République Tunisienne Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université de Sousse

Institut Supérieur des Sciences Appliquées et de Technologie de Sousse

DEPARTEMENT INFORMATIQUE

RAPPORT DE STAGE DE FIN D'ETUDES

En vue de l'obtention du diplôme de Licence Fondamentale en :

Sciences de l'Informatique

Option : Sciences de l'Informatique

Unity - Editeur Game Design

Elaboré par :

Hedi MLIKA

Encadré par :

Madame Maha KHEMAJA

Institut Supérieur des Sciences Appliquées et de Technologie de Sousse

Année Universitaire : 2018 /2019

Code Sujet : LF-SI19-030

Remerciments

Avant tout développement, il apparaît opportun de commencer ce rapport de stage par des remerciements, J'adresse mes sincères remerciements aux personnes qui m'ont aidé tout au long de ces quatre mois à la réalisation de ce projet.

En premier lieu, je remercie Mme Maha Khemaja, encadrante de l'Institut Supérieur des Sciences Appliquées et Technologies de Sousse, qui m'a donné de son temps ainsi que sa confiance pour accomplir la réalisation de ce projet avec beaucoup de patience et de pédagogie et pour ses conseils qui m'ont été d'un grand soutien.

De même, je tiens aussi à remercier tous les membres du jury qui m'ont honoré pour avoir accepté de prendre le temps de juger mon travail.

Enfin, je tiens à exprimer mon amitié et mon respect profonds envers ma famille, l'incubateur de cet institut et à tous ceux qui ont contribué de près ou de loin à la réussite et à la concrétisation de ce projet.

Je dédie ce projet à mes parents, ma sœur et à tous ceux qui m'aiment.

SOMMAIRE

LISTE DES TABLEAUX

LISTE DES FIGURES

ABSTRACT

Français

Ce projet consiste dans la création d'une extension de Unity3D partir d'une classe de Unity qui permet la création de modules et de fenêtres supplémentaires.

Cette extension va permettre au développeur, d'une part, de décrire son game design et de traduire son game design en objets préfabriqué automatiquement dans Unity.

D'une autre part, l'extension permettra au développeur de gérer les niveaux de manière visuelle et interactive en ayant la possibilité d'ajouter tout objet décrit dans le game design. Le développeur pourra relier chaque niveau créé grâce à des conditions préétablis que le joueur devra remplir pour passer au suivant.

En d'autres termes, le projet vise à rendre le moteur de jeu Unity orienté game design.

Mot clés: Game design, Unity, Extension

English

This project consists of creating an Unity3D extension from a Unity class that allows the creation of additional modules and windows.

This extension will allow the developer, on the first hand, to describe his game design and to translate his game design into prefabricated objects automatically in Unity.

On the second hand, the extension will allow the developer to manage the levels in a visual and interactive way by having the possibility to add any object described in the game design. The developer will be able to link each created level with pre-established conditions that the player will have to pass successfully to go to the following one.

In other words, the project aims to make Unity game engine game design-oriented.

Keywords: Game design, Unity, Extension

Bonne lecture

INTRODUCTION GENERALE

Un jeu vidéo (en anglais : video game) est un programme à but ludique qui se joue sur un appareil électronique doté d'une interface utilisateur permettant une interaction entre le joueur et un monde virtuel. Le joueur a en sa disposition des périphériques pour agir et réagir sur le jeu.

Le jeu vidéo se jouait au début sur des bornes d'arcade et il a évolué pour être jouable sur PCs, consoles et consoles portables et plus récemment sur smartphones et casques a réalité virtuelle.

Au fils de son histoire, le jeu vidéo a grandement contribué à l'évolution de la technologie et à la recherche scientifique ; notamment en 3D, en puissance de calcul, en réseaux, en supports de stockage et en simulations…etc.

Ce domaine souffre malgré lui de fortes controverses, car le jeu vidéo -comme loisir et phénomène de masse- soulève des interrogations et des critiques. Le jeu vidéo est à l'échelle de l'histoire des sociétés humaines une activité récente, les parents d'enfants nés dans les années 90 n'ont pour la plupart jamais joué à ce type de jeu dans leur enfance ou adolescence. [Voir Ref ig1, Page 57 : *Jeux vidéo*].

Un développeur de jeu vidéo est une personne physique ou morale comme une société ou une startup qui crée des jeux vidéo en rassemblant plusieurs domaines de compétences comme le game design, la programmation, la scénarisation, l'infographie, la musique, etc. Le développeur de jeux vidéo peut travailler au sein d'une entreprise, d'un collectif ou être indépendant.

Un développeur de jeu vidéo utilise divers outils pour créer un jeu comme des langages de programmation, des logiciels de graphisme ou des moteurs dédiés à la création de jeux.

Les moteurs de jeu sont de plus en plus utilisés, évitant de réinventer la roue et de réécrire indéfiniment du code commun à tous les jeux vidéo. Ils peuvent être sous forme d'API (CryEngine, LÖVE, Unity), ou bien d'outil graphique comportant des langages de programmation, comme dans le cas de Godot.

CHAPITRE PREMIER

Contexte Général

Ouverture

Ce rapport est divisé en 4 chapitres,

Celui-ci, le premier chapitre, est consacré au cadre général du projet et le problème à résoudre, c'est un chapitre introductif qui permet de contextualiser les chapitres suivants.

Le deuxième chapitre est réservé à l'étude générale du projet, l'analyse et à la spécification de besoins fonctionnels et non fonctionnels du projet et la méthodologie du travail choisi.

Au troisième chapitre, nous exposerons la partie conceptuelle de notre système.

Le dernier chapitre abordera la réalisation technique du projet, qui comprend les outils et techniques utilisés pour obtenir un produit final ou expérimental.

I – Cadre général

Le présent travail fait partie de la préparation du projet de fin d'études, dans le but d'obtenir une *licence fondamentale en sciences de l'informatique* de l'enseignement supérieur du prestigieux *Institut des sciences appliquées et de la technologie de Sousse*.

Ce stage s'est déroulé dans l'institut susmentionné pour une période de quatorze semaines à compter du 1er Février au 15 Mai 2019.

Ce projet comprend la conception et le développement d'une extension du Moteur de jeu Unity3D qui permet à un novice de mieux appréhender les bases du moteur grâce à des infobulles et à un expert d'accélérer la première phase de développement et le game design grâce à un éditeur de niveau intégré.

II – Contexte et thèmes à aborder

Afin de mieux se lancer dans le vif du sujet, une halte sur quelques termes qui seront utilisés tout au long du projet. Ces « termes » décrivent les thèmes généraux du

projet et seront très utile pour la compréhension des chapitres suivants dans but de mieux les appréhendés.

II .1 – Moteur de jeux

Un Moteur de jeu est un ensemble de fonctions, permettant le développement d'un jeu, avec accès à des fonctions haut-niveau directement (de la sorte, on masque toutes les opérations bas-niveau). De plus, il doit fournir une base, sur laquelle le programme est capable de tourner de manière autonome, jusqu'à ce que son arrêt soit requis.

On parlera alors de squelette, qui sera complété par tout jeu utilisant le moteur.

Le Moteur de jeu doit permettre de faire abstraction de toutes les contraintes liées à l'environnement, et ainsi faciliter le développement de jeux par la suite. En résumé, l'utilisateur du moteur (le développeur) doit se contenter de fonctions simples (chargements, rendus, mises à jour...). Cependant, plus le Moteur de jeu rend l'accès haut-niveau, moins l'utilisateur a de contrôle dessus (à moins de modifier directement le moteur). [Voir Ref 1.1, Page 58 : *Définissions sur les moteurs de jeux*].

II .2 – Unity Engine

Unity3D est un puissant moteur 3D multiplateforme et un environnement de développement convivial. Assez facile pour le débutant et assez puissant pour l'expert ; Unity devrait intéresser tous ceux qui souhaitent créer facilement des jeux et des applications 3D pour mobiles, ordinateurs de bureau, Internet et consoles. Le logiciel a la particularité d'utiliser un éditeur de script compatible mono (C#), UnityScript (un langage proche du JavaScript et inspiré d'ECMAScript et arrêté depuis la version 2017.2) et Boo (arrêté à la version 5.0) au lieu de Lua très utilisé dans les jeux vidéo.
Nous avons choisi Unity pour plusieurs raisons :

- Sa popularité par rapport à ses confrères et compères.
- Sa scalabilité et son extensibilité.
- Sa communauté
- Son potentiel en termes de création de jeu (game design)
- Ses mises à jour.

[Voir Chapitre 4 : *Réalisation*, Page 51]

II .3 – Game Design

Le game design — la conception de jeux en français — est le processus de création et de mise au point des règles et autres éléments constitutifs d'un jeu. L'expression, qui s'applique à tout type de jeu. L'une des règles du game design spécifique au jeu vidéo est la « règle des 3 C » pour Camera (Caméra), Control (Commandes), Character (Personnage). [Voir Ref 1.2, Page 58 : *Le game design*].

Elle définit ces trois éléments interdépendants et considérés comme cruciaux dans l'élaboration du système de jeu. Le principal travail du game designer est de concevoir un document intitulé « game design document » qui regroupe toutes les règles et propriétés du jeu.

C'est le cahier des charges des game designer et qui sera délivré au reste des développeurs s'il y en a.

II .4 – Développeur

Dans le monde du jeu vidéo, la notion de développeur englobe plusieurs professions et ne se réfère pas uniquement aux programmeurs.

A titre d'exemple :

- Un programmeur, un designer, un artiste : **Sont des développeurs.**
- Un expert marketing, Un directeur d'opération, un analyste : **Ne sont pas des développeurs**.

III – Problématique

Pour les éditeurs et développeurs du secteur du jeu vidéo, le marcher semble être une véritable mine d'or, mais le meilleur moyen de s'en assurer c'est de prendre le temps d'analyser ce marcher.

Pour ça, le mieux c'est de se renseigner auprès de l'agence française pour le jeu vidéo.

Leur site internet est l'un des meilleurs catalogues de sources qui références beaucoup d'articles sur le sujet.

Fondée en 2003, l'agence française pour le jeu vidéo (abrégée : AFJV) est une société filiale de Chips Interactive centrée sur l'information, l'emploi, les entreprises et les professionnels d'œuvres multimédia et du secteur du jeu vidéo.

Sa mission et de promouvoir les innovations, les créations techniques, artistiques et intellectuelles, favoriser l'emploi et la création d'entreprise au sein de la communauté des concepteurs, producteurs, éditeurs et distributeurs d'œuvres multimédia liés au secteur du jeu vidéo principalement.

Figure 1.1 : Logo de l'AFJV

D'après l'agence française pour le jeu vidéo et une étude de la part de la NPD[1] : Les dépenses suivies relatives au matériel (Consoles et PC), aux logiciels (Jeux vidéo) et aux accessoires (Périphériques de gaming), totalisaient 842 millions de dollars en avril 2019, soit un gain de 1% par rapport à l'année dernière. La croissance des dépenses en logiciels et en accessoires a compensé le recul du matériel.

Les ventes en dollars de logiciels de jeux vidéo pour consoles, portables et pour PC suivis ont atteint 427 millions de dollars en avril 2019, en hausse de 15% par rapport à l'année dernière. Mais depuis le début de cette l'année de 2019, les ventes en dollars de logiciels de jeux vidéo pour consoles, portables et PC ont atteint 1,9 milliard de dollars, soit 4% de plus par rapport à l'année dernière. [Voir Ref 1.3, Page 58 : *Rapport sur les ventes de jeux vidéo par l'NPD Group*].

[1] NPD: Market Research Company

Le jeu vidéo jouit d'un chiffre d'affaire (mondiale et par pays) assez conséquent et c'est ce qui le rend intéressant. [Voir Ref 1.4, Page 58 : *Revenu du secteur américain du jeu vidéo* et Ref 1.5, Page 58 : *Bilan annuel du marché français du jeu vidéo*]

Il y a toujours une corrélation entre dépense et vente, la récente étude de la NPD montre clairement que ce marché est à la hausse ; la consommation augmente ce qui pousse les entreprises à produire plus chaque année. Mais cela montre aussi, de façon indirecte, que le besoin en termes de main d'œuvre qualifié est à la hausse aussi.

Et le problème vient de là.

Pour développer un jeu vidéo on aura besoin de diverses qualifications ; du programmeur à l'artiste 3D, du game designer à l'analyste de marché, de l'expert en histoire à l'expert en psychologie…etc., mais si on veut développer un autre jeu, on aura surement besoin de remodeler les équipes. Cette industrie souffre ces deux dernières années d'un fort taux de licenciement, ce qui pousse beaucoup de développeurs à plier baguage pour une autre société, voire de se lancer en indépendant. C'est mon premier point.

Mais il y a autre chose, Dans cette industrie qui est caractérisée depuis le début des années 2010 par la progression affolante du nombre de jeux, le paysage du monde du jeux vidéo a grandement changé avec l'arrivée de studios et éditeurs de taille plus modeste de plus en plus puissants.

En toute objectivité, le domaine en lui-même est assez attirent, beaucoup de développeurs se sentent pousser des ailes et se lancent en indépendant dans une carrière solo, dans des PME[2] ou dans des Startups.

Cette industrie touche donc un très grand nombre de métiers différents qui -pour la grande majorité de ces développeurs à en devenir- n'ont aucune notion de programmation.

Pour un level designer, un artiste ou un testeur voulant travailler à son compte, la tâche est très ardue. En effet, outre le fait qu'il doit travailler à un rythme soutenu, il doit

[2] PME : Petite / Moyenne Entreprises

commencer à apprendre des compétences qu'il n'a jamais eu l'occasion de s'y former lors de son cursus, et notamment la programmation. C'est mon second point.

Mon premier et mon deuxième point montrent que les jeux vidéo indépendants deviennent de plus en plus « à la mode » mais les produits finaux souffrent dans la plupart du temps d'un manque - d'une part : de détails, de recherche et de réalisme qu'un game designer apporte et d'une autre part : de forme, de technique et de rigueur qu'une autre catégorie de développeur apporte.

C'est pour ça qu'on va essayer de créer un outil qui permet de consolider les deux profils et de subvenir aux besoins des travailleurs indépendants.

Mais comment améliorer la qualité des jeux vidéo indépendants ?

Conclusion

L'industrie du jeu vidéo atteint de nouveaux sommets ces deux dernières années car les jeux vidéo et les contenus associés sont devenus la nouvelle source de divertissement pour quasiment tout le monde mais requière néanmoins beaucoup de savoir-faire dans plusieurs domaines différents.

Ce chapitre est une partie introductive dans laquelle un aperçu de la problématique du sujet a été introduit.

Nous avons présenté le contexte général du projet et les thèmes le reliant.

Nous présenterons dans le chapitre suivant une analyse plus concrète de cet outil que j'ai mentionné juste avant de conclure et du marché auquel il s'y attache ainsi que les besoins fonctionnels et non fonctionnels.

CHAPITRE DEUXIEME

Analyse des besoins et Spécifications

Introduction

Dans ce chapitre nous allons faire une étude de l'existant afin, dans un premier temps, avoir une idée de ce qui se fait en matière de game design et dans un second temps, exposer une solution qui satisfera la problématique.

