LANOUZIERE Maxime – MOSNIER Bastien – PARIS Cédric

RIBIERE Laurent – SAYARH Nawhal

****

**Développement d'un jeu de type Puzzle Game à l'aide du Moteur de jeu Unity3D.**

Tuteur de projet : Pierre-Antoine PAPON

IUT de Clermont-Ferrand - Département Informatique

 Année 2015 - 2016

Nous autorisons la diffusion de ce rapport de projet sur l’intranet de l’IUT.

*Les mots en italique sont à retrouver dans le lexique du rapport.*

***REMERCIEMENTS***

Nous tenons à remercier dans un premier temps, toute l’équipe pédagogique de l’institut universitaire et technologique de Clermont-Ferrand ainsi que les intervenants professionnels responsables de notre formation pour l’aide et les conseils qu’ils nous ont apporté lors des différents suivis de projet.

**SOMMAIRE**

**REMERCIEMENTS………………………………………………………………………………..3**

**INTRODUCTION………………………………………………………………………………….5**

1. **PRESENTATION SYNTHETIQUE DU PROJET……………………..………………x**
2. **SITUATION INITIALE ET INTERET DU PROJET……………………………………….x**
3. **OBJECTIFS A REALISER…………………………………………………………………….x**
4. **MODALITES ET PLANNING PREVISIONNEL…………………………………………...x**
5. **ANALYSE ET CONCEPTION…………………………………………………………...x**
   1. **ANALYSE DU JEU A DEVELOPPER………………………………………………………..x**
      1. Présentation du moteur Unity………………………………………………………..x
      2. Présentation des autres outils utilisé……………………………………………..x
      3. Cas d'utilisation de l'application…………………………………………………….x
      4. Diagramme de classes………………………………………………………………….x
   2. **CONCEPTION DE L'APPLICATION…………………………….………………………….x**
      * 1. Génération d'un niveau…………………………………….………………………….x
        2. Édition d'un niveau………………………………………….…………………..……..x
        3. Menu de jeu……………………………………………………………………………….x
        4. Entité joueur……………………………………………………………………………….x
        5. Eléments jeu..................................................................................x
   3. **MODULES ET FONCTIONNALITES………………………………………………………..x**
      * 1. La gestion du son………………………………………………………………………..x
        2. La gestion de internationalisation………………………………………………….x
        3. Le système de sauvegarde…………………………………………………………..x
6. **BILAN DU TRAVAIL…………………………………………………………………….x**

**BILAN TECHNIQUE…………………………………………………………………………..x**

**CONCLUSION………………………………………………………………………………….x**

**ENGLISH SUMMARY…………………………………………………………………………x**

**BIBLIOGRAPHIE……………………………………………..………………………………x**

**LEXIQUE………………………………………………………………………………………..x**

**ANNEXE : DIAGRAMME DE CLASSE COMPLET DU JEU…………………..…….…..x**

**PRESENTATION DES SPRITES……………………………………………...x**

**INTRODUCTION**

Nous sommes un groupe d'étudiant en Informatique en deuxième année à l'Institut Universitaire et Technologique (I.U.T) de Clermont-Ferrand. Dans le cadre de notre cursus universitaire nous avons l'occasion de réaliser un projet de 5 mois dans le but de nous mettre en situation de travail d'équipe à long terme sur un sujet donné et de découvrir de nouveaux supports/moteurs de développement. Le sujet proposé par notre tuteur de projet est de réaliser un jeu 2D destiné aux plateformes mobiles.

Unity est un *moteur de jeu* professionnel dont la licence est gratuite tant que le chiffre d'affaire ne dépasse pas 100 000$/an. Unity permet de gérer le *Multi-Plateforme* et est utilisé à l'heure actuelle par beaucoup de développeurs de jeux. Il supporte et propose différents langages (JavaScript – C# - BOO), parmi lesquels nous avons choisi le C#, langage que nous connaissions déjà et qui nous semblait le plus adapté au projet.

De par les grandes avancées dans le domaine des technologies mobile et la démocratisation de celles-ci, l'intérêt de faire un jeu orienté Multi-plateforme est de toucher un maximum de «clients » potentiels. De plus l'intégration d'un tel jeu est facilité par *Google Play*. Cependant le développement mobile sous-entend d'exécuter le jeu sur des machines à performances moindres et à une taille d'écran réduite. De ce fait, la *conception* ainsi que la *qualité* du code auront un impact important sur le temps d’exécution.

Après vous avoir précisé les circonstances de la réalisation de notre projet, son but et les différentes étapes de sa mise en œuvre, nous aborderons les aspects développement, conception, fonctionnalités et tests de l’application. Nous finirons par la présentation de l’application en vous faisant part des difficultés rencontrées, accompagnées de possibles évolutions.

1. **PRESENTATION SYNTHETIQUE DU PROJET**
2. **Situation initiale et intérêt du projet**

Ce projet a pour but de répondre à un manque de jeux d'énigmes sur plateforme mobile.  Il y a également un intérêt économique puisque ce jeu contiendra un module de publicité de type *AdMob* et il pourra offrir la possibilité à l'utilisateur d'acheter du contenu pour progresser plus facilement dans le jeu.

Le produit final sera destiné aux utilisateurs possédant un *terminal* de type Android avec écran tactile. Le jeu pourra ainsi être exécuté par un smartphone ou une tablette et être publié sur le Google Play. Ce projet devra être réalisé sur 2 *périodes* soit 5 mois, des salles ayant été mises à notre disposition dans le cadre du projet les lundi matin de 8h à 12h au sein de l'IUT. Afin de se documenter et de nous former a l'utilisation du moteur de jeu Unity3D, nous utiliserons la documentation Unity ainsi que des vidéos de formation mis à notre disposition par notre tuteur. Nous travaillerons avec des environnement de développement comme *Visual Studio* ou *MonoDevelop* fournis avec Unity et des logiciels de traitements d'images pour les graphismes du jeu.

Nous avons décidé de développer le jeu en deux dimensions avec une vue du dessus. Le jeu se présente sous la forme d'une suite de niveaux de type énigme où l'utilisateur doit créer des chemins pour amener un voiture de la case de départ à une case d'arrivée.



1. **Objectifs à réaliser**

**Contraintes à respecter pour le jeu :**

Le jeu doit fonctionner sur les appareils mobile Android. L'application doit être concut a l'aide du moteur de jeu Unity3D et codé en langage C#. Le jeu doit être de type "Puzzle Game", c'est à dire basé sur la réflexion et où l'utilisateur doit placer des pièces ou des objets dans un ordre précis pour gagner.

