Paulin DOUX
Pierre MICHEL

12/02/2016
Ei5 - AGI - IHMRV

Développement d'un simulateur de conduite

Projet supervisé par :
Paul RICHARD

Remerciements

Nous souhaitons avant tout remercier M. Richard, notre enseignant tuteur, pour sa présence tout au long du projet, pour son aide et ses suggestions. Nous souhaitons également remercier Mme. Emmanuelle MENETRIER de la faculté de psychologie d'Angers et Mr Vincent BOUCHER du Cerema pour leur présence et leur implication tout au long du projet. Et pour conclure, nous souhaiterions remercier Mr Pierre GAC, étudiant à l'ISTIA ayant initialement commencé le projet, pour son aide précieuse ainsi que ses conseils.

Table des Matières :

Introduction

Dans le cadre de nos études au sein de l'ISTIA, il nous était demandé de choisir un projet à réaliser sur toute la durée du semestre. Afin de mener à bien ce projet, il nous était notamment demandé de mettre en place une méthodologie de gestion de projet.

Le programme Perceive

Le programme Perceive est une initiative menée par le LPPL (Laboratoire de Psychologie des Pays de la Loire) portée par Mme Emmanuelle Ménétrier. Ce programme implique plusieurs autres organismes tels que le laboratoire Laris, le CHU d'Angers et le CEREMA (centre d'études et d'expertise sur les risques, l'environnement, la mobilité et l'aménagement). Ce programme a pour objectif de déterminer comment les connaissances modulent les interactions d'individus avec l'environnement dans le cadre d'activités de la vie quotidienne. Cette recherche se fait via l'utilisation d'environnements virtuels.

Le projet

Le projet que nous avons décidé de gérer fait partie du projet Perceive en partenariat avec Mme Emmanuelle Ménétrier du LPPL et Vincent BOUCHER du CEREMA. L'objectif du projet étant de développer un simulateur de conduite. L'intérêt de ce simulateur est multiple. D'une part, il permettra de déterminer où regarde le conducteur pendant qu'il conduit. D'autre part, le simulateur permettra d'évaluer les capacités cognitives du conducteur.

Historique du projet

Le projet de simulateur de conduite a été initié en Mai 2015, lors du début du projet Perceive. Ce projet fut débuté par Mr Pierre GAC, étudiant de l'ISTIA, durant son stage de 4ème année. Ce dernier parvint à compléter la majorité des objectifs proposés au cours de son stage.

Nos objectifs

Notre principal objectif est de poursuivre le développement du ce projet en y intégrant un tout nouveau terrain de campagne. Au fur et à mesure du projet, d'autres objectifs ont été ajoutés par le biais de Trello, tel que l'intégration du volant. De plus, après contact avec la personne en charge du projet avant nous, nous nous sommes rajoutés quelques tâches, comme la correction de certains bugs restants et la refonte de l'interface pour tracer le rail de guidage.

Gestion du projet

Répartition des tâches

La répartition des tâches s'est faite de la manière suivante :

	Paulin	Pierre
Création de la zone de campagne		X
Intégration de nouvelles interfaces	X	X
Correction de bugs	X	
Refonte de l'interface de traçage du rail de guidage	X	

Méthode SCRUM

La méthodologie que nous avons appliquée était très similaire à *SCRUM*. La méthode *SCRUM* est une méthode *agile* pour la gestion de projet basée sur les *sprints* et sur beaucoup de communication avec les clients. Un *SCRUM Master* est généralement en charge de gérer le bon déroulement du projet. Dans notre cas, nous n'avions pas réellement de *SCRUM master*; nous étions en contact direct avec les clients. Pour gérer le projet, nous avons mis en place un *dashboard Trello*. Cet outil nous permettait, d'une part, de connaître nos objectifs à court terme, et d'autre part, de communiquer notre avancement à nos clients avec toute transparence.

Logiciels/Outils

Unity 3D

Unity 3D est un outil permettant de créer très facilement des jeux vidéo. Ce logiciel est devenu l'outil de développement incontournable pour les entreprises de développement de jeux vidéo indépendantes comme pour les grandes entreprises. Unity est également beaucoup employé pour le développement de *Serious Games*. C'est-à-dire des jeux ayant des buts éducatifs ou thérapiques. *Unity 3D* doit notamment sa popularité au fait qu'il est gratuit pour les entreprises ayant un revenu en dessous d'un certain seuil. Unity 3D est un logiciel très complet permettant de développer des applications de *réalité virtuelle* avec une grande facilité.

Figure 0.3.1 : Logo d'Unity 3D

Bitbucket/Git

Git est un outil dit de *versionning*. C'est-à-dire qu'il permet de travailler en équipe sur un même code tout en préservant les différentes versions du code. Cet outil est énormément utilisé lorsqu'il faut travailler à plusieurs sur un projet. Bitbucket est un service web permettant d'héberger de façon privée des projets Git. Ceci permet notamment à plusieurs personnes du monde entier de travailler sur un même projet.

Trello

Trello est un outil en ligne permettant de créer et partager des *dashboards*. Un dashboard est une page composée de plusieurs colonnes dans lesquelles on peut disposer des *Post-Its*. Ce logiciel peut être très utile pour une méthode SCRUM car il permet d'assigner des tâches à des utilisateurs et facilite la communication à distance pour des projets.

