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 deuxième année à l'Institut Universitaire et Technologique (I.U.T) Informatique de Clermont-Ferrand. Dans le cadre de notre cursus universitaire nous avons l'occasion de réaliser un projet qui a pour but de nous mettre en situation de travail à long terme sur un sujet et de découvrir de nouveaux supports/moteurs de développement, dans le but d'un jeu sur mobile.

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 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 du mobile et la démocratisation de celles-ci, le but 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* 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 de ce type sur plateforme mobile. Il nous a également été proposé car plus attractif pour des étudiants et de par l'absence de jeu de ce type sur internet.

Il y a également un intérêt économique puisque ce jeu contiendra un module de publicité de type *AdMob* et il offrira 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* avec écran tactile. Ce projet devra être réalisé sur 2 *périodes*, 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, nous utiliserons des sites internets, des vidéos de formation mis à notre disposition par notre tuteur (M. Papon). Des logiciels tels que des *IDE* et des éditeurs graphiques nous seront indispensables à la réalisation du projet. Les deux outils de développement utilisés, car mis à disposition par Unity, seront *Visual Studio* et *MonoDevelop*. Ainsi le jeu pourra être exécuté par un smartphone ou une tablette et être publié sur le Google Play.

Nous avons décidé de développer le jeu en 2 dimensions avec une vue du dessus. Le contexte du jeu est une suite de niveaux sous forme d'énigmes, où une voiture devra réaliser les chemins proposés par le joueur afin de résoudre ces énigmes. Le jeu possédera à la fois un mode solo et un mode **multijoueur.**



1. **Objectifs à réaliser**

**Contraintes à respecter pour le jeu :**

* Utilisation du Moteur Unity



* Utilisation du *gestionnaire de version* GitHub



* Utilisation du langage C#

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

* L'utilisation de ce produit devra pouvoir se faire sur des plateformes mobiles.

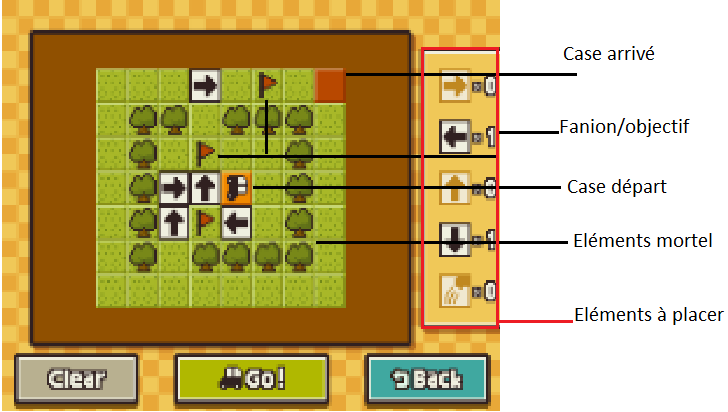
**Critères d'appréciation de la qualité du produit :**

* 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
* La possibilité d'augmenter la durée de vie du jeu (via un générateur de niveaux et un éditeur de niveaux)
* Un système de ressources et d'énergie
* Une ambiance sonore
* Un module de publicité
* Un mode Arcade
* Les différents menus adéquats

**Principes du jeu :**

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

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



La quantité d'éléments à placer est limité, une fois ceux-ci sur la terrain le joueur pourra cliquer sur « Run » pour lancer une simulation et en cas d'échec il retournera sur la fenêtre de « réflexion ». En cas de succès il accedera sur le niveau suivant après avoir vu une fenêtre de congratulations. Si le joueur rencontre un obstacle le personnage se détruira et le joueur retournera également sur la fenêtre de « réflexion ».

Afin d'augmenter la difficulté du jeu et d'ajouter des fonctionnalités, un système d'énergie (carburant) sera mis en place, les déplacements du véhicule consommeront de cette énergie, en récupérer d'autre sur la carte sera donc une nécessité.