Au-delà de ces quelques règles de base c'est à nous d'apporter des fonctionnalités supplémentaires (éditeur de carte, mode arcade...) de notre choix qui rendront le jeu plus intéressant et attractif pour le joueur.

**Contraintes pesant sur l'utilisation du produit :**

Il n'y a pas de contrainte notable pesant sur l'utilisation du produit. L'utilisateur doit simplement avoir un Smartphone ou une tablette Android pour pouvoir utiliser l'application.

**Critères d'appréciation de la qualité du produit :**

* Le jeu devra être facile d'utilisation pour toutes personnes même novices sur les technologies mobiles
* Le jeu devra avoir la fluidité adéquate demandée pour des plateformes mobiles
* Plusieurs niveaux jouables seront demandés
* Un interface clair pour accéder au jeu (Menus...)
* Une ambiance sonore
* Un module de publicité
* Un mode Arcade
* La possibilité d'augmenter la durée de vie du jeu (via un générateur de niveaux et un éditeur de niveaux)

**Principes du jeu :**

Le jeu se base sur le concept de l'énigme de la Minimobile du jeu "Professeur Layton et le Destin Perdu" sortie en 2010.



Case d'arrivée

Fanion / Objectif

Case de départ

Obstacle

Actions disponibles

Fig. n° : Interface du jeu Minimobile de Professeur Layton

L'utilisateur devra à partir d'une position de départ placer des éléments qui représente une action (flèches de changements de direction, sauts...) sur la carte afin que le personnage, symbolisé par une voiture, puisse récolter la totalité des ressources dites «objectifs» et arriver sur la case d'arrivée.

La quantité d'actions à placer sur la carte est limitée, l'utilisateur doit donc trouver le chemin le plus court possible pour amener son personnage a la case d'arrivée. Une fois les actions posées sur le terrain l'utilisateur devra cliquer sur "Play" pour faire avancer la voiture. En cas d'échec, c'est à dire si la voiture a rencontré un obstacle ou que la totalité des objectifs n'a pas été ramasser, l'utilisateur doit recommencer. En revanche, en cas de succès il accédera au niveau suivant après avoir vu une fenêtre de félicitation avec le score obtenu.

1. **Modalités et planning prévisionnel**

**Budget:**

Aucun budget n'est nécessaire à la réalisation du projet et au fonctionnement du jeu, le matériel pouvant être fournis à l'IUT si nécessaire.

**Planning:**

**Bastien**

1. **ANALYSE ET CONCEPTION**
   1. **ANALYSE DU JEU A DEVELOPPER**
      1. **Présentation du moteur de jeu Unity3D**

**MAXIME**

* + 1. **Présentation des autres outils utilisé**

Pour ce projet nous avons aussi utilisé le gestionnaire de versions Git/GitHub et le logiciel GraphicsGale pour les images du jeu.

**MAXIME**

**Git**

Pour notre projet, un gestionnaire de version était indispensable, pour que tout le monde puisse travailler sur des tâches différentes, sans avoir de problèmes pour réunir les différentes tâches. Un gestionnaire de versions permet de conserver et partager différentes versions du code-source d'un logiciel. On a choisi d'utiliser Git, car il est facile à utilisé, il est très répandu dans le milieu professionnel, c'est un logiciel libre et il fonctionne de manière décentralisé. Un gestionnaire de version décentralisé, permet aux membres d'une équipe à travailler chacun à son rythme, de façon désynchronisé ainsi que l'échanges des travaux respectifs. Cette désynchronisation a pour avantage :

* permet de ne pas être dépendant d'une seul machine.
* permet le travail hors-ligne.
* stockage de versions antérieurs à la version actuel d'un projet.
* réalisation plus rapide des tâches, car faites en réseau local
* etc …

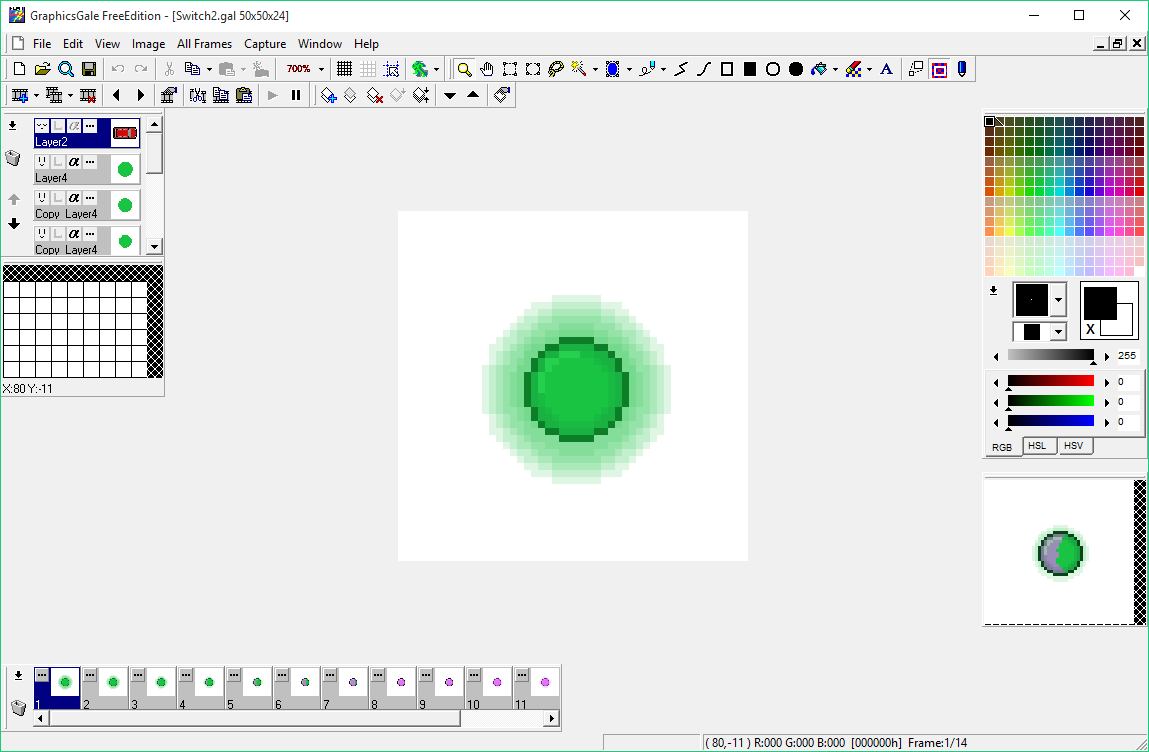
Mais le clonage d'un projet est long, car il faut copié tout l'historique du projet.