Partie 1 : Design d’une nouvelle zone

L’un des objectifs principaux de ce projet était d’ajouter une nouvelle zone au jeu. En effet, lorsque nous avions repris le développement du projet, une seule zone était présente : la ville. Ultimement, il serait désirable d’avoir un total de trois zones : la ville, la campagne, et une zone périurbaine. Ce projet étant relativement court, nous n’avons eu le temps que pour créer la zone de campagne. Cet objectif a surtout été un exercice de *level design*.

1) Méthodologie du design

À premier abord, le *level design* semble être une tâche compliquée. En effet, il s’agit avant tout de créer une zone en partant de rien tout en s’assurant du réalisme de celle-ci. De ce fait, il est important d’être très méticuleux dans la création de la zone. Voici donc quelques conseils que nous avons adoptés dans notre méthodologie de conception d’une zone.

1 - S’inspirer du monde réel

Tout d’abord, pour créer un terrain ressemblant au monde réel, il peut être très utile d’étudier comment le monde réel est disposé. En effet, il n’y a rien de mieux que le monde réel pour pouvoir trouver de l’inspiration pour son monde virtuel. De ce fait, nous avons passé un certain temps à étudier des cartes et à observer les décors que nous pouvons rencontrer dans une campagne. Pour l’anecdote, la route entre Rennes et Angers a été une grande source d’inspiration pour le design de la campagne.

Figure 1.1.1.1 : Photos de routes de campagne
(Source : https://www.google.fr/maps/)

Comme nous pouvons le constater, un décor de campagne est particulièrement vaste. En haut d'une colline, il est possible de voir à plusieurs kilomètres. Le relief est généralement composé de très larges collines. Cependant, la variation en hauteur reste néanmoins faible. Nous pouvons également constater qu'il y a très peu de constructions. Il y a généralement très peu de maisons isolées. La plupart des bâtiments que nous pouvons rencontrer sont sou des fermes, ou des maisons dans un village, ou éventuellement des usines. La végétation a une grande place dans le décor; elle est présente tout autour de la route. Après tout, la campagne est composée en grande partie de champs et de forêts.

2 - Faire attention aux détails

Peu importe le type de décor, les scènes du monde réel sont remplies de plein de petits détails. Ces détails peuvent être des buissons sur le côté de la route, des poubelles à côté d'une maison, du lierre poussant sur un poteau électrique, … Dans le cadre du développement d'une application de réalité virtuelle, il est tout simplement impossible de représenter tous ces détails afin d'avoir une scène variée et intéressante. Avoir trop de détails dans une scène peut causer de sérieux ralentissements à l'application. C'est pour cela qu'il est capital de faire particulièrement attention aux détails.

Figure 1.1.2.1 : Screenshot du jeu Grand Theft Auto 5
(Source : http://www.gamesradar.com/)

Faire attention aux détails peut signifier deux choses. D'une part, il ne faut pas noyer la scène de plein de détails inutiles. Ces détails prendraient trop de ressources de l'ordinateur pour trop peu de résultat. D'autre part, en revanche, il ne faut pas laisser la scène trop terne. Un strict minimum de détail ferait que le joueur ne s'ennuie pas en regardant la scène tout en préservant les ressources de l'ordinateur. Nous pouvons nous inspirer de l'image ci-dessus pour avoir une idée du niveau de détail que nous pouvons espérer avoir dans un jeu à monde ouvert. Le joueur voudra certainement regarder vers l'horizon, il peut donc être intéressant de faire en sorte que l'on puisse voir des éléments au loin.

3 - Rendre la scène vivante

Une scène statique, peu importe le niveau de détail, sera toujours moins intérressante qu'une scène avec du mouvement et de la vie. Le joueur sera bien plus captivé si celui-ci ne se sent pas seul dans un monde figé. Pour celà, deux choses peuvent être faites. D'une part, il est possible de faire en sorte que le décor paraisse plus vivant; par exemple en faisant bouger l'herbe et les feuilles des arbres avec du vent. D'autre part, il est possible de rendre la scène plus vivante en ajoutant des animaux ou d'autres voitures.

Figure 1.1.3.1 : Screenshot du jeu Watch_Dogs
(Source : https://i.ytimg.com/vi/Tm5ZGumYRao/maxresdefault.jpg)

Comme nous pouvons le constater dans le jeu *Watch_Dogs*, le décor est rempli de voitures et d'humains. Cet ajout contribue à un plus grand sentiment d'immersion dans l'environnement. Il ne faut cependant pas trop monde dans un décor de campagne. Pour paraître plus réaliste, le joueur ne doit pas voir d'humains en dehors des villages et le nombre de voitures doit lui aussi rester limité.

2) Génération procédurale de terrain

La première étape dans la création de la nouvelle zone a été de créer le terrain. Dans *Unity*, les terrains sont des modèles 3D facilement modifiables grâce à des outils pour sculpter le relief. En revanche, sculpter un terrain entier de sorte que celui-ci soit réaliste n'est pas facile. C'est pour cela que nous avons généré aléatoirement un terrain. Cette méthode a pour avantage de produire un terrain rapidement avec une géométrie semblable au relief que nous pouvons trouver sur Terre.