1. **Modalités et planning prévisionnel**

**Budget:**

Aucun budget n'est nécessaire à la réalisation et au fonctionnement du projet. Le matériel pouvant être fournis à l'IUT si nécessaire afin que nous puissions directement nous concentrer sur notre travail.

**Planning:**

**Bastien**

1. **ANALYSE ET CONCEPTION**
   1. **ANALYSE DU JEU A DEVELOPPER**
      1. **Présentation du moteur unity**

**MAXIME**

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

**MAXIME/NAWHAL**

* + 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.

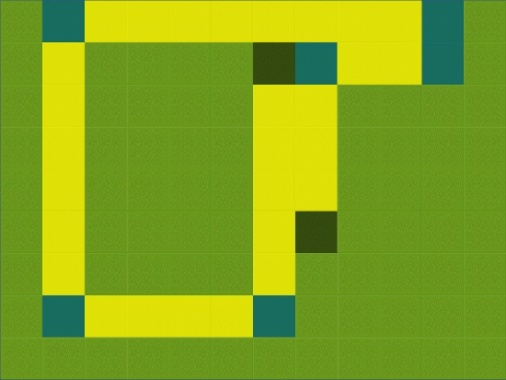
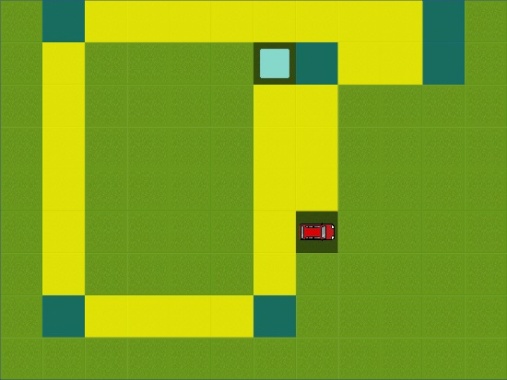


Fig. n° : Exemple de niveau générer pour 6 changements de directions

**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.



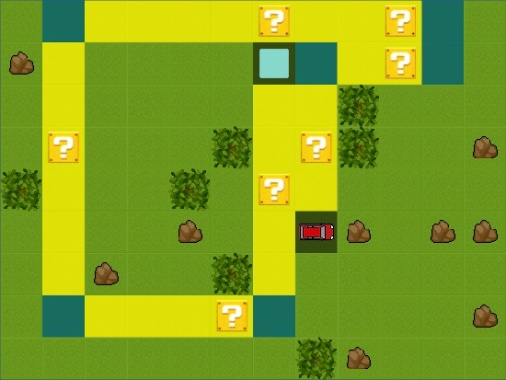


Fig. n° : Phases de génération d'un niveau

* + - 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**



Carte Personnalisée

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. **Menu de jeu**

**NAWHAL/CEDRIC/MAXIME**

**- Menu Classique (principale, Levels)**

**- Menu de Jeu**

* + - 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.



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

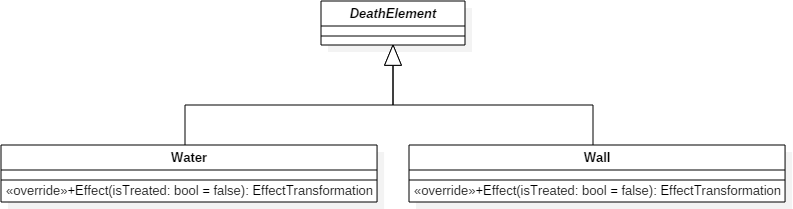
3: Applique les effets des éléments rencontrés

* + - 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).