Git ne repose pas sur un serveur centralisé. Git utilise la *fonction de hachage SHA-1* pour calculer la somme de contrôle , si un fichier n'est pas modifié, la somme de contrôle ne varie pas, et il n'y a qu'une version de stocker sur le disque. Dans le cas contraire, la somme varie et les deux versions du fichiers sont stockées sur le disque. Git possède deux types de structures de données, une base objet et un cache de répertoires, la base d'objet permet de stocker n'importe quel type d'objet , qui par l'intermédiaire de la fonction de hachage SHA-1, est indexé dans le cache de répertoires.

**GitHub**

GitHub est service web d'hébergement, qui sert d'interface au logiciel de versions Git. Git propose des comptes gratuits, mais limités à cinq projets publics, et des comptes payants, sans limitations. GitHub propose l'hébergement de projets, ce qui évite les pertes de travail et permet aussi de partager des projets. GitHub offre aussi des fonctionnalités qu'on retrouve sur des réseaux sociaux, comme suivre un projet ou une personne. Il permet aussi de créer un wiki et une page web pour un dépôt, il peut aussi intégrer un grand nombre de services externes, comme un chat basés sur les projets, un gestionnaire de versions. La particularité de GitHub repose sur le fork, qui permet d'avoir une branche du projet original et en être le chef. Nous utilisons l'interface GitDesktop, de GitHub, pour voir et partager les modifications ajoutées au projet, et si besoins changer de versions, avec une interface simple d'utilisation.

**GraphicsGale**

GraphicsGale est un logiciel très utilisé pour le PixelArt, forme de graphisme que nous avons choisi pour notre jeu. Il est particulièrement adapté à la création de Sprites, nom donné aux images utilisées dans les jeux vidéo principalement 2D. Il permet notamment de pré-visualiser l’animation de la Sprite pendant sa création, ce qui a nettement facilité la tâche, notamment dans la création d’animations comme celle du reflet sur la voiture, ou celle de l’interrupteur. L’interface de ce logiciel est facile à prendre en main, ce qui est parfait car la personne s’occupant du design est débutante dans le domaine.

* + 1. **Diagramme de cas d'utilisation de l'application**

Afin de bien définir les rôles des éléments de notre jeu, nous avons établi un *diagramme de cas d'utilisations*.

* L'entité Player est l'utilisateur. Il peut lancer et quitter le jeu, lancer une partie, changer les paramètres et créer une carte.

****

* + 1. **Diagramme de classes**

Une bonne architecture de projet est une nécessité afin qu'il ne soit pas trop lourd à exécuter, principalement pour un projet ayant pour but les plateformes mobiles.

L'architecture de notre jeu est divisée en 3 parties principales:

* La partie “Player”, qui s'occupe du comportement du personnage (voiture) :



* La partie “Element”, qui s'occupe du comportement des différents éléments du jeu ayant une interaction avec le personnage:



* La partie “Draggable”, qui s'occupe du comportement des éléments pouvant être déplacés par le joueur (et non le personnage):



* 1. **CONCEPTION DE L'APPLICATION**
     + 1. **Génération d'un niveau**

Notre application inclus un générateur aléatoire de niveaux qui est capable de générer des niveaux suivant différents paramètres qui sont le nombre d'actions nécessaires pour résoudre le niveau ainsi que la hauteur et la largeur du niveau en nombre de cases. Ce générateur est nécessaire au mode de jeu dit "Arcade" inclus dans notre application qui consiste à proposer à l'utilisateur une série de niveaux de plus en plus complexes, le but étant de ne jamais commettre d'erreur pour éviter de recommencer une série du début. Il peut aussi nous servir a générer les bases d'un niveau que l'on souhaite ajouter dans la liste des niveaux "pré-générés" proposés à l'utilisateur.

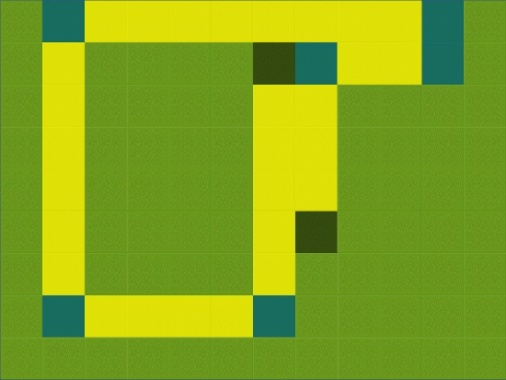
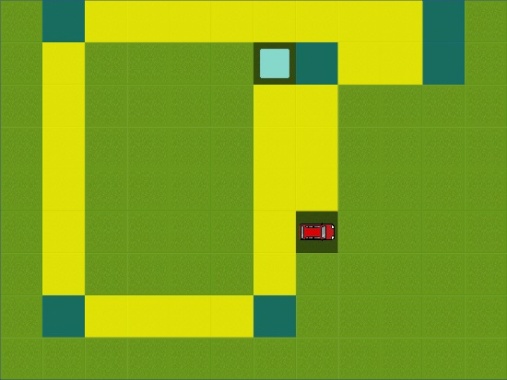


Fig. n° : Exemple de niveau générer pour 6 changements de directions

**Algorithme de Génération :**

Générer une énigme (niveau de jeu) qui soit suffisamment complexe pour que l'utilisateur ne découvre pas directement la solution n'est pas facile car il faut masquer les chemins évidents à emprunter pour résoudre l'énigme. La faible capacité de calcul d'un appareil mobile comparée à un ordinateur est également un point important à prendre en compte pour que l'utilisateur n'ait pas à attendre plusieurs dizaines de secondes avant d'obtenir un niveau fonctionnel. Nous avons donc conçu un algorithme qui prend en compte ces différentes considérations.

L'algorithme ne génère pas une carte aléatoire et cherche la solution la plus efficace mais est basé sur le principe inverse. Il va commencer par déterminer un chemin suivant le nombre d'actions nécessaire pour résoudre le niveau et va placer aux extrémités la case de départ et la case d'arrivée. Il va ensuite générer l'environnement autour de ce chemin de façon à ce que le chemin établit soit la ou l'une des seules solutions à l'énigme. Pour obliger l'utilisateur a passer par les chemins établis il suffit de placer correctement les objectifs sur le chemin puisqu'il faut absolument ramasser tous les objectifs pour gagner la partie. On placera donc de manière aléatoire des objectifs en nombre suffisant sur les différentes sections du chemin prédéfini. Pour terminer, on ajoute des obstacles pour bloquer d'autres solutions qui seraient encore possibles et rendre le niveau plus crédible.