1 - Bruit de Perlin

Il existe plusieurs méthodes de génération de relief. La plus courante est d'utiliser ce que l'on appelle le *Bruit de Perlin*. Ce dernier est ce que l'on appelle un bruit *cohérent*. C'est-à-dire que pour deux valeurs successives du bruit, l'écart entre ces deux valeurs restera faible. À l'opposé, un bruit non cohérent connaîtra des variations de valeurs parfois brutales.

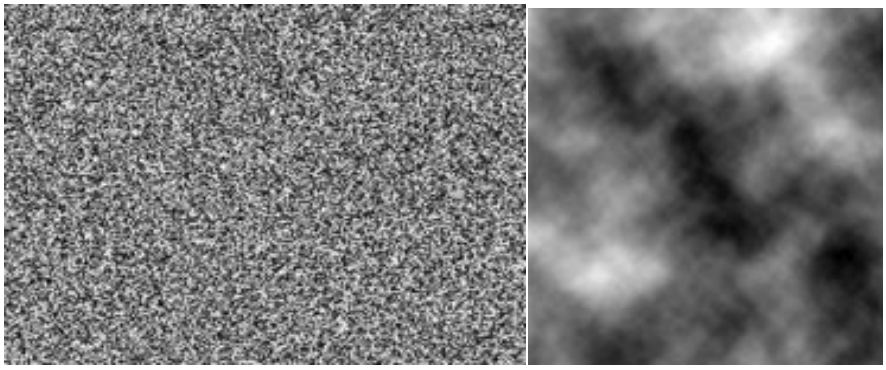

Figure 1.2.1.1 : Bruit blanc (gauche) et bruit de Perlin (droite)

Pour obtenir un bruit de Perlin, la première étape est générer un certain nombre de points équidistants. Ensuite, pour déterminer les valeurs des points non générés situés entre 2 points générés, il suffit de réaliser une interpolation.

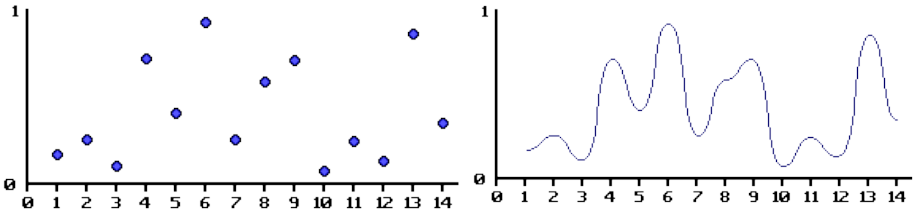

Figure 1.2.1.2 : Génération d'un bruit de Perlin 1D
(Source : http://freespace.virgin.net/hugo.elias/models/m_perlin.htm)

Sur l'image ci-dessus, nous avons un bruit en une dimension. Il est néanmoins possible de générer des bruits en 2D (ex : génération de terrain), en 3D (ex : génération de nuages), ou en n'importe quelle autre dimension. Afin de rendre le bruit plus réaliste, un seul bruit ne suffit pas. En général, plusieurs bruits sont combinés pour obtenir un résultat plaisant. Pour chacun des bruits, le nombre de points générés est une puissance de 2 différentes. Une persistance est appliquée sur chaque bruit de sorte que ceux avec un nombre de points plus élevé ont un impact plus faible.