I – Etude du Marché

Dans cette section, nous présenterons une étude basée sur des observations de différentes applications similaires à l'extension que nous allons vous présenter.

Tous comme notre extension, les applications que nous allons citer devront avoir la possibilité de :

- Designer des niveaux et des objets de façon visuelle et intuitive
- Les niveaux et objet créés pourront être facilement réutilisés dans la programmation d'un jeu.
- Eviter la traduction du game design. Le game design et la programmation doivent être incluent dans une seule et unique étape.

Cette étude va nous permettre d'identifier les forces et les faiblesses de différentes solutions existantes et d'identifier les besoins pour y répondre.

I .1 – Solutions existantes

A – Twine 2.0

Figure 2.1 : Logo de Twine 2.0

Twine 2.0 est un outil gratuit et en opensource créé par Chris Klimas pour la réalisation et la conception d'œuvres de fictions interactives sous la forme de pages Web. [Voir Ref 2.1, Page 58 : *Introduction à Twine 2.0*]

Caractéristiques

C'est actuellement l'outils de game design le plus utilisé sur le marché, il permet de :

- Créer et nommé un jeu (histoire)
- Créer un niveau (passage)
- Editer un niveau en y ajoutant du texte (l'histoire du niveau)
- Relier des niveaux entre eux
- Jouer l'histoire en mode test
- Lancer l'histoire
- Exporter sous format web

Avantages

L'un des avantages majeurs de Twine c'est son existence même, le manque de programmes qui unifient création de jeux et game design son très rares car la plupart des game designer opte pour le crayon et le papier. Mais Twine offre aussi une interface ultra simpliste et facile à utiliser, il est compatible sur Windows, OSX et Linux et offre même une interface Web identique grâce à ElectronJS.

Twine permet de relier les niveaux entre eux grâce à un système de script propre à lui. Il existe plusieurs types de script différents développés par Twine et/ou sa communauté, certains sont faciles mais limités aux simples conditions (du *Si Sinon*), d'autres difficile mais qui ne se limite qu'à la créativité des développeurs car elle se rapproche de la programmation procédurale tel qu'on la connait.

Twine subit constamment des mises à jour et possède aussi un mode nuit qui assombrie l'interface utilisateur qui est très pratique pour une longue utilisation et éviter d'avoir mal aux yeux.

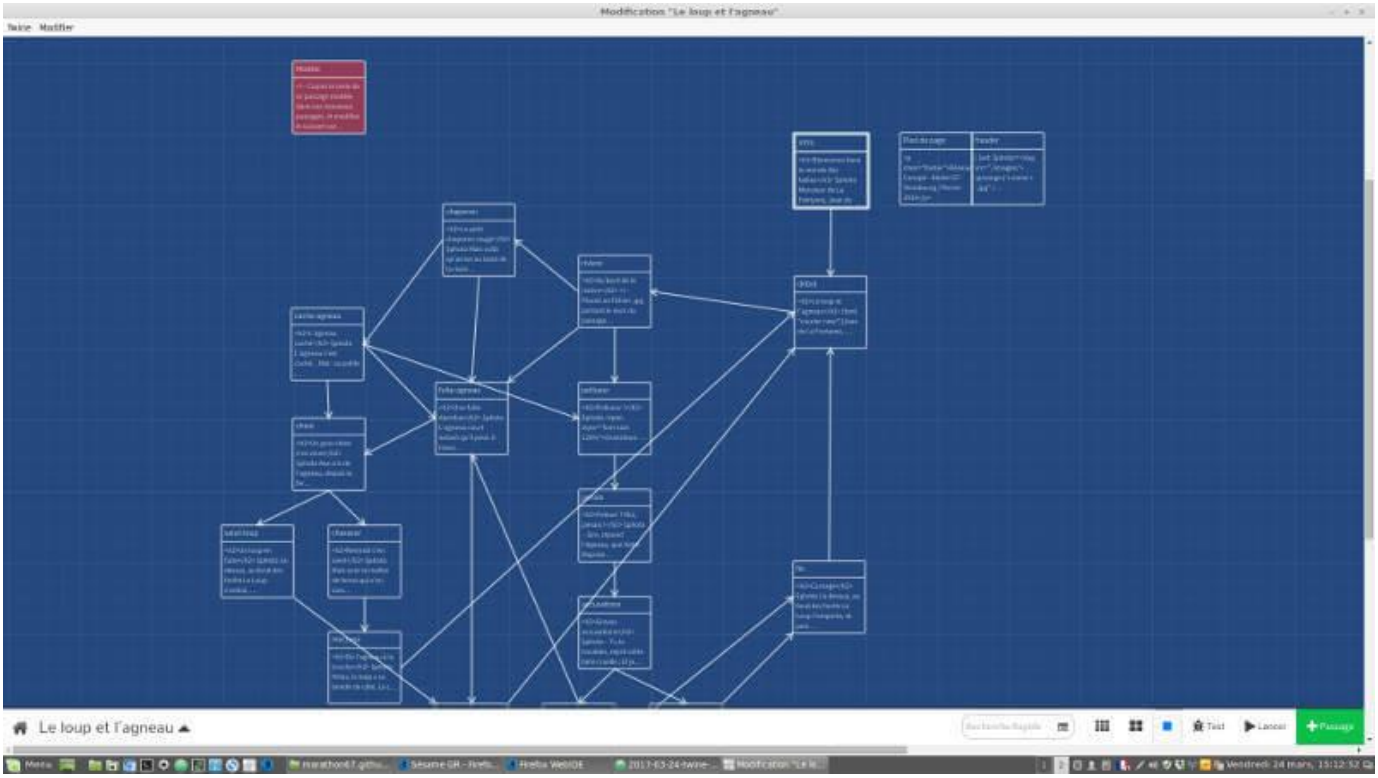

Figure 2.2 : Histoire et design du loup et l'agneau dans Twine

Inconvénients

Twine est un programme qui ne se limite qu'au jeux textuels et ne propose pas de moteur graphique.

Twine ne possède aucune aide à l'utilisateur visible dans son interface, le développeur peut vite lâcher prise s'il ne s'est pas déjà documenter.

Twine est un logiciel tier qui n'est pas intégré dans un moteur plus grand (Unity3D / Unreal Engine ...etc.). Il est certes utile, mais compétemment dispensable aux profits du cayon et du papier. Il ne propose pas non plus de réaliser une exportation vers ces derniers moteurs.

B – Diamond Visual Scripting

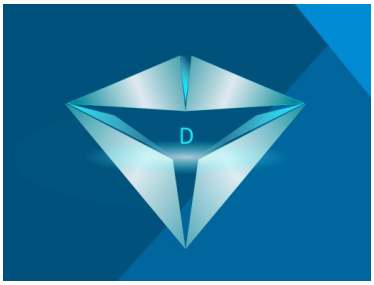

Figure 2.3 : Illustration officielle de diamond Visual Scripting

Diamond Visual Scripting est une extension de Unity3D qui permet d'ajouter des composants souvent récurant dans le code du jeu.

Caractéristiques

- Ajouter des publicité Unity.
- Un système d'évènement indépendant de Unity mais plus simple.
- Un accès direct au propriétés (matériaux, substances… etc.)
- Un transfert de données qui permet la communication des scripts générés.

Avantages

L'avantage de DVS[3] est sa polyvalence. C'est un produit qui permet de faire pleins de choses différentes au sein même de Unity, sans passer par l'inspecteur de Unity (Inspector)[4].

Il offre un gain de temps sur certaines phases -parfois obligatoires- du développement en quelques clics

Inconvénients

Ses fonctionnalités sont certes innovantes et pleines de bonne intentions mais son UI -radicalement différente de celle que Unity propose- rend la tâche beaucoup plus complexe.

De plus si le développement du jeu ne comporte aucune des fonctionnalités que DVS propose, l'interface devient inutile car DVS se concentre que sur des cas isolés du développement.

DVS est une extension ardue et pas facile à prendre en main pour les néophytes et comporte un certain nombre de bugs visuels qui peuvent ruiner l'expérience utilisateur pour un programme qui coute $40 sur l'asset store de Unity.

Voici un graphique comparatif des prix dans la catégorie « Editor Extension » dans l'Asset Store de Unity.

[Voir Ref 2.2, Page 58 : *Prix catégorie Editor Extension*]

[3] DVS : Diamond Visual Scripting

[4] Inspector : L'Inspecteur affiche des informations détaillées sur votre GameObject actuellement sélectionné.

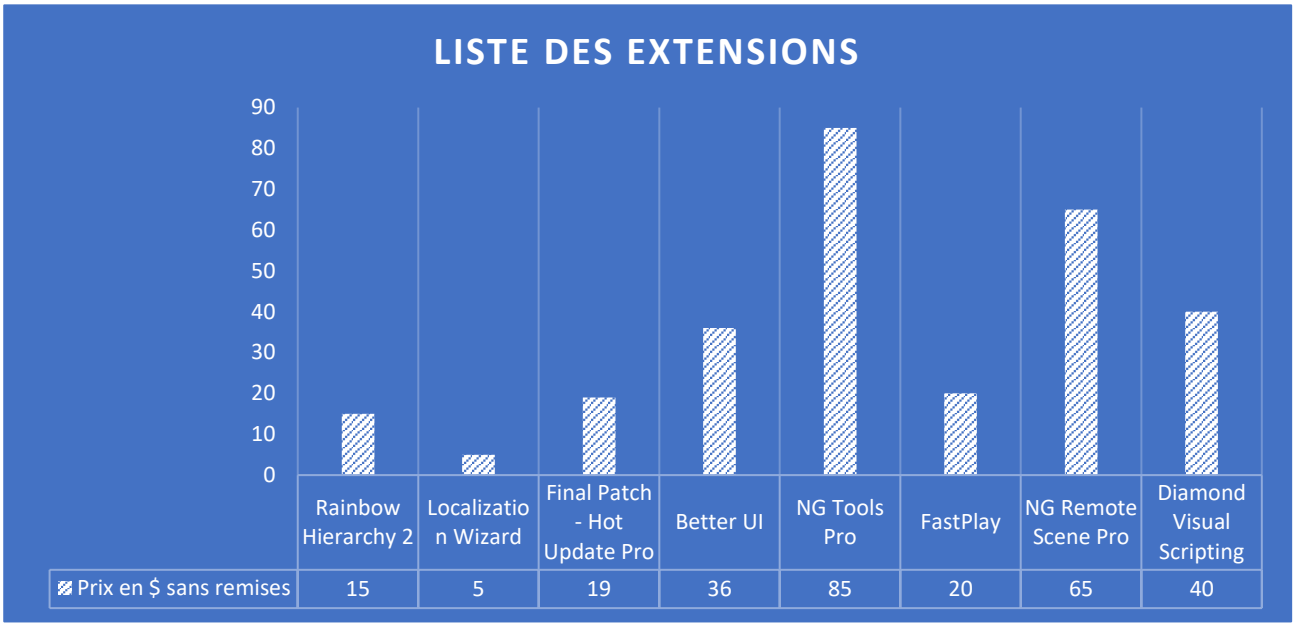

Figure 2.4 : Graph comparatif des extensions présentent dans l'Asset Store

I .2 – Critique de l'étude des solutions existantes

On peut d'ores et déjà dégager nos besoins depuis ces deux exemples.

L'exercice du game design se pratiquant généralement au crayon/papier, on trouve néanmoins des applications qui digitalise cette activité voir d'améliorer certains aspects, comme la possibilité de créer des conditions et de les visualiser. Mais la tâche du game designer et du level designer est compliquée pour un expert du domaine et négligée pour un novice et digitaliser la solution ne va pas nous rendre plus conscient de l'importance de cette activité et inversement, être un expert du design ne va le rendre apte à programmer des jeux d'envergure.

Pour ce qui est de la question du chapitre précèdent : *comment améliorer la qualité des jeux vidéo indépendants* ? C'est fusionner les activités de design et de programmation pour :

1. Ne pas perdre de temps de développement.
2. Prendre le temps de faire le design pour ne pas se lancer dans une aventure de développement en aveugle.
3. Rendre les deux activités complémentaires et accessible à tous.

La plupart des solutions existantes répondent qu'en partie aux besoins qui se dégagent. Les solutions Twine et diamond Visual Scripting sont les plus populaire mais sont pour l'un, dispensable et n'est limiter qu'à un seul type de jeu et l'autre est assez brouillon, n'est réservé qu'à une classe de programmeur d'élite et inutile dans beaucoup de cas.

Ces solutions ne prennent pas totalement en considération l'aspect du game design/level design et ne le traduit pas directement en code.

I .3 –Bilan de l'étude des solutions existantes

Si j'ai choisi d'exposer ces deux solutions c'est pour pointer du doigts deux choses :

1. Le manque sur le marché de logiciels orientés game design.
2. La redondance de certaines tâches dans Unity et les rendre plus visuels.

La solution que nous proposons c'est d'utiliser les avantages que propose Twine dans une extension Unity.

La solution devra être accessible à tout le monde et devrait rester utile aux programmeurs expérimentés.

Pour ce fait et vu qu'elle est surtout dédiée au game design, la solution devrais se concentrer sur les premières phases du développement du jeu sur Unity uniquement.

L'interface devrai suivre les codes UI/UX établi par Unity Technologies pour son logiciel (Unity3D).

La solution ne doit en aucun cas être utilisée en dehors de Unity et donc, doit y être incorporer et faire d'une pierre deux coups : éviter la redondance du travail et permettre un meilleur apprentissage et habituer les novices à l'interface de Unity.

II – Spécification des besoins

Comme nous voulons développer une interface intégrée dans le moteur de jeu Unity, le plus optimale est de créer une extension de ce dernier. Ici, nous allons présenter les acteurs en jeu, spécifier les besoins décrire le fonctionnement du système.

II .1 – Les acteurs

Un acteur est une entité externe qui interagit avec le système (opérateur, centre distant, autre système…). En réponse à l'action d'un acteur. Le système fournit un service qui correspond à son besoin. Les acteurs peuvent être classés (hiérarchie).

Le développeur Unity3D

C'est la personne primordiale du système et l'unique acteur qui peut interagir avec celui-ci. Il permet de :

- Créer un Modèle de Game design.
- Remplir le modèle de game design avec des éléments comme des scènes des types d'objet, des types de personnages non joueurs (PNJ)…etc.
- Editer son modèle de game design.

Je n'inclurai pas le joueur en tant qu'acteur pour deux raisons, la première est vu que l'extension n'est destinée qu'aux premières phases du développement du jeu, le développeur aura surement beaucoup de taches préalables à réaliser avant de publier son jeu et certains de ses GameObjets ne seront plus pris en compte par l'extensionsLa deuxième est que le processus de la publication est dépendant de Unity3D et non de l'extension.