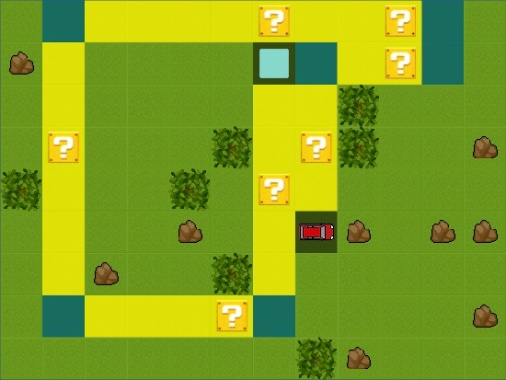


Fig. n° : Phases de génération d'un niveau

* + - 1. **Édition d'un niveau**

L'éditeur de niveau permet à l'utilisateur de créer ses propres niveaux de jeu en ayant la possibilité de placer sur la scène tous les objets accessibles dans les niveaux normaux (cases, objectifs...) et de choisir les actions qu'il souhaite donner à celui qui devra résoudre son énigme. L'utilisateur a évidement la possibilité de sauvegarder son niveau personnalisé et de le jouer pour le faire essayer dans son entourage.

Cet éditeur de niveau pourra également être utilisé plus tard pour partager des cartes personnalisées sous forme de fichier ou par l'intermédiaire d'une plateforme multi-joueurs.

**L'interface d'édition de niveau**



Zone d'édition où l'on peut placer des objets pour composer le niveau

Objet sélectionné

Ouvre et ferme le panneau latéral

Fig. n° : Interface d'édition de niveau

Bouton Caméra

Bouton Play

Bouton Suppression

Panneau actions

L'interface est composée de 3 panneaux. Le panneau de gauche peut être fermé pour ne pas gêner l'édition du niveau. Il contient différentes options, notamment pour sauvegarder ou récupérer une sauvegarde de niveau. Le panneau de droite contient tous les objets rangés par sections qui peuvent composer la carte. Enfin le panneau en bas permet de définir les actions qui seront accessible à celui qui jouera le niveau et en quelle quantité. La quantité est affichée pour chaque action (ici 0 sur chaque action). Pour incrémenter cette quantité, il faut cliquer soit sur la gauche de l'action pour incrémenter le compteur, soit sur la droite pour décrémenter le compteur.

L'utilité des 3 boutons en bas de l'interface est expliqué dans les parties suivante.

**Bouton Camera**

Le joueur peut éditer une carte qui s'étend au dela de la taille de l'écran. La camera est fixé par défaut pour éviter les mouvements de caméra non souhaités lorsque l'utilisateur touche l'écran. En cliquant sur le bouton "Camera", l'utilisateur peut donc activer le déplacement de la camera (Mode Caméra). cela a pour effet de désactiver l'ajout d'éléments sur le niveau mais l'utilisateur peut dès lors déplacer la caméra avec un mouvement de glissement sur l'écran. Pour refixer la caméra quand elle a était déplacé a la position voulue, il suffit de cliquer de nouveau sur le bouton pour désactiver le mode camera.

**Bouton Play**

L'édition d'un niveau a évidement pour but de pouvoir le jouer ou le faire jouer, c'est ce que permet ce bouton. Il a pour effet de charger la carte personnaliser dans une scène vide contenant uniquement un menu avec les actions défini dans l'éditeur de niveau.

**Bouton Suppression**

Ce bouton permet de supprimer des éléments posé par l'utilisateur sur la carte personnalisé. Quand il est cliqué, l'utilisateur ne peut plus poser d'objet sur la carte, au contraire, les objets sous ces doigts sont supprimés.

**Ajout / Suppression d'objets sur la carte**

La carte que l'utilisateur édite est placée dans un GameObject appelé TileMapEditor auquel est affecté deux scripts, PutEditorTiles et RemoveEditorTiles. Ces deux scripts agissent sur le contenu de la carte, le premier sert à poser des objets, l'autre les supprime. Il ne peuvent pas être actifs au même moment sinon l'un supprimerait ce que l'autre vient d'ajouter. Le script PutEditorTiles est donc actif par défaut mais un clic sur le bouton suppression désactive le script PutEditorTiles et active RemoveEditorTiles pour permettre la suppression d'objets sur la carte.

Ils fonctionnent sur un principe similaire : ils détectent la position des doigts sur l'écran et calculent la case pointée par l'utilisateur. Ensuite si PutEditorTiles est actif il ajoute l'objet sélectionné dans le menu à la carte, si RemoveEditorTiles est actif il supprime tout objet à cet position.