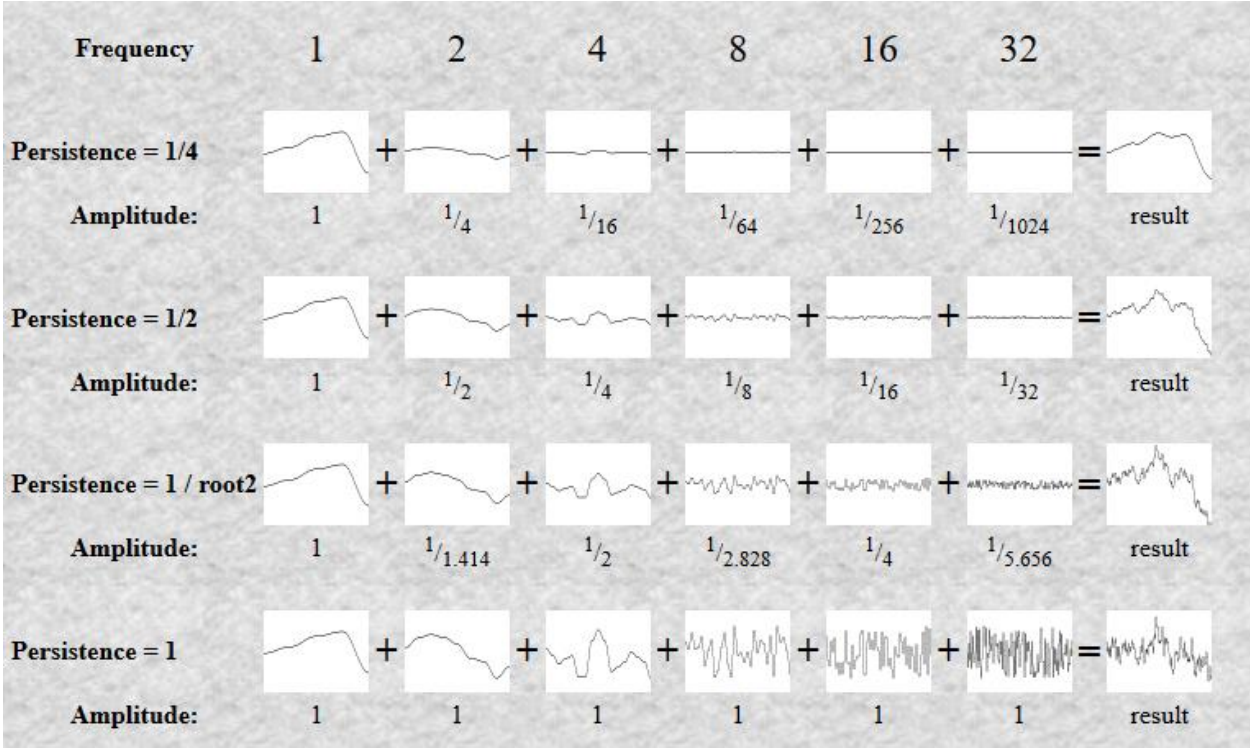

Figure 1.2.1.3 : Illustration de l'impact de la persistance
(Source : http://freespace.virgin.net/hugo.elias/models/m_perlin.htm)

En modulant le nombre de bruits à combiner, leur amplitude, et la persistance de chacun, il est alors possible d'obtenir le résultat que nous désirons. Comme précisé plus tôt, le relief que nous souhaitons doit être relativement plat et avec peu de variation. De ce fait, nous devons éliminer les hautes fréquences le plus possible. En d'autres termes, nous devons choisir une faible persistance afin de ne pas avoir de petites bosses partout sur le terrain.

2 - Logiciel : Terragen

Pour générer notre terrain, nous avons utilisé le logiciel *Terragen*. Ce logiciel permet de créer des *heightmaps (cartes de relief)* directement exploitables par *Unity*. L'interface de Terragen peut paraître imposante vu le nombre colossal d'options à notre disposition. Cependant, en cherchant un peu, nous pouvons remarquer les trois seules options dont nous avons besoin : l'amplitude, le nombre de couches, et la persistance. C'est en jouant uniquement avec ces trois valeurs que nous sommes parvenus à un résultat concluant.

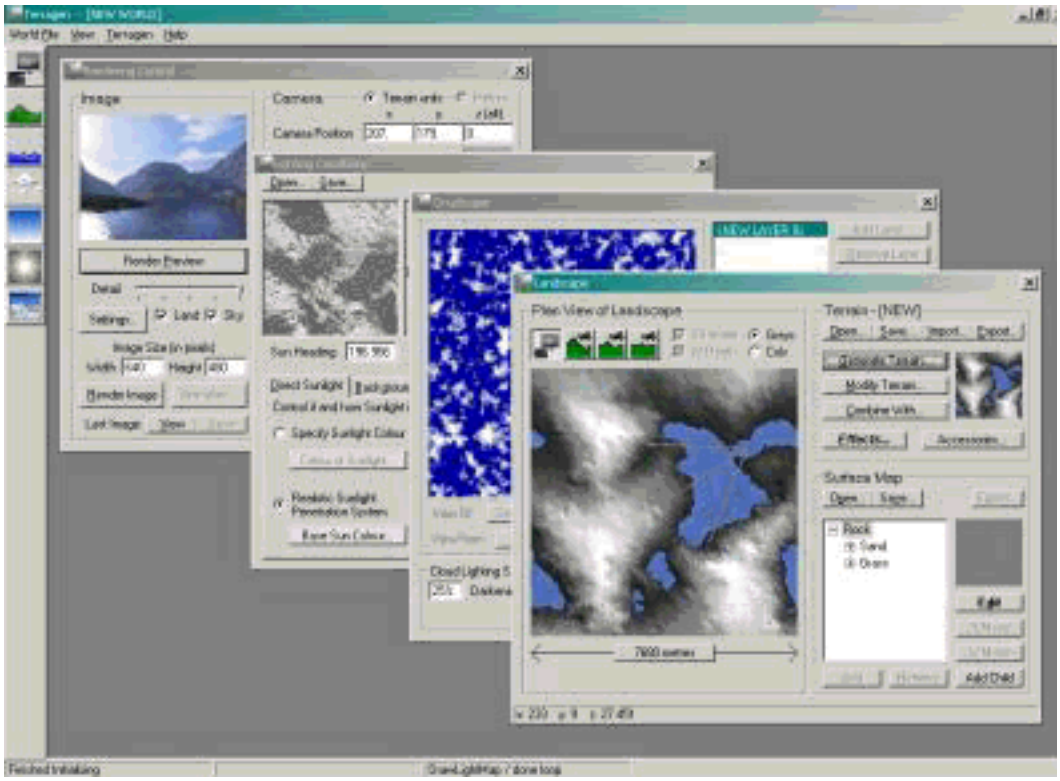

Figure 1.2.2.1 : Interface du logiciel Terragen

3 - Ajustements sur le terrain généré

Une fois le terrain généré et importé sous Unity, il reste quelques détails à régler. En effet, même en gardant une persistance faible dans le logiciel *Terragen*, il se peut qu'il y ait encore des bosses un peu trop imposantes. *Unity* possède un outil de *"smooth"* permettant d'uniformiser un peu le terrain et de ce fait de supprimer les bosses.

