

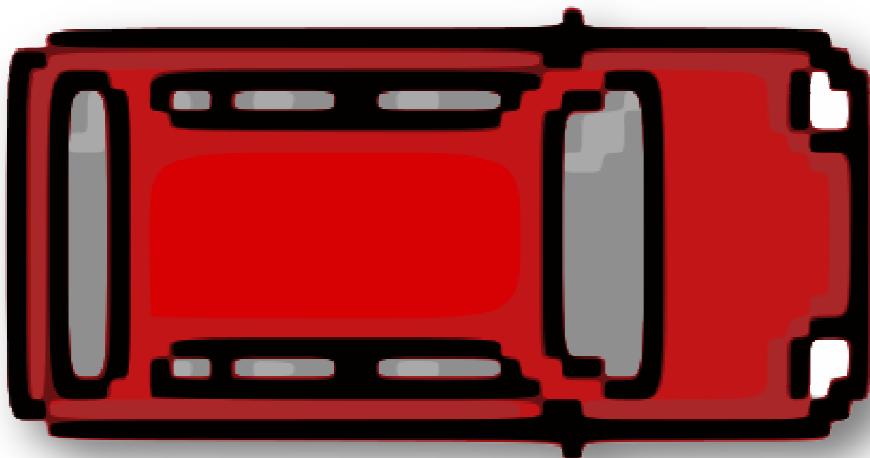
LANOUZIERE Maxime – MOSNIER Bastien – PARIS Cédric  
RIBIERE Laurent – SAYARH Nawhal



# 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.

## *Remerciements*

Nous tenons à remercier, dans un premier temps, notre tuteur de stage M. Pierre-Antoine PAPON pour nous avoir donné l'opportunité de réaliser ce projet et pour nous avoir guidé dans sa réalisation.

Nous souhaitons également remercier 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*

<b>Introduction .....</b>	<b>5</b>
<b>I. Présentation synthétique du projet .....</b>	<b>6</b>
I.1. Situation initiale et intérêt du projet .....	6
I.2. Objectifs à réaliser .....	6
I.3. Modalités et planning prévisionnel .....	8
<b>II. Analyse et conception .....</b>	<b>12</b>
II.1. Analyse du jeu à développer .....	12
II.1.1. Présentation du moteur de jeu Unity3D .....	12
II.1.2. Présentation des autres outils utilisés .....	13
II.1.3. Diagramme de cas d'utilisation .....	15
II.1.4. Diagramme de classe de l'application .....	16
II.2. Conception de l'application .....	17
II.2.1. Génération d'un niveau .....	17
II.2.2. Édition d'un niveau .....	19
II.2.3. Menus .....	22
II.2.4. Entité joueur .....	26
II.2.5. Éléments du jeu .....	28
II.3. Modules et fonctionnalités .....	30
II.3.1. Gestion du son .....	30
II.3.2. Gestion de l'internationalisation .....	31
II.3.3. Système de sauvegarde .....	32
<b>III. Bilan technique .....</b>	<b>34</b>
<b>Conclusion .....</b>	<b>35</b>
<b>Abstract .....</b>	<b>36</b>
<b>Lexique .....</b>	<b>37</b>
<b>Webographie .....</b>	<b>38</b>
<b>Annexes .....</b>	<b>39</b>

Les mots en italique sont à retrouver dans le lexique page 37

# ***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 multiplateforme 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 connaissons déjà et qui nous semblait le plus adapté au projet.

De par les grandes avancées dans le domaine des technologies mobiles et la démocratisation de celles-ci, l'intérêt de faire un jeu orienté multiplateforme est de toucher un maximum de « clients » potentiels. 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.

# I. Présentation synthétique du projet

---

## I.1. Situation initiale et intérêt du projet

Ce projet a pour but de répondre à un manque de jeux d'énigmes sur plateformes mobiles. 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 lundis matin de 8h à 12h au sein de l'IUT. Afin de se documenter et de nous former à l'utilisation du moteur de jeu Unity3D, nous utiliserons la documentation Unity ainsi que des vidéos de formation mises à notre disposition par notre tuteur. Nous travaillerons avec des environnements de développement comme Visual Studio ou MonoDevelop, qui est fourni avec Unity, et des logiciels de traitement 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 une voiture de la case de départ à une case d'arrivée.



## I.2. Objectifs à réaliser

### Contraintes à respecter pour le jeu

Le jeu doit fonctionner sur les appareils mobiles Android. L'application doit être conçue à 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 une fluidité adéquate pour des plateformes mobiles
- Plusieurs niveaux jouables seront demandés
- Une interface claire 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 niveau et un éditeur de niveau)

## Principes du jeu

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

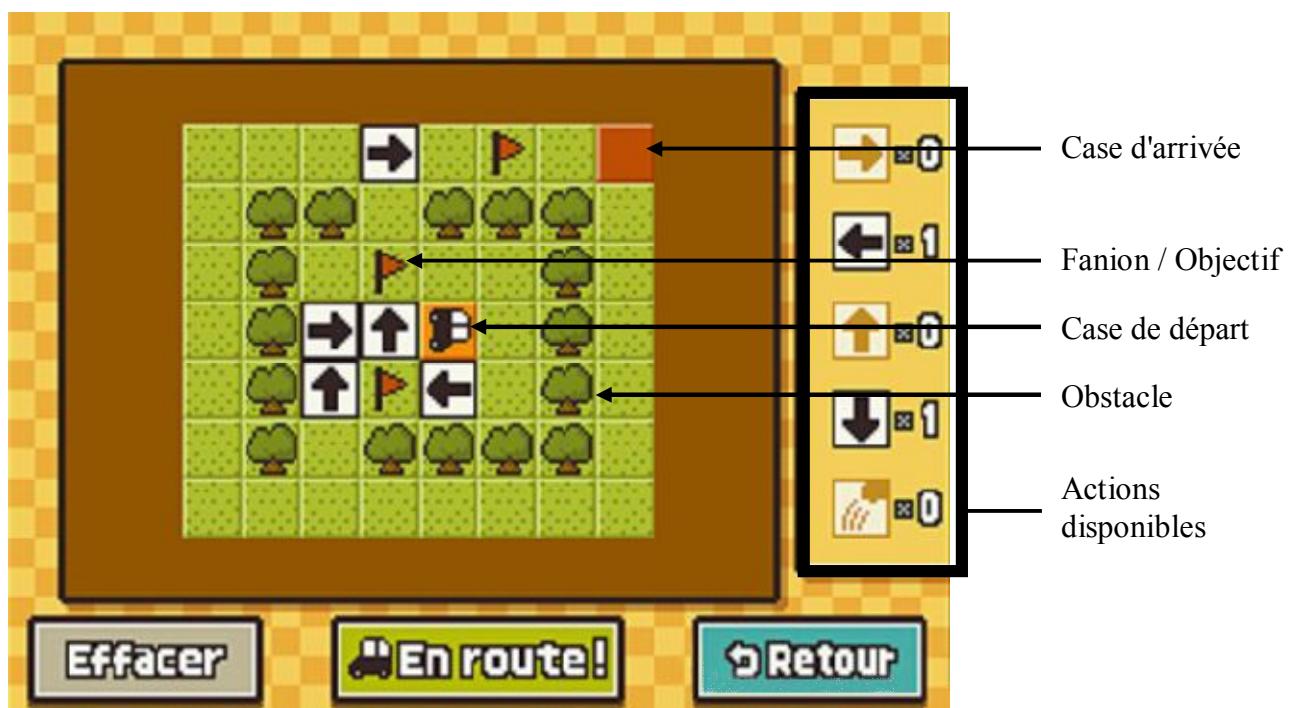


Fig. n°1 : Interface du jeu Professeur Layton

L'utilisateur devra à partir d'une position de départ placer des éléments qui représentent 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 à 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 ramassée, l'utilisateur doit recommencer. En revanche, en cas de succès il accèdera au niveau suivant après avoir vu une fenêtre de félicitations avec le score obtenu.

## ***I.3. Modalités et planning prévisionnel***

### **Budget**

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

### **Planning**

Ce projet s'est étendu sur deux périodes de deux mois et demi. Nous avons fait en sorte pour chacune de ces périodes de définir et de répartir équitablement les tâches à réaliser en début de période pour que chacun puisse organiser son temps de travail.

Durant la première période, une partie des tâches a été réalisée par l'ensemble du groupe, notamment la formation à Unity, le début de la modélisation objet du projet et la création d'une partie des éléments du jeu (flèches, obstacles...). D'autres tâches ont été confiées spécifiquement aux membres de l'équipe.

- Cédric s'est vu attribuer la création de l'entité "Joueur" dans le jeu, d'un éditeur de niveau, ainsi que de la retranscription des niveaux du jeu original.
- Maxime a eu pour objectif de se former dans le but de créer un système de sauvegarde.
- Bastien s'est vu confier la gestion de la résolution des niveaux, et donc l'implémentation des cases de départ et d'arrivée, ainsi que des objectifs récupérables.

- Laurent a eu pour responsabilité de créer un système audio, ainsi que de gérer l'internationalisation de notre application pour qu'elle soit accessible dans plusieurs langues.
- Nawhal devait se charger de la création des textures et autres graphismes pour le jeu, ainsi que du menu de jeu.

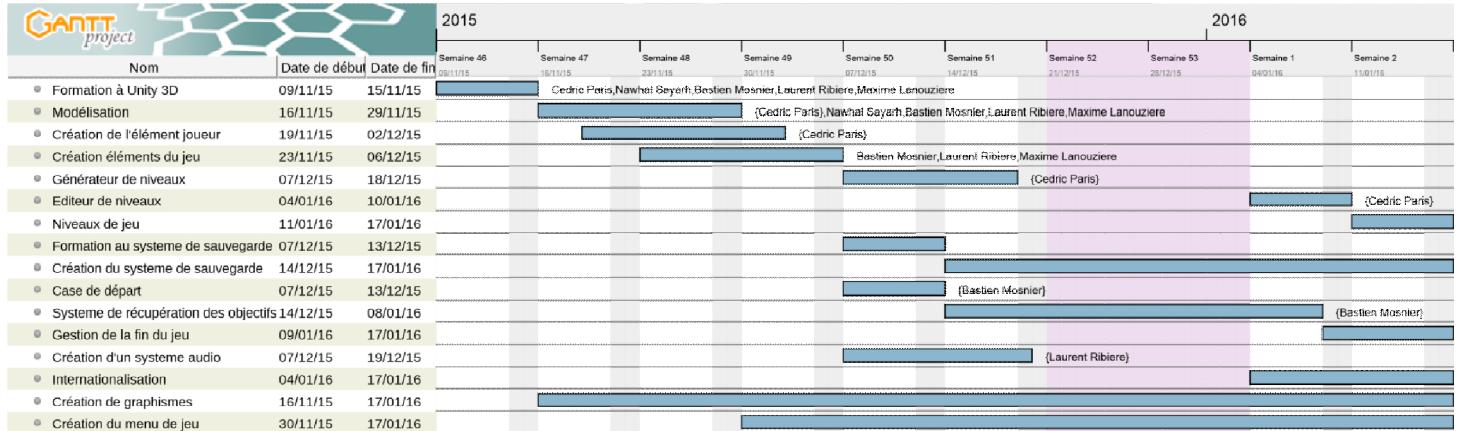


Fig. n°2 : Diagramme de Gantt prévisionnel de la première période



Fig. n°3 : Diagramme prévisionnel des ressources de la première période

Au terme de cette période, nous avons pu constater deux écarts majeurs par rapport à notre Gantt prévisionnel:

- Le système de sauvegarde avait pris du retard, suite à des problèmes techniques ainsi qu'à des problèmes de santé de Maxime.
- Le système audio qui, bien que presque fini, était inachevé à cause d'un bug que nous n'avions pas eu le temps de corriger.