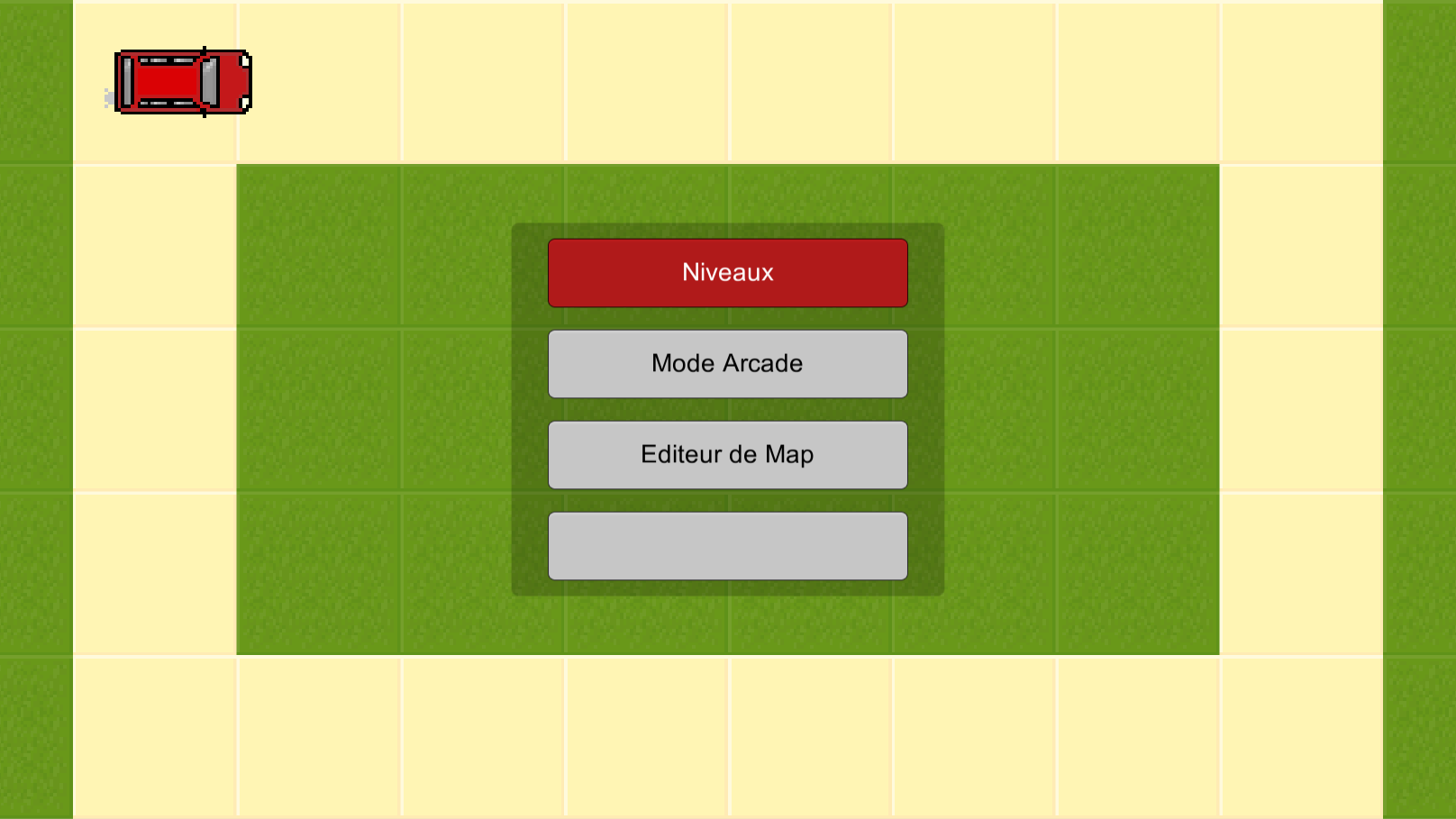


Position calculée pour positionner l'objet

Position du doigt de l'utilisateur

Fig. n° : Éditeur de niveaux : position écran / position calculée

* + - 1. **Menus**
         1. **Menu Principal**

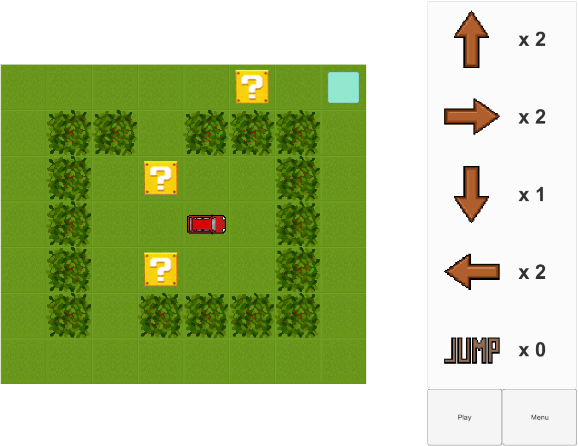
Le menu principal est le menu permettant d’accéder aux différents aspects de notre jeu. Lorsque nous lançons le jeu, le menu principal est le premier écran que l’on voit.

Différents boutons s’offrent alors à nous :

* Le premier bouton nous amène dans une interface de choix de niveaux dont nous parlerons plus loin.
* Le deuxième bouton mène vers le mode Arcade, mode de jeu utilisant le générateur de niveaux, présenté plus haut, afin que le joueur soit confronté à une série de niveaux, sachant qu’une simple erreur conduit à la fin de la partie.
* Le troisième bouton mène vers l’éditeur de niveaux, également détaillé plus haut.

Afin de pouvoir naviguer de ce menu à une autre *Scene*, le script SceneLoader est associé au menu. Ainsi, quand on appuie sur un bouton du menu, la méthode LoadScene() chargera la bonne *Scene*.

* + - * 1. **Menu de jeu**

Ce que l’on appelle menu du jeu est la barre latérale présente en jeu qui permet à l’utilisateur de choisir et poser les éléments de mouvement : flèches et saut. Elle est composée de 5 boutons, un par élément de mouvement, à côté desquelles sont affichés les nombres d’éléments que le joueur peut encore poser sur le jeu.

Il est facile à prendre en main : si l’on veut poser un élément de mouvement dans le jeu, il suffit de faire glisser son doigt du bouton de l’élément que l’on veut à la case où on veut poser notre élément. Si on change d’avis et que l’on veut enlever un élément de mouvement du jeu, il suffit de le glisser vers le bouton du menu qui lui correspond et il disparait.

Malgré son apparente facilité, l’implémentation du menu a posé un problème principal : l’interaction UI / non-UI.

Sur Unity, nous disposons de plusieurs plans pour poser nos objets. Un objet dit UI, User Interface, est un objet qui se situe sur le plan UI. Sur ce plan, les objets ne bougent pas avec la caméra. Ce plan est très utilisé que ce soit pour avoir une barre de vie ou pour avoir certains boutons accessibles en tout temps à partir de la vue de jeu. Tout objet n’appartenant pas à ce plan est alors appelé non-UI.

Pour revenir à notre menu de jeu, ce dernier est un objet UI, car si la carte de jeu est plus grande que l’écran ou que le joueur veut zoomer, le menu ne doit pas bouger. Cependant, le menu doit générer des éléments de mouvements non-UI car, eux, doivent être zoomés et bougés si l’utilisateur zoome et bouge la carte. C’est ici qu’apparaît le problème de l’interaction UI / non-UI.

Ce qui rend cette interaction difficile est qu’il n’est pas possible de passer un objet d’un plan à l’autre. En effet, les éléments UI ont un élément de positionnement RectTransform différent de l’élément de positionnement Transform des autres éléments, car contrairement aux autres c’est par rapport à l’écran et non au monde qu’ils doivent se placer. On ne peut transformer l’un en l’autre, on ne peut donc pas transformer un élément UI en non-UI et vice-versa. C’est pourquoi il est impossible de passer des éléments entre le plan UI et les autres. Ceci empêche également la plupart des évènements de passer entre un plan et l’autre, rendant l’interaction entre les deux plans très compliquée.

Dans notre première implémentation de menu, nous contournions le problème en ne passant pas par des évènements entre le menu et les éléments de mouvement mais entre le doigt et les éléments puis entre le doigt et le menu. Pour glisser un élément du jeu au menu, nous faisions comme suit :

* Quand un élément commence à se faire glisser par un doigt, il l’indique au menu.
* Si le doigt passe sur le menu et qu’un élément se fait glisser, le menu dit à l’élément de s’autodétruire.
* Le menu trouve le texte correspondant à l’objet et augmente le compte d’éléments.

Le problème de cette implémentation était que n’importe quelle classe pouvait faire croire qu’un élément se faisait glisser et ce défaut d’*encapsulation* pouvait résulter en des destructions d’éléments de mouvements sans raisons apparentes.