3) Ajout de décors au terrain

Une fois le terrain créé, la seconde et plus longue étape fut de le décorer. En effet, cette partie fut la plus longue car c'était à ce moment qu'il fallait se montrer le plus méticuleux. Les principaux éléments que nous avons ajoutés au terrain sont les routes, les batiments, et la végétation.

1 - Routes

Pour faire un simulateur de conduite, il va sans dire qu'il est nécessaire d'avoir des routes sur lesquelles conduire. C'est pour cela que le design de la zone de campagne a été centré autour des routes. C'est-à-dire que une fois les routes placées, nous avons réfléchit au placement des autres éléments de décor. Pour choisir comment placer les routes, nous avons dessiné une carte de la zone avec les lacs et les rivières. À partir de là, nous avons pu réaliser un plan de la zone (Voir image ci-dessous). Un village, une ferme et une forêt furent placés une fois les routes dessinées.

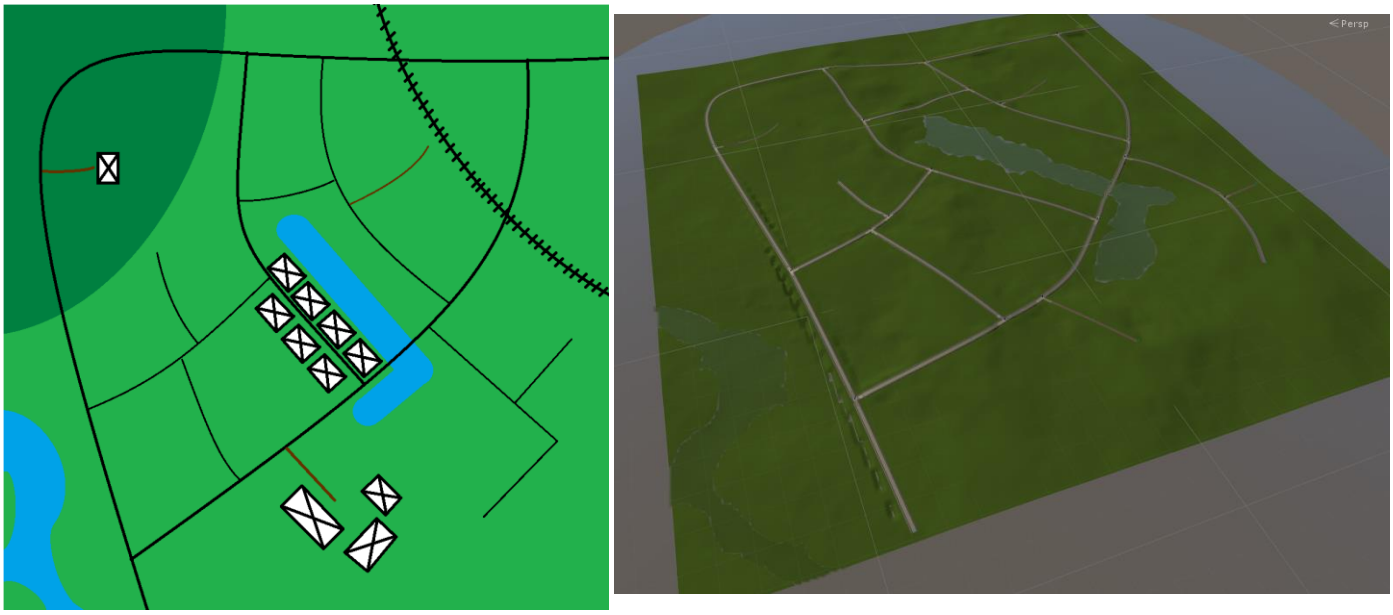

Figure 1.3.1.1 : Carte de la campagne (gauche) & Campagnes avec les routes (droite)

Tracer les routes n'a pas été une tâche particulièrement difficile. En effet, nous avions à notre disposition un package *Unity* destiné à cet usage. *EasyRoads* est un package Unity permettant de tracer un itinéraire. Des modèles 3D des routes seront alors générées et disposés sur le terrain. Avoir ce package a été un gain de temps considérable car il nous aura permis de tracer les routes très rapidement.

2 - Maisons

Comme précisé précédemment, nous avons décidé de créer un petit village au centre de la campagne afin de rendre la zone un peu plus vivante et diversifiée. En plus de ce village, plusieurs autres bâtiments ont été ajoutés tout autour de la zone telle qu'une ferme. La majorité des modèles de maisons ont été trouvé sur *TF3DM*, site proposant des modèles 3D gratuitement.

Figure 1.3.2.1 : Village en campagne

3 - Végétation

Figure 1.3.3.1 : Scènes de campagne avec de la végétation

Comme précisé dans la partie sur la *méthodologie du design*, nous avons constaté que la végétation est une partie capitale dans le décor. Sans elle, la zone serait encore relativement vide. Comme nous pouvons le constater sur les images ci-dessus, même simplement ajouter de l'herbe autour de la route réduit ce impression de vide.

