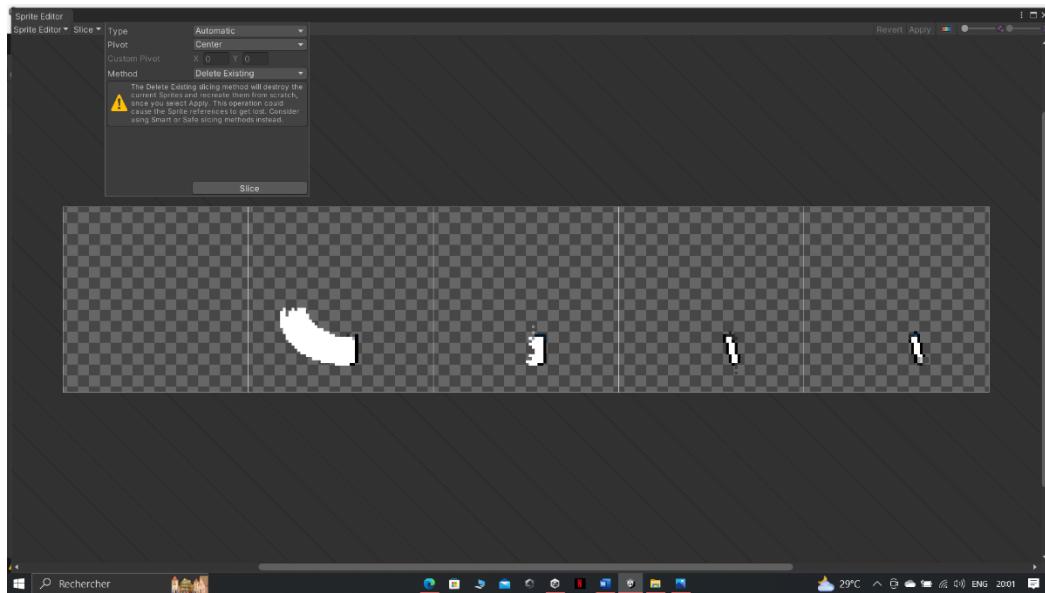


Figure 33 : Découpe de l'effet d'attaque du joueur.

### Attribuer un Tag :

Dans UNITY, tous les objets peuvent avoir un **Tag**. Lorsque l'on créer un objet, il possède le tag ‘Untagged’ qui est le tag par défaut de tous les objets. Cependant, nous aurions besoin d'attribuer un tag spécifique au joueur que nous allons appeler ‘Player’ afin de pouvoir le détecter dans notre code pour des référencement :

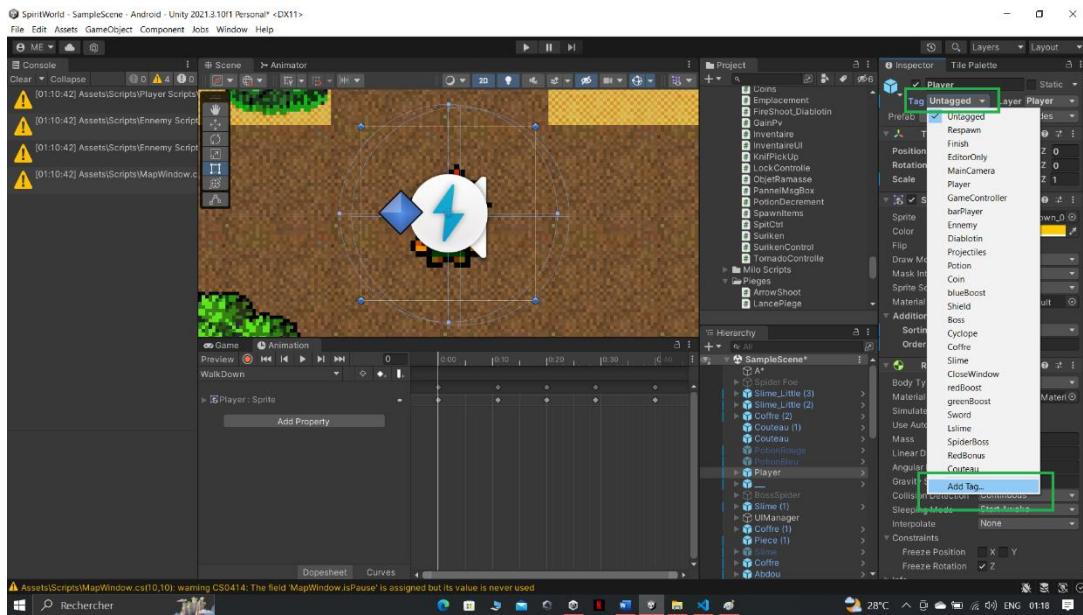
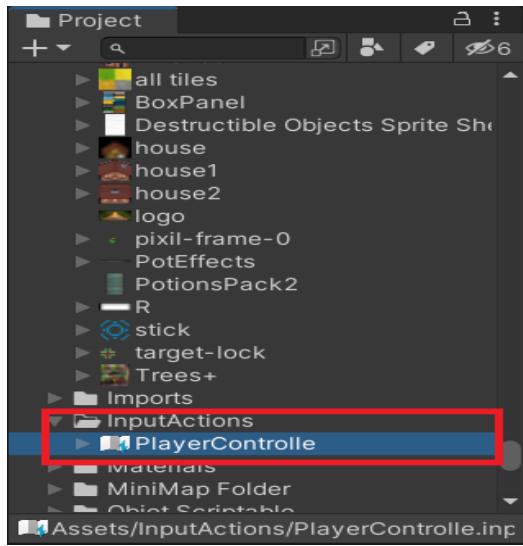


Figure 34 : Ajout d'un Tag ‘Player’ à l'objet joueur.

- Input Actions

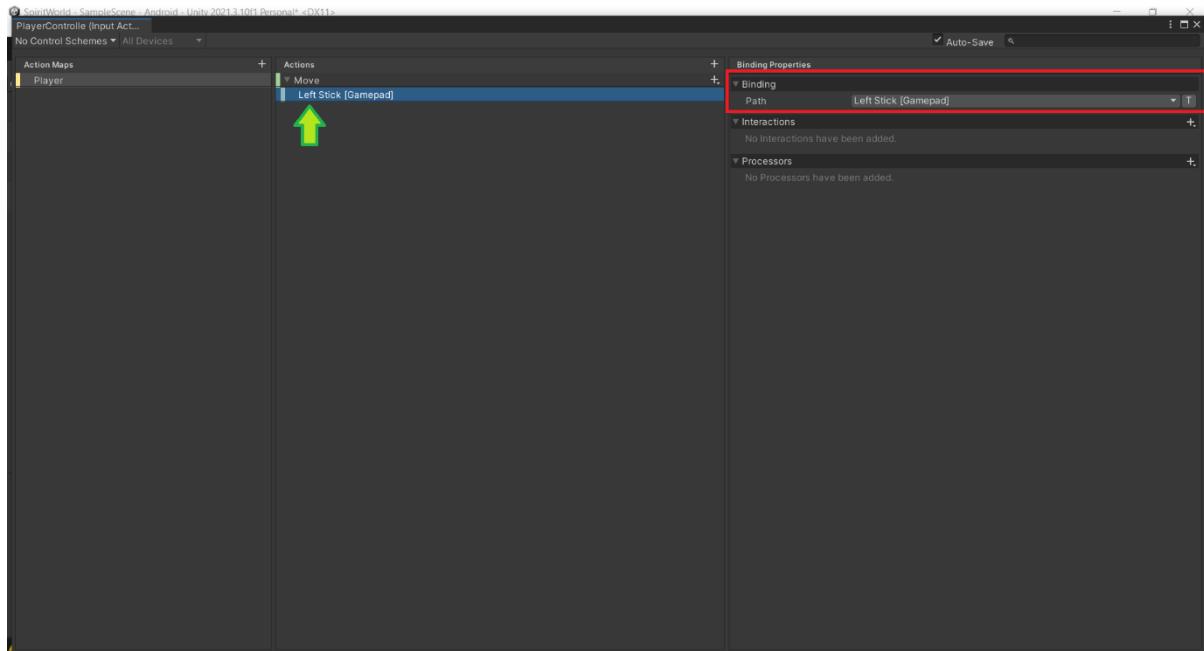
Nous allons à présent utiliser le New Input System de unity pour faciliter le contrôle du déplacement avec un joystick grâce à des Input Actions.

Voici comment en créer :



**Figure 35 : Crédit d'un InputActions tactile (PlayerControlle)**

Dans un dossier créé appelé ‘InputActions’, on fait un clique droit puis aller dans Create => Input Actions. De ce fait, nous créons un InputAction que nous allons configurer en double-cliquant dessus :



**Figure 36 : Configuration de l'InputAction**

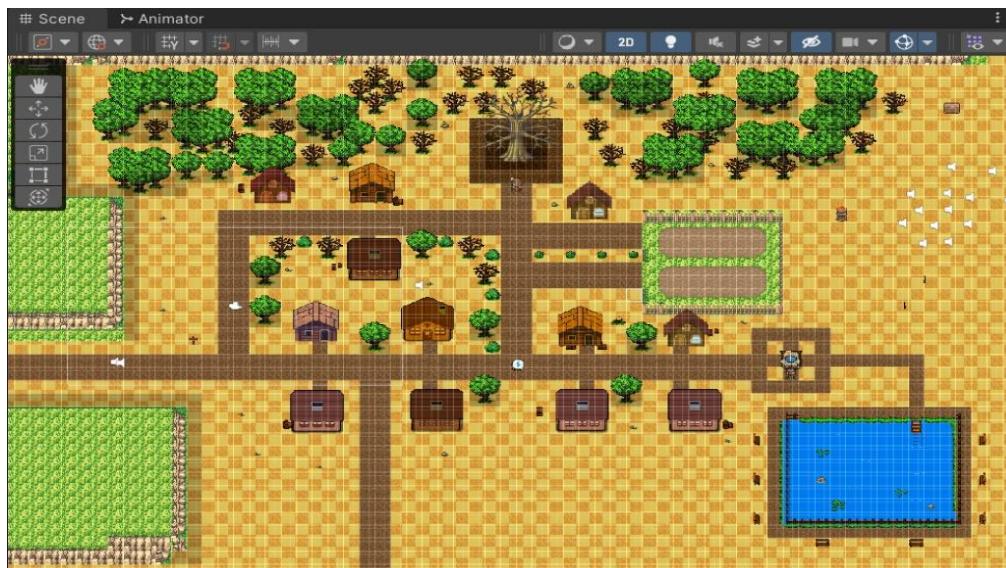
Dans le menu de l'inputAction que nous avons créé, on va tout d'abord créer un Action Map en cliquant sur le petit (+) de la section ‘Action Map’ puis donner un nom. Ensuite dans la section ‘Actions’, nous allons créer une action ‘Move’ de la même manière que l’ActionMap. Dans la section Action Properties, nous allons configurer le type d’action et le type de contrôle.

Le type d'action sera de type (Value) et le type de contrôle sera un Vector2 pour désigner les axes (x, y) du joystick pour déplacer le joueur.

Maintenant il nous faut un Binding ‘Système d’entrée’ pour accepter des entrées provenant de diverses sources (clavier, souris, manettes, écrans tactiles, etc.). Cela permet de gérer les entrées de manière plus uniforme, indépendamment du matériel utilisé par les joueurs. Dans notre cas, nous choisissons Left Stick [Gamepad]. Enfin, la configuration du système d’entrée du joueur est fin prête.

A présent nous pouvons nous concentrer sur le développement de l’environnement. Nous allons créer un village qui représentera le village de Tindaba, puis deux autres villages voisins où il faudra passer par le foret, territoire occupé par les monstres invoqué par l’esprit maléfique, des lacs, un labyrinthe et la zone de combat avec le boss final qui est l’esprit maléfique.