Nous avons alors pris une toute autre approche. À présent, le menu en lui-même est purement visuel car c’est les boutons qui gèrent le menu à présent. Chaque bouton du menu a un fonctionnement décrit dans le script MenuButton. Chaque MenuButton est associé à un élément de mouvement, que ce soit une des flèches ou le saut. Ces MenuButton sont équipés de Collider2D, élément permettant de détecter les collisions avec d’autres objets. Le déroulement est donc celui-ci :

* Un objet équipé d’un Collider2D a une collision avec le Collider2D du MenuButton.
* Le MenuButton vérifie que ce soit l’élément de mouvement qui lui soit associé, si ce n’est pas le cas il ne fait rien.
* Le MenuButton dit à l’élément de s’autodétruire et augmente le compte d’élément qui lui est associé.

Pour ce qui est de faire glisser un élément de mouvement du menu de jeu à la carte, nous bravons la difficulté d’une autre manière. Nous avons créé un élément UI déplaçable pour chacun des éléments de mouvement. A chaque fois qu’il reste un élément ou plus de disponible dans le menu, un de ces éléments UI déplaçable est généré au-dessus du MenuButton correspondant. Ainsi, le joueur doit juste déplacer cette élément UI là où il veut son élément de mouvement, et l’élément UI déplaçable génèrera cet élément de mouvement avant de s’autodétruire.

* + - 1. **Entité joueur**

On appel "Joueur" l'avatar de l'utilisateur (voiture) qui se déplace sur la carte. Il est composé de 3 grandes entités: une qui le représente graphiquement (classe Player), une qui s'occupe de gérer ses déplacements (classe PlayerMovementController) et une dernière dont le rôle est de détecter les éléments qui se trouvent devant lui. Cette dernière entité est composée de deux observateurs d'éléments. Le premier sert à détecter les éléments que le joueur rencontre physiquement (obstacles). Le second détecte les éléments qui sont sous le joueur et qui ont un effet direct sur son comportement ou qu'il peut attraper ou actionner, par exemple une flèche qui fera changer le joueur de direction ou un objectif qu'il doit ramasser pour gagner.



Fig. n° : Diagramme de classe du Joueur

Le Joueur est programmé selon un principe simple : il avance en ligne droite et à vitesse constante en attendant qu'un élément posé sur son chemin agisse sur son comportement (changement de direction...). L'entité Joueur répète en permanence le même schéma :

Le PlayerMovementController demande aux observateurs d'éléments si ils ont détecté un élément. Si c'est le cas pour l'observateur d'obstacle, le PlayerMovementController va lancer un événement indiquant que le Joueur doit exploser et dans ce cas l'entité qui s'occupe du rendu graphique du Joueur va afficher l'animation d'explosion.

Si le second observateur d'éléments a détecté un élément, le PlayerMovementController va prendre en compte l'effet de cet élément - par exemple en modifiant la direction courante - et le rendu graphique du joueur va changer en conséquence si nécessaire.

Ensuite le PlayerMovementController fait avancer le Joueur dans la direction courante et on recommence le même schéma.

On peut résumer ce fonctionnement répétitif par le diagramme suivant :

**Contrôleur de mouvement**

PlayerMovementController

**Représentation graphique**

Player

**Observateur d'éléments physiques (obstacles...)**

ElementObserver

**Observateur d'éléments à effets ou actionnables**

ElementObserver

1: interroge

2: interroge

4: notifie les changements graphiques

3: Applique les effets des éléments rencontrés

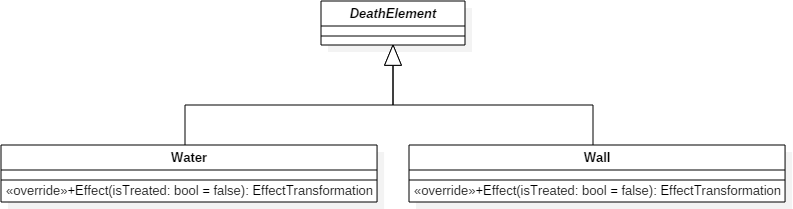
Fig. n° : Schéma du fonctionnement de l'entité Joueur

* + - 1. **Eléments du jeu**

On appel "élément" tous ce qui est posé sur la carte de jeu et qui interagie avec le Joueur. Ces éléments de jeu peuvent influer sur son comportement (changement de direction...), être ramasser (Objectifs) ou actionner(leviers de ponts...). Les éléments de jeu sont divisible en 4 parties distinctes selon l'effet qu'il applique sur le Joueur.

* **Les "Death Elements" ou éléments entrainant la mort**

Les "Death Elements" sont des éléments entrainant la mort du joueur lors de sa collision avec eux. Ils peuvent être à la fois liquides (Water) ou solide (Wall).

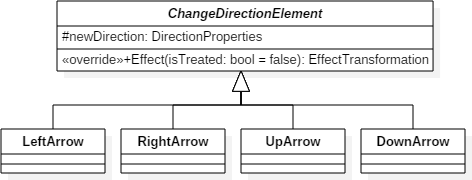


C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Water.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Mountain.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Tree.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Rock.png

Fig. n° : Hiérarchie des "Death Elements"

* **Les "Change Direction Elements" ou éléments modifiant la direction**

Les "Change Direction Elements" sont les éléments qui on un effet sur la direction du joueur. Quand le joueur les rencontres, ils lui impose une des quatres direction possible pour le joueur (Nord, Est, Sud, Ouest).

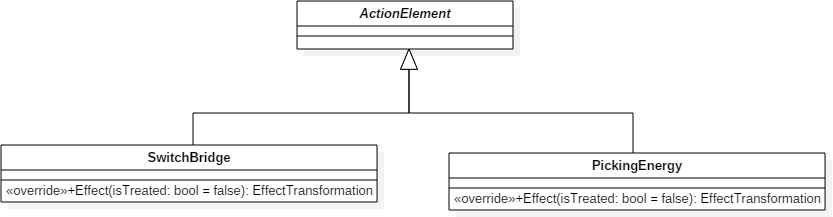


C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Arrows.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Arrows.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Arrows.png

C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Arrows.png

Fig. n° : Hiérarchie des "Change Direction Elements"

* **Les "Action Elements" ou éléments actionnables**

 Les "Action Elements" sont des éléments que le joueur peut actionner ou ramasser quand il passe dessus. C'est notament le cas des objectifs que le joueur doit collecter pour terminer le niveau.