4) Optimisation de l’affichage

Une fois le terrain parfaitement décoré, le travail ne s’arrête pas là. En effet, selon le niveau de détail, il se peut que la scène connaisse des ralentissements. Il est généralement considéré dans l’industrie du jeu vidéo que 30 images par seconde sont le strict minimum à obtenir pour une expérience de jeu plaisante. Les deux méthodes suivantes peuvent permettre d’améliorer les performances du jeu de façon considérable. Pour un jeu se passant dans un monde ouvert, ces méthodes deviennent essentielles.

1 - Light map

Dans *Unity*, il existe trois façons pour afficher les ombres : *dynamique*, *baked* ou *mixed*. Par défaut, c’est l’option *dynamique* qui est sélectionnée. Cette méthode est de loin la plus inefficace car elle consiste à recalculer à chaque image la disposition des ombres. Ce mode est utile pour calculer les ombres d’objets qui ne bougent pas pour un environnement. Le deuxième mode, *baked*, permet de pré-calculer les ombres de tous les objets. Comme cela, le jeu n’a pas à les recalculer durant son exécution. C’est cette méthode qu’il faut appliquer aux éléments statiques du décor. Le dernier mode, *mixed*, est un mélange des deux méthodes. C’est celle-là que nous avons utilisée car elle permet de pré calculer les ombres des objets statiques tout en calculant les ombres dynamiquement des objets se déplaçant.

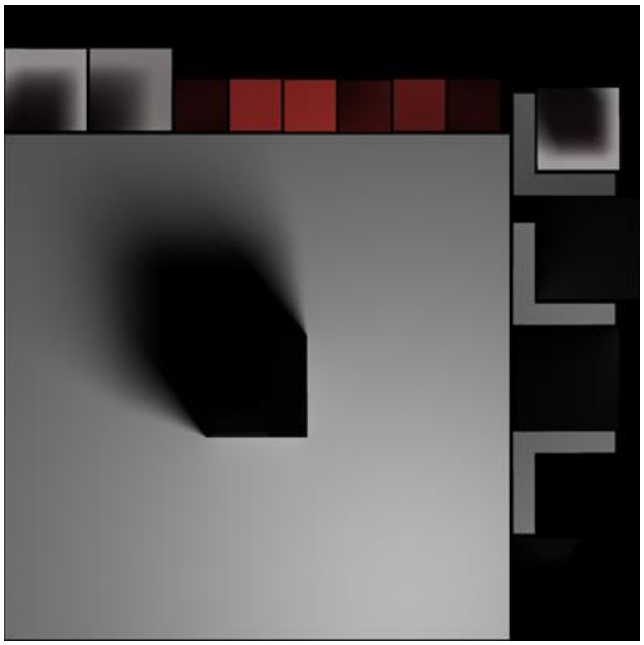

Figure 1.4.1.1 : Exemple de lightmap sur un cube

2 - Occlusion Culling

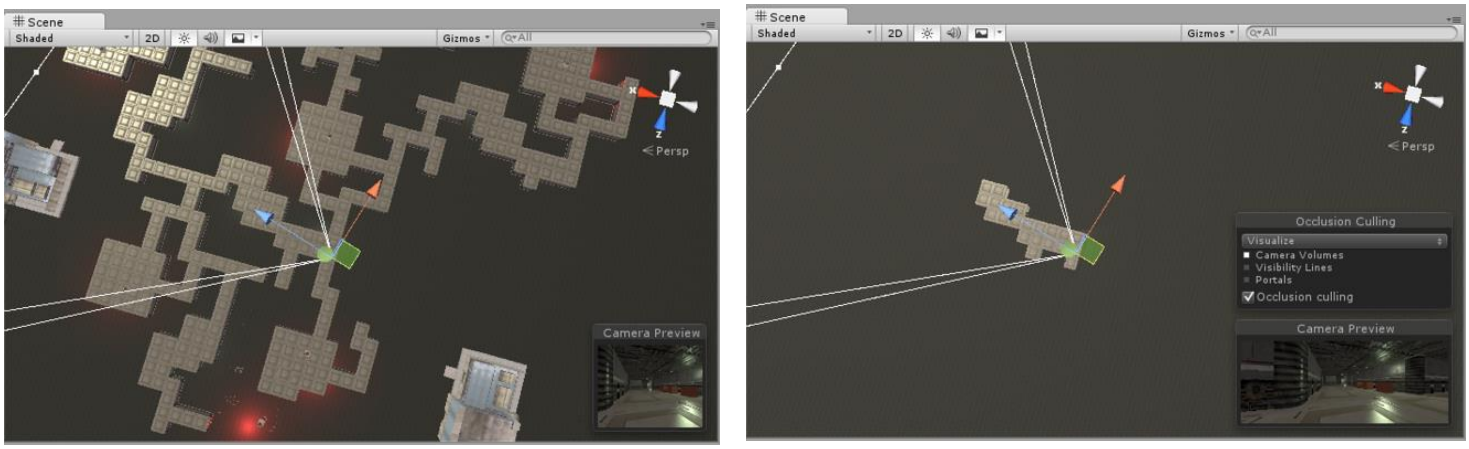

Figure 1.4.2.1 : Exemple d'utilisation de l'Occlusion Culling
Sans Occlusion à gauche, avec à droite

La deuxième méthode proposée par *Unity* se nomme *l'Occlusion Culling*. Avec cette méthode, il est possible d'épargner à l'ordinateur des quantités impressionnantes de calculs. Pour faire simple, avec cette méthode, l'ordinateur ne dessine pas ce que le joueur ne peut pas voir. Comme on peut le constater sur les images ci-dessus, sans *occlusion*, tous les objets seront affichés; y compris ceux que le joueur ne pourra pas voir. C'est une perte immense de temps. Avec *occlusion*, uniquement les objets visibles par le joueur pourront être affichés. Ce qui économise du temps et des ressources pour l'ordinateur.