II .2 – Les besoins fonctionnels

Vu que c'est une extension Unity et qui s'adresse aux développeurs (L'acteur). Notre extension doit répondre aux exigences suivantes :

- **Créativité :** L'extension doit permettre au développeur de créer ce qu'il veut à partir de l'extension dans Unity et ne doit pas le ralentir ou l'interdire de faire ce que Unity propose déjà.
- **Flexibilité :** L'extension doit permettre au développeur d'hériter nos classes, redéfinir/surcharger ses méthodes, de modifier et d'ajouter certains composants.

II .3 – Les besoins non fonctionnels

Mis à part les différents besoins fonctionnels soulignés, notre extension doit tout de même répondre à d'autres critères qui serait maladroit de négliger et doit quand même se conformer à quelques standards de qualité :

- **Fluidité :** Notre extension doit être le plus fluide possible et ne doit pas poser de problèmes de performance à Unity.
- **Intégrité :** L'extension ne doit pas contenir de surcouche graphique, elle doit rester par défaut pour être le plus fidèle possible à l'interface Unity.

II .4 – Fonctionnement Général et cas d'utilisations

Dès l'ouverture de Unity3D et de l'onglet « Level Editor », La première étape pour le développeur consiste à choisir son modèle de game design (Game design Template).

Vu que le game design est une formalisation de l'idée du jeu (Un peu comme un cahier des charges et UML réuni), l'extension lui permet d'inclure ou non certaines parties du game design.

La première étape est sous forme de formulaire à cases à cocher lui permettant de choisir s'il veut inclure des éléments suivant au game design :

- Nom du jeu.
- Analyse du jeu.
- Genre et thème.
- Plateforme.
- Audience cible.
- Personnages et histoire.
- Gameplay (En français : Jouabilité).
- Esthétique du jeu.

Nom du jeu et Gameplay sont obligatoire est sont cochés d'office. Elles apparaissent pour lui montrer qu'elles existent.

Ne rien cocher c'est passer outre l'extension. L'extension ne doit pas obliger son utilisateur à l'utiliser.

La deuxième étape consiste à remplir les huit (ou moins) éléments cités ci-dessus.

Nom du jeu : Un champ de texte avec le nom du jeu comme output.

Analyse du jeu : Un champ de texte avec le pitch du développeur, il doit contenir dans quelques lignes seulement un descriptif du jeu. Contient aussi la liste des niveaux et leurs noms.

Genre et thème : une liste contenant les différents genres (Jeu de rôle, action, simulateur…etc.) et thèmes (Futuriste, Zombie, Educations, Histoire…etc.).

Plateforme : Une liste de sélection de la plateforme de sortie (Playstation 4, PC, Mac, Linux, XBOX One, Nintendo Switch, Androïd, iOS, Web et Smart TV).

Audience Cible : Champ de texte pour décrire son audience cible (Ex : l'Age requit)

Personnages et histoire : permet de créer des types de personnages non-joueurs (Boss, Ennemie 1, Ennemie 2, Ami, Vendeur…etc.) avec les propres mots du développeur. Cette catégorie permet aussi de créer des personnages et les assigner à un type et à un niveau.

Gameplay : Permet de décrire le monde et les règles du jeu :

Vue 2D du jeu, Mouvements principaux du joueurs et gravité et actions principales du joueur (tirer, voler, courir).

Esthétique du jeu : permet de décrire la charte graphique et l'interface du joueur.

Vu que l'extension est destinée principalement aux game designers, l'extension offre la possibilité pour les huit éléments de créer automatiquement le Game Design Document, le programme créera un document PDF avec toutes les informations remplit qui survivera comme cahier des charges au développeur.

Arrivés à la troisième étape, l'extension va générer une vue subjective du Game design dans des scènes avec -dans les scènes- les objets et personnages crées.

Cette « vue » des scènes est sous forme de nœuds qu'on pourra relier entre elles, ajouter de nouvelles, supprimer certaines et modifier leurs contenus.

La figure suivante illustre le diagramme des cas d'utilisation générale de mon extension.

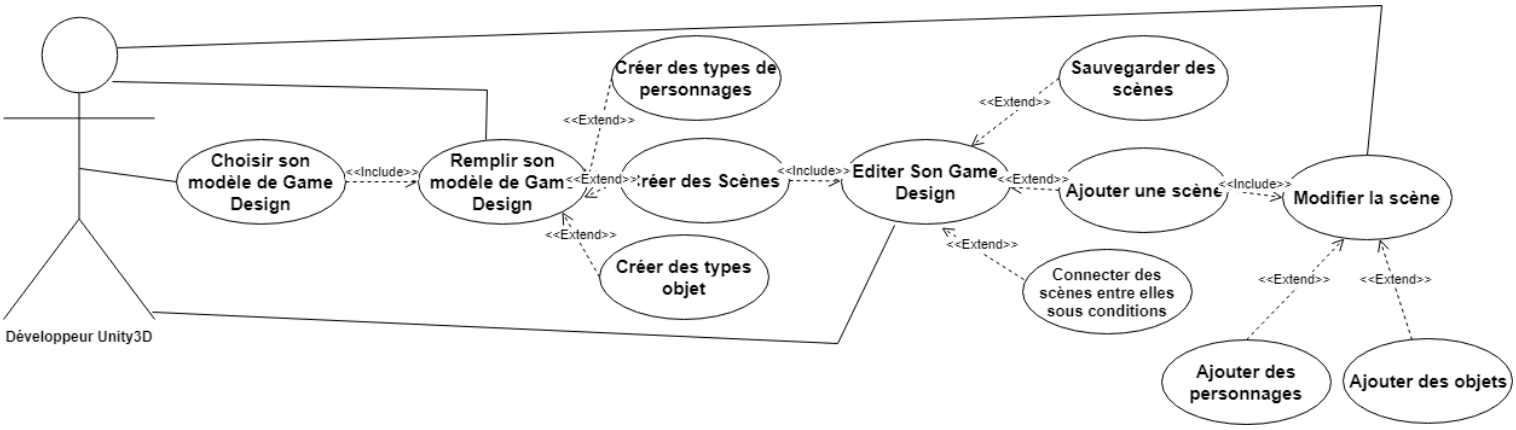

Figure 2.5 : Diagramme de cas d'utilisation général

III – Analyse et Démarche à suivre

Développer une extension pour Unity est radicalement différent du développement d'un jeu sur Unity.

Unity permet de créer des classes (plus connues sous le nom de « scripts ») C# grâce à MonoBehaviour[5]. Mais cette classe ne nous permet pas d'aboutir et de répondre aux besoins soulevés.

```
NewBehaviourScript.cs ×
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class NewBehaviourScript : MonoBehaviour
6   {
7       // Start is called before the first frame update
        0 references
8       void Start()
9       {
10      
11      }
12
13      // Update is called once per frame
        0 references
14      void Update()
15      {
16      
17      }
18  }
19
```

Figure 2.6 : Script Unity de base

[5] MonoBehaviour : C'est la classe de base à partir de laquelle chaque script Unity dérive

Pour y remédier il faut commencer par modifier *MonoBehaviour* par *EditorWindow*.

EditorWindow ne nous permet pas de créer des jeux mais de modifier l'interface de Unity (en créant de nouvelles fenêtres par exemple). Les méthodes comme Start() et Update() seront dès alors obsolète vu qu'elle interagissent avec un jeu et non une fenêtre personnalisée.

EditorWindow est un excellent choix qui va nous permettre de créer notre extension et répondre à nos besoins.

IV – Etude de la méthodologie de travail

La méthodologie de travail est le processus de développement / séquence d'étapes ordonnées pouvant aboutir à un logiciel complet, une application web ou mobile, une extension d'un logiciel existant ou la modification d'un système existant. Le but est de produire en ayant un certain standard qualité et qui doit répondre aux différents besoins des utilisateurs finaux tout en respectant les coûts en temps de développement.

IV .1 – Les différentes méthodologies de travail

NB : *J'utiliserai les noms anglais des méthodologies pour éviter toutes confusions*

Méthodologie	Description	Avantages	Inconvénients
Waterfall	Comme son nom l'indique, l'approche en cascade suit la logique d'une chute d'eau. Une fois que l'eau a dévalé le flanc de la montagne, elle ne peut plus remonter, mais seulement continuer son chemin. Ainsi, dès qu'une étape du projet est terminée, l'équipe passe à l'étape suivante ; il n'y a pas (ou peu) de retour en arrière. L'idée est d'avancer naturellement, étape par étape, jusqu'à atteindre l'objectif final en suivant une direction claire et précise.	La méthode waterfall est simple, facile à mettre en place, logique et structurée. Elle s'adapte parfaitement à des projets qui répondent à des objectifs clairement identifiés ainsi qu'aux projets où la qualité prime sur le coût et les délais.	L'inconvénient majeur de cette approche est son manque de flexibilité à cause de son déroulement séquentiel. En effet, la méthode waterfall ne laisse aucune place aux changements et aux imprévus.

V Model	Le V-Model se concentre sur une méthode typiquement en cascade qui suit des phases strictes étape par étape. Alors que les étapes initiales sont des phases de conception générales, les étapes progressent de façon de plus en plus granulaire, menant à la mise en œuvre et au codage, et finalement, à travers toutes les étapes de test avant la fin du projet.	Convient aux projets restreints : en raison de la nature rigoureuse du modèle V et de sa conception linéaire. Idéal pour la gestion du temps : dans le même ordre d'idées, V-Model est également adapté aux projets qui doivent respecter une échéance stricte et respecter les principales dates importantes pendant tout le processus	Manque d'adaptabilité : semblable aux problèmes auxquels est confronté le modèle de cascade traditionnel sur lequel repose le modèle V, l'aspect le plus problématique du modèle V est son incapacité à s'adapter aux changements nécessaires pendant le cycle de vie du développement.
2TUP[6]	Le 2TUP propose un cycle de développement en Y, qui dissocie les aspects techniques des aspects fonctionnels. Il commence par une étude préliminaire qui consiste essentiellement à identifier les acteurs qui vont interagir avec le système à construire, les messages qu'échangent les acteurs et le système, à produire le cahier des charges et à modéliser le contexte (le système est une boîte noire, les acteurs l'entourent et sont reliés à lui, sur l'axe qui lie un acteur au système on met les messages que les deux s'échangent avec le sens).	2TUP est un modèle réaliste et naturel qui conserve le caractère « étapiste » de waterfall mais l'intègre dans une approche itérative. Dans 2TUP, le risque est un facteur qui est tenu en compte explicitement.	Il est difficile de faire comprendre au client le mode l'opération de ce modèle. L'évaluation des risques exige une expertise pointue et repose sur une capitalisation de l'expérience vécue.
Agile (SCRUM)	La méthodologie Agile s'oppose généralement à la méthodologie traditionnelle waterfall. Elle se veut plus souple et adaptée, et place les besoins du client au centre des priorités du projet. A l'origine, cette approche a été créée pour les projets de développement web et informatique. Aujourd'hui, la méthode Agile est de plus en	L'avantage majeur de l'approche Agile est sa flexibilité. Les changements du client et les imprévus sont pris en compte et l'équipe projet peut réagir rapidement. Autre atout : la collaboration et la communication fréquente avec le client, ainsi que sa forte implication dans le	Comme le dialogue est privilégié, la méthode Agile laisse peu de place à la documentation, ce qui peut poser problème en cas de changement d'équipe projet, par exemple. Le client doit être disponible et s'intéresser à son

[6] 2TUP : Two Tracks Unified Process

	plus répandue car elle est adaptable à de nombreux types de projets, tous secteurs confondus.	projet. Une relation de confiance se tisse entre le client et l'équipe projet.	projet afin de s'assurer qu'il répondra parfaitement à ses besoins. Tous les clients n'ont pas le temps, ni l'envie de s'impliquer pleinement dans la réalisation d'un projet.
Extreme Programming	Extreme Programming ou extreme agile est l'une des méthodologies Agile. Il partage tous les principes Agiles, y compris une forte implication des clients dans le processus de développement logiciel, une bonne communication au sein des équipes et des cycles de développement itératifs. XP[7] pousse à l'extrême des principes simples.	Le principal avantage de Extreme Programming est que cette méthodologie permet aux sociétés de développement de logiciels de réduire les coûts et le temps nécessaires à la réalisation du projet. Les économies de temps sont possibles car XP met l'accent sur la livraison en temps voulu des produits finis. La simplicité est un avantage supplémentaire des projets de programmation extrême. Les développeurs qui préfèrent utiliser cette méthodologie créent un code extrêmement simple qui peut être amélioré à tout moment.	Dans les projets XP, la documentation des défauts n'est pas toujours satisfaisante. L'absence de documentation sur les défauts peut conduire à l'apparition de bugs similaires à l'avenir. Un autre inconvénient de XP est que cette méthodologie ne mesure pas l'assurance qualité du code. Cela pourrait causer des défauts dans le code initial.

Tableau 2.1 : Comparaison des différentes méthodologies de travail

[Voir Ref 2.3, Page 58 : *Comparaison des différentes méthodologies de travail*]

IV .2 – Méthodologie choisie

Pour ce projet j'ai choisi XP car c'est le meilleur compromis entre temps, usure et couts.

L'XP nous permettra de mieux nous adapter en cas de besoins changeants et de mieux nous concentrer sur la partie réalisation du projet.

[7] XP: Extreme Programming

Conclusion

Dans ce chapitre nous avons défini les besoins fonctionnels et non fonctionnel et ce que l'extension doit respecter à partir d'une étude de marcher. Nous avons établi le fonctionnement général et les cas d'utilisation pour mieux entamer l'exécution du travail demandé.

Nous allons voir dans les chapitres suivant plus de détails sur le dessous des cartes du projet et concrètement mieux comprendre son fonctionnement interne.

CHAPITRE TROISIEME

Etude Conceptuelle

Introduction

Une méthode de conception est une démarche générale reflétant une philosophie de présentation et de suivi du système. C'est aussi est un procédé qui a pour objectif de formaliser les étapes préliminaires du développement d'un système afin de rendre ce développement plus fidèle aux besoins du client. Elle propose des outils spécifiques permettant un suivi efficace de l'information relative au système. La phase de conception permet de décrire de manière non ambiguë, le plus souvent en utilisant un langage de modélisation, le fonctionnement futur du système, afin d'en faciliter la réalisation. Mon choix se porte sur le langage UML[8].

L'UML est un langage de modélisation orientée objet, elle est développée dans le but de définir un standard pour la modélisation des applications construites à l'aide des objets. Il est utilisé pour spécifier un logiciel ou pour le concevoir.

Le modèle produit par une conception orientée objet est en général une extension du modèle issu de la spécification, il l'enrichit de classe dites techniques qui n'intéressent pas l'utilisateur final du logiciel mais seulement ses concepteurs.

I – Diagramme de Classes

I .1 – Définition

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Une classe est une description abstraite d'un ensemble d'objet ayant des propriétés similaires, un comportement commun et des relations communes avec d'autres objets. Elle décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe.

Les classes peuvent être liées entre elles grâce au mécanisme d'héritage qui permet de mettre en évidence des relations de parenté. D'autres relations sont possibles entre des classes, chacune de ces relations est représentée par un arc spécifique dans le diagramme de classes. Elles permettent de modéliser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples.