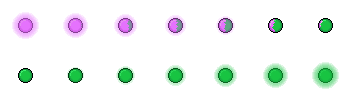
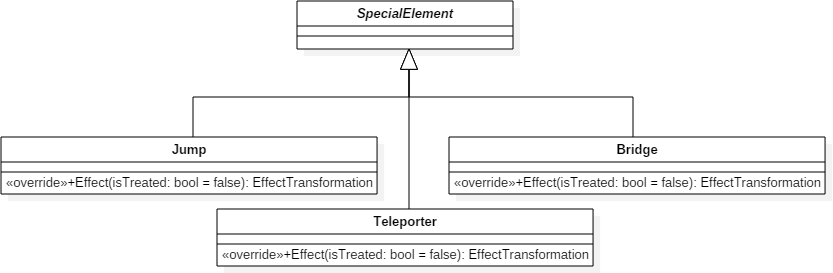
C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\ObjectifTemporaire.png

Fig. n° : Hiérarchie des "Action Elements"

* **Les "Special Elements" ou éléments à effets spéciaux**

Les "Special Elements" sont les éléments qui ne peuvent pas être classés dans une des catégorie précédente car leur effet est très particulier. Par exemple, l'effet de l'élément Pont est variant suivant si il est ouvert ou fermé. si le pont est fermé, il n'y a pas d'effet sur le joueur mais si le pont est ouvert alors son effet est d'entrainer la mort du joueur.



C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Teleport.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Bridge.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Jump.png

Fig. n° : Hiérarchie des "Special Elements"

La hierarchie complète des éléments dans notre application est résumé par le diagramme en annexe.

* 1. **MODULES ET FONCTIONNALITES**
     + 1. **La gestion du son**

**LAURENT**

Unity offre la possibilité de jouer des sons. Ainsi il a été possible de jouer un son lors de certaines collisions. Nous avons décidé de déléguer cette tâche à une classe SoundControl. Cette classe possède une méthode “Play” qui permet de jouer un son en fonction d'une audio\_music, qui possède une audioSource (piste de musique).

La classe ButtonSound permet le contrôle du son par l'utilisateur via des bouttons (pause/Play/Mute/UnMute). La méthode “OnMouseDown”, détecte un clique sur la souri, et appellera la méthode du SoundControl adéquate. La hiérarchie du son est la suivante :



Les sons (musique en l'occurence ici), sont en sont “Hill Of Sorrow” et “Murya”, ils ont été récupérer sur youtube, et son en “*.Mp3*”

* + - 1. **La gestion de l'internationalisation**

Notre application est traduite automatiquement suivant la langue de l'appareil sur lequel elle est lancé. Nous avons fait en sorte que les appareils français ait une application en français et que les autres appareils aient une application en anglais. Ces traductions sont basé sur des fichiers JSON qui font correspondre une traduction à une clé qui est identique sur chaque fichier. De cette façon le fichier anglais (string\_en.json) fait correspondre à la clé "textPlay" la valeur "Play" alors que le fichier français (string\_fr.json) fait correspondre à la même clé "textPlay" la valeur "Jouer". Ainsi il est très simple de rajouter une langue, il suffit de créer un fichier Json pour cette langue et l'application pourra être traduite (par exemple string\_es.json pour une traduction espagnol).

**Mécanisme de traduction**

Quatre classes sont impliqué dans la traduction des textes de notre application.

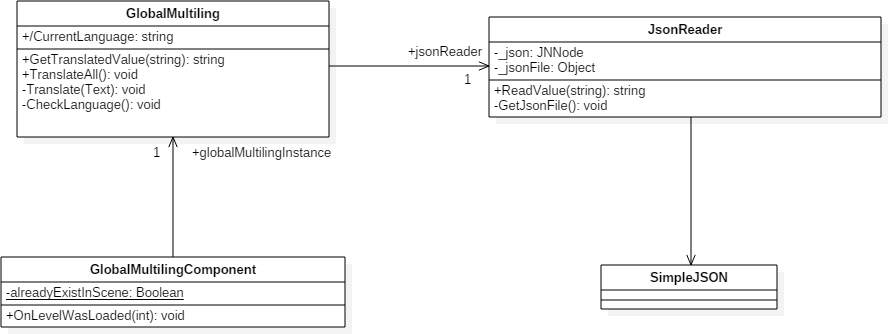


Fig. n° : Classes impliquées dans le mécanisme de traduction

En réalité, l'entité SimpleJson n'est pas une classe mais le nom d'un fichier mis a notre disposition par la communauté Unity qui contient plusieurs classes permettant de lire un fichier JSON et d'en extraire des couples clé - valeur. Pour facilité l'explication du mécanisme de traduction nous le considérerons comme une seule classe. Les classes GlobalMultiling et GlobalMultilingComponent sont les classe qui s'occupe directement de traduire les textes dans l'application. GlobalMultiling est capable de traduire la totalité des textes d'une scène ou de traduire uniquement un texte donné. GlobalMultilingComponent ne fait qu'utiliser la classe GlobalMultiling mais elle a la particularité de pouvoir être associé a un GameObject dans une scène. Enfin la classe JsonReader fait le lien entre ces deux classes et la classe SimpleJson. Il suffit de lui donner une clé et elle s'occupe de demander la lecture d'un fichier JSON pour obtenir la valeur associée.

Le fait que la classe GlobalMultilingComponent puisse être associé a un GameObject lui permet de détecter le chargement d'un nouvelle scène. Quand elle détecte qu'une scène est chargé elle recherche dans cette nouvelle scène tout les textes et demande leur traduction au JsonReader. Ainsi la totalité des scènes de l'applications sont traduite à l'utilisateur.

Pour les éléments qui ne font pas partie directement des scènes (messages affiché a l'utilisateur...) et qui apparaissent uniquement quand l'utilisateur fait certaines actions, on demande la traduction du texte à afficher directement à la classe GlobalMultiling.

* + - 1. **Le système de sauvegarde**

Le système de sauvegarde peut être séparé en deux parties distinctes: la sauvegarde de la progression de l'utilisateur et la sauvegarde d'un niveau du jeu.

**Sauvegarde de la progression**

La classe ProgressionSave a pour responsabilité de sauvegarder la progression de l'utilisateur dans le jeu, c'est a dire enregistrer quels niveaux il a déjà terminé et avec quel score. ProgressionSave utilise la classe PlayerPrefs fournis par Unity qui permet de conserver de petite données entre plusieurs sessions de jeu. Pour chaque donnée que l'on enregistre on associe une clé. On peut ainsi récupérer les donnée a partir de leurs clés (par exemple le score obtenu au niveau 2 avec la clé "ScoreLevel2").

Au niveau graphique la progression est représenté par des étoiles pour chaque niveau, trois étoile indiquant un bon score et une indiquant un score faible. Les niveaux non débloqué apparaissent grisé et ne sont pas accessible. Pour connaitre les étoiles a afficher et les niveaux non débloqué, on enregistre le dernier niveau débloqué et le score pour tous les niveaux précédents.