5) Améliorations possibles

Bien que le projet soit terminé, il reste néanmoins plusieurs améliorations qui pourraient être apportées à la zone. D'une part, d'un point de vue design, il manque encore quelques éléments tel que les lignes électriques qui pourraient être ajoutées. De plus, la zone n'est pas encore peuplée. Il faudrait donc ajouter des voitures sur les routes et éventuellement des animaux dans les champs. D'autre part, la scène n'est pas encore complète car il n'est pas encore possible d'utiliser les options de simulation dans la zone. Ainsi, il faudrait également ajouter certain objets présents dans la scène de ville notamment pour afficher les menus et gérer *l'Intelligence Artificielle* des voitures.

Partie 2 : Refonte du choix d’itinéraire

Etant difficile de travailler à deux sur une même scène, il a été décidé que une seule personne travaillerait sur la scène de la ville. Afin de savoir et comprendre ce qui pouvait être changé sur cette scène, nous avons pris contact avec la personne qui été en charge du projet avant nous. Cette personne nous a alors indiqué qu’il y avait des problèmes avec le système de rail et de planification d’itinéraire. Nous nous sommes donc penchés sur ce problème.

1) Ancien système

1 - Présentation

Notre projet possède un système de planification de route, permettant de régler différentes choses, tel que les feux de carrefours, les routes bloquées mais également de planifier un trajet qui sera alors automatiquement suivit par l’utilisateur grâce à un rail invisible de guidage.

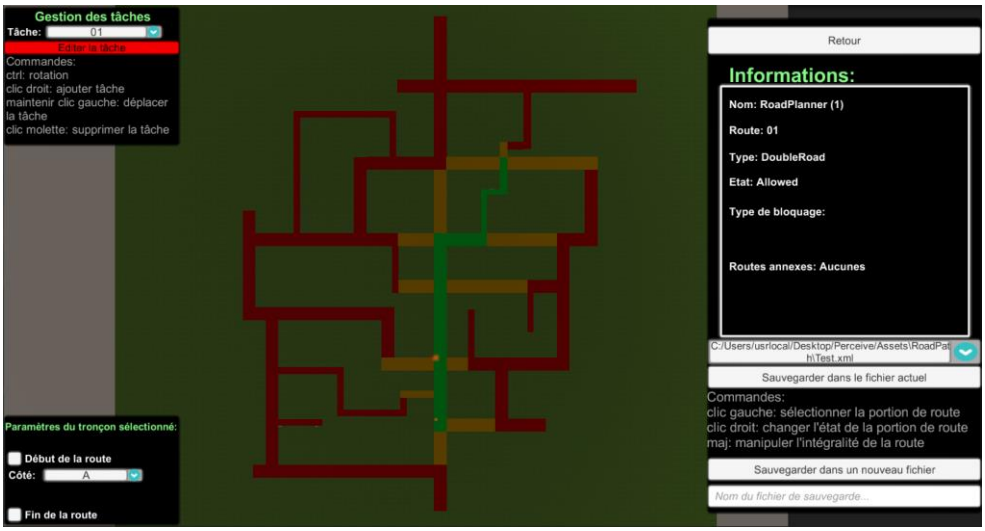

Figure 2.1.1.1 : Ancien planificateur de route avec réglage de route

L’ancien système de planification d’itinéraire se présentait sous la forme suivante, en rouge se situe les routes qui ne sont pas praticables, en orange les routes qui sont bloquées et en vert, celles qui serviront à créer le rail de guidage. Le choix du point de départ et de fin se fait grâce au menu en bas à gauche. Un autre menu en bas à gauche étant présent pour régler les feux des carrefours après leurs sélections. En haut à gauche se situait le menu pour gérer les taches. Et finalement à droite le menu de sauvegarde et de chargement.

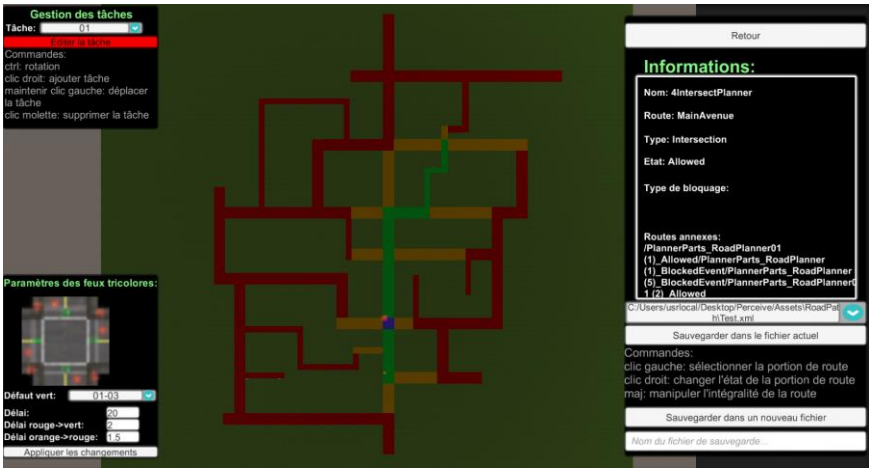

Figure 2.1.1.2 : Ancien planificateur de route avec réglage de carrefour

Le choix de l'itinéraire se faisait donc en changeant, grâce à un clic droit, la couleur des routes et en définissant un point de départ, ainsi qu'un point de fin.