[Voir Ref 3.1, Page 58 : *Diagramme de classes*]

[8] UML : Unified Modeling Language

I .2 – Diagramme de classes Général

Pour des raisons de lisibilité et vu que j'utilise des classes avec beaucoup d'attributs j'ai fait le choix de ne pas inclure les variables et méthode dans ce diagramme.

Mais aucune inquiétude, les classes présentes seront chacune détaillés (à part) dans les pages qui suivent.

Ce diagramme mettra en évidence la structure et la hiérarchie des classes que j'ai créé.

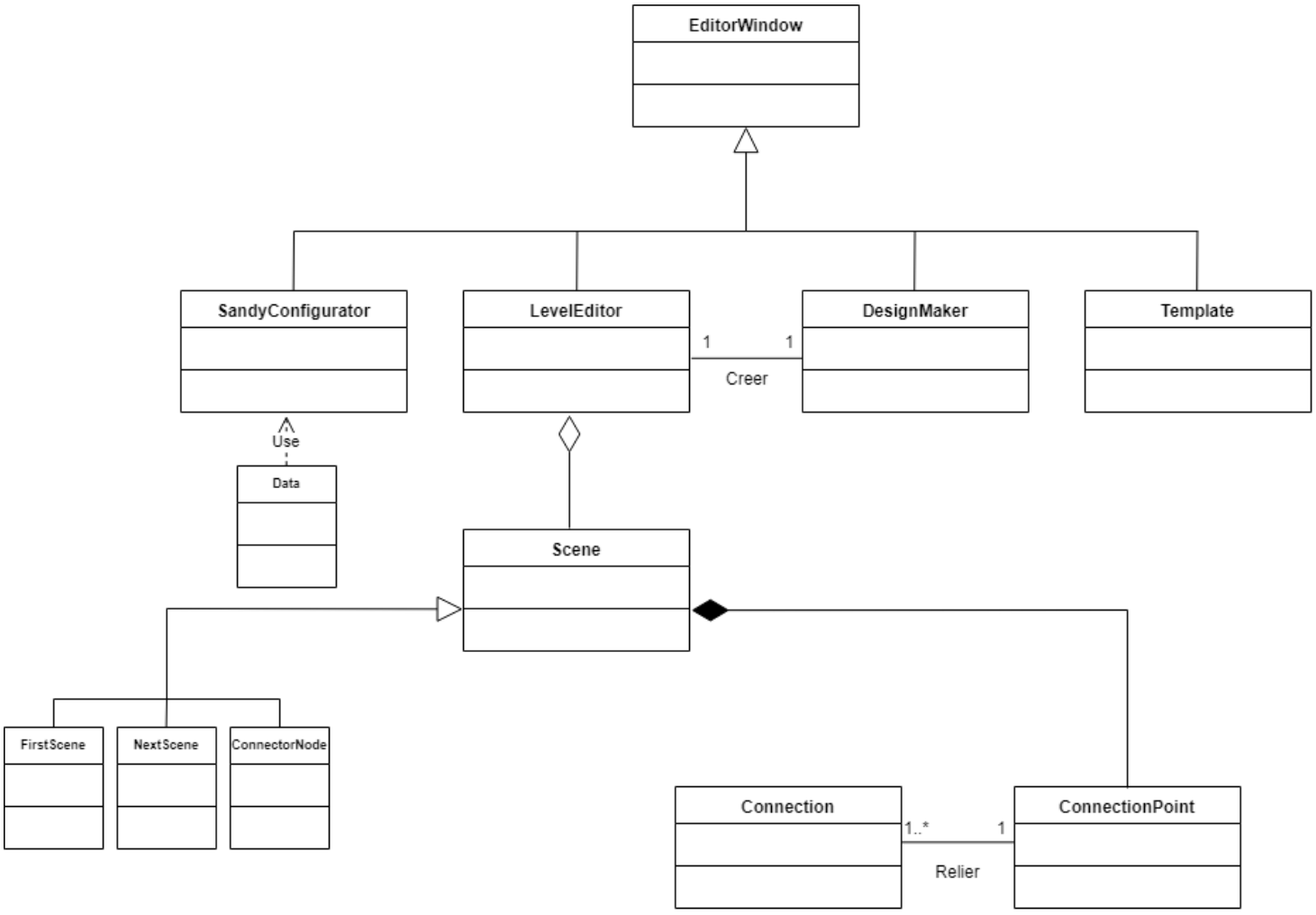

Figure 3.1 : Diagramme de classe

EditorWindow

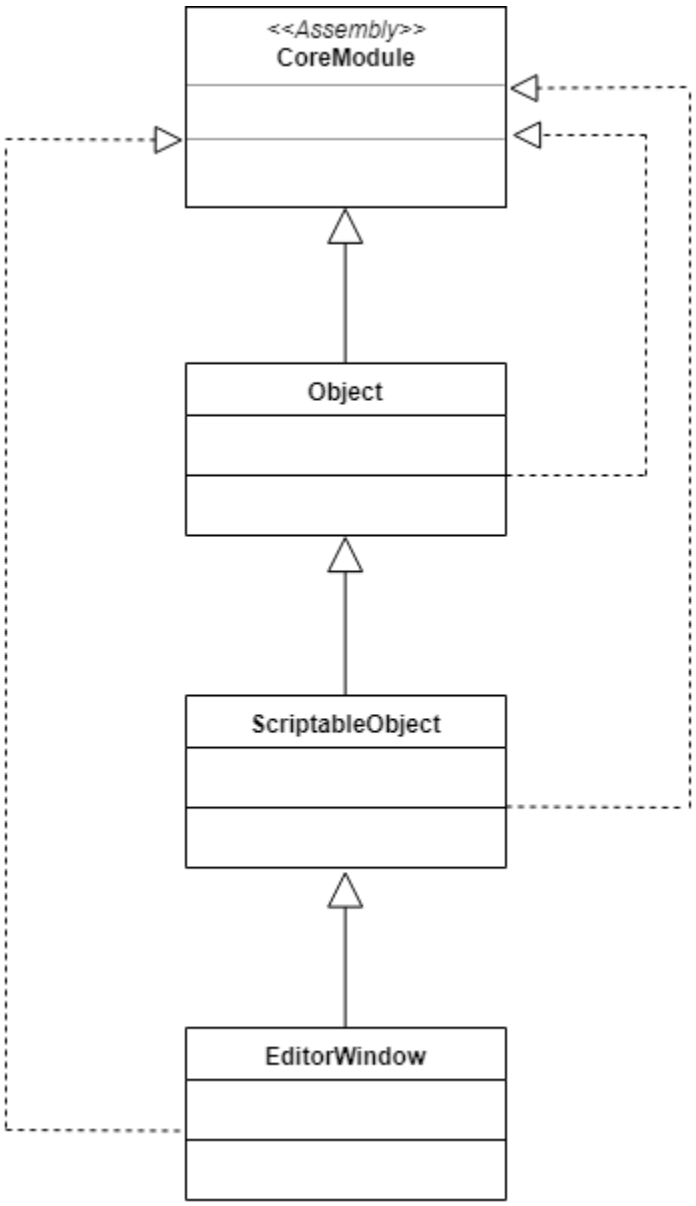

Figure 3.2 : Hiérarchie de la classe EditorWindow dans Unity3D

La figure ci-dessus illustre les classes à partir des quelles EditorWindow est dérivée dans Unity3D.

Classe	Description
EditorWindow	La classe EditorWindow est une classe qui permet de créer votre propre fenêtre d'édition personnalisée pouvant flotter librement ou être ancrée sous la forme d'un onglet, tout comme les fenêtres natives de l'interface Unity3D.
ScriptableObject	La classe ScriptableObject est une classe à partir de laquelle nous pouvons hériter si nous souhaitons créer des objets qui n'ont pas besoin d'être attachés à des objets de jeu. Celle-ci est particulièrement utile pour les actifs destinés uniquement à stocker des données.
Object	La classe Object est la classe de base pour tous les objets que Unity3D peut référencer
CoreModule	CoreModule implémente les classes de base nécessaires au fonctionnement de Unity3D.

Tableau 3.1 : Description de la classe EditorWindow et des classes auxquelles elle est dérivée

Template et DesignMaker

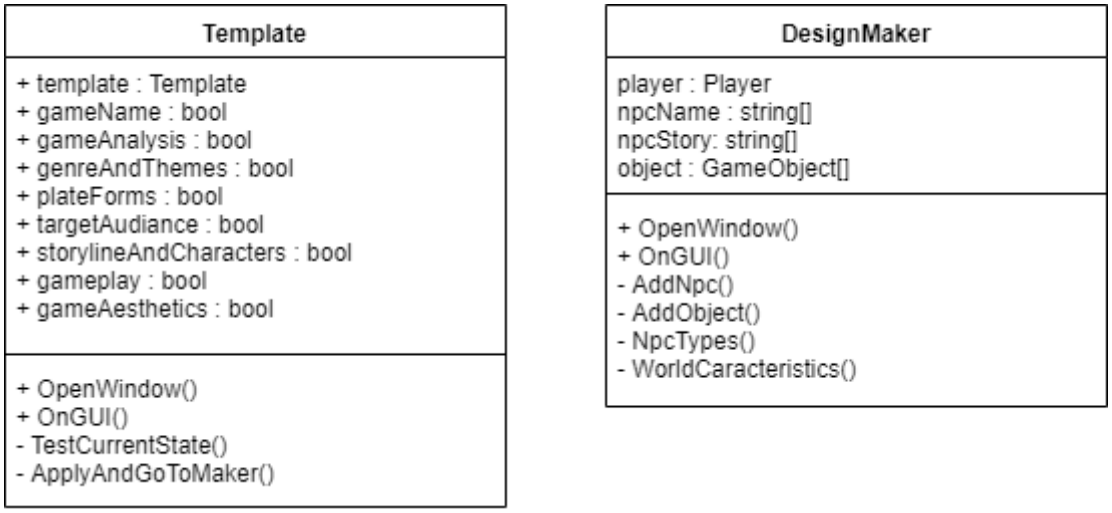

Figure 3.3 : Details des classes Template et DesignMaker

La classe Template permet de décrire les éléments que l'acteur veut inclure dans le projet représenté par les huit variables booléennes. gameName et gameplay sont toujours vraies.

Classe Template		
Variable	Type	Description
-		
Méthode	Type de Retour	Description
OpenWindow()	Void	OpenWindow() est une méthode abstraite que je redéfinie et qui permet d'instancier une fenêtre dans Unity3D grâce à l'onglet « Window » de Unity3D.
OnGUI()	Void	OnGUI() est un méthode abstraite que je redéfinie et qui permet d'afficher le contenue de la fenêtre, OnGUI() rafraichie le contenue de la fenêtre à chaque fois que l'acteur effectue une action dans celle-ci (Bouger la souris compte aussi comme une action).
TestCurrentState()	Void	TestCurrentState() permet de savoir dans quel état de l'éditeur, l'acteur se trouve : 1 pour Template, 2 pour DesignMaker et 3 pour LevelEditor.
ApplyAndGoToMaker()	Void	ApplyAndGoToMaker() permet de sauvegarder les choix de l'acteur dans un fichier JSON et ouvre DesignMaker. NB : le JSON n'est pas utile au programme, mais une simple prévention en cas de crash de Unity3D.

Tableau 3.2 : Description des variables et méthodes de la classe Template

LevelEditor

Figure 3.4 : Detail de la classe LevelEditor

LevelEditor est la plus grande partie du programme et permet d'instancier de façon linéaire les scènes créer dans DesignMaker.

Les scènes sont représentées sous forme de nœuds qu'on peut relier en eux dans une grille infinie.

Classe LevelEditor		
Variable	**Type**	**Description**
instanceCreated	int	La variable instanceCreated reflète le nombre de scènes crées
scenes	List<Scene>	La variable scenes est une liste non chainée de toutes les scènes crées.
connections	List<Connection>	La variable connections est une liste non chainée de toutes les connections entre les scènes crées.
inPointStyle, outPointStyle	GUIStyle	Les variables de type GUIStyle reflète le style de l'élément en

		question (un peu comme du CSS dans le Web). Ici, inPointStyle et outPointStyle sont les points de connections de chaque nœud.
selectedInPoint, selectedOutPoint	ConnectionPoint	Les variables selectedInPoint et selectedOutPoint sont les points de connections du nœuds actuellement en sélection.
offset	Vector2	La variable offset permet grâce à DrawGrid() de dessiner une grille.
drag	Vector2	La variable drag permet grâce à OnDrag() de bouger librement a l'intérieur de la grille.
Méthode	**Type de Retour**	**Description**
OnEnable()	Void	OnEnable() est une methode rédéfinie de Unity3D qui permet d'afficher la grille directement quand LevelEditor est créé.
DrawScenes()	Void	DrawScenes() permet de dessiner une nouvelle scène.
DrawConnections()	Void	DrawConnections() permet de créer une nouvelle connection.
ProcessEvent()	Void	ProcessEvent() permet de gérer tous les évènements de souris.
ProcessNodeEvents()	Void	ProcessNodeEvents() permet de gérer tous les évènements de souris liés aux scènes.
DrawConnectionLine()	Void	DrawConnectionLine() permet de dessiner une nouvelle connection.
ProcessContextMenu()	Void	ProcessContextMenu() permet de créer un menu quand l'acteur fait « clic droit ».
SaveCurrent(), LoadCurrent()	Void	SaveCurrent() et LoadCurrent() permettent de sauvegarder le statut courant de LevelEditor et de le recharger.
SceneID()	Int	SceneID() permet d'atribuer a chaque nouvelle scène créée un numéro. Gère aussi la variable instanceCreated.

OnClickInPoint(), OnClickOutPoint()	Void	OnClickInPoint() permet de créer une nouvelle connection à partir d'un point d'entrée (InPoint) ou la conclure à partir d'un point de sortie (OutPoint), OnClickOutPoint() est l'inverse de OnClickInPoint(). Ce menu permet de choisir le type de scène à créer, sauvegarder et charger.
OnClickRemoveScene()	Void	OnClickRemoveScene() permet de supprimer un nœud déjà créer. Si le nœud en question possède des fils, tous les fils sont supprimés eux aussi.
OnClickRemoveConnection()	Void	OnClickRemoveConnection() permet de supprimer une connection en deux nœuds.
CreateConnection()	Void	CreateConnection() permet d'ajouter à la connection qui vient de se créée toutes les références de type Parent/fils aux deux scènes concernées par la connection.
ValidConnectionRules()	Bool	ValidConnectionRules() vérifie si pour une connection créée : - La connection est unique (évite le dédoublement de connections) - La connection doit être de type différents (ConnectorNode ≠ FirstScene ou NextScene). - Le nombre de connections ne dépasse pas 9. - Un nœud doit avoir un seul parent Si l'une de ces règles est violé, la connection ne s'effectue pas.
CreateName()	Void	CreateName() attribue un nom a une scène parent si aucun nom ne lui est attribuée.