**Sauvegarde de niveaux**

La sauvegarde de niveaux est gérée par la classe LevelSave. Cette classe est capable d'écrire et de lire des données dans un fichier XML. Pour sauvegarder un niveau, on sérialise son contenu (carte et actions) dans un fichier. Pour cela, elle a besoin des classes TileMapSave et Vector3Save.

- La classe Vector3Save est une classe sérialisable permettant de stocker les coordonnées d'un Vector3, ce qui nous permet de sauvegarder la position d'un élément.

- La classe TileMapSave est une classe sérialisable qui contient la liste des éléments d'une scène (plus précisément de la TileMap) avec leur position sous forme de Vector3Save. Elle contient aussi le nombre d'actions utilisables par le joueur pour résoudre le niveau, sous la forme d'un dictionnaire associant un entier à un élément.

Ainsi, lorsque l'on souhaite sauvegarder le contenu de notre level, LevelSave parcours chaque élément de la TileMap, et les ajoute à une TileMapSave avec leur position. Le dictionnaire d'éléments utilisables passé en argument à la fonction est lui aussi ajouté à la TileMapSave. Ensuite cet ensemble est sérialisé dans un fichier.

Il est ensuite possible de récupérer la liste des fichiers de niveaux sauvegardés, et d'en sélectionner un pour le charger. On va alors réaliser la démarche inverse. De ce fichier, on va récupérer une TileMapSave contenant les différents éléments du jeu.

On va alors pour chaque élément récupérer le nom du préfab à instancier (chaque élément du jeu ayant un préfab, il suffit de savoir quel préfab doit être placer et de l'instancier) ainsi que sa position. On vide alors la TileMap actuelle pour s'assurer que le niveau est vide, avant d'y insérer chaque élément lu dans le fichier. Enfin on renvoie le dictionnaire contenant la liste des éléments utilisables qui était stocké dans la TileMapSave récupérée.

*Lexique:*

*Sérialisation / Sérialiser :*

*Sérialiser un objet signifie le changer en un flux que l'on pourra sauvegarder. Dans ce projet, la sérialisation consiste à écrire le contenu de nos objet au format xml dans un fichier de sauvegarde.*

* 1. **TESTS**

**A FAIRE**

1. **BILAN TECHNIQUE**

Ce projet avait pour objet de développer un jeu vidéo en deux dimensions. Ainsi, nous avons mis au point un jeu mobile capable d'être utilisée aussi bien en mode tactile qu'avec une souris.

Les deux périodes de projet nous ont permis d'appliquer concrètement nos connaissances mais aussi de les enrichir. Nous avons très fortement renforcer nos connaissances du langage C#. Nous avons également appris à utiliser le moteur Unity3D, le gestionnaire de version Git utilisant le serveur de stockage GitHub.

Ce qui a finalement pris le plus de temps dans le développement a été la création du menu. Il fallait un menu détectant parfaitement le déplacement d'un objet dans la partie du jeu et celle où sont stocker les éléments. De même qu'il était nécessaire d'avoir un menu ergonomique, simple d'utilisation.

**CONCLUSION:**

**Rappel du sujet :** Création d'un jeu pour plateformes mobiles avec le moteur Unity3D et le langage C#.

Dans le cadre de ce projet, les objectifs ont été très clairs. Il nous fallait réaliser un jeu vidéo en C#. Afin de développer cette application, il nous a été demander d'utiliser le moteur Unity3D. Cela nous a faciliter le développement du jeu grâce aux fonctionnalités que ce moteur nous apporte. La conception avant le développement a été une étape clef du projet afin de savoir dans quelle direction s'orienter. Nous avons ainsi décrit l'analyse et le développement de l'application avec de nombreuse illustrations et explications d'utilisations.

Ce projet était une opportunité qui s'est révélée être très intéressante et enrichissante, dans la mesure où il nous a permis d'enrichir notre connaissance du langage de programmation C# ainsi que l'utilisation du moteur Unity3D. En effet, nous avons été confrontés à un nouveau mode de développement, avec un travail en groupe réel, sur une longue période, où il fallait s'entre aider afin de régler des problèmes pouvant être fréquents. Cela a également été notre première expérience dans le développement d'un jeu vidéo. Où il était nécessaire de se mettre réellement à la place du joueur, afin de développer. Ce projet nous a également permis d'approfondir certaines méthodes de travail, à savoir la plateforme de collaboration GIT, qui s'est avérée très utile pour le travail en équipe. De plus, travailler en groupe de cinq étudiants était une excellente expérience qui nous a donné un avant-goût du travail d'équipe en entreprise, tout en développant nos compétences humaines, telles que la communication, l'ouverture d'esprit et l'organisation. Chaque membre du groupe a pu appliquer les connaissances et les méthodes de travail qu'il avait préalablement étudié en cours à l'IUT de Clermont-Ferrand, au département informatique.

**LEXIQUE**

**Terminal** : Désigne à l’origine l’extrémité d’un réseau informatique (point de communication entre l’homme et la machine). On emploie ce terme par abus de langage pour faire référence à un appareil informatique quelconque (smartphone, tablette, ordinateur).

**Périodes** : Zone temporaire signifiant 7 semaines de cours active, au sein de l'IUT.

**IDE** : (Environnement de développement) constitue un ensemble d’outils pour augmenter la productivité des programmeurs.

**Editeurs graphiques** : Collection de programmes permettant de manipuler et de traiter des dessins ou images.

**Visual Studio** : Suite de logiciels de développement pour Windows.

**MonoDevelop** : Logiciel de développement lié à Unity

**Moteur de jeu** : Ensemble de composant logiciel qui effectuent des calculs de géométrie et de physique utilisés dans les jeux vidéo

**Multi-Plateforme** : Un logiciel multiplate-forme est un logiciel conçu pour fonctionner sur plusieurs plates-formes, c'est-à-dire le couple liant ordinateur et système d'exploitation.

**Google Play** : Boutique en ligne créée par Google (le 6 mars 2012) par fusion des services Android Market.

**Conception** : Ensemble d'activités qui effectue simultanément l'étude des données et l'étude des traitements à effectuer.

**Qualité** : Appréciation d'un logiciel basée sur de nombreux indicateurs.

**AdMob** : Société de publicité sur mobile fondée par Omar Hamoui en 2006.

**Gestionnaire de version** : Consiste à maintenir l'ensemble des version d'un ou plusieurs fichiers. Essentiellement utilisé dans le domaine de la création de logiciels, elle concerne surtout la gestion des codes source.

**.Mp3** : Algorithme de compression audio

**Internationalisation** : Permet la compatibilité d'un logiciel avec les différentes parties du monde.

**diagramme de cas d'utilisations** : Donne une version globale du comportement fonctionnel d’un système.