Village de Tindaba :



**Figure 37 : Interface du Village de Tindaba**

Foret :

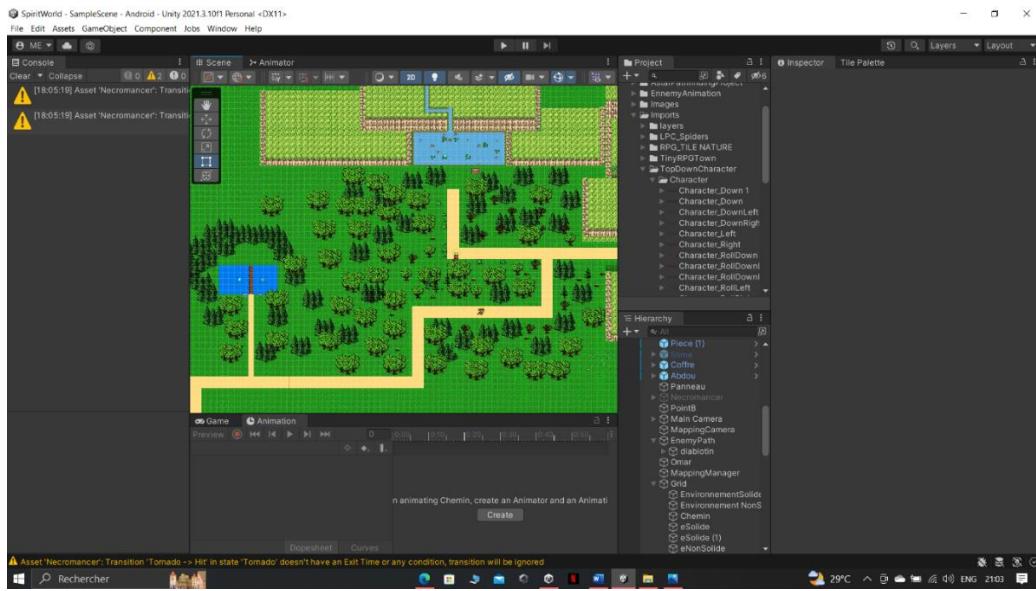


Figure 38 : Interface de la foret

Village Voisin :

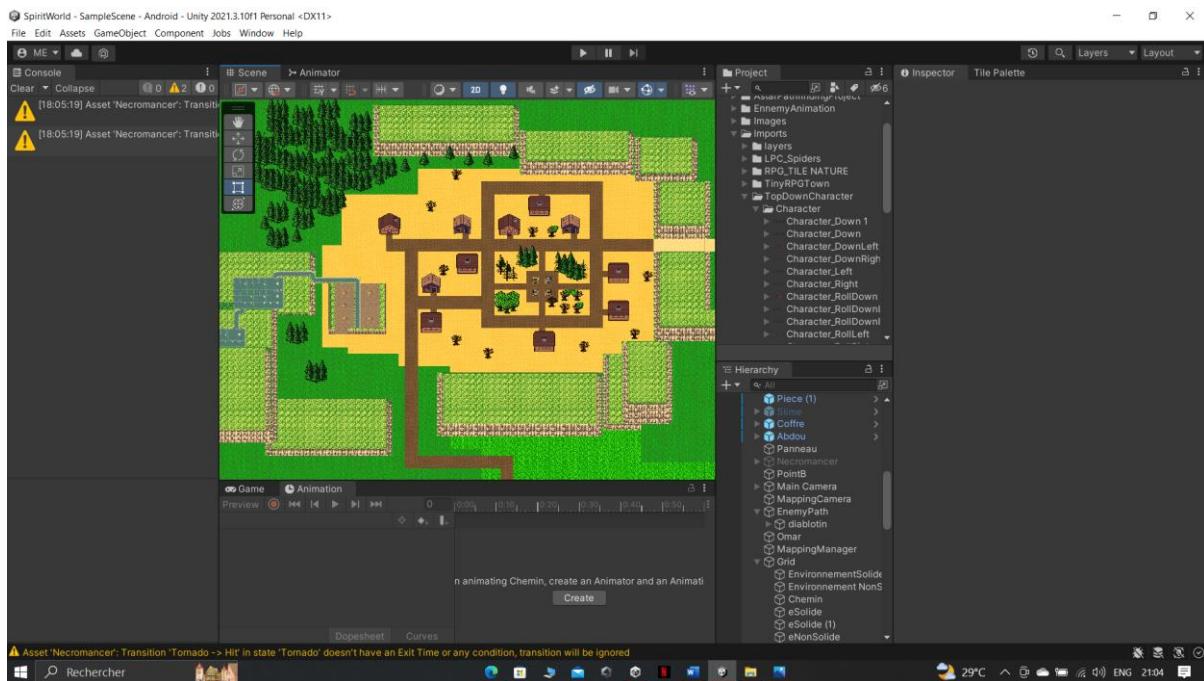
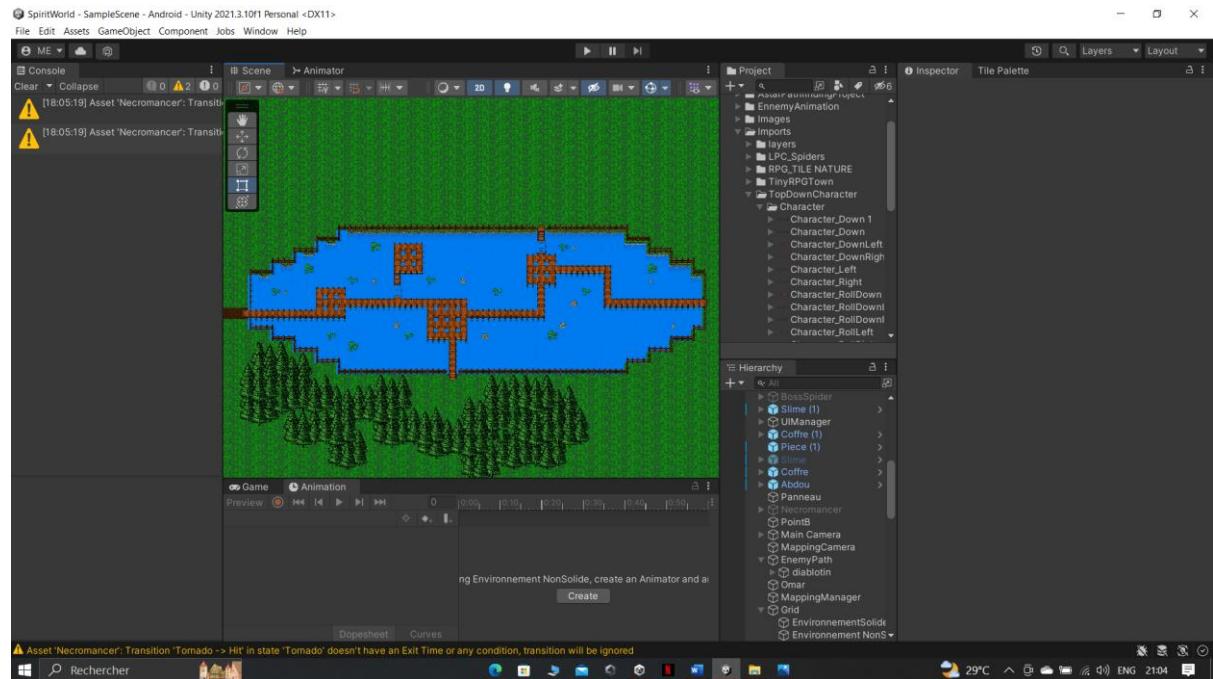
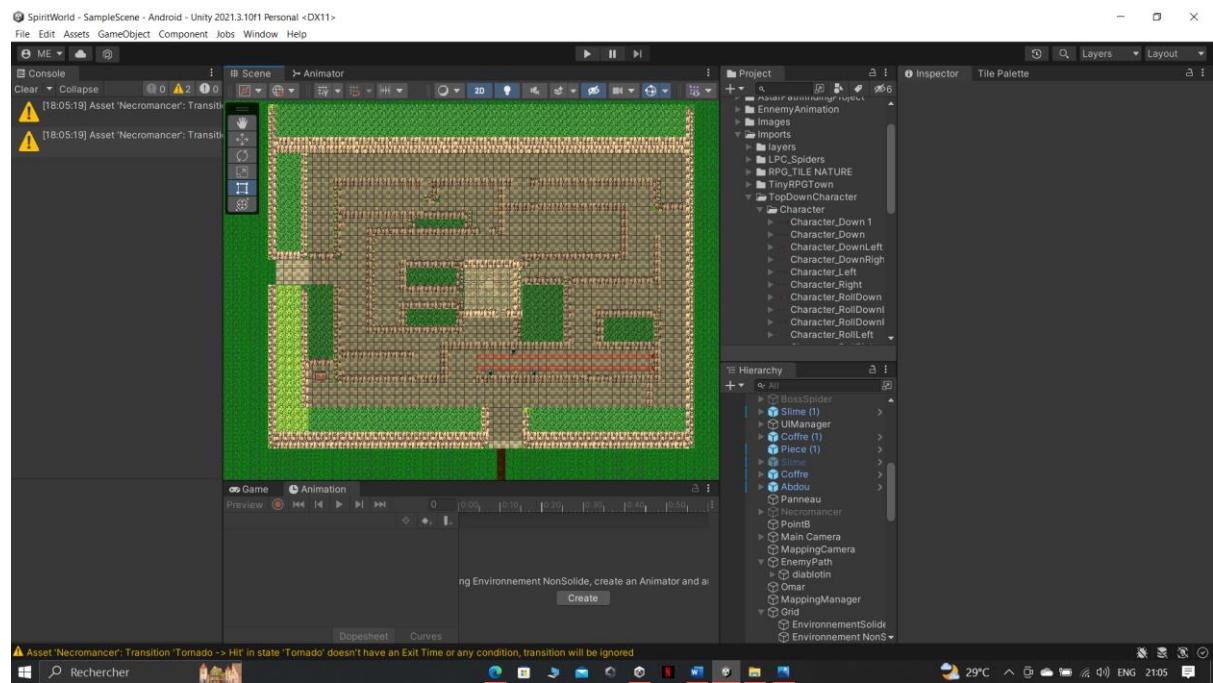
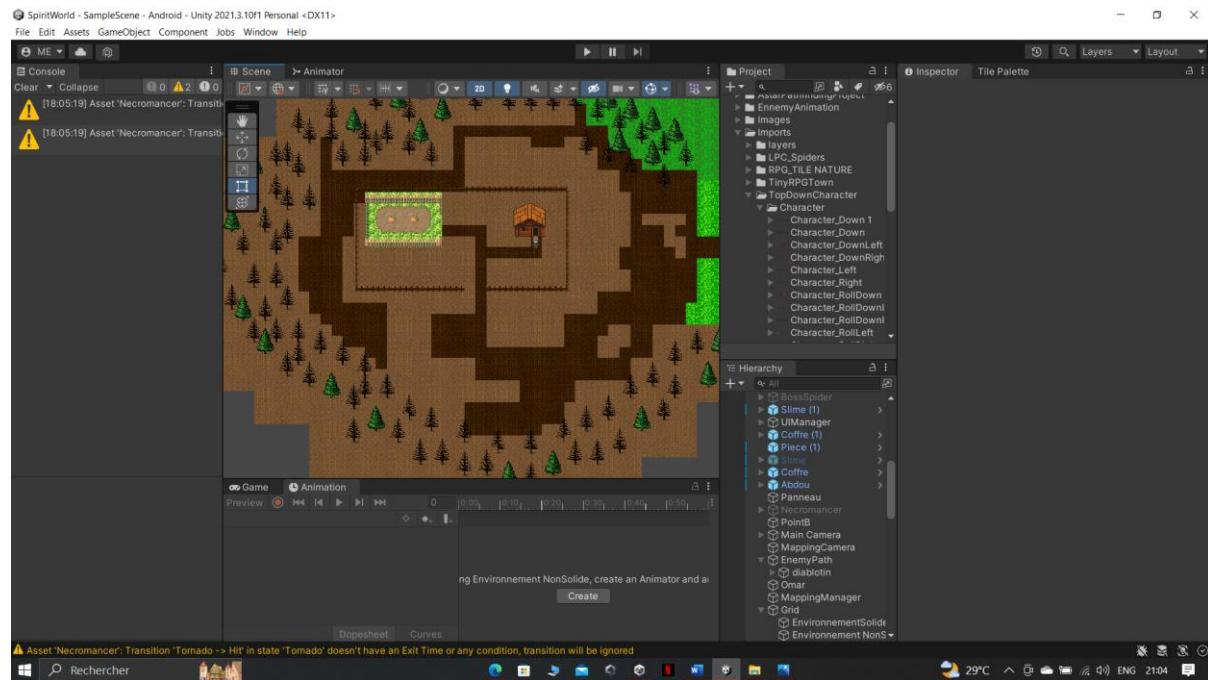


Figure 39 : Interface du village voisin

**Lac :****Figure 40 : Interface du Grand lac.****Labyrinthe :****Figure 41 : Interface du labyrinthe**

Territoire du Grand Saltigue :



**Figure 42 : Interface du territoire du Grand Saltigue.**

Nous avons conçu un environnement avec plusieurs territoires que le joueur pourra désormais explorer en combattant des monstres, effectuant des quêtes ou en explorant les lieux.