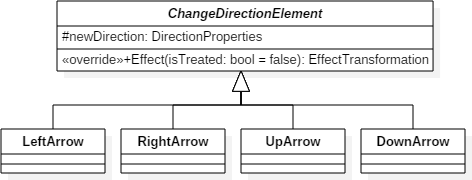


C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Water.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Mountain.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Tree.pngC:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\Rock.png

Fig. n° : Hiérarchie des "Death Elements"

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

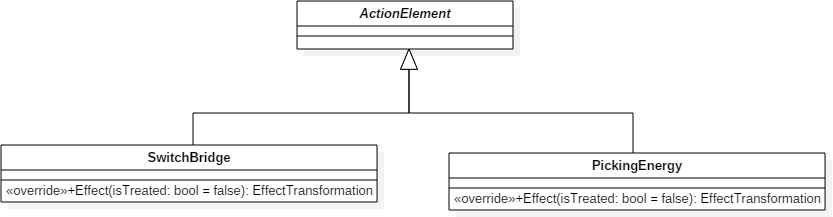
Les "Change Direction Elements" sont les éléments qui on un effet sur la direction du joueur. Quand le joueur les rencontres, ils lui impose une des quatres direction possible pour le joueur (Nord, Est, Sud, Ouest).



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

* **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.

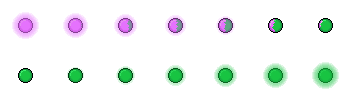
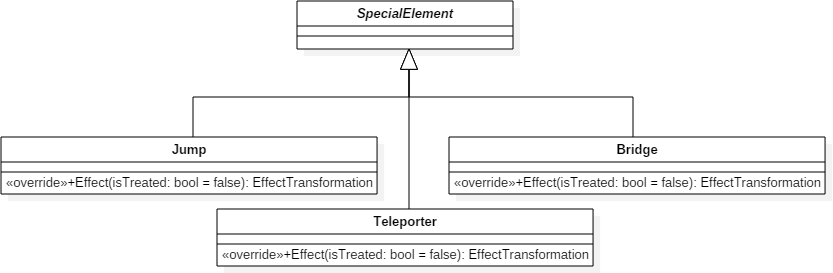
C:\Users\Cedric\Desktop\PuzzleGameProject\Assets\Resources\ObjectifTemporaire.png

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 des éléments dans notre application est résumé par le diagramme suivant:

* 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**

Afin de pouvoir effectuer une traduction efficace du jeu, l'utilisation de Json nous a parut la plus adapté. Avent de commencer quoi que ce soit, il a été nécessaire de prendre la classe “SimpleJSON”, mis à disposition par la communautée.

La traduction pure est délégué à la classe “SimpleJSON”, à laquel nous avons ajouter une classe, “JsonReader”, qui s'occupe de la liaison entre le text à traduire et “SimpleJSON” faisant la traduction. “GlobalMultiling” permet tant qu'à lui de vérifier la langue du la machine de l'utilisateur et d'appeller JsonReader pour lui déléguer la traduction de chaque élément traduisible. “GlobalMultilingComponent” s'occupe de gérer l'existance d'un GlobalMultiling. Voici la hiérarchie de l'*internationalisation* :



A chaque fois qu'un level est load la méthode “OnLevelWasLoaded” de GlobalMultilingComponent détruit le nouveau GlobalMultiling qui est créer afin de gérer le nombre d'instance de ce dernier. Après cela, la méthode “TranslateAll” du GlobalMultiling est appelé qui pour chaque élément traduisible appellera la méthode “Translate” qui utilise le JsonReader pour appeler le SimpleJSON.

* + - 1. **Le système de sauvegarde**

**MAXIME / BASTIEN**

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 but de sauvegarder la progression du joueur dans le jeu. Au niveau graphique la progression sera illustrée par des étoiles, qui représenteront le score réalisé pour un niveau, sous les boutons des niveaux déjà complétés dans le menu des niveaux, et par la désactivation des boutons des niveaux supérieurs au niveau après le dernier niveau complété. Au niveau du code ProgressionSave utilisé la classe « PlayerPrefs » de Unity, cette classe permet de garder et récupérer des données entre des sessions de jeu, pour récupérer et changer la valeur du niveau actuel. Cette valeur est récupérée pour savoir quel bouton il faut désactiver ou pour lui ajouter un score. Pour obtenir la première étoiles il faut finir le niveau, pour accéder à plus d'étoiles il faut faire le niveau dans un temps imparti qui est fixe pour tout les niveaux.

**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'éléments 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 étant 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.