Tableau 3.3 : Description des variables et méthodes de la classe LevelEditor

Scene

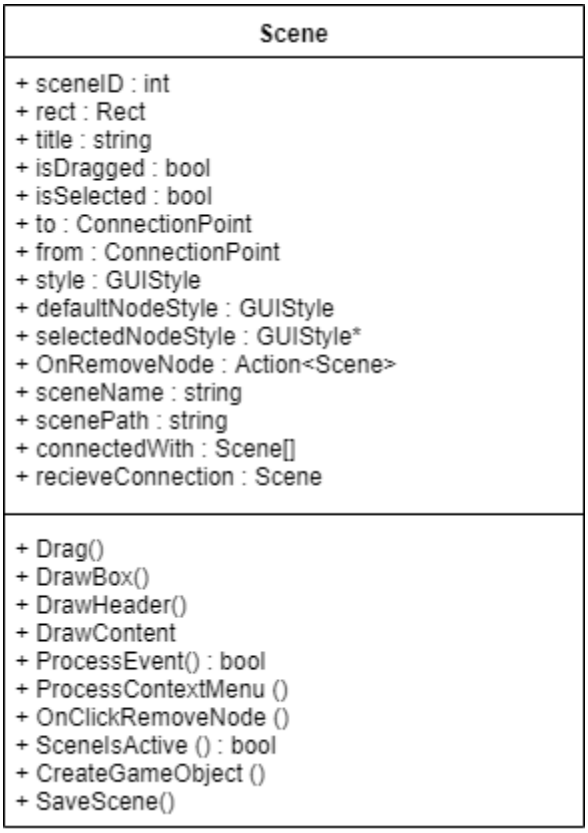

Figure 3.5 : Details de la classe Scene

Classe Scene		
Variable	**Type**	**Description**
sceneID	int	La variable sceneID est l'identifiant de la scène obtenue grâce à la méthode SceneID() de LevelEditor.
rect	Rect	La variable scenes est une liste non chainée de toutes les scènes crées.
title, sceneName	string	Les variables title et sceneName représente le nom de la scène. La différence est que title est zone de texte.
isDragged	bool	La variable isDragged vérifie si le nœud bouge dans la grille ou non.
isSelected	bool	isSelected vérifie si le nœud est actif ou non. isDragged et isSelected sont complémentaires.
to from	ConnectionPoint	to et from sont les points de connection de la scène.

OnRemoveNode	Action<Scene>	OnRemoveNode est un délégué qui renvoie à une méthode. Si la méthode s'active, la scène va être supprimer. Dans C#, un délégué de type Action est identique au délégué Func, à la différence que le délégué Action ne renvoie pas de valeur. En d'autres termes, un délégué Action peut être utilisé avec une méthode ayant un type de retour vide.
scenePath	string	La variable scenePath est le chemin de la scène dans Unity3D quand la scène sera sauvegardée.
connectedWith	List<Scene>	connectedWith les la liste les scènes fils connectés à la scène.
revieveConnection	Scene	revieveConnection représente la scène parent.
Méthode	**Type de Retour**	**Description**
Drag()	Void	Drag() permet de faire bouger le nœud dans la grille.
DrawBox()	Void	DrawBox() est la partie du nœud qui contient les points de connections, DrawHeader() contient l'identifiant de la scène et le nom de la scène et DrawContent() contient tout le reste (Objets, boutons, champs à remplir…etc.).
CreateGameObject()	Void	CreateGameObject() crée dynamiquement un objet dans la scène.
ProcessEvent()	Void	ProcessEvent() permet de gérer tous les évènements de souris à l'intérieur du nœud.
SaveScene()	Void	SaveScene() permet de sauvegarder la scene si elle est active.

Tableau 3.4 : Description des variables et méthodes de la classe Scene

Connection et ConnectionPoint

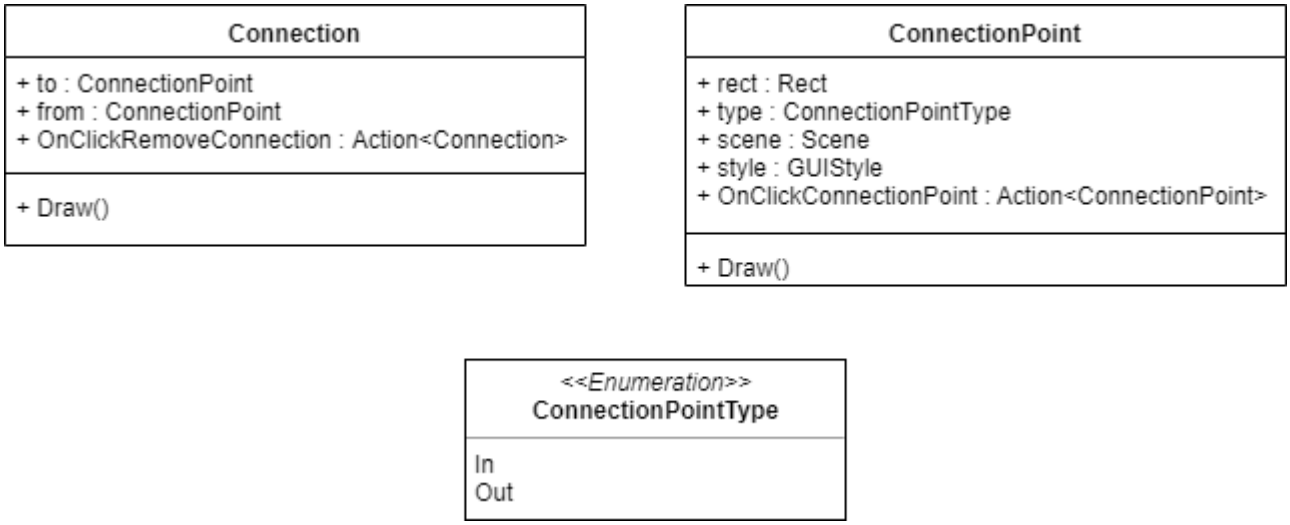

Figure 3.6 : Details des classes Connection et ConnectionPoint et de L'énumération ConnectionPointType

ConnectionPoint représente le bouton alors que Connection représente la liaison elle-même.

Classe Connection		
Variable	**Type**	**Description**
from	ConnectionPoint	from : de quelle scène la connection vient.
to	ConnectionPoint	to : à quelle scène la connection ira.
OnClickRemoveConnection	Action<Connection>	Permet de supprimer une connection
Méthode	**Type de Retour**	**Description**
Draw()	Void	Draw() permet dessiner une connection

Tableau 3.5 : Description des variables et méthodes de la classe Connection

Classe ConnectionPoint		
Variable	**Type**	**Description**
rect	Rect	Délimite la zone cliquable du point de connection
type	ConnectionPointType	Type du point de connection : - In : Se trouve à gauche du nœud de la scène en question - Out : Se trouve à droite du nœud de la scène en question.
scene	Scene	Référence de la scène ou se trouve le point de connection instancié.
style	GUIStyle	Le style graphique du point de connection.
OnClickConnectionPoint	Action<ConnectionPoint>	Permet de cliquer sur un point de connection dans le seul but de créer une connection dans LevelEditor.
Méthode	**Type de Retour**	**Description**
Draw()	Void	Draw() permet dessiner le point de connection

Tableau 3.6 : Description des variables et méthodes de la classe ConnectionPoint

ConnectorNode

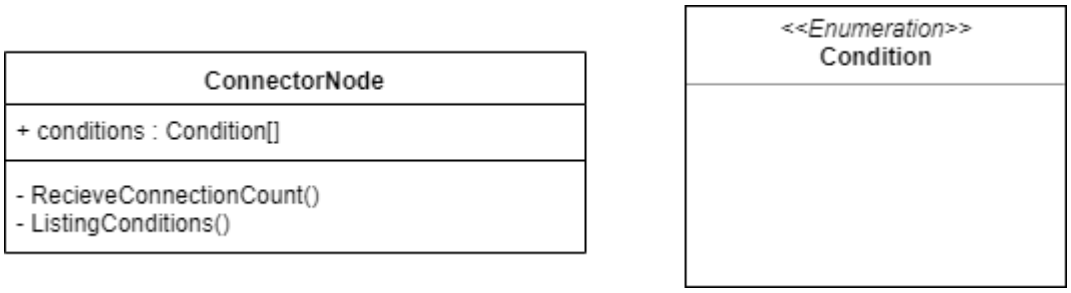

Figure 3.7 : Details de la classe ConnectorNode et de l'énumération Condition

ConnectorNode n'est pas une scène à proprement parler mais elle en a l'apparence. ConnectorNode est comme une scène mais qui en connecte deux autres.

Elle permet de visualiser les connections entre un parent et ses fils. Elle permet aussi de gérer les conditions de transitions entre deux scènes.

L'acteur aura la possibilité d'éditer la classe Condition.

Classe ConnectorNode		
Variable	**Type**	**Description**
conditions	Conditions[]	Liste de toutes les conditions à afficher
Méthode	**Type de Retour**	**Description**
RecieveConnectionCount()	Void	Affichage du nombre de connections effectuées dans connectedWith
ListingConditions()	Void	Permet à l'acteur de choisir la condition voulue pour chaque scène dans connectedWith

Tableau 3.7 : Description des variables et méthodes de la classe ConnectorNode

SandyConfigurator

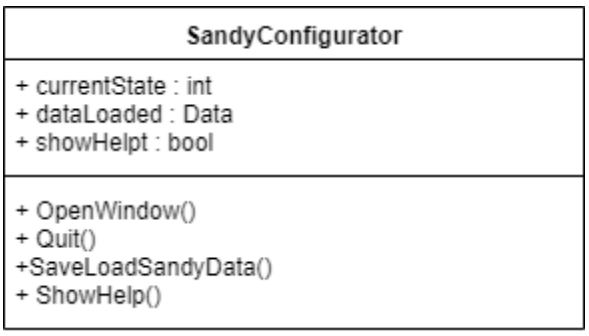

Figure 3.8 : Details de la classe SandyConfigurator

SandyConfigurator permet de gérer les données qui transitent entre Template, DesignMaker et LevelEditor.

Classe SandyConfigurator		
Variable	**Type**	**Description**
currentState	int	Permet de savoir si l'acteur est dans la phase Template, DesignMaker ou LevelEditor.
dataLoaded	Data	Contient la référence au variables qui transitent entre Template, DesignMaker et LevelEditor.
Méthode	**Type de Retour**	**Description**
OpenWindow	Void	OpenWindow() est une méthode abstraite que je redéfinie et qui permet d'instancier une fenêtre dans Unity3D grâce à l'onglet « Window » de Unity3D. Son but est de permettre a l'acteur de cocher ou non l'aide dans l'interface.
Quit()	Void	Quitter l'éditeur en sauvegardant.
SaveLoadSandyData()	Void	Permet de gérer les données qui transitent entre Template, DesignMaker et LevelEditor.
ShowHelp()	Void	ShowHelp() permet ou non d'afficher les infobulles et la documentation a l'acteur.

Tableau 3.8 : Description des variables et méthodes de la classe SandyConfigurator