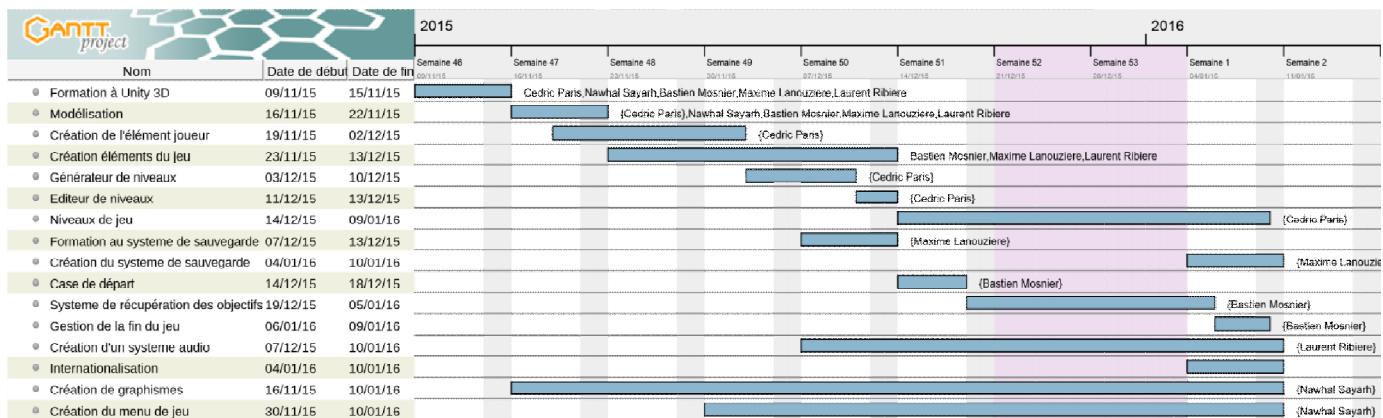


Fig. n°4 : Diagramme de Gantt réel de la première période

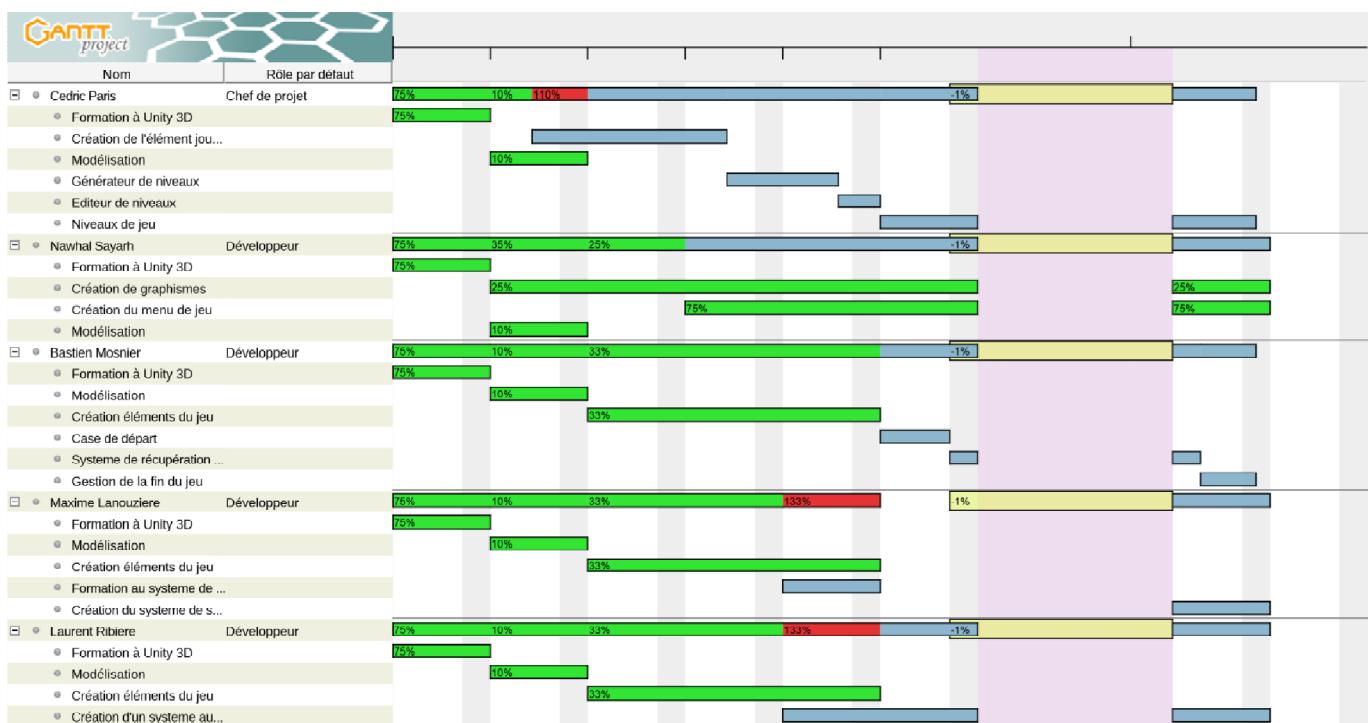


Fig. n°5 : Diagramme réel des ressources de la première période

Sur la seconde période, nous avons procédé de la même façon. Les tâches qui avaient pris du retard ont dû être terminées. De nouvelles tâches ont été assignées, comprenant la création d'éléments de jeu supplémentaires, la conception d'un générateur de niveau, la mise au point d'un mode arcade se basant sur ce générateur et l'ajout du module de publicité *Admob*.

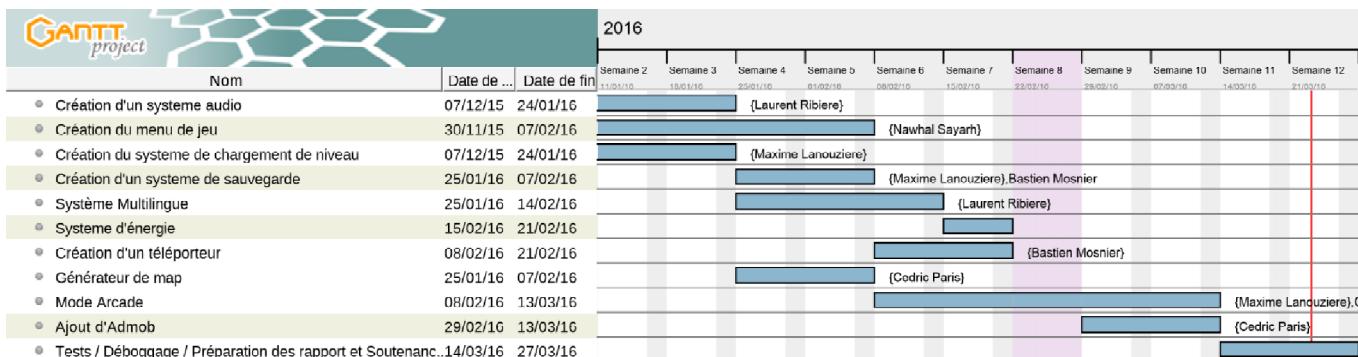


Fig. n°6 : Diagramme de Gantt prévisionnel de la seconde période

Au final, ces tâches ont toutes pu être réalisées. Cependant, du retard a été pris sur le menu de jeu, qui a été achevé plus tard que prévu. La première implémentation du menu fonctionnait, mais elle était basée sur une mauvaise conception. Nous avons donc décidé de recommencer ce menu afin que son implémentation soit plus propre au niveau conceptuel.

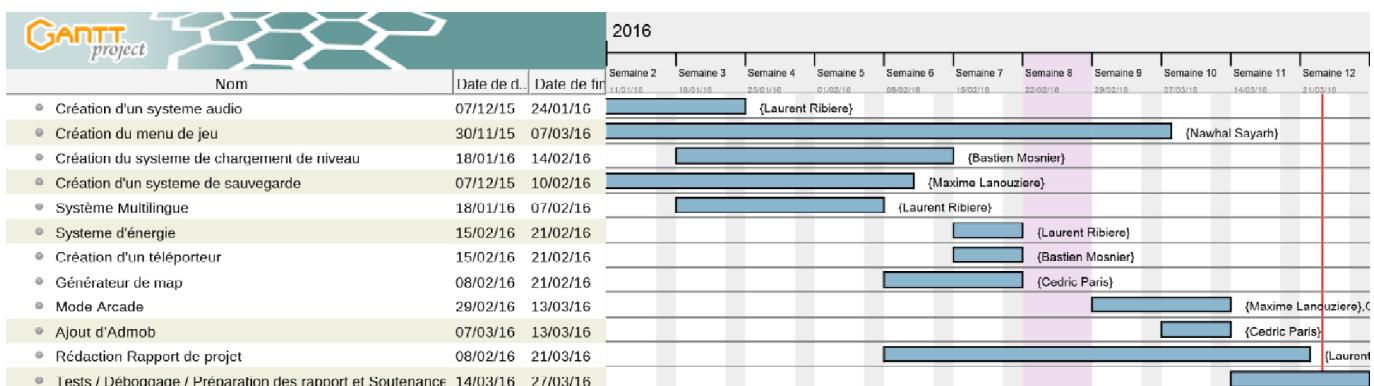


Fig. n°7 : Diagramme de Gantt réel de la seconde période

La totalité des diagrammes présentés dans cette partie sont disponible en version agrandie en annexe.

## *II. Analyse et conception*

### *II.1. Analyse du jeu à développer*

#### *II.1.1. Présentation du moteur de jeu Unity3D*

Unity3D est un moteur de jeu multiplateforme (iOS, Android, Web, Windows, Mac, PlayStation4, Xbox 360...) développé par Unity Technologies. Unity 3D est idéal pour ce projet puisqu'il propose une licence gratuite sans limitations au niveau du moteur depuis le 2 Août 2013. Il est très répandu dans l'industrie du jeu vidéo, que ce soit pour les gros studios ou pour les indépendants.

Un moteur de jeu est un ensemble de composants logiciels qui s'occupe notamment de produire les caractéristiques (gravité, collisions entre objets, calcul de position...) d'un monde virtuel dans lequel se déroule le jeu. Il a également à sa charge la gestion des entrées / sorties (clic de la souris, haut-parleur...) ou encore le rendu graphique.