### 7.3 Animation de l'environnement et des personnages

L'animation est ce qui va donner vie à nos personnages et à l'environnement. Nous animerons les éléments piégés du jeu comme des lance qui s'active lorsque le joueur est proche, des lances flammes, des lames cacher sous le sol etc... Puis nous animerons les personnages du jeu comme le joueur, les PNJ, et les ennemis.

Prenons le cas d'une lance qui s'active et offre des dégâts au joueur lorsqu'il est en contact avec la lance :

Nous avons déjà créé deux animations : UP et Down pour le déclenchement et le retrait de la lance.

Pour créer une animation on ouvre la fenêtre Animation, puis on clique sur ‘Create New Clip’. Unity va nous demander l'emplacement de l'Animation pour l'enregistrer puis nous pouvons commencer à animer. Pour animer la lance dans la fenêtre Animation, nous allons sélectionner toutes les frames de la lance qui se dirigent vers le haut (Pour l'animation UP) et pareil pour

les frames du bas (Pour l'animation Down). Plusieurs points se forment représentant les frames de chaque milliseconde.

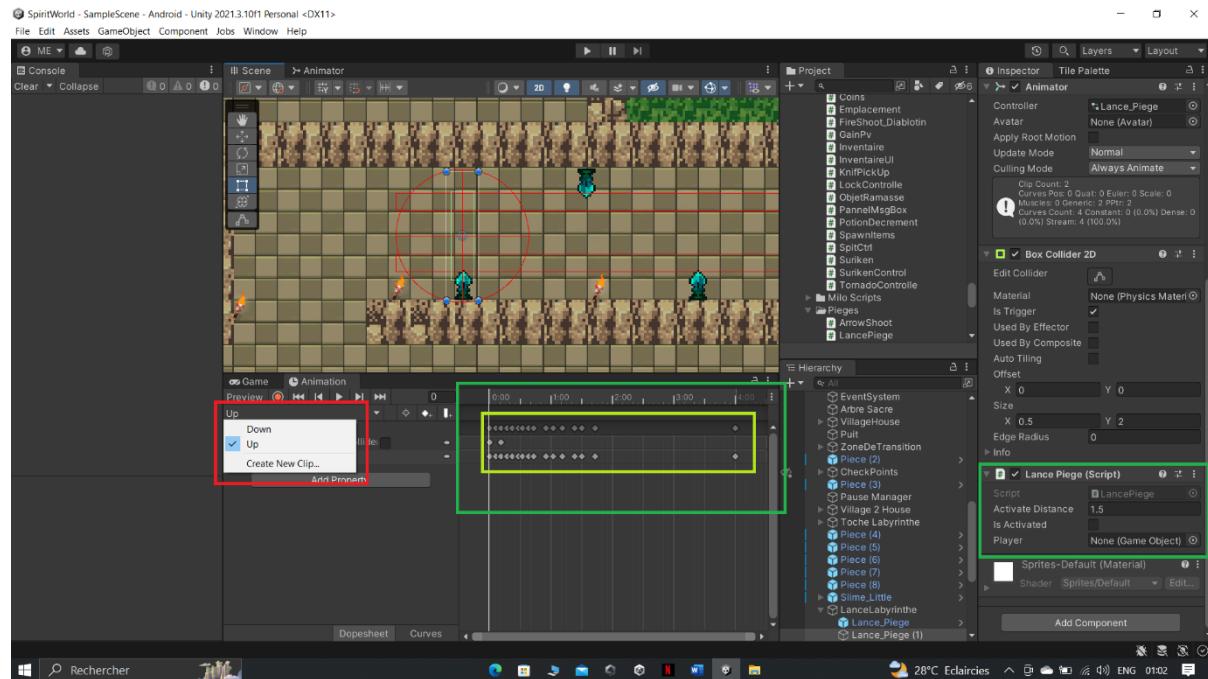


Figure 43 : Animation (Up & Down) du piège Lance

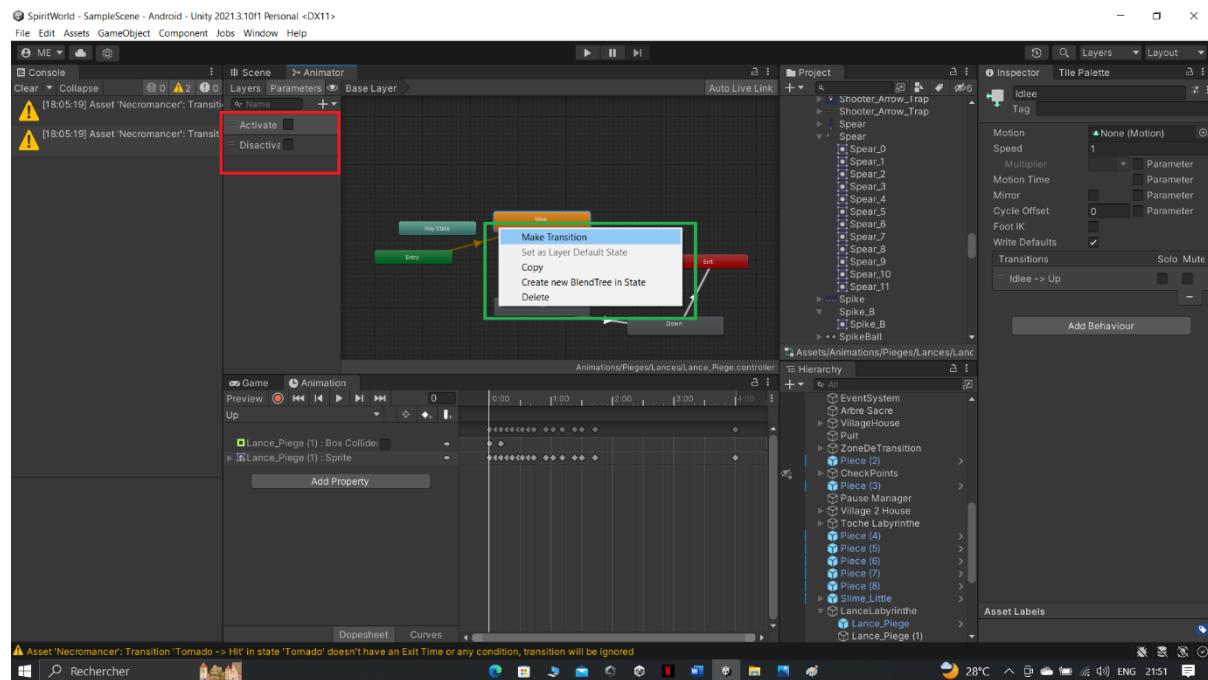


Figure 44 : Interface de l'Animator

Pour contrôler les différentes animations nous aurons besoin de l'Animator, un outil d'animations sur UNITY qui va nous aider à gérer les animations de façon à ce qu'elle s'exécute selon un paramètre qu'on utilisera comme conditions. Dans le cas de notre lance, les paramètres seront de type Booléen : Activate, Deactivate.

Dans l'Animator, nous avons les différentes animations UP et Down, nous avons aussi créer un état vide 'Idle' pour désigner un état de commencement. Donc on effectue une transition de Idle à l'Animations UP, puis comme conditions on attribue Activate a true et de l'Animation UP a Down on attribue Activate a false

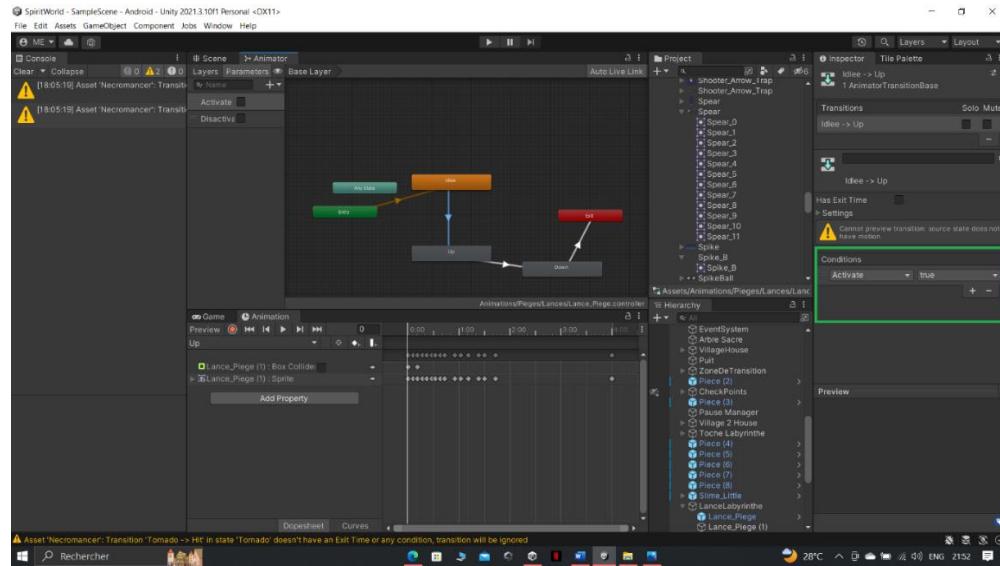


Figure 45 : Transition de l'animation Idle à l'animation Up

L'Animator nous permet de gérer les animations dans nos codes, après avoir bien configurer les animations dans l'Animator, nous pouvons les gérer dans le script de la lance que nous avons créé et ajouter : lancePiege :

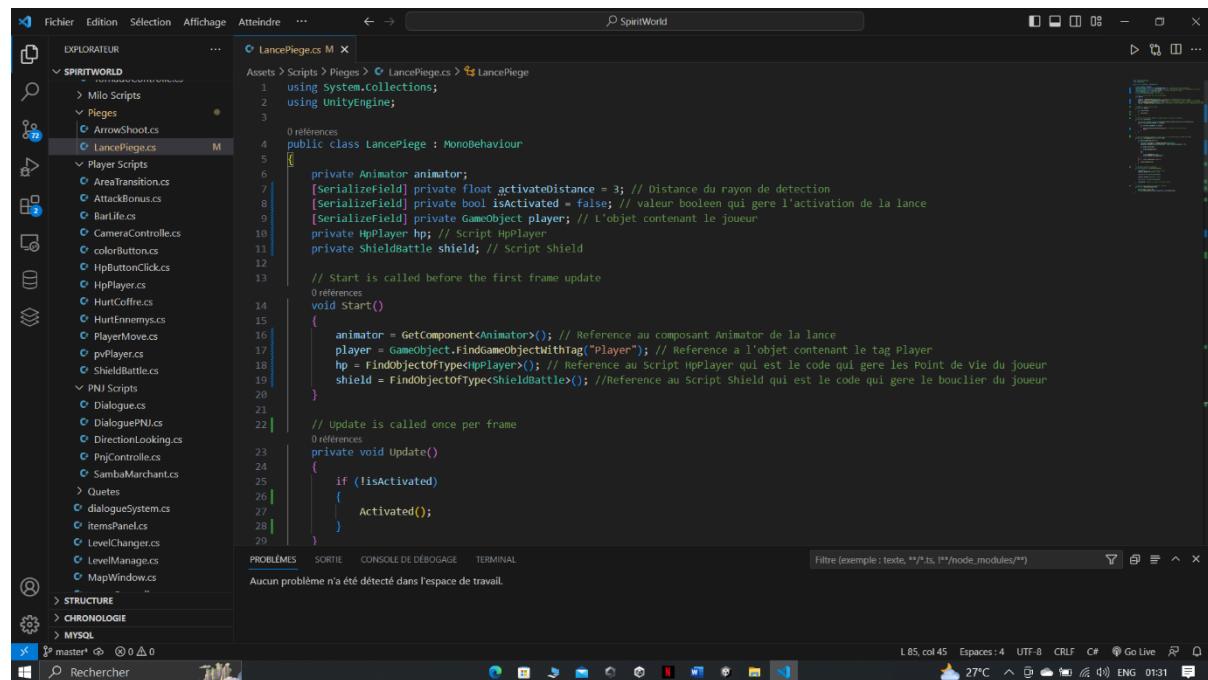
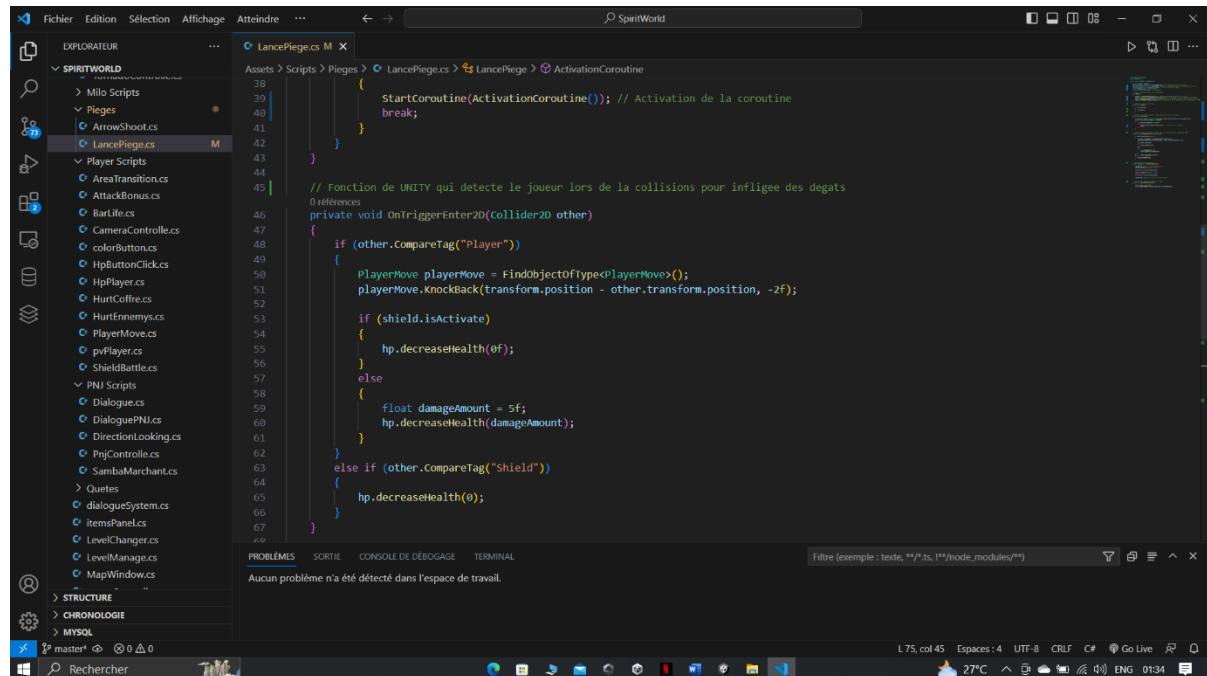