II – Diagramme d'activités

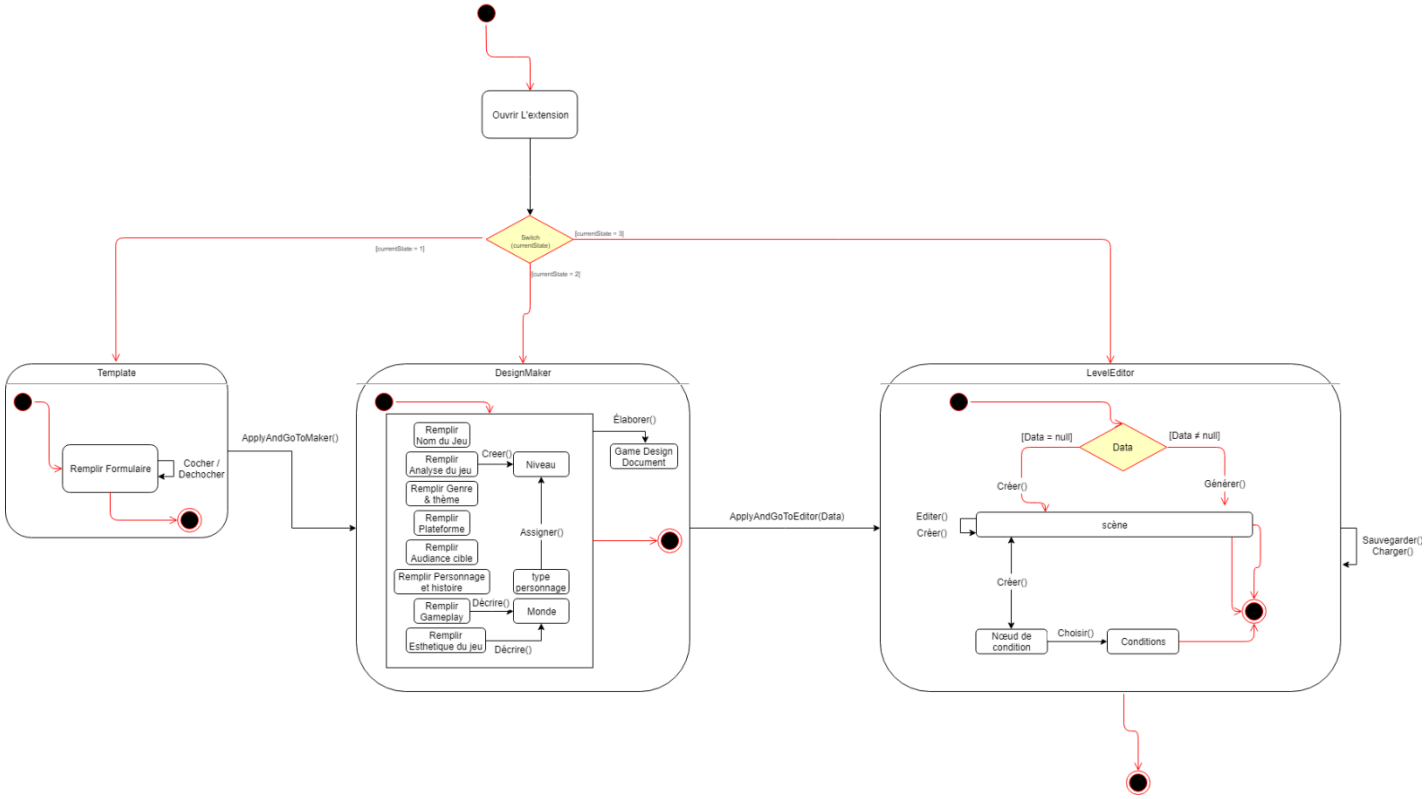

Figure 3.9 : Diagramme d'activités

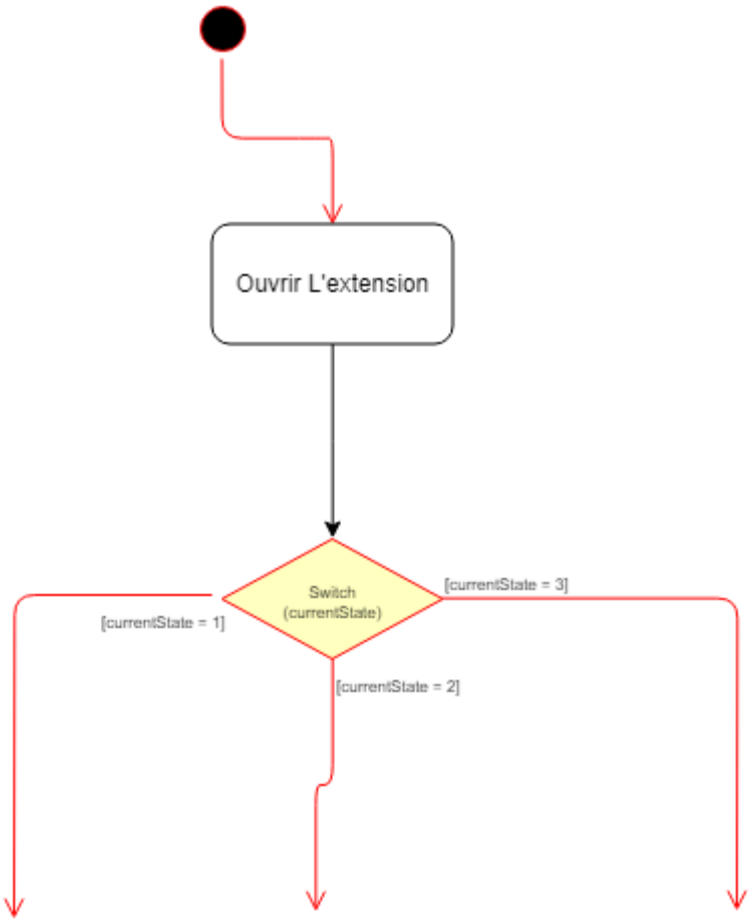

Figure 3.10 : Diagramme d'activités, Ouverture de l'extension

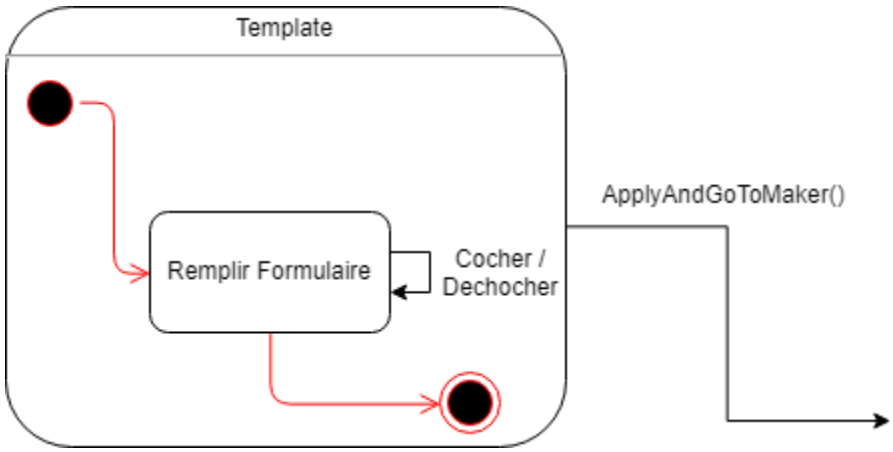

Figure 3.11 : Diagramme d'activités, partie Template

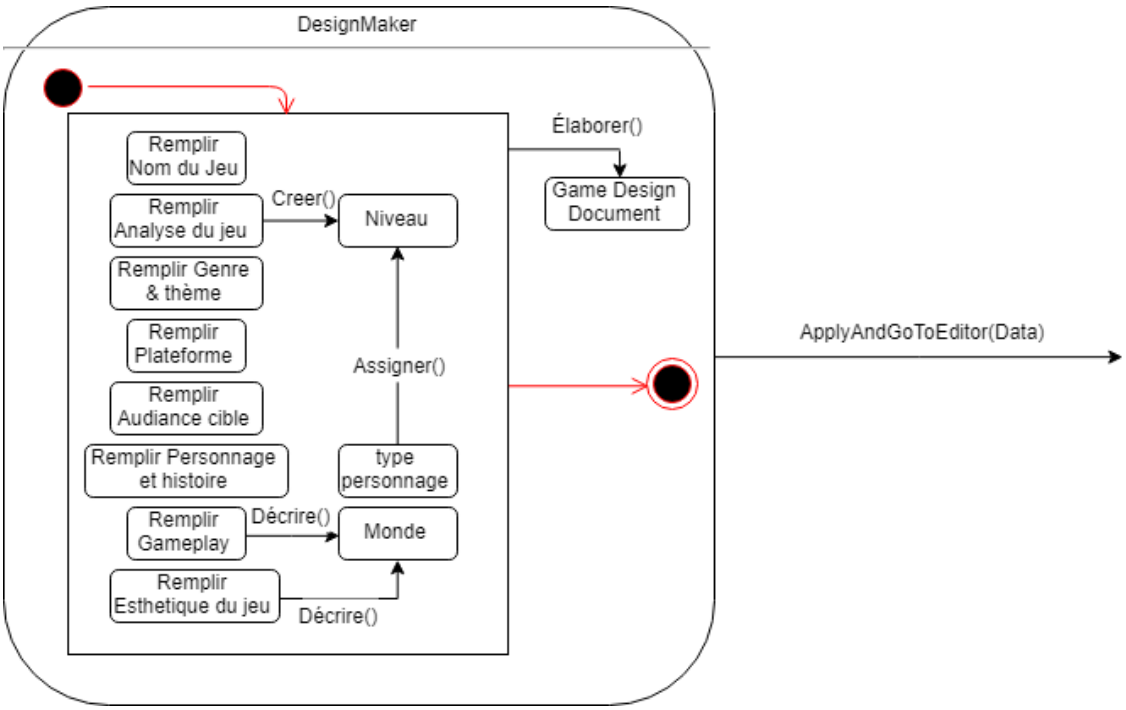

Figure 3.12 : Diagramme d'activités, partie Créateur de design

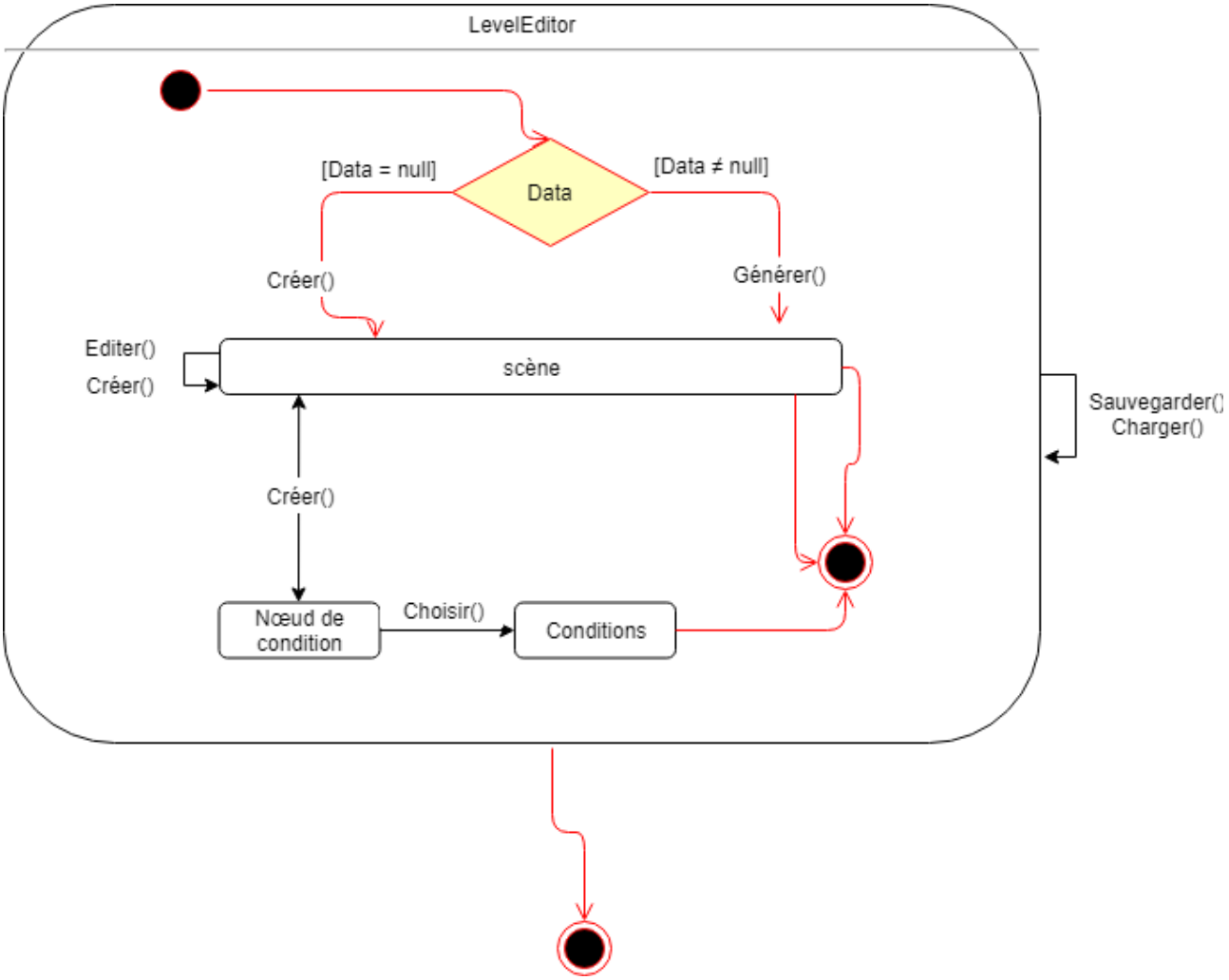

Figure 3.13 : Diagramme d'activités, partie Editeur de niveaux

Conclusion

Dans cet avant dernier chapitre, nous avons vu ce qui compose l'extension grâce au diagramme de classe et son fonctionnement profond grâce au diagramme d'activités.

CHAPITRE QUATRIEME

Réalisation

Introduction

Ce dernier chapitre présente la partie de la réalisation et la mise en œuvre des différents composants décrits au niveau du chapitre précédent. Dans un premier temps, nous présenterons l'environnement matériel et logiciel. Ensuite, nous décrirons le travail réalisé en détaillant quelques captures d'écrans des fonctionnalités réalisées.

I – Environnement de travail

I .1 – Environnement Matériel

Pour développer cette application, j'ai eu recoure à un ordinateur avec un processeur intel 7ème génération 7700HQ cadencé à 3Ghz, 12Go de RAM, une mémoire morte de 1To et une carte graphique Nvidia GTX 1050 avec une VRAM de 4Go.

Pour utiliser l'extension (Nous faisons ici référence à l'acteur), le développeur a besoin de ce système minimum requis :

- Windows 7 SP1+, 8, 10, version 64-bit seulement ; macOS 10.12+
- CPU : SSE2 instruction set
- GPU : Carte graphique compatible DX10

Nous notons cependant que cette configuration ne permet pas d'utilisation de Unity (et donc, notre extension) dans de très bonnes conditions et de confort. Nous recommandons une configuration d'entrée de gamme de l'année indiquée dans la version de Unity utilisée. En toute subjectivité, notre configuration nous a permis le développement de l'extension de Unity sans accroche ni « lag »

[Voir Ref 4.1, Page 58 : *Configuration minimum requise*]

I .2 – Environnement Logiciel

Pour les logiciels j'utilise Unity3D et Visual Studio Code pour le développement du site et de l'extension et Adobe Illustrator CC 19 pour le logo. J'utilise aussi git (gitLab) pour faire le suivi du projet et pour éviter la perte de ce dernier.

J'ai choisi Unity Engine pour sa flexibilité et son immense communauté active. En effet grâce au langage C# et le panel monstrueux de bibliothèques Unity, Cette technologie laisse la libre cour a l'imagination et à la créativité et me permet de créer l'outil sans devoir à créer les objets physiques et interactions de base qui y sont déjà intégré.

Notez bien que seul le langage C# est compatible avec Unity.

Unity 3D

Unity est un moteur de jeu multiplateforme (smartphone, ordinateur, consoles de jeux vidéo et Web) développé par Unity Technologies. Il est l'un des plus répandus dans l'industrie du jeu vidéo, aussi bien pour les grands studios que pour les indépendants

du fait de sa rapidité aux prototypages et qu'il permet de sortir les jeux sur tous les supports.

Il a la particularité de proposer une licence gratuite dite « Personal » avec quelques limitations de technologie avancée au niveau de l'éditeur, mais sans limitation au niveau du moteur.

Nous avons utilisé la version 2018.3.0f2 pour le développement de l'extension, une version antérieure à celle présente sur le site officiel, mais aucun problème de compatibilité n'a été soulevé/remarqué.

Pour l'heure, aucune annonce de Unity Technologies pour une future mise à jour de leur moteur ne présage une possible menace de compatibilité pour notre extension. [Voir Ref 4.2, Page 58 : *Définition de Unity*]

Figure 4.1 : Logo Unity

Visual Studio Code

Visual Studio Code (VS Code) est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS.

Figure 4.2 : Logo Visual Studio Code

Il comprend la prise en charge du débogage et le contrôle Git intégré. VS Code est hautement personnalisable, permettant aux utilisateurs de changer le thème, les raccourcis clavier, les préférences et d'installer des extensions qui ajoutent des fonctionnalités supplémentaires. Le code source est gratuit et open source et publié sous la licence permissive MIT. [Voir Ref 4.3, Page 58 : *Définition de VS Code*]

Gitlab

GitLab est un logiciel libre de forge basée sur git proposant les fonctionnalités de wiki,

Figure 4.3 : Logo GitLab

un système de suivi des bugs, l'intégration continue et la livraison continue. Développé par GitLab Inc et créé par Dmitriy Zaporozhets et par Valery Sizov, le logiciel est utilisé par plusieurs grandes entreprises informatiques incluant IBM, Sony, le centre de recherche de Jülich, la NASA, Alibaba, Oracle, Invincea, O'Reilly Media, Leibniz Rechenzentrum, le CERN, European XFEL, la GNOME Foundation, Boeing, Autodata et SpaceX. [Voir Ref 4.4, Page 58 : *Définition de GitLab*]

Adobe Illustrator CC 19

Adobe Illustrator est un logiciel de création graphique vectorielle. Il fait partie de la gamme Adobe, peut être utilisé indépendamment ou en complément de Photoshop, et offre des outils de dessin vectoriel puissants. Les images vectorielles sont constituées de courbes générées par des formules mathématiques.