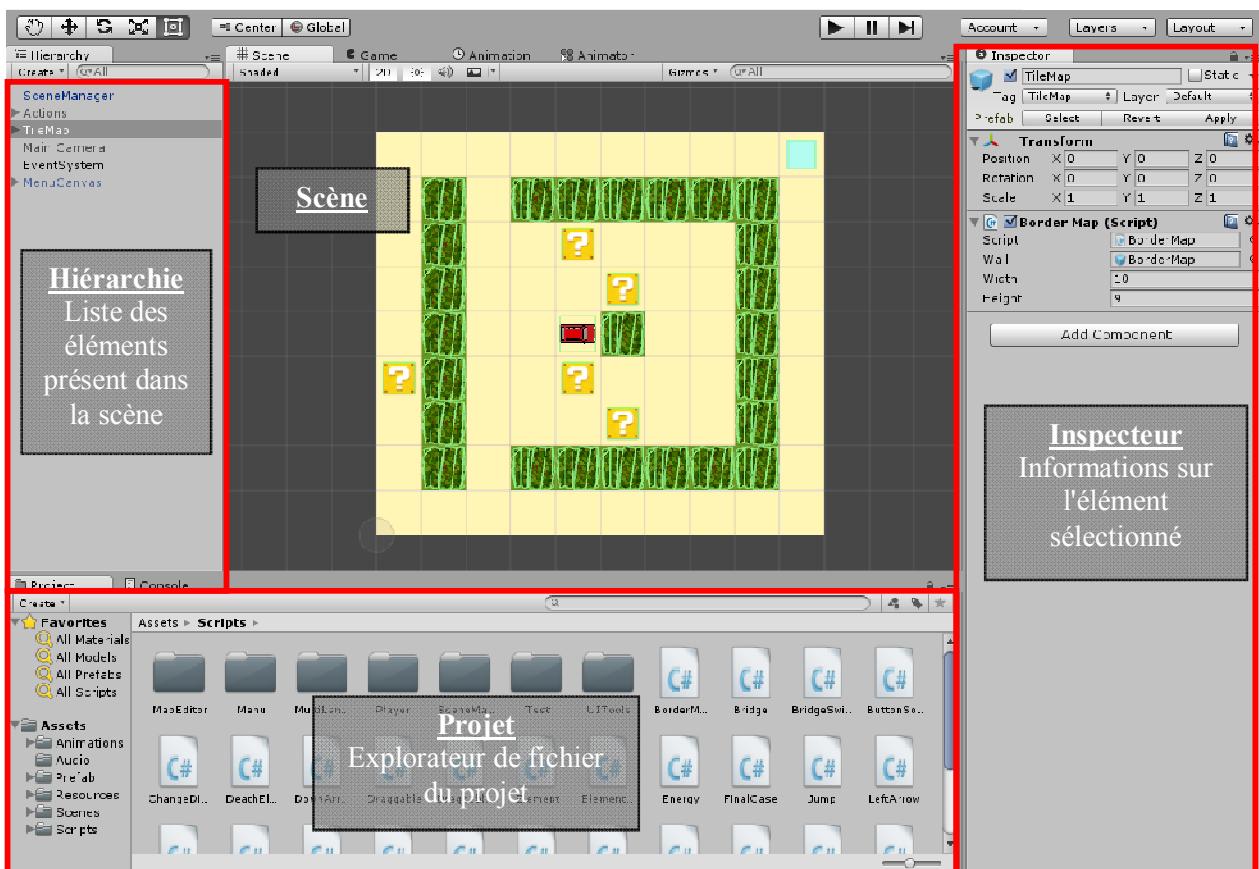


Fig. n°8 : Interface du logiciel Unity

## *II.1.2. Présentation des autres outils utilisés*

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

### **Git**

Git est un gestionnaire de version, c'est à dire un logiciel qui permet de conserver et de partager différentes versions du code-source d'un logiciel. Nous avons choisi d'utiliser Git car il est très répandu dans le milieu professionnel et car c'est un logiciel libre qui fonctionne de manière décentralisée.

L'intérêt d'utiliser un système de gestion de version est avant tout de pouvoir sécuriser le travail qui a déjà été produit. En effet, la dernière version du projet et l'historique des versions précédentes est conservé en local sur l'ordinateur de chaque membre de l'équipe. Ainsi en cas de problème sur un ordinateur ou si une grosse modification du code pose problème, il est possible de revenir à une version antérieure fonctionnelle à tout moment.

Un autre avantage d'un gestionnaire de version est de permettre de ne pas être dépendant d'une seule machine. Chaque membre de l'équipe peut travailler sans accès internet sur un tâche séparée, en ayant son propre historique local. C'est uniquement une fois qu'il considère qu'une partie de son travail est aboutie qu'il le partage avec les autres.



### **GitHub**

GitHub est un service web d'hébergement, qui utilise le système de gestion de version Git. Il permet de déposer gratuitement sur un serveur le code source d'un projet le rendant accessible à n'importe quel moment. Plus précisément, on dépose sur la plateforme GitHub l'équivalent d'un dépôt Git local, c'est à dire à la fois le projet actuel mais également son historique.

GitHub propose des comptes gratuits, mais limités à des projets publics, et des comptes payants pour avoir des projets privés. Il offre aussi des fonctionnalités qu'on retrouve sur des réseaux sociaux, comme suivre un projet ou le travail d'une personne en particulier.

Nous avons utilisé le logiciel GitHub Desktop qui propose les mêmes fonctionnalités que Git mais avec une interface graphique permettant d'être plus efficace et de suivre plus facilement les modifications sur le dépôt.

## **GraphicsGale**

GraphicsGale est un logiciel très utilisé pour le PixelArt, forme de graphisme que nous avons choisie 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.

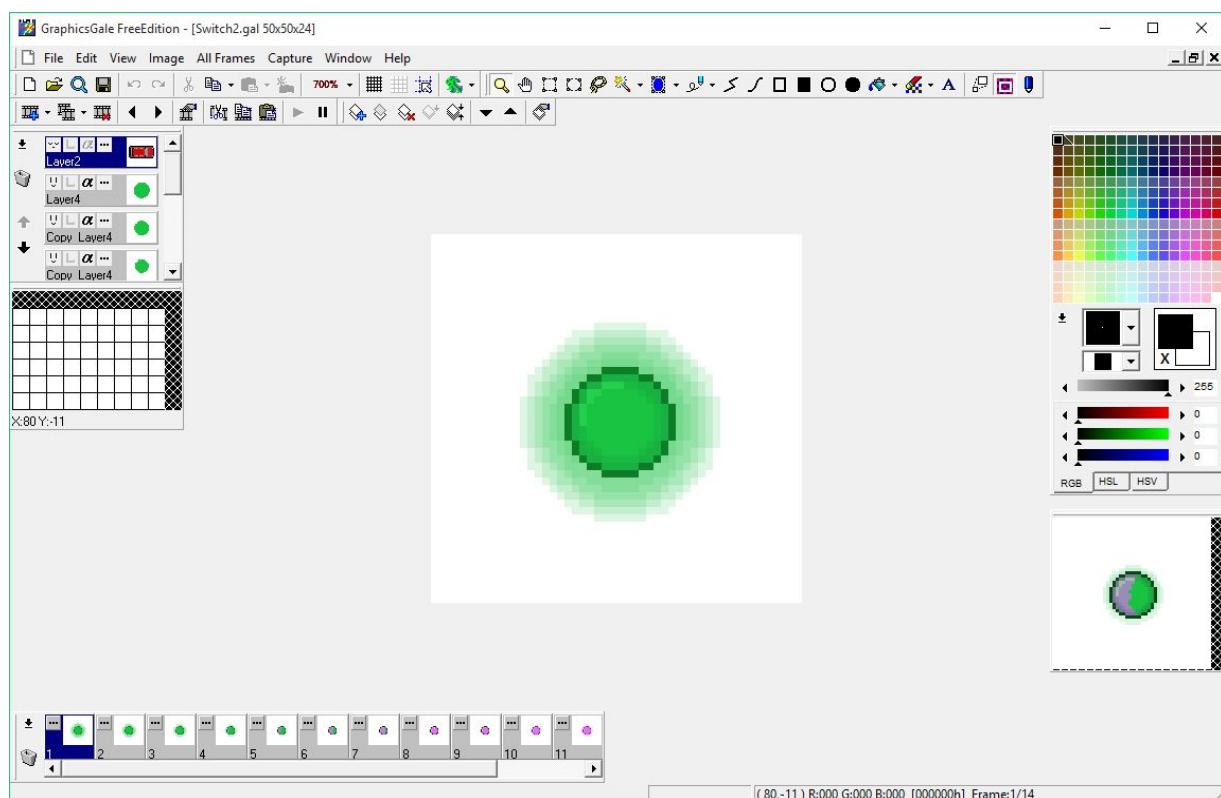


Fig. n°9 : Interface du logiciel GraphicsGale

### *II.1.3. Diagramme de cas d'utilisation*

Afin de déterminer les différentes parties qu'il fallait développer pour obtenir un jeu intéressant et fonctionnel, nous avons établi la liste des actions auxquelles l'utilisateur aura accès et les fonctionnalités qu'il devait pouvoir utiliser.

Ces fonctionnalités et actions sont résumées de manière simplifiée dans le diagramme de cas d'utilisation suivant :

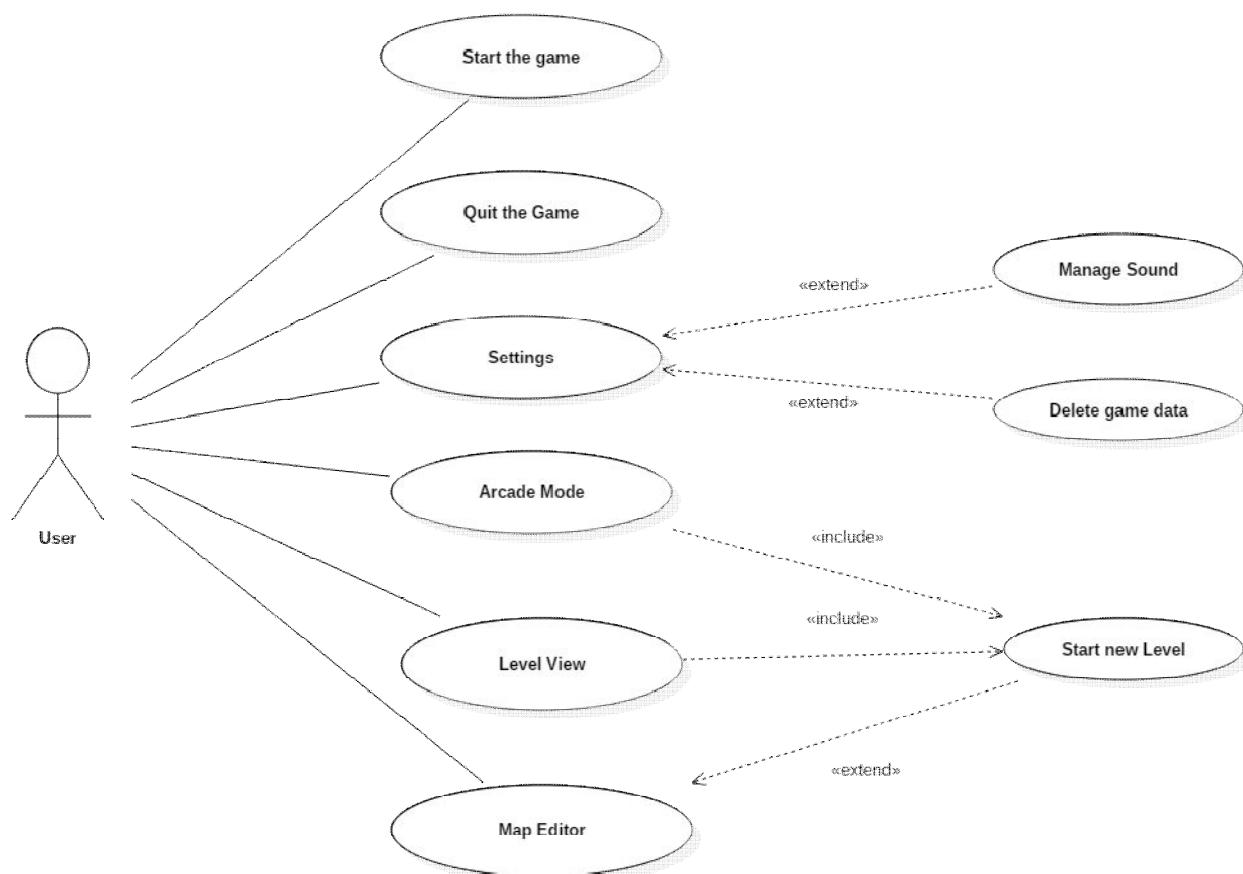


Fig. n°10 : Diagramme de cas d'utilisation de l'application

## *II.1.4. Diagramme de classe de l'application*

En début de projet, nous avons modélisé les éléments principaux de l'application pour déterminer les liens entre les différentes parties à réaliser mais également pour que chaque membre de l'équipe puisse travailler une partie en étant conscient des liens qu'il existe avec celles des autres. Le diagramme de classe de l'application est disponible en annexe.