Figure 46 : Script (LancePiege) de la Lance

1. **Using Directives** : Ces directives indiquent que le script utilise les espaces de noms **System.Collections** et **UnityEngine**, ce qui permet d'accéder aux classes et fonctionnalités nécessaires.
2. **Class Declaration** : La classe principale est appelée **LancePiege** et elle hérite de **MonoBehaviour**, la classe de base pour les scripts Unity.
3. **Variables privées** :
  - **animator** : Une référence à l'Animator attaché à cet objet.
  - **activateDistance** : La distance à partir de laquelle le piège s'activera lorsque le joueur s'approche.
  - **isActivated** : Un booléen pour suivre l'état d'activation du piège.
  - **player** : Une référence au GameObject du joueur.
  - **hp** : Une référence à un composant (script) de gestion de la santé du joueur.
  - **shield** : Une référence à un composant de gestion du bouclier du joueur.
4. **Start()** : Cette méthode initialise les variables, en trouvant les références aux composants nécessaires et au joueur.
5. **Update()** : La méthode **Update** est appelée à chaque trame (frame) du jeu. Dans ce cas, elle vérifie si le piège n'est pas déjà activé, et si c'est le cas, elle appelle la méthode **Activated()**



```

using System.Collections;
using UnityEngine;

public class LancePiege : MonoBehaviour
{
    public float activateDistance = 2f;
    private bool isActivated = false;
    private Player hp;
    private Shield shield;

    void Start()
    {
        StartCoroutine(ActivationCoroutine());
    }

    void Update()
    {
        if (!isActivated)
        {
            return;
        }
    }

    void ActivationCoroutine()
    {
        while (true)
        {
            if (transform.position - hp.transform.position < activateDistance)
            {
                StartCoroutine(OnTriggerEnter2D());
            }
            yield break;
        }
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            if (shield.isActivate)
            {
                hp.decreaseHealth(5f);
            }
            else
            {
                float damageAmount = 5f;
                hp.decreaseHealth(damageAmount);
            }
        }
        else if (other.CompareTag("shield"))
        {
            hp.decreaseHealth(5f);
        }
    }
}

```

**Figure 47 : Méthode OnTriggerEnter2D du script de la Lance**

6. **Activated()** : Cette méthode est chargée de vérifier si le joueur est à proximité du piège, et si c'est le cas, elle lance une coroutine (**ActivationCoroutine()**) pour activer le piège.

```

    // Coroutine qui gère l'animation
    private IEnumerator ActivationCoroutine()
    {
        isActivated = true; // La lance est activée
        animator.SetBool("Activate", true);

        yield return new WaitForSeconds(4f);

        animator.SetBool("Activate", false);

        yield return new WaitForSeconds(4f);

        isActivated = false; // La lance n'est plus activée
    }

    // Fonction pour dessiner le rayon dans l'éditeur
    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, activateDistance);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            if (isActivated)
            {
                other.GetComponent<HealthManager>().TakeDamage(damage);
            }
        }
        else if (other.tag == "Shield")
        {
            if (isActivated)
            {
                other.GetComponent<HealthManager>().TakeDamage(0);
            }
        }
    }
}

```

**Figure 48 : Coroutine ActivateCoroutine() du Script LancePiege**

7. **OnTriggerEnter2D(Collider2D other)** : Cette méthode est appelée lorsque quelque chose entre en collision avec le collider du piège. Si l'objet en collision est le joueur (étiqueté avec "Player"), cela déclenche un effet de recul sur le joueur et inflige des dégâts à sa santé. Si l'objet est le bouclier du joueur, aucun dégât n'est infligé.
8. **ActivationCoroutine()** : Cette coroutine gère le comportement d'activation et de désactivation du piège. Pendant 4 secondes, le piège est activé visuellement (animation), puis il est désactivé pendant 4 secondes.
9. **OnDrawGizmosSelected()** : Cette méthode est appelée lorsque l'objet est sélectionné dans l'éditeur Unity. Elle dessine une sphère rouge autour du piège pour représenter la distance d'activation.

Nous avons enfin créé notre premier piège qui jouera un rôle dans notre environnement 2D. En effectuant les bonnes pratiques en animation et en ayant acquis les bases en programmation, nous pouvons créer de la logique pour tous les objets d'un jeu et c'est ce qui les rends amusants. Pour les autres pièges du jeu nous resterons sur la même procédure en alliant graphisme et programmation, nous donnons vie à un univers amusant et immersive.

## 7.4 Création des entités autonome ( IA )

Dans cette section du chapitre, nous allons créer nos entités autonomes, autrement dit : Les PNJ et les ennemis.

Les PNJ, sont ceux qui nous permettront de recevoir des quêtes, de faire la rencontre de nouveaux personnages qui nous aideront dans notre quête. Nous allons voir comment mettre

en place un système de dialogue et de déplacement autonome point par point. Après avoir vu comment découper les trames (frames) d'un PNJ (page 51), nous allons créer un PNJ nommé 'Abdou'

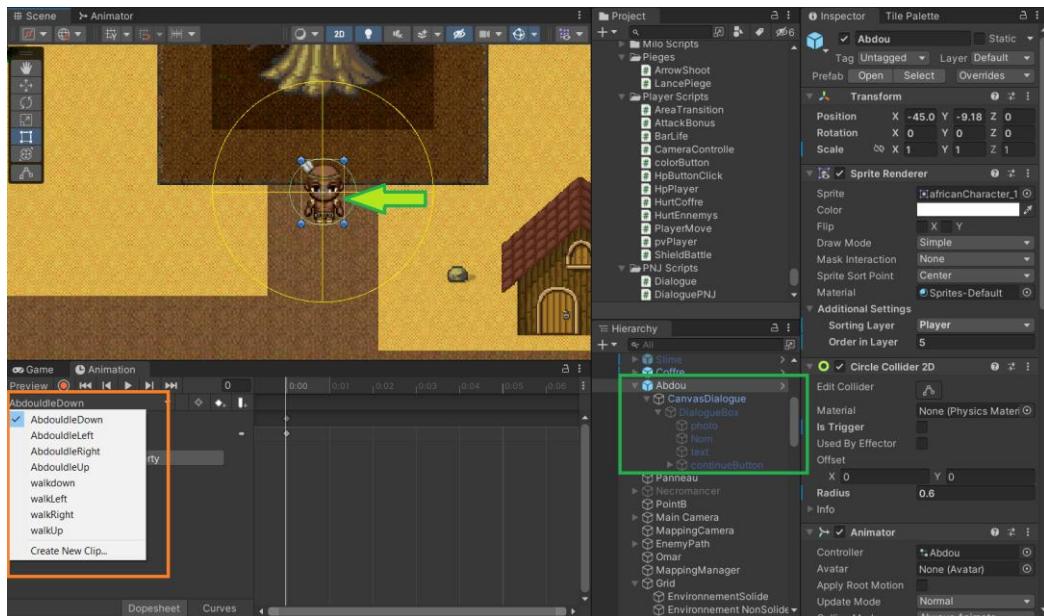


Figure 49 : Crédit d'un PNJ (Abdou)

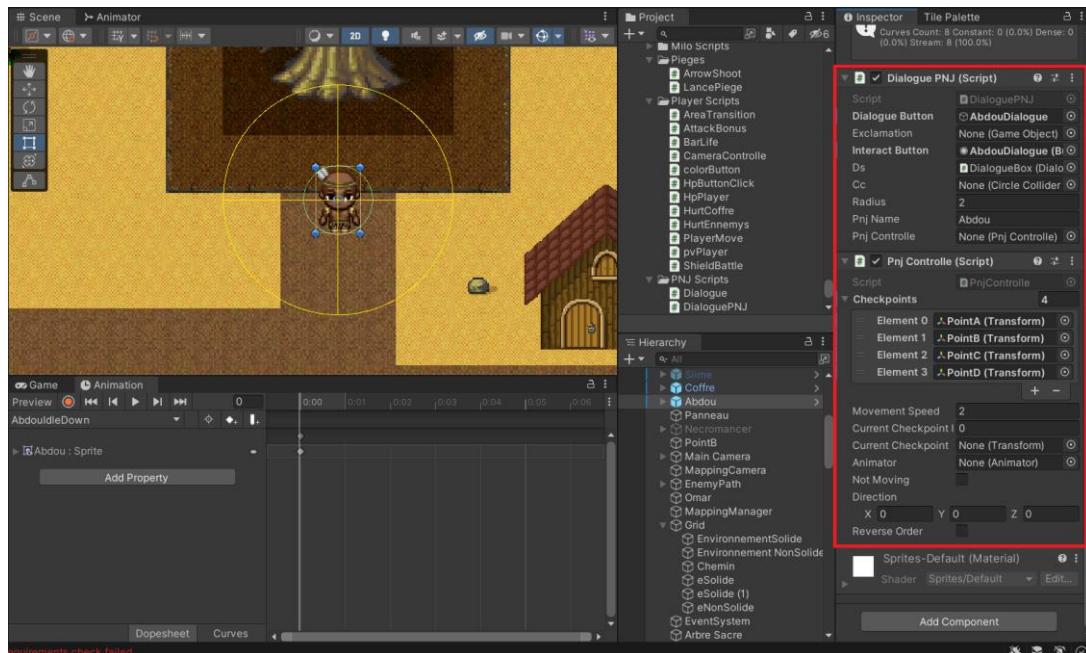


Figure 50 : Scripts du PNJ

Comme nous le voyons, nous avons glissé un Sprite d'un PNJ que nous avons nommé 'Abdou' dans la scène pour le représenter dans notre plan de travail. Ce PNJ contient un Circle Collider 2D pour avoir une collision circulaire avec les objets ce qui lui offre une meilleure fluidité de collision. Les animations sont mises en place pour l'arrêt et le déplacement du PNJ Abdou.

Deux scripts sont attachés au PNJ : DialoguePNJ et PnjControlle.

On remarque aussi un canvas attaché au joueur dans la Hierarchy, ce canvas contient la boite de dialogue (DialogueBox) de Abdou lorsque le joueur entre en contact avec Abdou, un bouton apparaît à l'écran nous permettant de démarrer une discussion avec Abdou. Ce bouton va appeler la fonction ‘Interact()’ du script DialoguePnj de l'objet Abdou pour déclencher la discussion lors du clique.