Illustrator CC a été publié avec Creative Cloud (résultat du changement de stratégie d'administration d'Adobe). La 17ème version a été la première à être vendue uniquement

Figure 4.4 : Logo Adobe Illustrator CC

dans un modèle de service basé sur un abonnement, en ligne avec les autres logiciels de l'ancien logiciel Creative Suite. [Voir Ref 4.5, Page 58 : *Définition de Adobe Illustrator CC*]

II – Communication et Approche marketing

II .1 – Nom

Le nom que j'ai choisi (Sandy) est un hommage à Alexander Shafto "Sandy" Douglas qui était un professeur britannique de science de l'informatique, crédité de la création du premier jeu sur ordinateur dotés de graphismes intitulé OXO (aussi connu sous le nom Noughts & Crosses), un jeu de tic-tac-toe créé en 1952 sur l'ordinateur EDSAC à l'Université de Cambridge.

Sandy est considéré pour certains comme l'un des pères fondateurs du jeu vidéo.

II .2 – Logo

Figure 4.5 : Concept pour logo original pour l'extension

II .3 – Site Web

L'extension Sandy sera disponible sur l'asset store de Unity et sera visible comme suit.

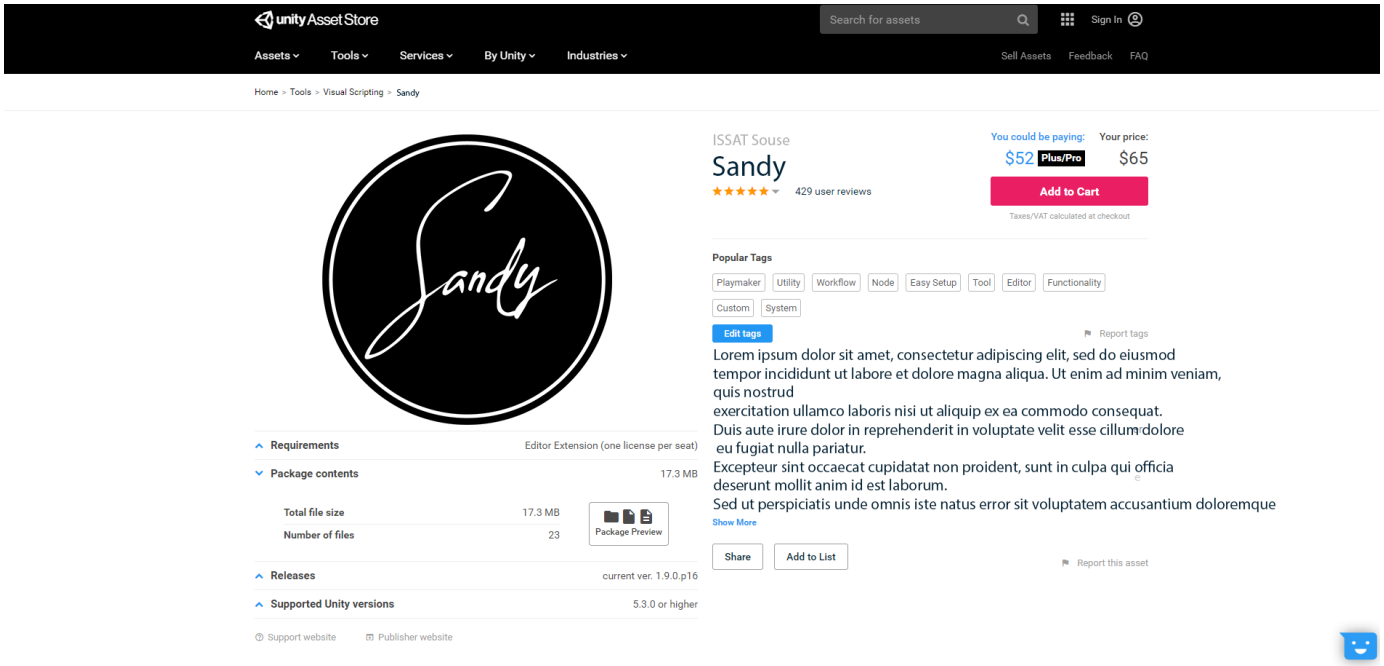

Figure 4.6 : Aperçu de la page « Asset Store » de l'extension

III – Interfaces de l'extension

La figure 4.7 présente l'onglet Window (présent de base dans Unity) dans lequel on accède à l'extension Sandy.

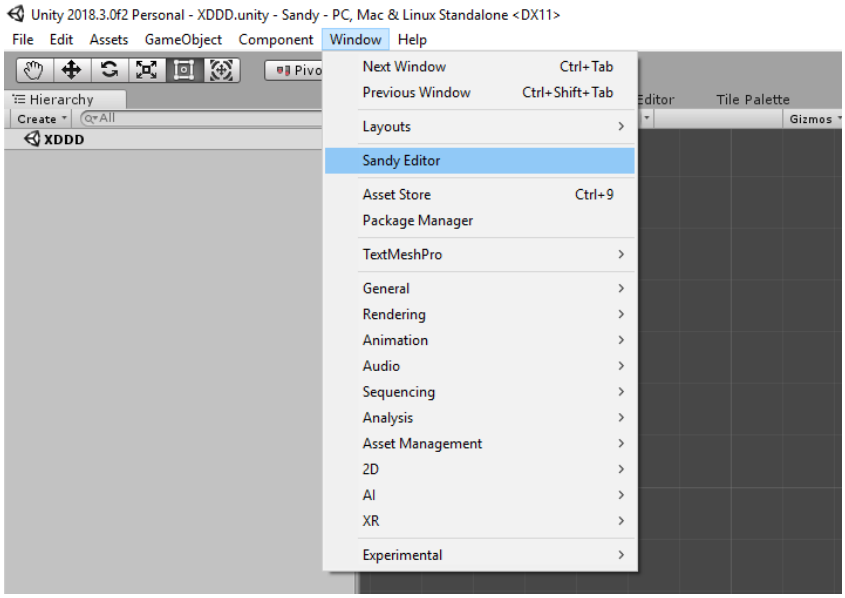

Figure 4.7 : Ouverture de l'extension

La figure 4.8 présente l'interface « Template ». Cette interface présente une série de cases à cocher pour choisir les éléments du Game Design Document qu'on veut remplir.

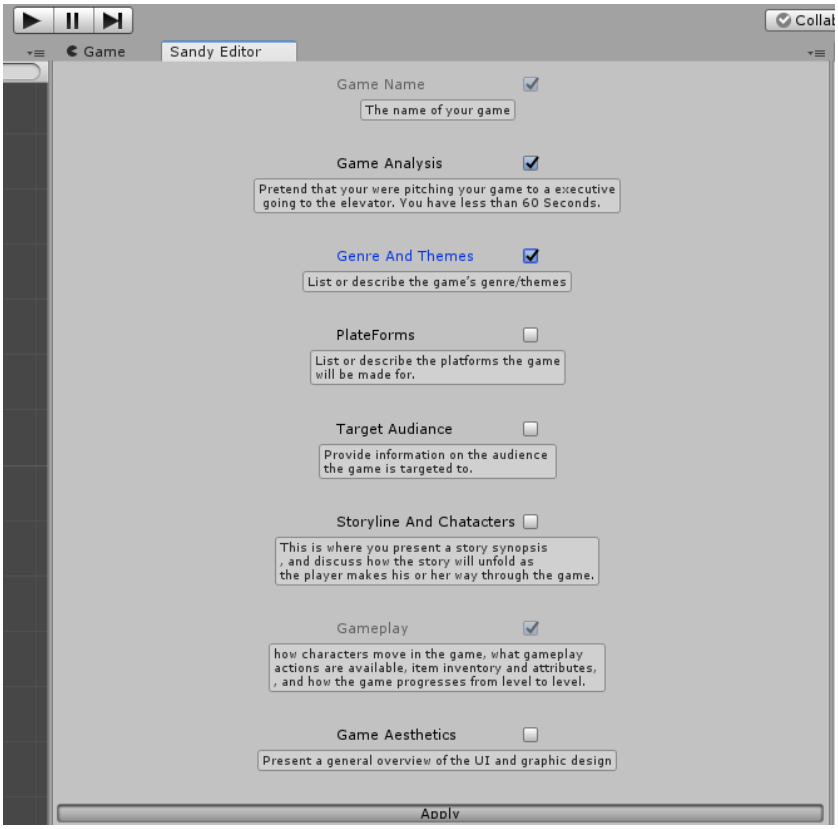

Figure 4.8 : Interface Template

La figure 4.9 nous montre la sauvegarde crée lorsque le développeur appui sur le bouton « Apply ». La sauvegarde sera créée en locale sur son dossier personnel.

Figure 4.9 : Sauvegarde 1

La figure 4.10 nous montre l'interface « Design Maker » cette interface nous permet la description concrète du jeu, certains paramètres seront sauvegardés pour la suite.

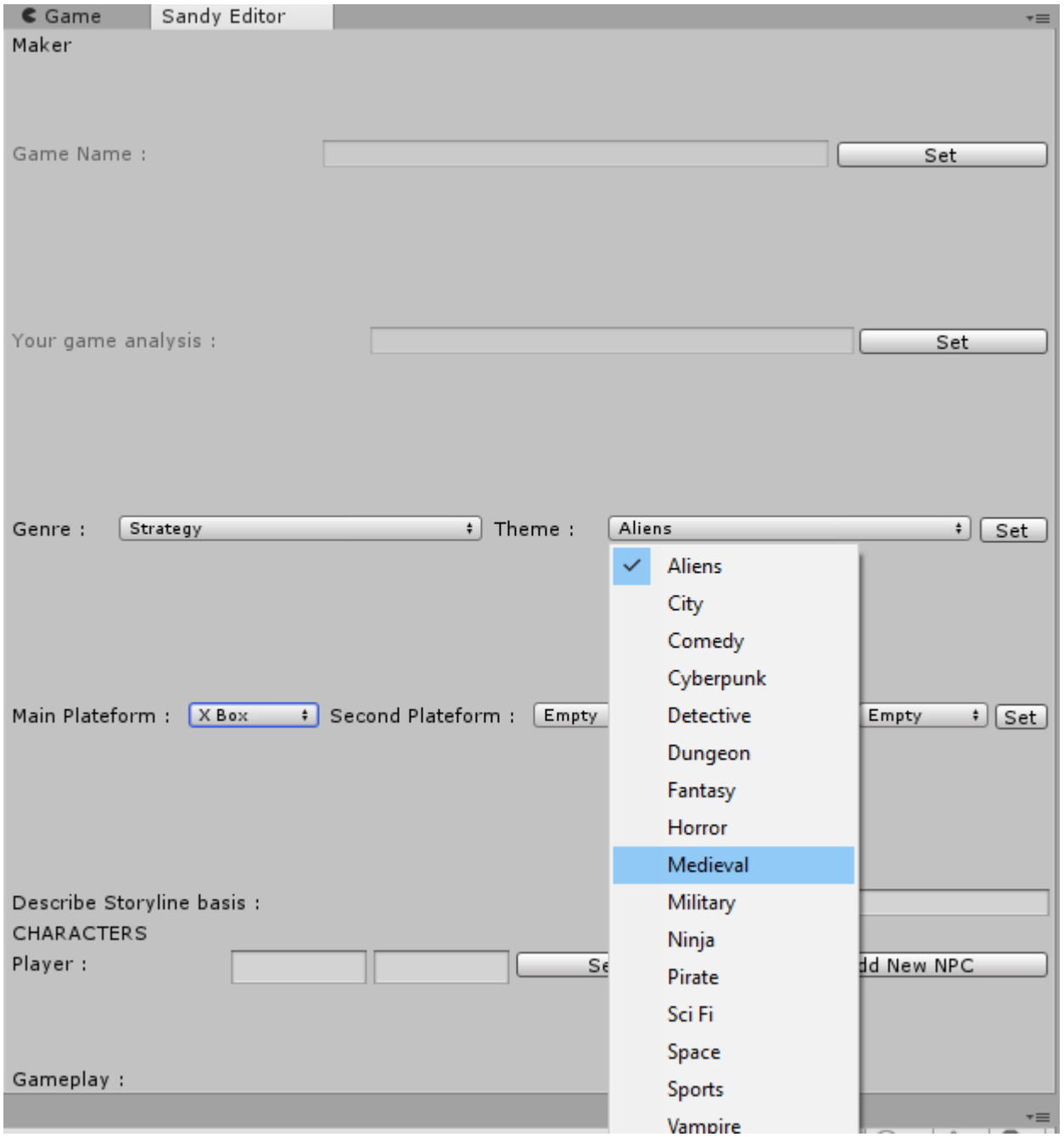

Figure 4.10 : Interface Design Maker

La figure 4.11 Décrit l'interface « level designer » cette interface permet la création de **scènes** (représentés par les grands nœuds) et des **conditions** qui lient et connectent les scènes (représentées par les petits nœuds). Nb : les nœuds de même type ne peuvent pas se connecter entre elles.

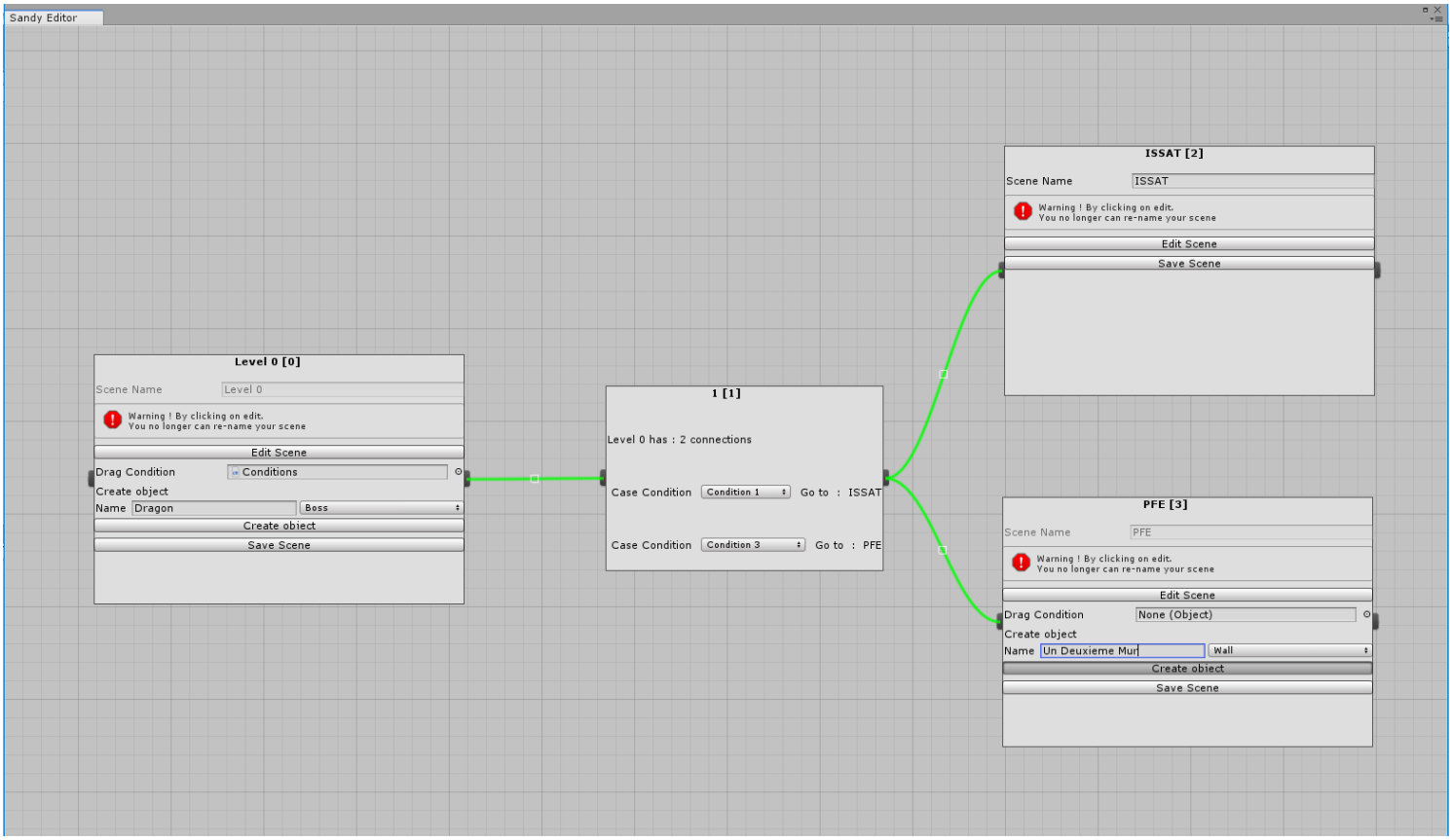

Figure 4.11 : Interface Level Editor

Les nœuds sont disposés dans une grille

Cas : Clic Droit sur la grille ➔ Fait apparaitre un menu pour créer un nœud de type *Scène* ou de Type *ConditionNode*.