Unity est un moteur de jeu qui fonctionne sous forme de scripts, c'est à dire que l'on peut appliquer un comportement à un objet de la *scène* par l'intermédiaire d'une classe (qui hérite de MonoBehaviour) sans qu'elle n'ait de lien avec le reste de l'application. Ainsi le clic d'un bouton peut être géré par une classe SceneLoader qui écoute les clics et qui charge une scène sans que cette classe ne soit jamais référencée ou instanciée ailleurs.

Pour cette raison, notre diagramme de classe est réduit et ne contient pas la totalité des classes qui figure dans notre application. Si c'était le cas, une grande partie des classes serait isolées, sans aucune interaction avec les autres classes du diagramme. Nous n'avons donc représenté dans ce diagramme que la partie centrale de l'application (Joueur et interactions avec les cases et les éléments de case) qui ne s'apparente pas à de la programmation sous forme de script.

## ***II.2. Conception de l'application***

### *II.2.1. Génération d'un niveau*

Notre application inclut 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 à générer les bases d'un niveau que l'on souhaite ajouter dans la liste des niveaux "pré-générés" proposés à l'utilisateur.



Fig. n°11 : Exemple de niveau généré 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 établi soit la ou l'une des seules solutions à l'énigme. Pour obliger l'utilisateur à 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.

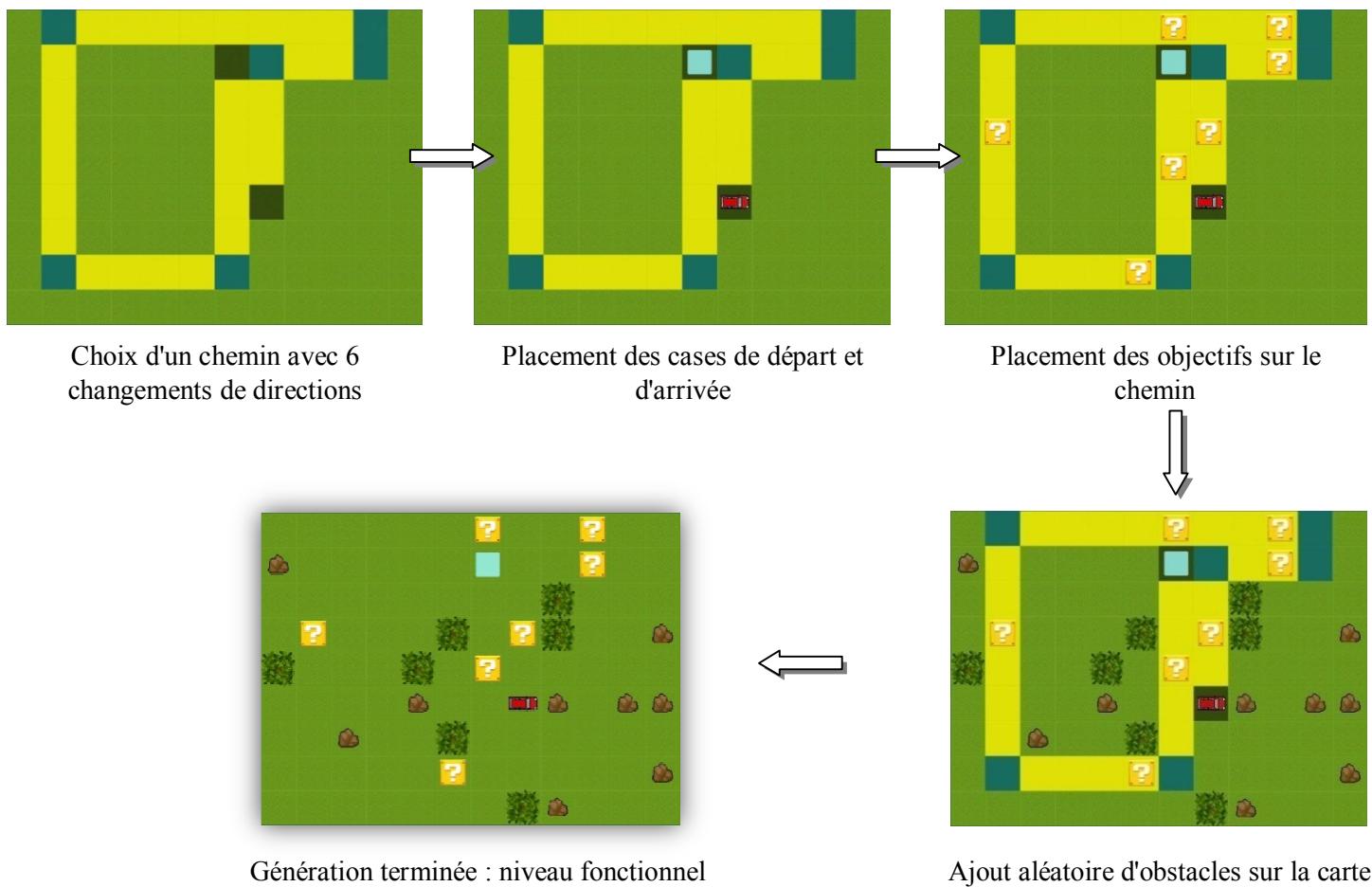


Fig. n°12 : Phases de génération d'un niveau

## II.2.2. 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 évidemment 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



Fig. n°13 : Interface d'édition de niveau

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 accessibles 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 expliquée dans les parties suivantes.

### **Bouton Camera**

Le joueur peut éditer une carte qui s'étend au-delà de la taille de l'écran. La caméra est fixée par défaut pour éviter les mouvements de caméra non souhaités lorsque l'utilisateur touche l'écran. En cliquant sur le bouton "Caméra", 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 été déplacée à la position voulue, il suffit de cliquer de nouveau sur le bouton pour désactiver le mode caméra.

### **Bouton Play**

L'édition d'un niveau a évidemment 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 personnalisée dans une scène vide contenant uniquement un menu avec les actions définies dans l'éditeur de niveau.

### **Bouton Suppression**

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

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

La carte que l'utilisateur édite est placée dans un *GameObject* appelé TileMapEditor auquel sont affectés 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. Ils 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.

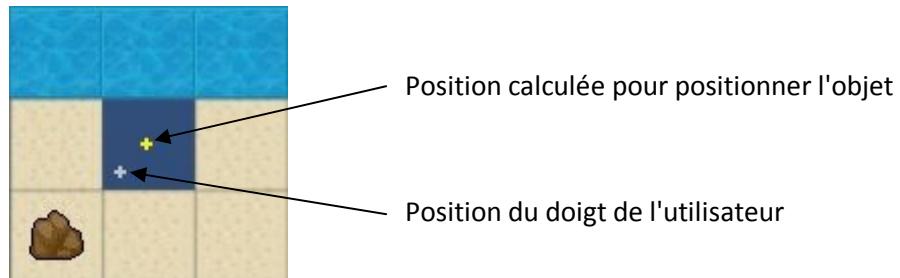


Fig. n°14 : Éditeur de niveau : position écran / position calculée

## *II.2.3. Menus*

### 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.

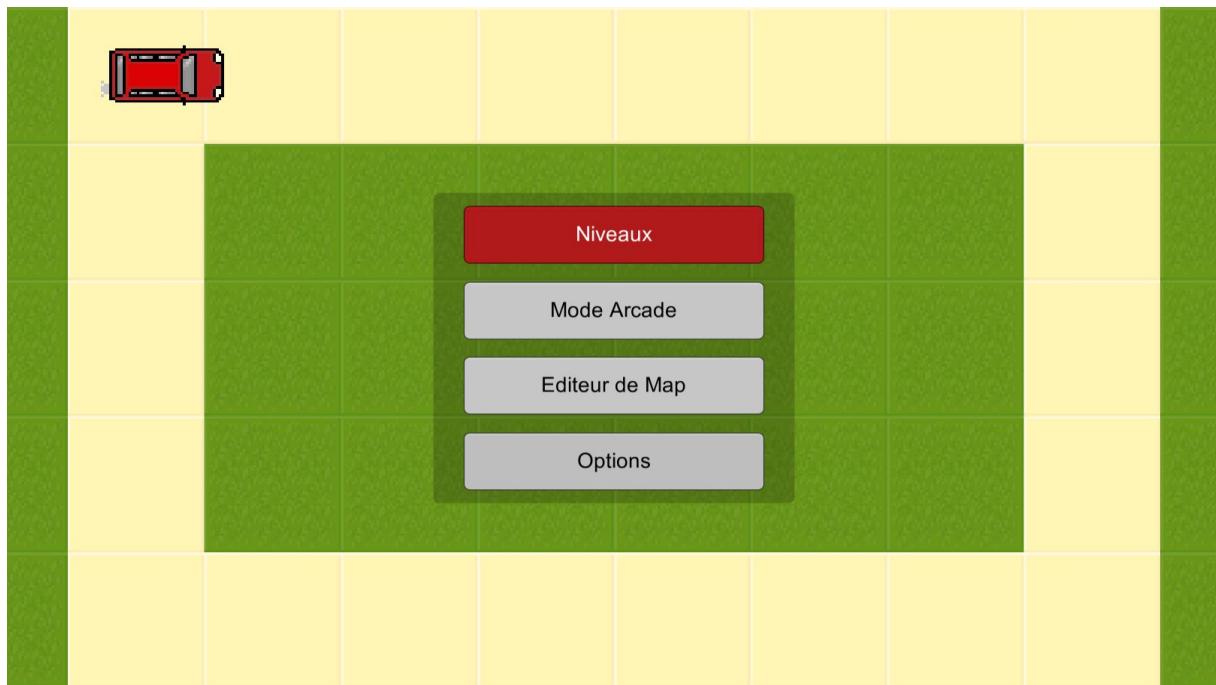


Fig. n°15 : Menu principal

Différents boutons s'offrent alors à nous :

- Le premier bouton mène à 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 niveau, 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.
- Le quatrième bouton mène au menu d'options.