Nous pouvons aussi remarquer le Script DialogueSystem attaché à l'objet ‘DialogueBox’. Ce script va gérer les phrases qu'Abdou devra dire et lorsque l'on appuie sur le bouton ‘Suivant’, il continuera la discussion avec une nouvelle phrase grâce au dictionnaire de donnée :

- public Dictionary<string, string[]> pnjDialogues = new Dictionary<string, string[]>();

Enfin, nous vient le script pour le déplacement autonome du PNJ ‘PnjControlle’.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PnjControlle : MonoBehaviour
{
    public Transform[] checkpoints; // Les points de contrôle (PointA, PointB, C, D)
    public float movementSpeed = 5f; // Vitesse de déplacement du PNJ
    [SerializeField] private int currentCheckpointIndex; // L'index du point de contrôle actuel
    [SerializeField] private Transform currentCheckpoint; // Le point de contrôle actuel
    [SerializeField] private Animator animator; // Référence à l'Animator du PNJ
    public bool notMoving;
    [SerializeField] private Vector3 direction;
    [SerializeField] private bool reverseOrder; // Indique si le PNJ doit suivre les points de contrôle dans l'ordre inverse

    void Start()
    {
        currentCheckpointIndex = 0;
        currentCheckpoint = checkpoints[currentCheckpointIndex];
        animator = GetComponent<Animator>();
        notMoving = false;
        reverseOrder = false;
    }

    void Update()
    {
        DeplacerCheckpoints();
    }

    void DeplacerCheckpoints()
    {
        if (!notMoving)
        {
            if (reverseOrder)
            {
                ...
            }
            else
            {
                ...
            }
        }
    }
}

```

**Figure 51 : Script PnjControlle du PNJ ‘Abdou’**

## 1. Variables publiques :

- **checkpoints** : Un tableau de Transform contenant les points de contrôle que le PNJ doit suivre.
- **movementSpeed** : La vitesse de déplacement du PNJ.
- **notMoving** : Un booléen qui indique si le PNJ doit arrêter de bouger.
- **animator** : Une référence à l'Animator du PNJ pour gérer les animations.
- **direction** : La direction vers laquelle le PNJ se déplace.
- **reverseOrder** : Un booléen qui indique si le PNJ doit suivre les points de contrôle dans l'ordre inverse.

**DeplacerCheckpoints()** : Cette méthode gère le déplacement du PNJ entre les points de contrôle. Si le PNJ ne doit pas s'arrêter (**!notMoving**), elle calcule la direction vers le point de contrôle actuel, déplace le PNJ en fonction de la direction et de la vitesse, gère les animations et vérifie si le PNJ a atteint le point de contrôle.

2. **StopWalking()** : Cette méthode arrête le mouvement du PNJ en mettant à jour les animations et la variable **notMoving**.
3. **WaitForAnimation()** : Cette coroutines arrête le mouvement du PNJ, le met en pause pendant une courte période (**1.5f** secondes) pour simuler une pause entre les animations, puis réactive le mouvement du PNJ.

Ce script contrôle le déplacement du PNJ entre les points de contrôle définis. Le PNJ peut suivre les points de contrôle dans l'ordre ou dans l'ordre inverse. Le script gère également les animations du PNJ en fonction de son mouvement et de son arrêt, et il permet d'arrêter temporairement le mouvement pour laisser place aux animations.

Finissons avec les ennemis. Nous allons créer un ennemi (Slime) qui pourchassera et attaquera le joueur en lui infligeant des dégâts lorsque le joueur est à portée et si le joueur est en dehors de la portée de l'ennemi, l'ennemi revient à sa position de départ. Nous allons faire en sorte que le Slime soit détecté et sélectionné lorsque le joueur appuis sur lui sur l'écran de son téléphone et nous pourrions ensuite lui jeter des couteaux pour lui infliger des dégâts à distance.

Ennemis Slime :

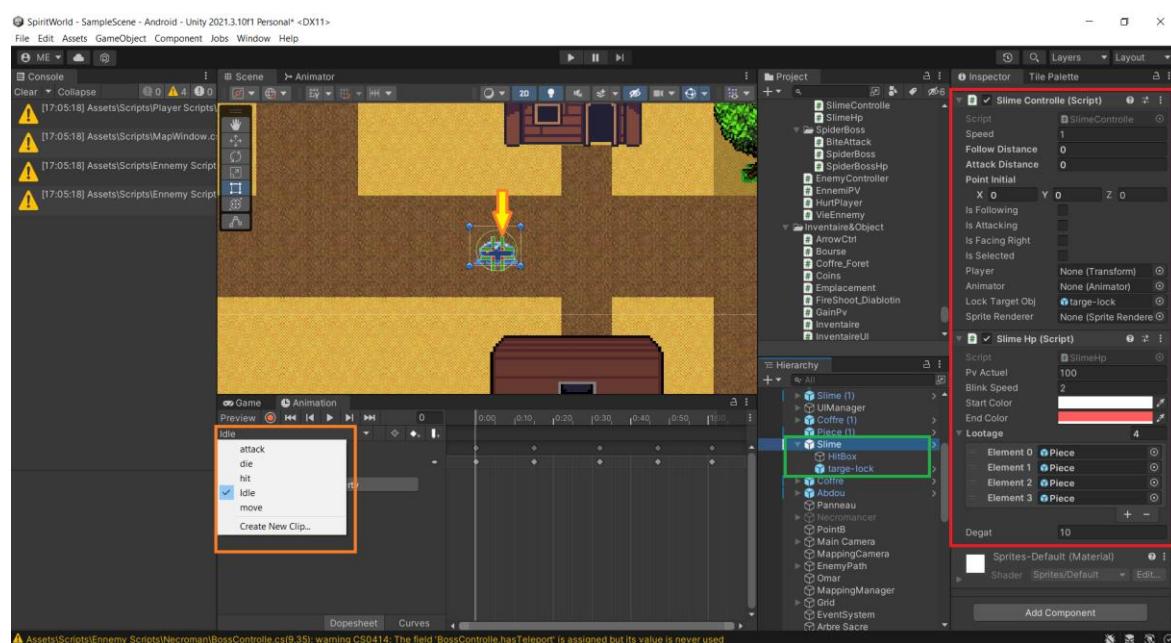
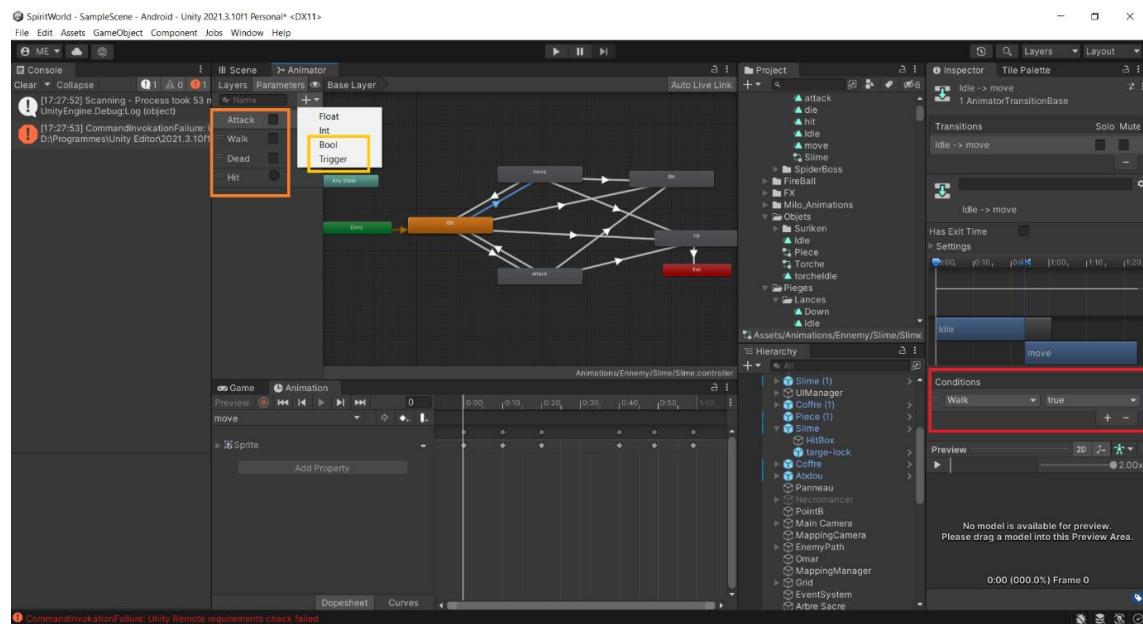


Figure 52 : Crédit de l'ennemi ‘Slime’

Nous avons ajouté un Slime dans notre scène, nous avons déjà découpé les sprites du Slime pour faire des animations. Nous avons déjà créé les animations de la même manière que l'étape d'animation du piège de la lance (page 72). Deux scripts ont été créés sur le Slime, ‘SlimeControlle’ sera responsable du contrôle du slime pour le déplacement et l’attaque et la sélection du slime par le toucher tactile de l’écran.

Dans la Hiérarchie, on a ajouté un objet HitBox qui sera responsable du contact avec le joueur lors de l’attaque du Slime pour infliger des dégâts au joueur, et nous avons ajouté un objet target-lock qui représente le symbole vert sur le slime pour montrer qu’il est sélectionné lorsque le joueur appuie sur le Slime.



**Figure 53 : Animations de l’ennemi ‘Slime’**

Dans l’Animator du Slime, nous avons les différents états des animations que nous avons créées : **attack – die – hit – idle – move**.

L’animation Idle qui est de couleur orange dans l’Animator est l’état par défaut c’est-à-dire qu’il sera la première animation jouer.

Nous avons créé 4 paramètres dont 3 booléens : Attack – Walk – Dead, et un paramètre Trigger (Déclencheur) : Hit.

L’état Idle est relié à tous les autres en formant des transitions. Par exemple, la transitions de Idle => move possède comme condition : Walk => true. Ce qui veut dire que dans le code du Slime, lorsque par exemple le joueur est à portée de vue du slime, le Slime va se diriger vers sa cible en jouant l’animation Walk si la condition est à true. Et pour revenir à l’état Idle lorsque le joueur s’éloigne du Slime, on refait une transition de l’état move => Idle en ajoutant comme condition : Walk => false.

Voyons maintenant notre code SlimeControlle qui gère le système de contrôle du Slime :

```

public class SlimeController : MonoBehaviour
{
    public float speed;
    [SerializeField] private float followDistance;
    [SerializeField] private float attackDistance;
    [SerializeField] private Vector3 pointInitial;
    [SerializeField] private bool isFollowing = false;
    [SerializeField] private bool isAttacking = false;
    [SerializeField] private bool isSelected = false;
    private string playerTag = "Player";
    public Transform player;
    public Animator animator;
    private Suriken suriken;
    public GameObject lockTargetObj;
    // Start is called before the first frame update
    void Start()
    {
        animator = GetComponent<Animator>();
        player = GameObject.FindGameObjectWithTag(playerTag).transform;
        suriken = FindObjectOfType<Suriken>();
        spriteRenderer = GetComponent<SpriteRenderer>();
        lockTargetObj.SetActive(false);
        pointInitial = transform.position;
    }
}

```

Figure 54 : Script de contrôle de l'ennemi 'SlimeController'

### Variables publiques et privées :

- public float speed; : La vitesse à laquelle le Slime se déplace.
- [SerializeField] private float followDistance; : La distance à laquelle le Slime commencera à suivre le joueur.
- [SerializeField] private float attackDistance; : La distance à laquelle le Slime attaquera le joueur.
- [SerializeField] private Vector3 pointInitial; : La position initiale du Slime.
- [SerializeField] private bool isFollowing = false; : Indique si le Slime est en train de suivre le joueur.
- [SerializeField] private bool isAttacking = false; : Indique si le Slime est en train d'attaquer le joueur.
- [SerializeField] private bool isFacingRight = false; : Indique si le Slime fait face à droite.
- public bool isSelected = false; : Indique si le Slime est sélectionné.
- private string playerTag = "Player"; : Le tag utilisé pour trouver le joueur.
- public Transform player; : La référence au transform du joueur.
- public Animator animator; : La référence à l'Animator attaché au Slime.
- private Suriken suriken; : Une référence à un objet appelé "Suriken".
- public GameObject lockTargetObj; : Un objet utilisé pour indiquer que le Slime est sélectionné.

### Méthode OnMouseDown :

- Appelée lorsque le Slime est cliqué. Gère la sélection et la désélection du Slime ainsi que d'autres actions similaires pour d'autres types de personnages.

### Méthode PositionDepart :

- Fait déplacer le Slime vers sa position initiale. Gère également la rotation du personnage en fonction de sa direction.

### Méthode FollowPlayer :

- Gère le comportement de suivi du joueur par le Slime en fonction de la distance entre eux. Gère également la rotation et l'animation.

### Méthode AttackPlayer :

- Gère le comportement d'attaque du Slime en fonction de la distance entre lui et le joueur. Gère la rotation, l'animation et l'arrêt du déplacement.

### Méthode Flip :

- Inverse l'orientation du Slime en changeant l'échelle du transform.

Voyons maintenant le système de point de vie du Slime dans notre script : SlimeHp