Cas : Clic Droit sur une scène ➔ Fait apparaitre un menu pour supprimer une scène de n'importe quel type. (Cela supprime tous ses fils)

La figure 4.12 Montre le fonctionnement des nœuds de type **scènes** et la création d'objects dynamique.

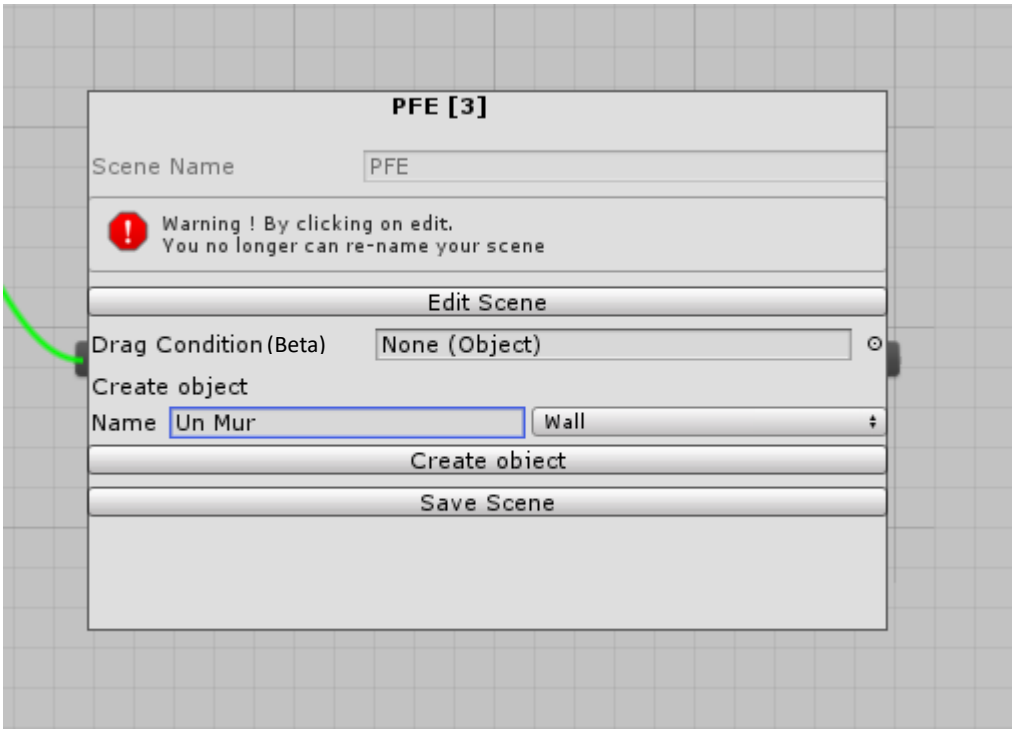

Figure 4.12 : Exemple de Nœud de type scene

En vert on distingue une Connection entrante depuis *Level 0* à *PFE*. Si le joueur réussi la condition 3 (que le développeur peut programmer avec C#) lors du *Level 0* alors le joueur passe au niveau *PFE*.

Cas : Clic sur *Edit Scene* ➔ Affiche le menu juste en dessous du bouton *Edit Scene*.

Drag Connection permet d'ajouter un script personnalisé que le nœud *ConnctionNode* (fils) prendra en charge.

Create Object permet de créer un **Personnage** avec un type ou un **Objet** *(Nb1 : Les objets ne sont pas encore implémentés dans notre code mais sont considéré comme des personnages).*

(Nb2 : personnages ou objet, tout est considéré comme un GameObject dans Unity)

Save permet de sauvegarder la scène.

Les figures 4.13 et 4.14 montrent la création des scènes et objet dynamiquement dans Unity.

Figure 4.13 : Les object créés dans une scène

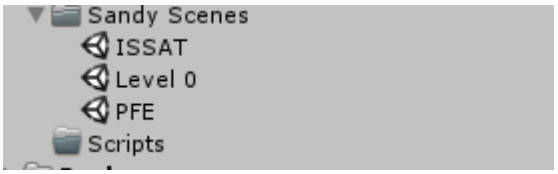

Figure 4.14 : Les scène créées

La figure 4.13 montre les objets crées dans une scène dans Unity.

La figure 4.14 montre les scènes sauvegardées. Elles sont sauvegardées dans un dossier appelé *Sandy Scenes* – accessibles à tout moment par le développeur.

La figure 4.15 montre la sauvegarde générée par level Editor lorsque le développeur décide de sauvegarder son travail pour ne pas risquer de le perdre.

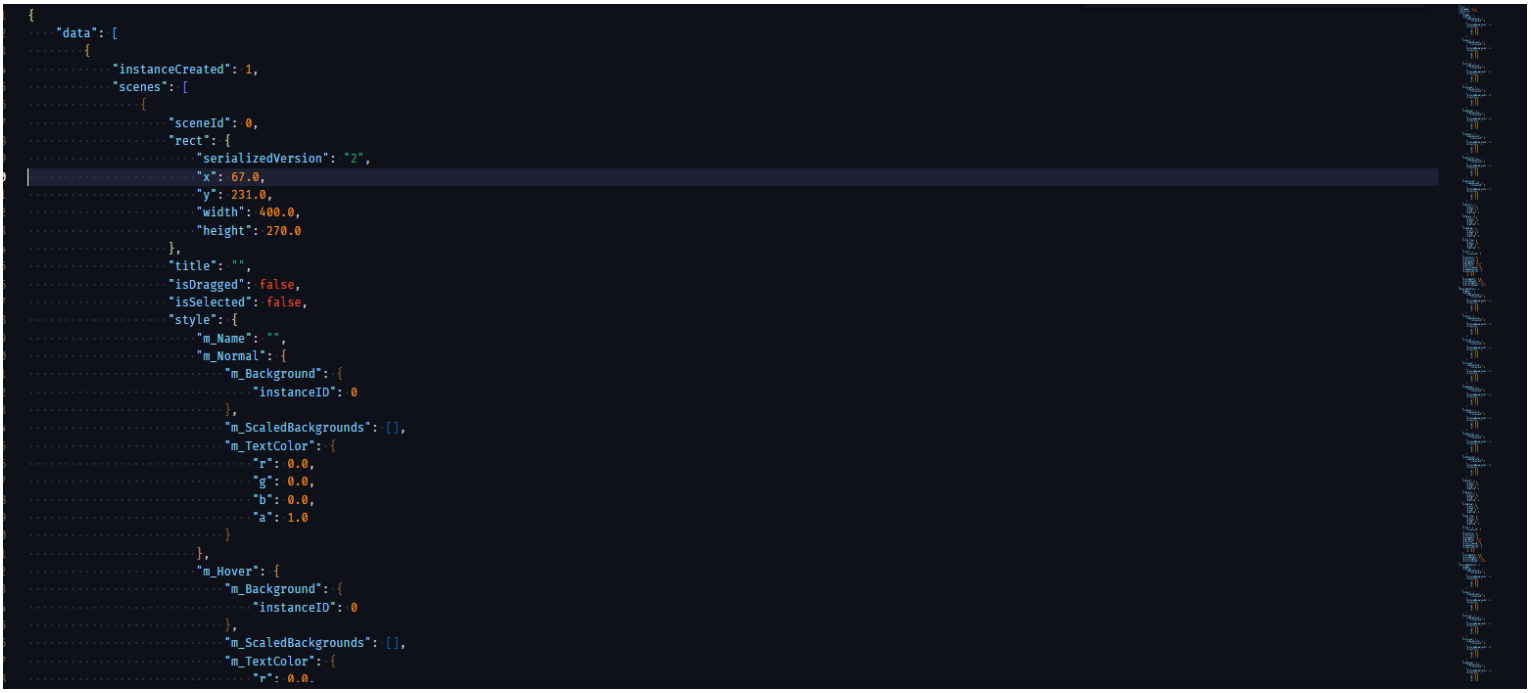

Figure 4.15 : Sauvegarde 2

CONCLUSION GENERALE

Au terme de ce projet de fin d'études, je tiens à souligner que sa réalisation était d'un très grand bénéfique pour moi car c'était une bonne occasion pour consolider mes connaissances théoriques dans le domaine de la programmation de moteurs de jeu et du game design.

Il est évidement que ce projet n'est ni une œuvre parfaite ni un projet fini mais j'ai tenu à ce qu'il soit à la fin de ce stage à la hauteur de mes espérances professionnelles ainsi qu'à la communauté des développeurs du monde entier en espérant qu'elle trouve dans mon travail une bonne solution pour les différents problèmes que cette dernière rencontre.

Le game design est un art méconnu au sein même du milieu des développeurs de jeux vidéo, j'espère que ce projet fera en sorte de mieux le mettre en valeur car un bon jeu vidéo c'est avant tout un bon game designer qui est derrière.

J'ai consacré le tier de mon temps pour rendre l'extension le plus scalable possible, en perspectives, cette dernière peut être grandement améliorée.

BIBLIOGRAPHIE

Livres :

- Langage de modélisation UML, Fréderic Julliard.
- Theory of Fun for Game Design, Raph Koster.

Site Web :

- https://unity.com/
- https://docs.unity3d.com/
- https://docs.microsoft.com/en-us/dotnet/csharp/
- https://www.draw.io/
- https://about.gitlab.com/

ANNEXE

L'Institut supérieur des sciences appliquées et de technologie de Sousse ou ISSATSO est un établissement scientifique relevant de l'Université de Sousse (Tunisie). Il est créé en 2001, il compte plus de 2000 étudiants, plus de 300 enseignants, plus de 20 clubs et association et 5 startups incubées dans l'incubateur dont je suis l'un des fondateurs pour une surface de 27000 m^2.

Sources

Figure 1.1 : Logo de l'AFJV..... 06

https://www.afjv.com/logo/index.php

Figure 2.1 : Logo de Twine 2.0..... 11

https://twinery.org/

Figure 2.2 : Histoire et design du loup et l'agneau dans Twine 13

https://www.bac-a-sable.eu/twine/twine-quelle-version-choisir/

Figure 2.3 : Illustration officielle de diamond Visual Scripting 13

https://forum.unity.com/threads/diamond-visual-scripting.477293/

Figure 2.4 : Graph comparatif des extensions présentent dans l'Asset Store 15

https://www.gameassetdeals.com/

Figure 2.6 : Script Unity de base..... 20

https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Figure 4.1: Logo Unity..... 46

https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Unity_Technologies_logo.svg/1200px-Unity_Technologies_logo.svg.png

Figure 4.2: Logo VS Code..... 46

https://code.visualstudio.com/assets/updates/1_35/logo-stable.png

Figure 4.3: Logo GitLab..... 47

https://about.gitlab.com/images/press/logo/png/gitlab-logo-gray-rgb.png

Figure 4.4: Logo Adobe Illustrator CC..... 47

https://i.pinimg.com/originals/3f/bb/2f/3fbb2f0aa414d20476517f78061b7459.png

Figure 4.5: Concept pour logo original pour l'extension..... 48

https://i.pinimg.com/originals/3f/bb/2f/3fbb2f0aa414d20476517f78061b7459.png

Ref ig1 : Définitions sur les jeux vidéo..... 01

https://fr.wikipedia.org/wiki/Jeu_vid%C3%A9o

Ref 1.1 : Définitions sur les moteurs de jeux..... 04

http://www.poj.b3dgs.com/page.php?lang=fr&section=course_engine/1_intro

Ref 1.2 : Le game design..... 05

https://wikimonde.com/article/Game_design

Ref 1.3 : Rapport sur les ventes de jeux vidéo par l'NPD Group..... 06

https://www.afjv.com/news/9701_april-2019-u-s-video-game-sales-report-from-the-npd-group.htm

Ref 1.4 : Revenu du secteur américain du jeu vidéo..... 06

https://www.afjv.com/news/8448_us-video-game-industry-revenue-reaches-36-billion-in-2017.htm

Ref 1.5 : Bilan annuel du marché français du jeu vidéo..... 07

https://www.afjv.com/news/9546_chiffre-d-affaires-du-marche-francais-du-jeu-video-en-2018.htm

Ref 2.1 : Introduction à Twine 2.0..... 12

http://twinery.org/wiki/

Ref 2.2 : Prix catégorie Editor Extension 14

https://assetstore.unity.com/

Ref 2.3 : Comparaison des différentes méthodologies de travail.....23

https://www.planzone.fr/blog/tendances-management-projet-2017

https://www.planzone.fr/blog/topic/gestion-de-projet/page/15

https://www.supinfo.com/articles/single/5322-v-model-qu-est-ce-que-c-est-comment-utiliser

https://fr.slideshare.net/mbruchet/video2-agilite-etscalabiliteentreprise

http://nocremetz.free.fr/polytech/methodologies/methodologieUML.pdf

Ref 3.1 : Diagramme de classe 27

https://www.wikizero.com/fr/Diagramme_de_classes

Ref 4.1 : Configuration minimum requise 45

https://unity3d.com/unity/system-requirements

Ref 4.2 : Définition de Unity 46

https://blogs.unity3d.com/2019/04/16/introducing-unity-2019-1/

Ref 4.3 : Définition de VS Code 46

https://en.wikipedia.org/wiki/Visual_Studio_Code

Ref 4.4 : Définition de GitLab 47

https://about.gitlab.com/company/

Ref 4.5 : Définition d'Adobe Illustrator CC 19..... 48

https://fr.wikipedia.org/wiki/Adobe_Illustrator