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

## 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.

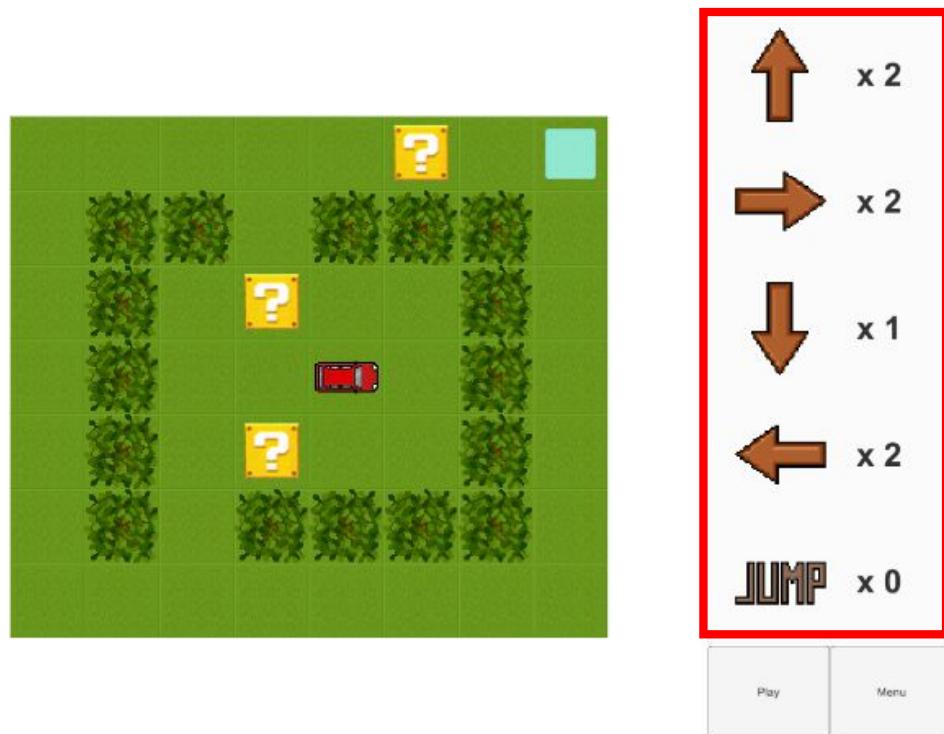


Fig. n°16 : Le menu de jeu dans un niveau

Il est facile à prendre en main : si l'on veut poser un élément dans le jeu, il suffit de faire glisser 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 du jeu, il suffit de le glisser vers le bouton du menu qui lui correspond et il disparaît.

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 zoomé 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 événements 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 événements entre le menu et les éléments de mouvement mais entre le doigt et les éléments puis entre le doigt et le menu. Par exemple, si l'on voulait glisser un élément du jeu au menu, nous faisions comme suit :

```
L'élément se fait déplacer  
L'élément dit au menu qu'elle se fait déplacer  
Le menu sauvegarde la référence de l'élément  
Lorsqu'on passe son doigt sur le menu :  
    Le menu regarde s'il a une référence sauvegardée  
    Si oui :  
        Le menu dit à l'élément de s'autodétruire  
        Le menu augmente le nombre d'éléments restant  
    Fin si  
Fin lorsque
```

Le problème de cette implémentation était que n'importe quelle classe pouvait faire croire qu'un élément se faisait glisser et ceci pouvait résulter en des destructions d'éléments sans raisons apparentes.

Nous avons alors pris une toute autre approche. À présent, le menu en lui-même est purement visuel car ce sont 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. Par exemple, si l'on veut glisser un élément du jeu au menu, nous faisons comme suit :

L'élément a une collision avec le collider du MenuButton.

Si l'élément est celui associé au MenuButton

    Le MenuButton dit à l'élément de s'autodétruire

    Le MenuButton augmente le nombre d'éléments restant

Fin si

Pour ce qui est de faire glisser un élément du menu de jeu à la carte, nous bravons la difficulté d'une autre manière. Pour chaque élément, il existe un élément UI déplaçable. Cet élément est placé sur chaque MenuButton s'il reste au moins 1 élément à poser. Ainsi, le joueur doit juste déplacer cet élément UI là où il veut placer son élément, et l'élément UI sera remplacé par un élément de jeu.

## II.2.4. Entité joueur

On appelle "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.

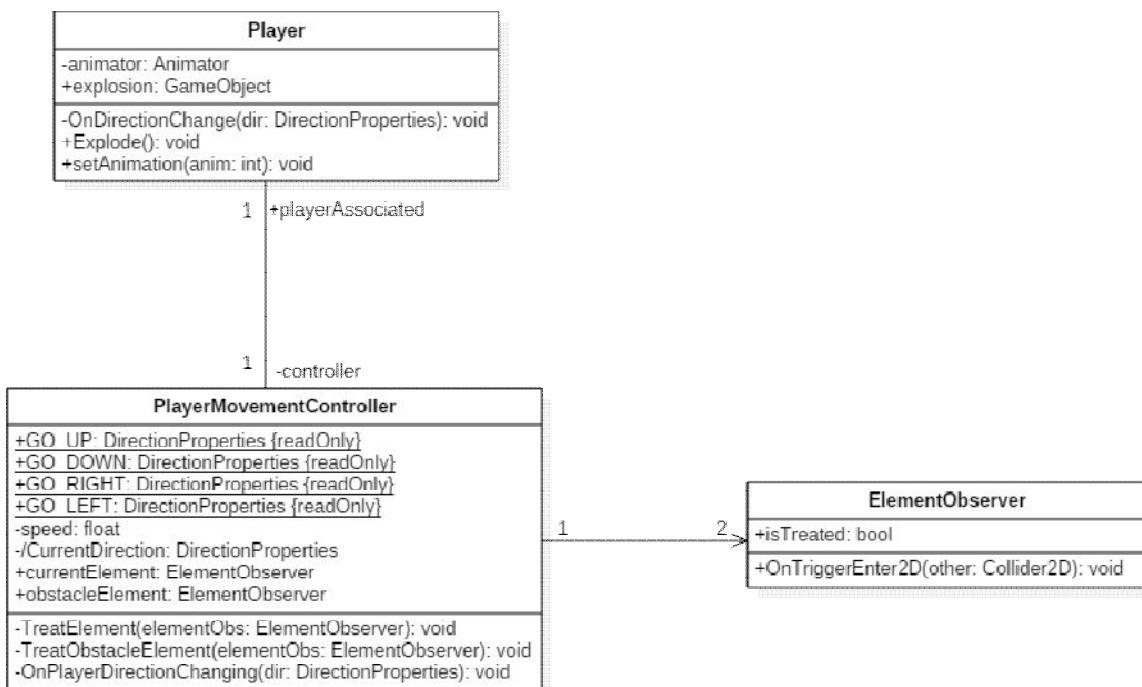


Fig. n°17 : Diagramme de classe du Joueur

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 s'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 :

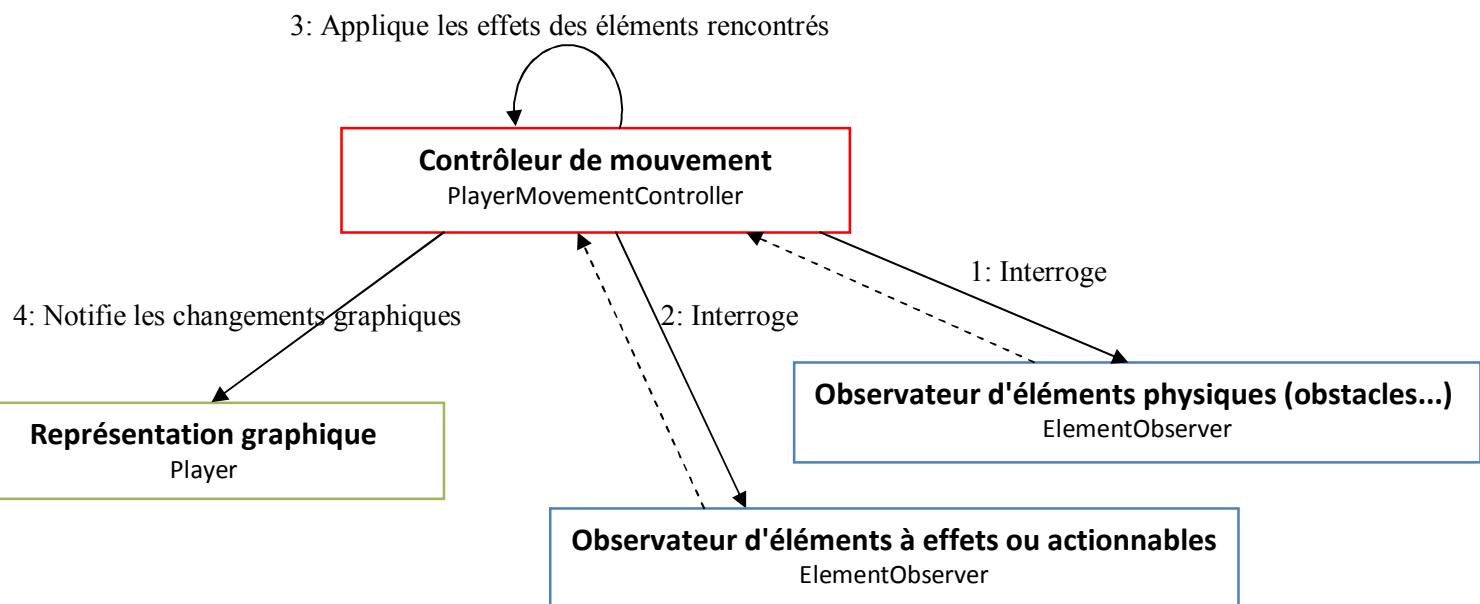


Fig. n°18 : Schéma du fonctionnement de l'entité Joueur

## II.2.5. Éléments du jeu

On appelle "élément" tout ce qui est posé sur la carte de jeu et qui interagit avec le Joueur. Ces éléments de jeu peuvent influer sur son comportement (changement de direction...), être ramasser (Objectifs) ou s'actionner(leviers de ponts...). Les éléments de jeu sont divisibles en 4 parties distinctes selon l'effet qu'ils appliquent sur le Joueur.

- **Les "Death Elements" ou éléments entraînant la mort**

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

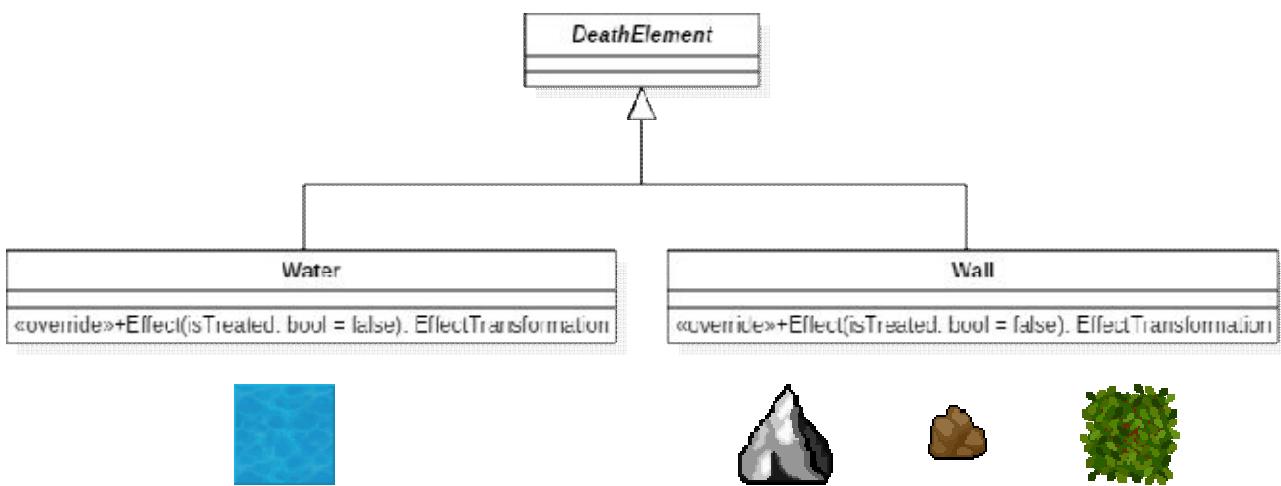


Fig. n°19 : Hiérarchie des "Death Elements"

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

Les "Change Direction Elements" sont les éléments qui ont un effet sur la direction du joueur. Quand le joueur les rencontre, ils lui imposent une des quatre directions possibles pour le joueur (Haut, Bas, Droite, Gauche).