```

Fichier Édition Sélection Affichage Atteindre ... ← → SpiritWorld
EXPLORATEUR ... Assets > Scripts > Enemy Scripts > Slime > SlimeHp.cs > SlimeHp > Start
SPLITWORLD
Diablotin
DiablotinController.cs
HpDiablotin.cs
LittleSlime
HpLittleSlimes.cs
LittleSlime.cs
Necromon
BossController.cs
FlameAttack.cs
HpNecromones.cs
RayonAttacks.cs
Slime
SlimeAttack.cs
SlimeController.cs
SlimeHp.cs
SpiderLoses
EnemyController.cs
EnnemiPV.cs
HurtPlayer.cs
Viennemy.cs
Inventaire&Object
ArrowController.cs
Bourse.cs
Coffee_Foret.cs
Coins.cs
Emplacement.cs
FireShoot_Diablotin.cs
GainPV.cs
PROBLÈMES SORTIE CONSOLE DE DÉBOUTAGE TERMINAL
Aucun problème n'a été détecté dans l'espace de travail.
123, col 23 Espaces : 4 UTE-B CR LF ⌘ Go Live ⌘ D
STRUCTURE CHRONOLOGIE MYSQL
master* 0 △ 0

```

Figure 55 : Script de Points de vie de l'ennemi Slime ‘SlimeHp’

Ce script gère la santé, les dégâts et la mort du "Slime" dans le jeu. Nous avons inclus également des fonctionnalités pour faire clignoter le personnage lorsque sa santé est faible et pour générer des objets de butin (loot) lorsqu'il meurt. Voici une explication détaillée du code :

#### 1. Variables privées et sérialisées :

- **[SerializeField] private int pvActuel;** : Les points de vie actuels du Slime.
- **[SerializeField] private float blinkSpeed;** : La vitesse de clignotement lorsqu'il subit des dégâts.
- **[SerializeField] private Color startColor = Color.white;** : La couleur de base du SpriteRenderer du Slime.
- **[SerializeField] private Color endColor = new Color(1f, 0.627f, 0.627f);** : La couleur vers laquelle le SpriteRenderer va clignoter.
- **private Animator animator;** : La référence à l'Animator attaché au Slime.

- **private SpriteRenderer spRenderer;** : La référence au composant SpriteRenderer du Slime.
- **public List<GameObject> lootage = new List<GameObject>();** : Une liste d'objets de butin à générer lorsque le Slime meurt.
- **public int degat = 10;** : La quantité de dégâts infligés lorsque le Slime est touché.
- **private CircleCollider2D circle;** : Une référence au composant CircleCollider2D du Slime.

## 2. Méthode DegatRecu :

- Réduit les points de vie du Slime en fonction des dégâts infligés, déclenche une animation de coup reçu et gère sa mort si ses points de vie atteignent zéro.

## 3. Méthode Mort :

- Désactive le collider, gère les animations de mort et de désactivation des autres animations, puis détruit l'objet Slime après un délai.

## 4. Méthode BlinkCoroutine :

- Une coroutine qui fait clignoter la couleur du SpriteRenderer entre la couleur de départ et la couleur de clignotement, donnant l'effet visuel de faible santé.

## 5. Méthode LootSpawnsCoroutine :

- Une coroutine qui attend un court délai après la mort du Slime, puis instancie des objets de butin aléatoires autour de sa position.

Vient enfin le dernier script (SlimeAttack) de l'objet HitBox du Slime qui s'active lorsque le Slime joue l'animation d'attaque pour toucher le joueur et lui infliger des dégâts :

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SlimeAttack : MonoBehaviour
{
    void Start()
    {
        hpPlayer = FindObjectOfType<HpPlayer>();
        shieldBattle = FindObjectOfType<ShieldBattle>();
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            if (other.CompareTag("Player"))
            {
                if (shield.isActivate)
                {
                    hpPlayer.decreaseHealth(5f);
                }
                else
                {
                    float damageAmount = 5f;
                    hpPlayer.decreaseHealth(damageAmount);
                    PlayerMove pm = FindObjectOfType<PlayerMove>();
                    pm.KnockbackX(transform.position - other.transform.position, -25f);
                }
            }
        }
        else if (other.CompareTag("Shield"))
        {
            hpPlayer.decreaseHealth(5f);
        }
    }
}

```

Figure 56 : Script d'attaque du Slime ‘SlimeAttack.cs’

Le script utilise les références aux scripts « HpPlayer » et « ShieldBattle » pour appliquer les dégâts au joueur et gérer l'état du bouclier.

Voici une explication :

## 1. Variables privées :

- **HpPlayer hpPlayer ;** : Une référence à un script qui gère la santé du joueur.
- **ShieldBattle shield ;** : Une référence à un script qui gère l'état du bouclier du joueur.

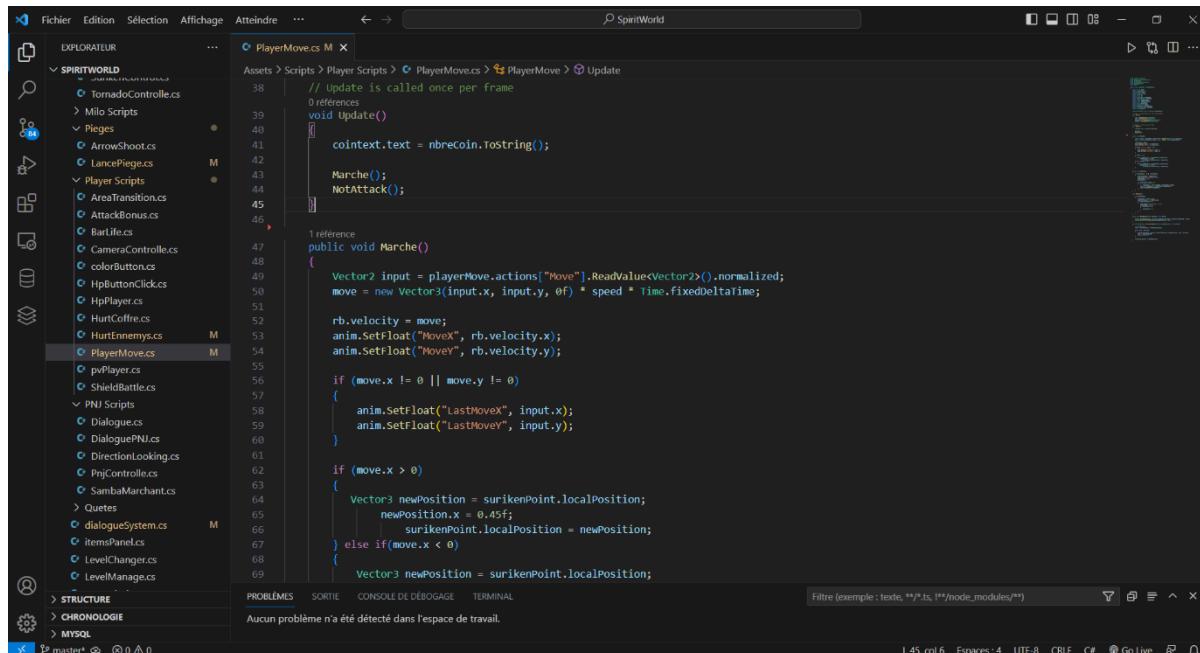
## 2. Méthode OnTriggerEnter2D :

- Cette méthode est appelée lorsque le collider du Slime entre en collision avec un autre collider dans la scène.
- Si le collider en collision a le tag « Player », le script vérifie ensuite si le bouclier du joueur est activé (**shield.isActive**). S'il l'est, cela signifie que le joueur bloque l'attaque avec le bouclier et aucune perte de santé n'est subie (**hpPlayer.decreaseHealth(0f)**).
- Si le bouclier n'est pas activé, alors le joueur subit des dégâts (**hpPlayer.decreaseHealth(damageAmount)**) et il est repoussé légèrement (**KnockBack**).
- Si le collider en collision a le tag « Shield », cela signifie que l'attaque a touché le bouclier du joueur, donc aucune perte de santé n'est appliquée (**hpPlayer.decreaseHealth(0)**).

## 7.5 Programmation du joueur

Dans ce chapitre, nous allons créer les différents scripts du joueur pour le contrôle et l'attaque, l'inventaire, les points de vie et le bouclier.

Commençons par le Script de contrôle : **PlayerMove**



```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMove : MonoBehaviour
{
    void Update()
    {
        // Update is called once per frame
        void update()
        {
            context.text = nbrecoin.ToString();

            Marche();
            NotAttack();
        }
    }

    public void Marche()
    {
        Vector2 input = playerMove.actions["Move"].ReadValue<Vector2>().normalized;
        move = new Vector3(input.x, input.y, 0f) * speed * Time.deltaTime;

        rb.velocity = move;
        anim.SetFloat("moveX", rb.velocity.x);
        anim.SetFloat("moveY", rb.velocity.y);

        if (move.x != 0 || move.y != 0)
        {
            anim.SetFloat("LastMoveX", input.x);
            anim.SetFloat("LastMoveY", input.y);
        }

        if (move.x > 0)
        {
            Vector3 newPosition = surikenPoint.localPosition;
            newPosition.x = 0.45f;
            surikenPoint.localPosition = newPosition;
        } else if (move.x < 0)
        {
            Vector3 newPosition = surikenPoint.localPosition;
        }
    }
}

```

Figure 57 : Script de contrôle du joueur ‘PlayerMove.cs’

## 1. Variables publiques et sérialisées :

- **public float speed;** : La vitesse de déplacement du joueur.
- **public Rigidbody2D rb;** : La référence au composant Rigidbody2D du joueur.
- **public Vector3 move;** : Le vecteur de déplacement du joueur.
- **public Animator anim;** : La référence à l'Animator attaché au joueur.
- **public float TempsAttaque;** : Le temps que dure une attaque.
- **public float compteurAttaque;** : Le compte à rebours pour gérer le temps de l'attaque.
- **public float attackCount;** : Le nombre d'attaques effectuées.
- **public bool isAttacking;** : Indique si le joueur est en train d'attaquer.
- **public AudioSource audios;** : La référence à l' AudioSource pour les sons.
- **public AudioClip[] attackAudios;** : Un tableau d'effets sonores d'attaque.
- **public TextMeshProUGUI cointext;** : La référence au composant TextMeshProUGUI pour afficher le nombre de pièces.
- **public int nbreCoin;** : Le nombre de pièces collectées par le joueur.

## 2. Méthode Update :

- Appelle les méthodes **Marche()** (pour gérer le déplacement) et **NotAttack()** (pour gérer l'attaque) à chaque frame.
- Met à jour l'affichage du nombre de pièces (**cointext**).

## 3. Méthode Marche :

- Récupère les entrées de mouvement du joueur via l'action "Move" définie dans le **PlayerInput**.
- Calcule le vecteur de déplacement en fonction de la vitesse et du temps.
- Applique la vitesse au **Rigidbody2D**.
- Met à jour les paramètres d'animation pour gérer les animations de déplacement.

## 4. Méthode OnAttack :

- Gère l'attaque du joueur. Si le compteur d'attaque est inférieur à 3 et que le joueur n'attaque pas déjà, il déclenche une animation d'attaque et incrémenté le compteur d'attaque.
- Joue un son d'attaque aléatoire si des clips audio sont disponibles.

## 5. Méthode NotAttack :

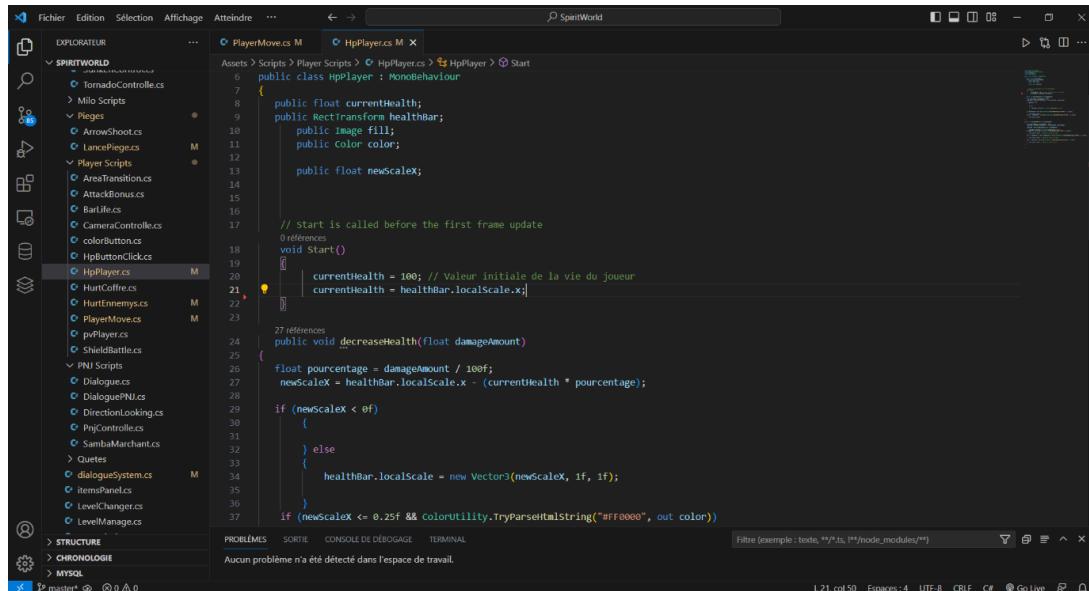
- Gère le comportement du joueur lorsque l'attaque est en cours. Il arrête la vitesse du joueur, décrémente le compteur d'attaque et désactive l'animation d'attaque lorsque le temps d'attaque est écoulé.