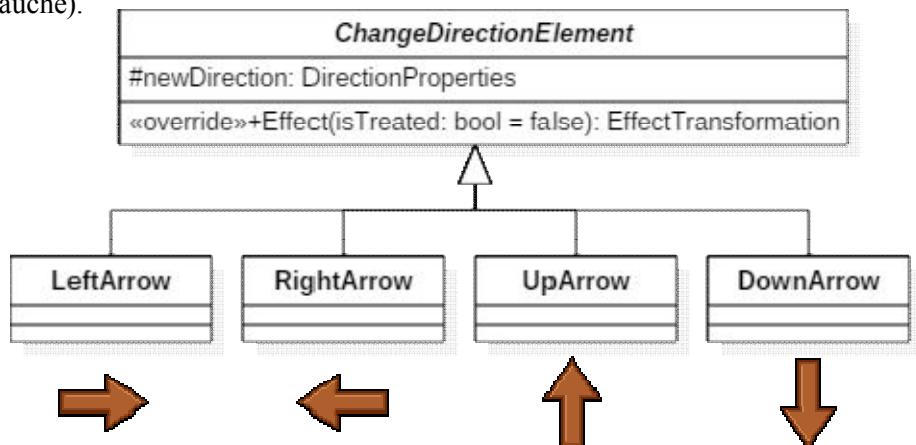


Fig. n°20 : Hiérarchie des "Change Direction Elements"

- **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 notamment le cas des objectifs que le joueur doit collecter pour terminer le niveau.

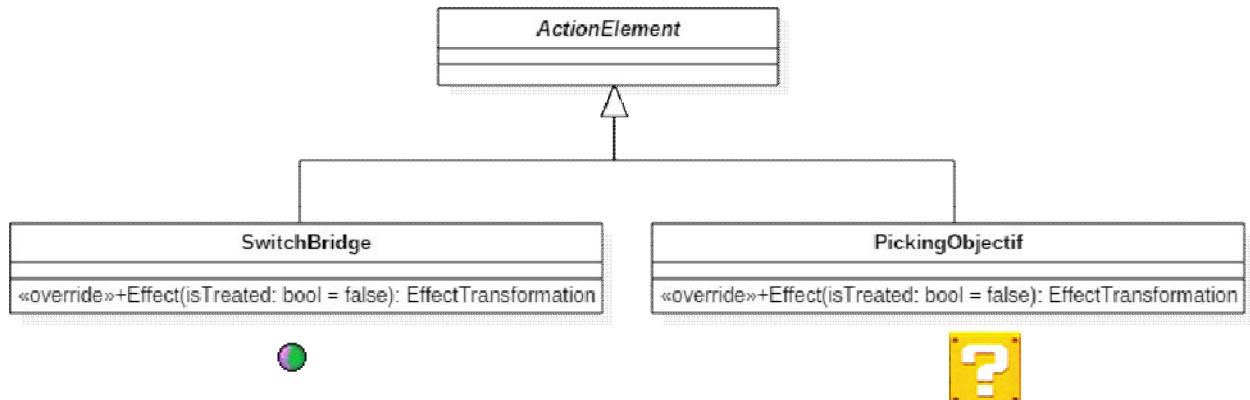


Fig. n°21 : Hiérarchie des "Action Elements"

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

Les "Special Elements" sont les éléments qui ne peuvent pas être classés dans une des catégories précédentes car leur effet est très particulier. Par exemple, l'effet de l'élément Pont est variant suivant s'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'entraîner la mort du joueur.

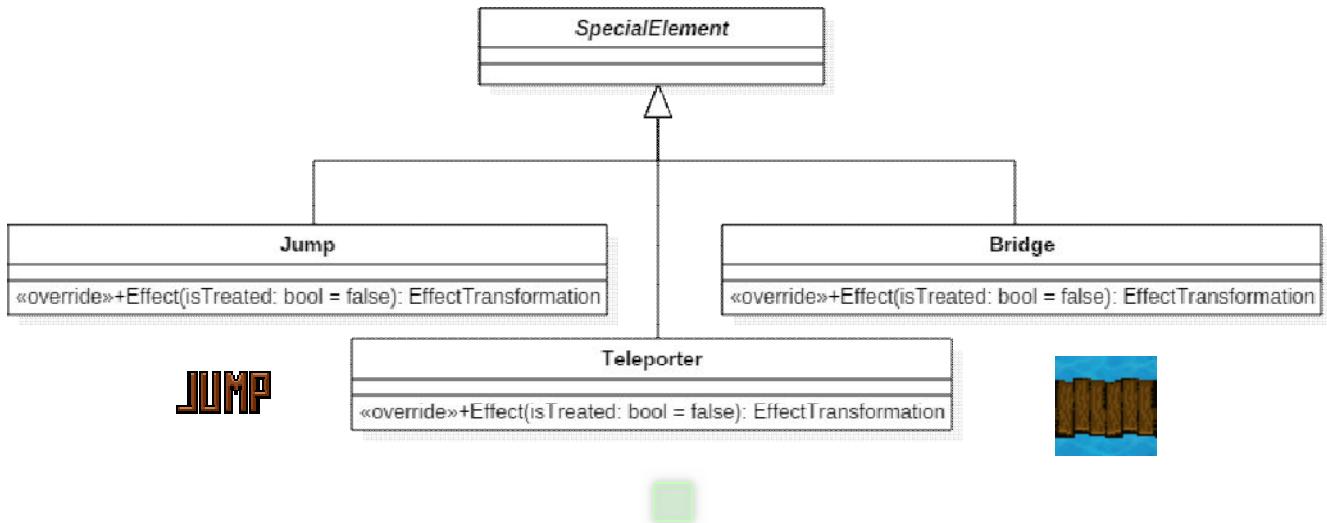


Fig. n°22 : Hiérarchie des "Special Elements"

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

## II.3. Modules et fonctionnalités

### II.3.1. La gestion du son

Nous avons choisi d'ajouter des effets sonores à notre jeu car nous considérons que c'est un élément important pour apprécier et s'immerger dans un jeu.

Pour que les sons dans une application ne soient pas désagréables, il faut que la musique se poursuive même si l'affichage change, c'est pour cette raison que la gestion de ces sons a été déléguée à un *GameObject* particulier qui n'est pas détruit lors du chargement d'une nouvelle scène, contrairement aux *GameObjects* "normaux". Ainsi, il n'y a pas d'interruption dans la musique.

Ce comportement particulier est autorisé par Unity mais il faut enregistrer le *GameObject* comme un objet à ne pas détruire en cas de changement de scène. Ce *GameObject*, appelé SoundController, est donc chargé dans la toute première scène de l'application et ne sera jamais détruit.

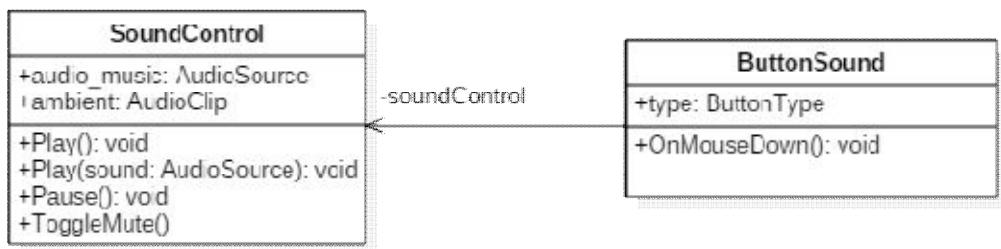


Fig. n°23 : Diagramme du Contrôleur de son

Le *GameObject* SoundController est géré par une classe nommée SoundControl. Elle connaît un  *AudioSource* qui joue les sons et un  *AudioClip* qui représente le son joué – le fichier audio d'une certaine façon. Le rôle de cette classe est donc de recevoir les requêtes liées au son et d'effectuer les actions demandées, comme par exemple changer la musique jouée par l' *AudioSource*.

Sur la figure ci-dessus nous avons représenté des boutons qui agissent sur le son pour donner un exemple précis d'appel au SoundControl. La classe BoutonSound connaît le SoundController de la scène et peut donc interagir avec lui. Quand on clique sur le bouton, la méthode `OnMouseDown()` est appelée. Cette méthode va simplement appeler `Play()`, `Pause()` ou `ToggleMute()` de la classe SoundControl suivant le type du bouton. Cette classe se chargera ensuite d'agir en conséquence sur le son.

## II.3.2. 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 aient une application en français et que les autres appareils aient une application en anglais. Ces traductions sont basées 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 car il suffit de créer un fichier Json pour cette langue et l'application pourra être traduite – par exemple string\_es.json pour une traduction espagnole.

### Mécanisme de traduction

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

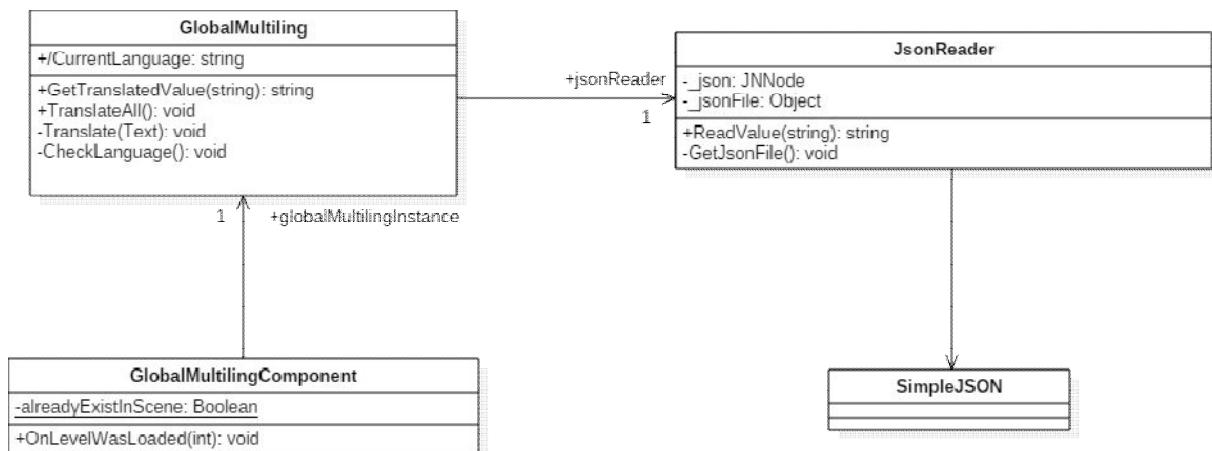


Fig. n°24 : Classes impliquées dans le mécanisme de traduction

En réalité, l'entité SimpleJson n'est pas une classe mais le nom d'un fichier mis à 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 faciliter l'explication du mécanisme de traduction nous le considérerons comme une seule classe. Les classes GlobalMultiling et GlobalMultilingComponent sont les classes qui s'occupent 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 associée à 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 associée à un GameObject lui permet de détecter le chargement d'une nouvelle scène. Quand elle détecte qu'une scène est chargée elle recherche dans cette nouvelle scène tous les textes et demande leur traduction au JsonReader. Ainsi toutes les scènes de l'application sont traduites à l'utilisateur.