## 6. Méthode KnockBack :

- Génère un effet de recul lorsque le joueur subit un impact. Il utilise une position cible et une force pour déplacer le joueur de manière graduelle.

Le script que nous avons créé gère les déplacements, les attaques et les comportements associés du personnage joueur dans le jeu. Il utilise des entrées de mouvement du joueur, des animations, des sons et des interactions pour créer une expérience fluide et interactive.

Passons au script HpPlayer qui gère les points de vie du joueur.



```

public class HpPlayer : MonoBehaviour
{
    public float currentHealth;
    public RectTransform healthBar;
    public Color color;
    public float newscaleX;

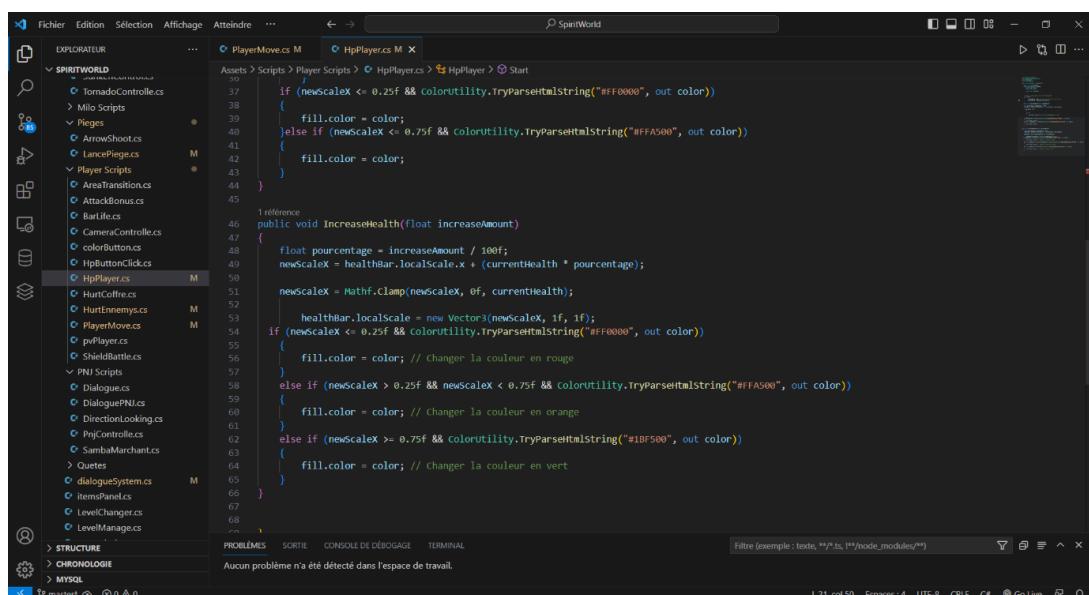
    void Start()
    {
        currentHealth = 100; // Valeur initiale de la vie du joueur
        currentHealth = healthBar.localScale.x;
    }

    public void decreaseHealth(float damageAmount)
    {
        float pourcentage = damageAmount / 100f;
        newscaleX = healthBar.localScale.x - (currentHealth * pourcentage);

        if (newscaleX < 0f)
        {
        }
        else
        {
            healthBar.localScale = new Vector3(newscaleX, 1f, 1f);
        }
        if (newscaleX <= 0.25f && colorUtility.TryParseHtmlString("#FF0000", out color))
    }
}

```

Figure 58 : Méthode DecreaseHealth du Script HpPlayer.cs



```

public class HpPlayer : MonoBehaviour
{
    public float currentHealth;
    public RectTransform healthBar;
    public Color color;
    public float newscaleX;

    void Start()
    {
        if (newscaleX <= 0.25f && colorUtility.TryParseHtmlString("#FF0000", out color))
        {
            fill.color = color;
        }
        else if (newscaleX <= 0.75f && colorUtility.TryParseHtmlString("#FFA500", out color))
        {
            fill.color = color;
        }
    }

    public void increaseHealth(float increaseAmount)
    {
        float pourcentage = increaseAmount / 100f;
        newscaleX = healthBar.localScale.x + (currentHealth * pourcentage);

        newscaleX = Mathf.Clamp(newscaleX, 0f, currentHealth);

        healthBar.localScale = new Vector3(newscaleX, 1f, 1f);
        if (newscaleX < 0.25f && colorUtility.TryParseHtmlString("#FF0000", out color))
        {
            fill.color = color; // changer la couleur en rouge
        }
        else if (newscaleX > 0.25f && newscaleX < 0.75f && colorUtility.TryParseHtmlString("#FFA500", out color))
        {
            fill.color = color; // changer la couleur en orange
        }
        else if (newscaleX >= 0.75f && colorUtility.TryParseHtmlString("#1B8500", out color))
        {
            fill.color = color; // Changer la couleur en vert
        }
    }
}

```

Figure 59 : Méthode IncreaseHealth du Script HpPlayer.cs

### 1. Variables publiques et références :

- **public float currentHealth;** : La santé actuelle du joueur.
- **public RectTransform healthBar;** : La référence au composant RectTransform de la barre de santé.
- **public Image fill;** : La référence à l'image de remplissage de la barre de santé.
- **public Color color;** : La couleur utilisée pour changer la couleur de la barre de santé.
- **public float newScaleX;** : La nouvelle échelle en X de la barre de santé.

### 2. Méthode decreaseHealth :

- Réduit la santé du joueur en fonction du montant de dégâts donné.
- Calcule le pourcentage de dégâts par rapport à la santé maximale.
- Met à jour la nouvelle échelle en X de la barre de santé en fonction du pourcentage de dégâts.
- Restreint la nouvelle échelle en X pour éviter des valeurs négatives ou supérieures à la santé maximale.
- Change la couleur de la barre de santé en fonction de la nouvelle échelle et des seuils (rouge, orange, vert).

### 3. Méthode IncreaseHealth :

- Augmente la santé du joueur en fonction du montant de soins donné.
- Calcule le pourcentage de soins par rapport à la santé maximale.
- Met à jour la nouvelle échelle en X de la barre de santé en fonction du pourcentage de soins.
- Restreint la nouvelle échelle en X pour éviter des valeurs supérieures à la santé maximale.

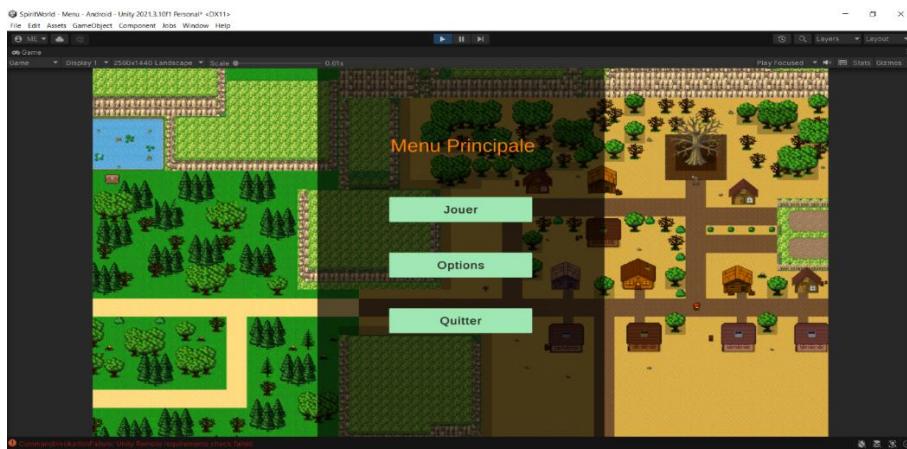
Ce script gère la barre de santé du joueur en mettant à jour visuellement la barre et en ajustant la couleur en fonction de la quantité de santé restante. Il inclut des fonctions pour réduire la santé en cas de dégâts et augmenter la santé en cas de soins. Lorsque l'on récupère une potion dans l'inventaire et qu'on clique sur le bouton de la potion, le bouton va appeler la méthode IncreaseHealth pour augmenter les points de vie de la barre de vie, et si le joueur est touché par un ennemi, l'ennemi va appeler la méthode DecreaseHealth pour réduire la barre de vie du joueur.

## 7.6 Présentation du jeu vidéo

Dans cette dernière section du chapitre, nous allons présenter un petit aperçu du jeu avec des captures d'écran. Après avoir effectué toutes les démarches et ayant coder tous les éléments du jeu, nous pouvons à présent jouer !

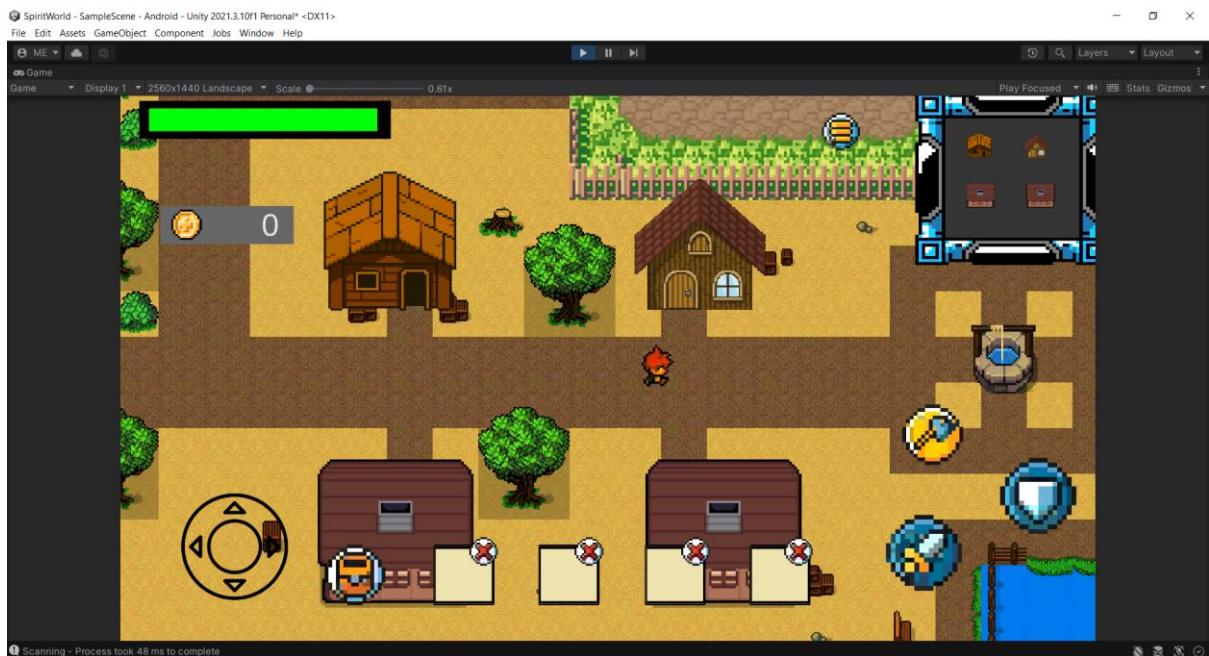
En rentrant dans le jeu, nous sommes dirigés dans le Menu Principale avec plusieurs options

En appuyant sur Jouer, nous démarrons le jeu :

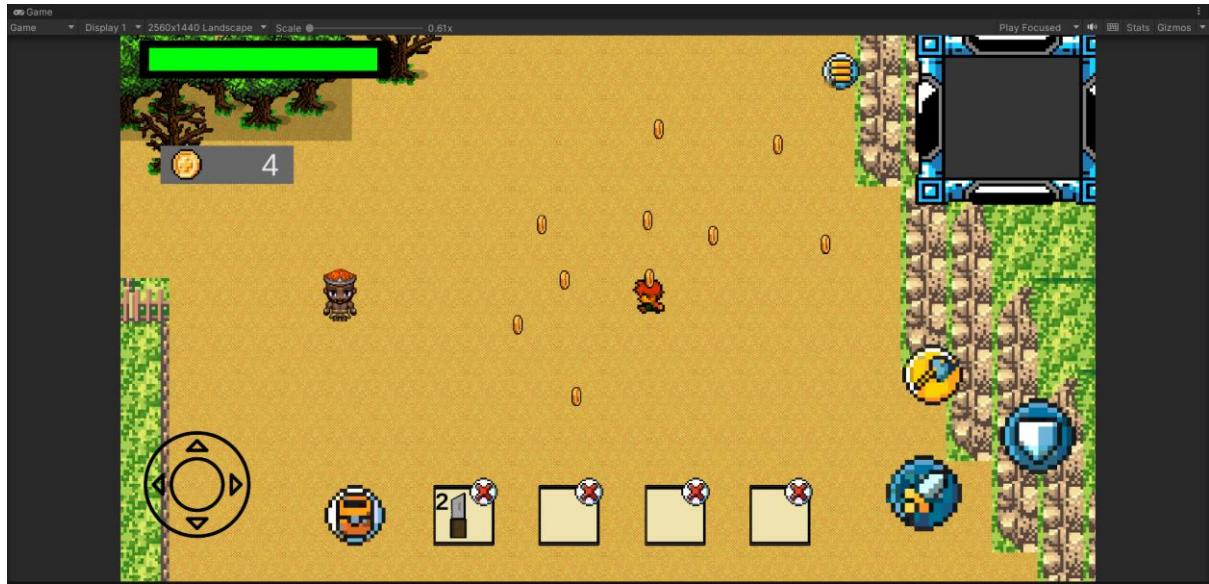
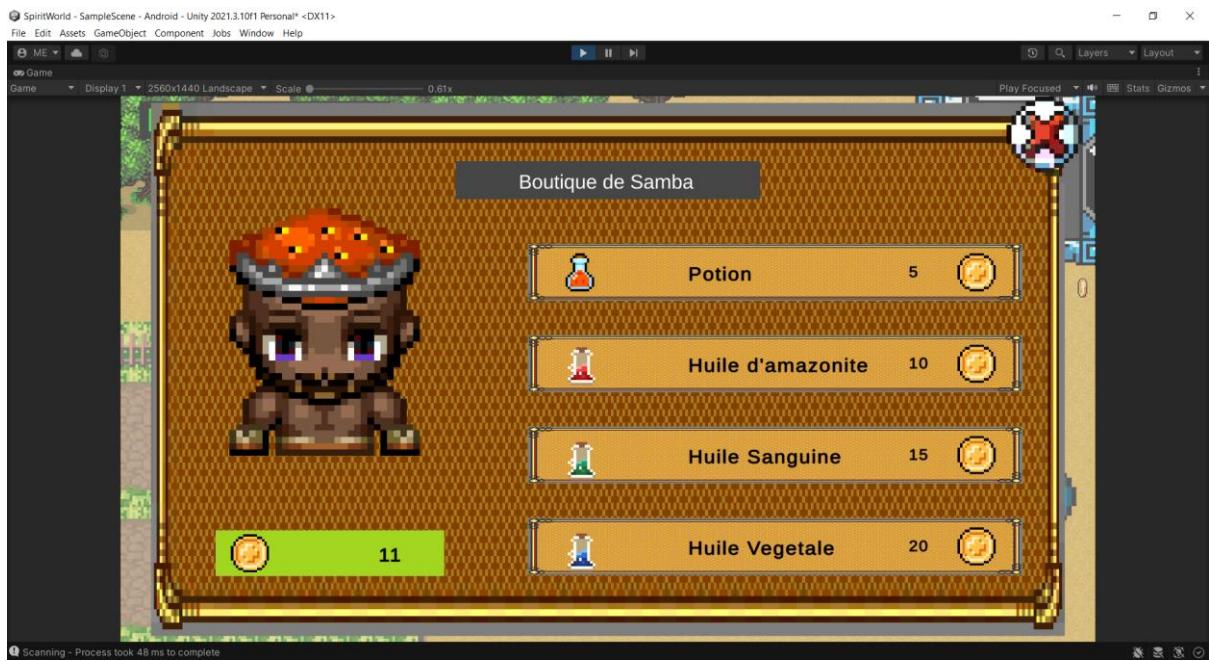


**Figure 60 : Interface du Menu Principale**

Nous commençons dans le village de Tindaba en Casamance et nous pouvons nous déplacer.



**Figure 61 : Déplacement du joueur**

**Figure 62 : Récolte de pièces.****Figure 63 : Interface de la boutique de Samba**

En récupérant des pièces, nous pouvons discuter avec le marchand ‘Samba’ qui possède des potions de régénérations, ainsi que des huiles pour notre épée qui permet d’augmenter les dégâts. Si on achète une potion qui coute (5 pièces) par exemple, le nombre de pièces que l’on possède diminue et le bouton de la potion s’affiche dans notre inventaire. Nous pouvons l’utiliser en cliquant dessus en cas de besoin.



**Figure 64 : Dialogue avec le PNJ ‘Abdou’**

Nous rencontrons un PNJ Abdou avec qui nous pouvons discuter ou démarrer une quête.

Un ennemi Diablotin nous attaque lorsqu'on est à sa portée et nous lance des projectiles, utilisons le bouclier en cliquant sur l'icône du bouclier :



**Figure 65 : Activation du bouclier de protection du joueur**

Grace au bouclier, les projectiles ne touchent pas le joueur.



**Figure 66 : Ennemi ‘Diablotin’ vaincu**

Apres avoir tuer le Diablotin, nous pouvons utiliser notre potion pour remplir nos points de vie :



**Figure 67 : Utilisation de la potion de Soin.**



**Figure 68 : Sélection d'un ennemi pour verrouiller la cible.**

Ici le joueur sélectionne un ennemi pour verrouiller la cible et lui jeter un couteau pour l'attaquer à distance.

Nous avons enfin développé et conçu notre jeu RPG 2D sur mobile. Pour importer notre jeu sur un appareil mobile, nous devons connecter le téléphone Android puis dans Unity on se rend dans File, Build Setting puis on clique sur Build pour construire notre jeu sur un appareil mobile.

## **Conclusion du mémoire**

Au terme de ce mémoire, nous avons plongé dans l'univers dynamique et fascinant de la conception et de la programmation de jeux vidéo pour plateformes mobiles en utilisant la plateforme de jeu UNITY. Nous avons exploré en profondeur les différentes étapes nécessaires à la création d'un jeu mobile Africain, en comprenant les principes fondamentaux de conception ainsi que les défis techniques et artistiques qui émergent lors de ce processus.

L'ensemble de ce mémoire reflète notre engagement envers la création d'expériences ludiques et engageantes qui captivent les joueurs à travers le monde.

L'Objectif de ce projet est de concevoir et réaliser un jeu vidéo 2D sur mobile avec des cultures Africaines, qui a comme but de s'intéresser à des RPG Africain en montrant qu'il serait temps que les jeux moderne s'intéressent plus aux cultures africaines pour en faire des jeux captivants et uniques tout en offrant une nouvelle catégorie de jeu dans les Stores du mobile.

Pour accomplir ce projet, nous avons étudié les différentes étapes de conception d'un univers 2D en s'appuyant sur les graphismes et l'interface utilisateur, et nous avons utilisé des langages et outils de programmations pour la logique du jeu tel que C# avec Visual Studio Code pour l'écriture du code et Git pour la gestion de version des scripts.

En nous lançant dans ce voyage, nous avons non seulement acquis de nouvelles compétences, mais nous avons également contribué à l'évolution de l'industrie du jeu vidéo en apportant un nouveau visage dans l'industrie du gaming Africain.

## Webographie

**Opportunité de l'industrie du jeu mobile :**

<https://jeuxvideopro.fr/les-opportunites-de-l-industrie-des-jeux-video-mobiles/>

**Perspectives de croissance de l'industrie des jeux mobiles :**

<https://lesafriques.com/2021/11/que-reserve-2022-pour-l-industrie-des-jeux-en-afrique/>

**Aurion :** [https://en.wikipedia.org/wiki/Aurion:\\_Legacy\\_of\\_the\\_Kori-Odan](https://en.wikipedia.org/wiki/Aurion:_Legacy_of_the_Kori-Odan)

**Modèle économique courant :** <https://mordorintelligence.com/fr/industry-reports/africa-gaming-market>

**Tendance actuelle :** <https://www.jeuneafrique.com/676370/economie-entreprises/nouvelles-technologies-le-boom-des-jeux-video-sur-mobile-en-afrique/>

**Concurrence :** <https://www.dynamique-mag.com/article/marche-gaming-mobile-sature.6818>

**Difficulté a se démarquer :** <https://siecledigital.fr/2022/08/08/jeux-video-difficultes-q2-2022/>

**Monétisation :** <https://pausetonecran.com/la-monetisation-dans-les-jeux-video-vous-connaisez/>

**Principaux éléments du jeu réussi :** <https://fr.yeeply.com/blog/guide-creer-jeu-mobile-4-etapes/>

**Anthony Cardinal, Professeur en ligne sur la conception de jeu video :**

<https://www.udemy.com/courses/search/?q=anthony+cardinale&src=sac&kw=antho>

**Youtubeur Brackeys :** [www.youtube.com/@Brackeys](https://www.youtube.com/@Brackeys)

**Youtubeur TutoUnityFr :** [www.youtube.com/@TUTOUNITYFR](https://www.youtube.com/@TUTOUNITYFR)

**Youtubeur CodeMonkey :** [www.youtube.com/@CodeMonkeyUnity](https://www.youtube.com/@CodeMonkeyUnity)

## Table des matières

Dédicace .....	I
Remerciement .....	II

Résumé .....	III
Liste des abréviations .....	IV
Glossaire .....	V
Liste de tableau et des figures.....	VI
Sommaire.....	IX
Introduction générale .....	1

## **Première partie : Conception et programmation des jeux plateforme mobile. .... 3**

<b>Chapitre 1 – Présentation générale .....</b>	<b>4</b>
1.1 Présentation du thème.....	4
1.2 Choix et intérêt du sujet.....	4
1.3 Problématique .....	4
1.4 Étapes du processus.....	5

## **Chapitre 2 – Conception et programmation des jeux mobiles .....7**

2.1 Méthodologie et approche adoptée .....	7
2.1.1 Principes de base .....	7
2.1.2 Défis spécifiques.....	8
2.2 Langages de programmation orientée objet.....	8
2.3 Technologies et outils de développement.....	10
2.4 Défis techniques de la programmation de jeux .....	12
2.5 Bonnes pratiques en matière de programmation de jeux.....	13

## **Deuxième partie : Analyse des jeux Mobiles sur l'Afrique. .... 14**

<b>Chapitre 3 – Les opportunités offertes par l'industrie des jeux mobile</b>	<b>15</b>
3.1 Introduction .....	15
3.2 Perspectives de croissance de l'industrie des jeux mobiles .....	15
3.3 Modèles économiques courants.....	18

## **Chapitre 4 – Les défis de l'industrie .....**20

4.1 Introduction .....	20
4.2 Concurrence intense dans l'industrie des jeux mobiles .....	21

4.3 Monétisation équilibrée et éthique des jeux .....	23
<b>Chapitre 5 – Meilleures pratiques pour la conception et la programmation de jeux vidéo mobiles .....</b>	<b>24</b>
5.1 Introduction .....	24
5.2 Les principaux éléments d'un jeu mobile réussi.....	25
5.3 Considérations ergonomiques et de jouabilité du gameplay .....	26
<b>Troisième partie : Réalisation du projet.....</b>	<b>28</b>
<b>Chapitre 6 – Mise en place de l'environnement de développement pour une plateforme mobile .....</b>	<b>29</b>
6.1 Introduction.....	29
6.2 Installation et configuration des outils de développement .....	29
6.3 Importation des Sprites pour l'environnement 2D .....	34
6.4 Architecture du jeu .....	39
<b>Chapitre 7 – Développement du jeu mobile en 2D .....</b>	<b>43</b>
7.1 Introduction.....	43
7.1.1 Synopsis .....	43
7.2 Création du personnage et de l'environnement RPG 2D .....	44
7.3 Animation de l'environnement et des personnages.....	51
7.4 Création des entités autonome ( IA ).....	55
7.5 Programmation du joueur .....	63
7.6 Présentation du jeu vidéo .....	66
Conclusion du mémoire .....	72
Webographie.....	73
Table des matières.....	74