Pour les éléments qui ne font pas partie directement des scènes (messages affichés à 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.

### *II.3.3. 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 à 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 petites données entre plusieurs sessions de jeu. Pour chaque donnée que l'on enregistre on associe une clé. On peut ainsi récupérer les données à partir de leurs clés (par exemple le score obtenu au niveau 2 avec la clé "ScoreLevel2").

Au niveau graphique la progression est représentée par des étoiles pour chaque niveau, trois étoiles indiquant un bon score et une indiquant un score faible. Les niveaux non débloqués apparaissent grisés et ne sont pas accessibles. Pour connaître les étoiles à afficher et les niveaux non débloqués, 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 rendre la carte et les actions *serializables*, on les encapsule simplement dans deux classes serializables, TileMapSave et Vector3Save.

- La classe Vector3Save est une classe serializable permettant de stocker les coordonnées d'un Vector3. Un Vector3 est un vecteur dont les coordonnées sont du type (x, y, z). Les Vector3 représente donc la position d'un élément dans la scène en trois dimensions.
- La classe TileMapSave est une classe serializable qui contient la liste des éléments d'une scène (plus précisément de la carte à sauvegarder) avec leurs positions 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 niveau, LevelSave parcourt chaque élément de la TileMap (carte), et les ajoute à une TileMapSave avec leur position. Le dictionnaire d'éléments utilisables, passé en argument à la méthode, 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 et leur position.

On va recréer chaque élément à partir de son *prefab* et le placer à la bonne position dans la TileMap. Enfin on renvoie le dictionnaire contenant la liste des actions nécessaires à l'utilisateur pour finir le niveau que l'on vient de charger.

### ***III. 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 utilisé 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 renforcé 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 de jeu. Il fallait un menu détectant parfaitement le déplacement d'un objet de la carte au menu. De même, il était nécessaire d'avoir un menu ergonomique et 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 demandé d'utiliser le moteur Unity3D. Cela nous a facilité 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 nombreuses 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 d'apprendre à utiliser le 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 les problèmes rencontrés par les membres de l'équipe. Cela a également été notre première expérience dans le développement d'un jeu vidéo, où il est nécessaire de se mettre réellement à la place du joueur, pour comprendre ses attentes. Ce projet nous a également permis d'approfondir certaines méthodes de travail avec la plateforme de collaboration GitHub, 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 Informatique de Clermont-Ferrand.

# ***Abstract***

---

We are a group of students pursuing a two-year university diploma in Computer Science in the University Institute of Technology in Aubière. As part of the university curriculum, we have the rare opportunity to work on the development of a game. This puzzle game project enabled us to discover the Unity 3D Game Engine. The objective of this project was to create a two-dimensional game. This game is intended to be played via smartphone, whatever operating system it may have.

The topic of our project was given by Mr Pierre-Antoine PAPON, a research and teaching assistant. He realised that there is a lack of puzzle games in the smartphone games' market, which is how he came up with this project.

The game works as follows: the player has to take a toy car from a point A to a point B. The player can move the toy car by placing movement actions represented by four arrows and a jump item. He only has a certain number of each movement action to finish the level. If the toy car hits an obstacle, it explodes and the player loses. The toy car has to collect all items labelled as objectives for the player to win.

In addition to the main game, we also created several game features. The first one is the Level Generator. Its algorithm can generate a whole map, including the number of movement actions available. This feature is used by the Arcade Mode which consists in a never-ending series of levels. In this mode, the goal is to get the path for the car right the first time on every level. Whenever the player loses in the level, he loses the game and his score is equal to the number of levels he cleared.

Another game feature we added is the Level Editor. This feature allows us and the player to create levels directly through the game and to test them while creating them. We can also save and load these levels for later use.

After 5 months of hard work, we succeeded in developing a functional puzzle game for smartphones. Even if we did not implement all the features we wanted, like the Play Store Services, we created the game that was asked for.

Overall, this project was a good learning experience. Firstly, we discovered the Unity 3D Game Engine and how it works. This game engine, being widely used amongst freelance developers, is a great asset to our technical skills. We also learned how to use GitHub, a version control system, which is also widely used amongst developers. We improved our personal skills such as the capacity to work both autonomously and in a team as well as organizational skills. We gained experience in our programming skills in C# programming. Had we had more time, we would have improved our game by creating and/or implementing other game features such as teleporters, a fuel bar or a multiplayer mode.

# *Lexique*

---

**Terminal** : Désigne à l'origine l'extrémité d'un réseau informatique. 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 actives, au sein de l'IUT.

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

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

**Préfab** : Un préfab correspond à la sauvegarde de l'état d'un GameObject avec tous ses fils et composants. On peut ainsi conserver le Prefab d'un GameObject pour l'utiliser à un autre moment sans avoir à reconstruire entièrement le GameObject avec toutes ses propriétés.

**GameObject** : Tous les objets présents dans une scène Unity sont appelés GameObject (personnages, images, boutons...). Il est possible d'ajouter des scripts à ces GameObject pour modifier leur comportement. Ils peuvent être également hiérarchisés pour faciliter leur utilisation : par exemple, le GameObject "bouton" pourra être un fils du GameObject "menu".

**Scène** : Une scène Unity peut s'apparenter à une scène de théâtre, elle contient des objets (GameObjects) et des actions ont lieu dans cette scène (mouvements de personnages...). Une scène Unity représente donc ce que l'utilisateur voit à l'écran quand il joue. Toujours de la même manière qu'au théâtre, on peut changer de scène (de la scène de menu à la scène du niveau 1).

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

# ***Webographie***

---

<https://unity3d.com/>

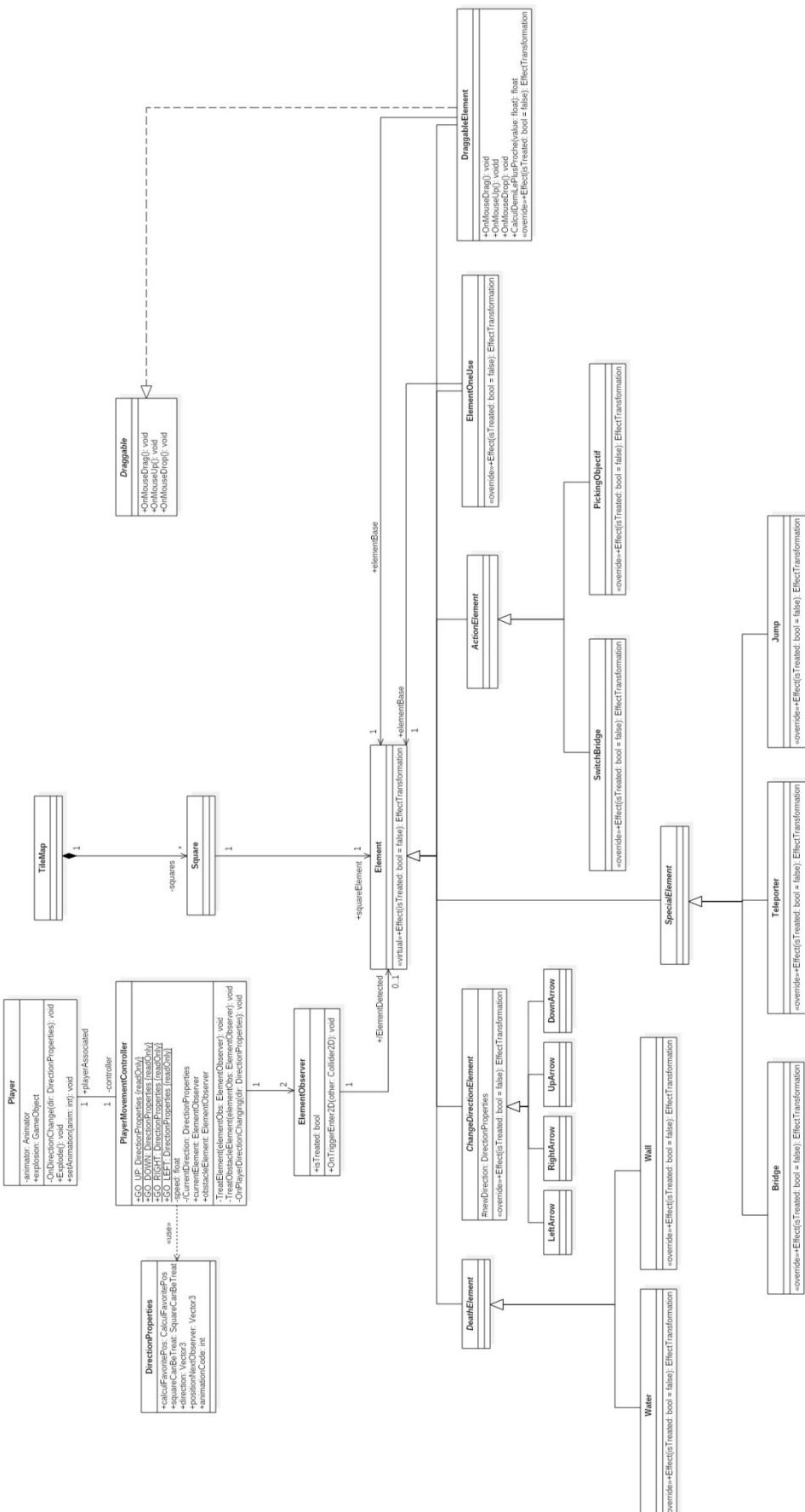
<http://answers.unity3d.com/>

<http://docs.unity3d.com/ScriptReference/>

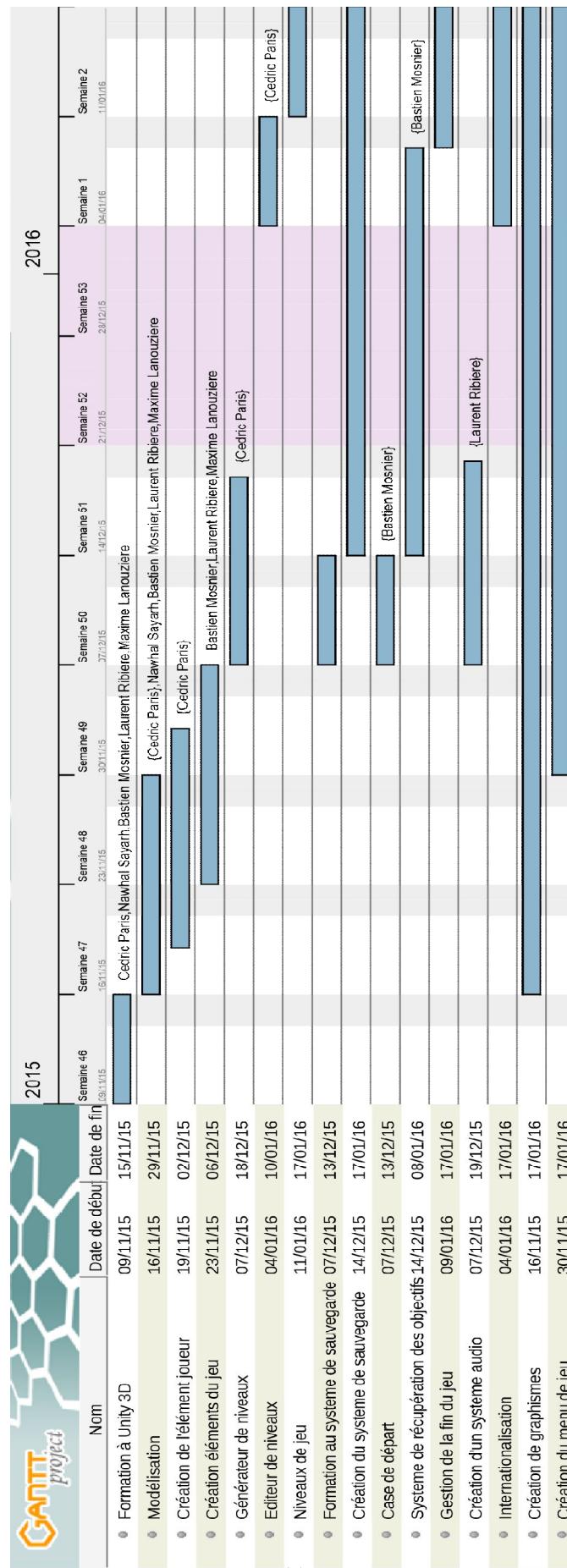
<https://www.youtube.com/>

<https://www.lynda.com/Unity-3D-training-tutorials/>

## *Annexe 1: Diagramme de classe*



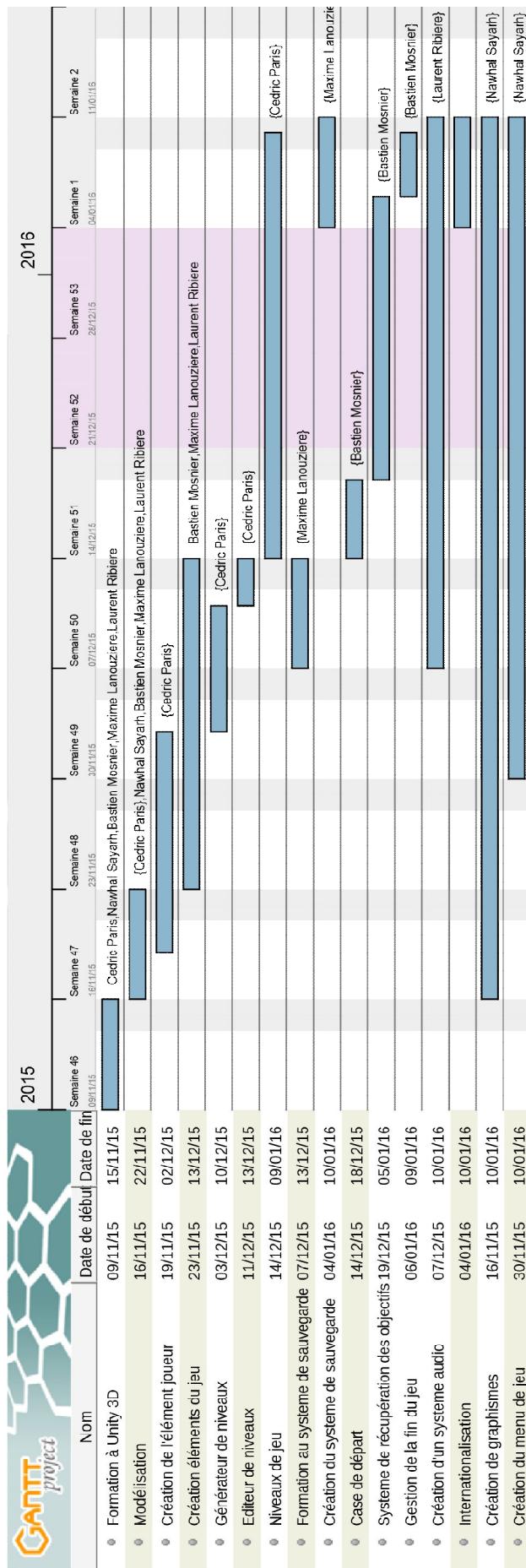
## Annexe 2 : Gantt prévisionnel de la première période



## Annexe 3 : Diagramme prévisionnel des ressources de la première période

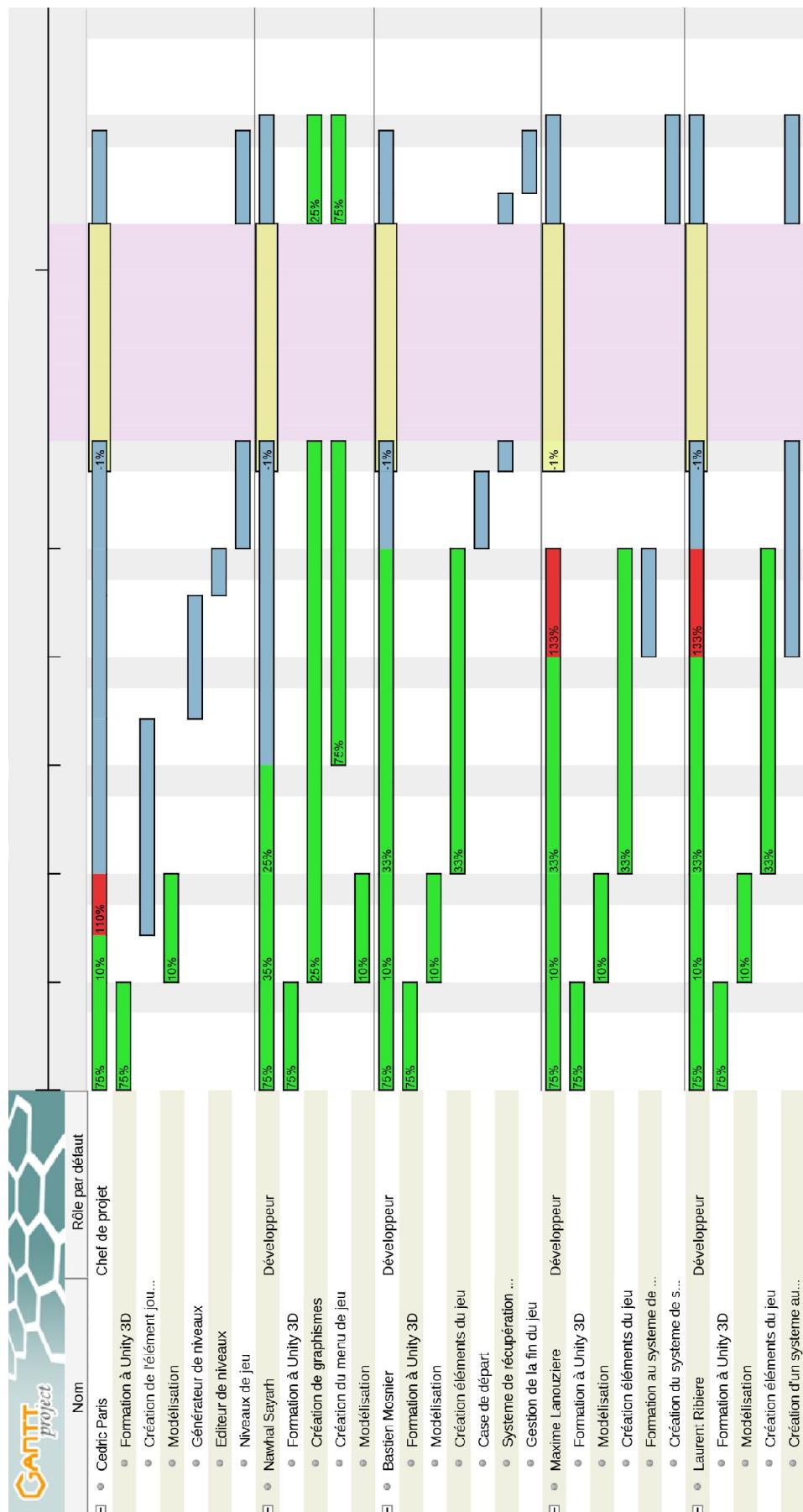


## Annexe 4 : Gantt réel de la première période



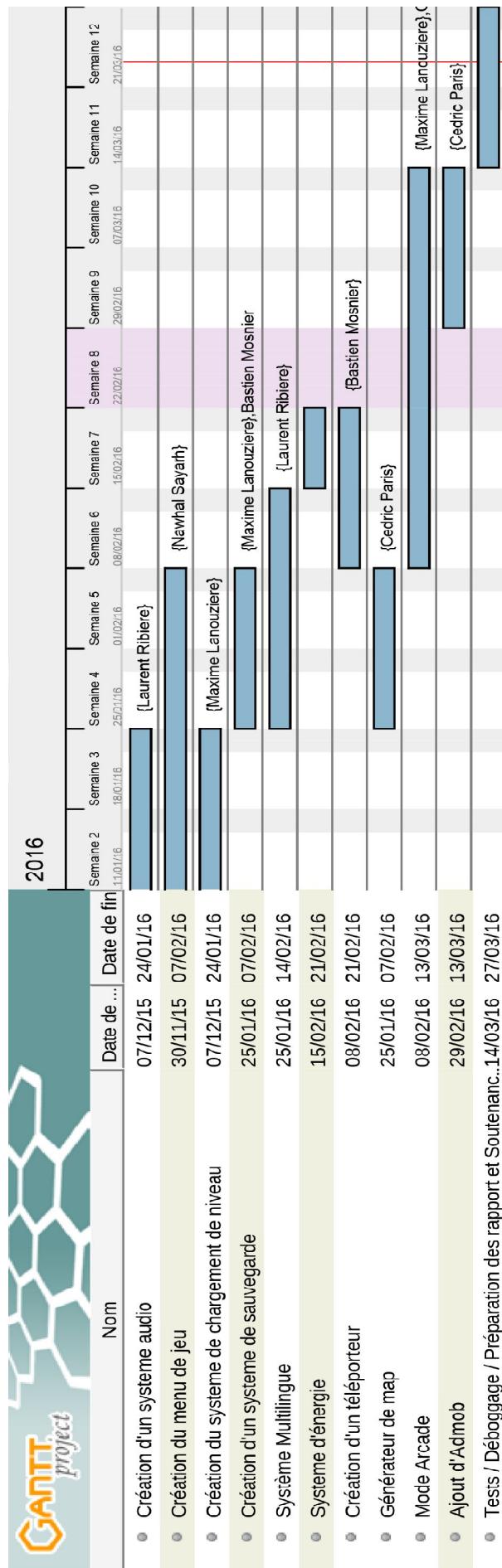
## Annexe 5 : Diagramme réel des ressources de la première période

---



## Annexe 6 : Gantt prévisionnel de la seconde période

---



## Annexe 7 : Gantt réel de la seconde période

---