2 - Les limites du système

Si nous nous intéressons à ce système, c'est que celui-ci présentait des limites et des défauts. Le premier, et plus important, était dans le *pathfinding* pour le tracé du rail. Lorsque l'algorithme devait prendre un chemin vers le sud ou l'ouest de la carte, celui-ci prenait le point de navigation à l'opposé de la route et parcourait la route en sens inverse. Entrainant ainsi, une désorientation de l'utilisateur et surtout des sorties de route et traverser à travers les immeubles.

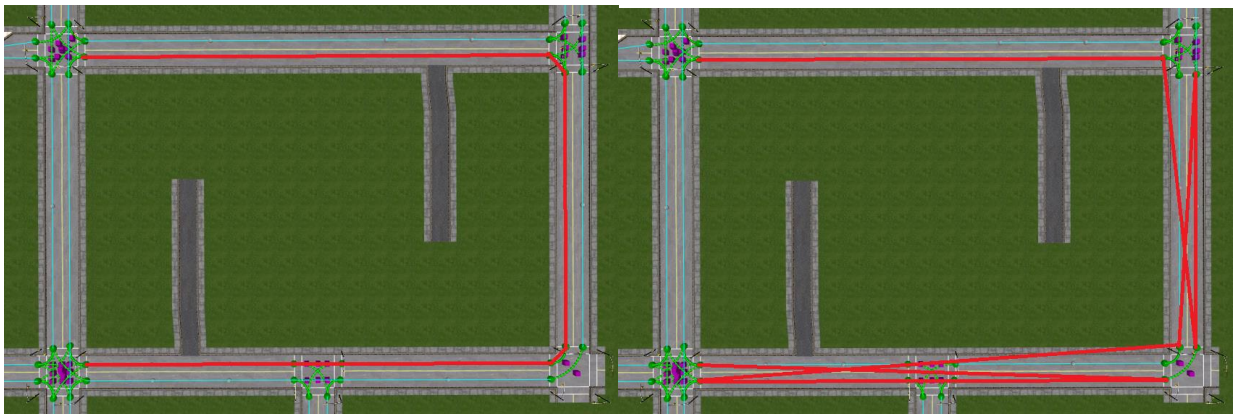

Figure 2.1.2.1 : Route planifiée pour le rail (gauche) & Route effectuée avec le rail (droite)

Un autre problème était l'incapacité à faire des croisements ou des boucles au sein même du parcours. Il était impossible de passer par un carrefour et de repasser par ce même carrefour pour prendre une autre route.

2) Refonte du système

Pour pallier à ces problèmes, il a été décidé de changer la méthode de traçage des routes, tout en conservant le système pour les carrefours ainsi que pour la tâche. Ce travail demande une bonne compréhension de l'existant, et cela malgré le manque de documentation. Nous avions cependant la possibilité de contacter la personne en charge du projet avant nous.

1 - Compréhension du code

Les premières semaines sur cette partie du projet furent axées sur la compréhension de l'existant, afin de déterminer ce qui pouvait, ou devait être gardé et ce qui devait être changé. Ce travail nous a permis de comprendre comment fonctionnait le système de route et de *waypoint* ainsi que comprendre comment placer ces derniers sur le nouveau terrain. Une fois les scripts compris, nous avons commencé par modifier le script ayant le moins d'impact sur le reste de l'application, celui calculant le rail, le pathfinding.

2 - Correction de l'algorithme de pathfinding

La correction du pathfinding fut sans grand impact sur le fonctionnement du reste de l'application, car il s'agit d'une seule fonction qui n'est appelée qu'une fois en début de parcours. Il convenait d'abord de déterminer les paramètres d'entrée et de sortie de la fonction et une fois cette tache effectuée, il a été nécessaire de faire la fonction qui aller structurer les données comme nous le souhaitions. Le code de la fonction ne mérite pas que l'on s'attarde plus que ça dessus et va être changé par la suite lors de la refonte de l'interface, mais au bout de deux jours de travail, la fonction était terminée et remplissait son rôle.

3 - Refonte de l'interface

Afin de rendre l'interface plus simple à utiliser et d'intégrer le futur système de choix d'itinéraire, nous avons dû modifier l'interface déjà existante. Le projet étant le premier projet que nous réalisions sous Unity 5, il a été nécessaire de comprendre comment fonctionnait le nouveau système d'interface intégré dans Unity 5. Cela pris un certain temps, et après quelques essais sur un projet vide, nous sommes arrivés à un résultat satisfaisant. Afin de minimiser les changements à faire au niveau du script et l'interface, il a été choisi de conserver les menus de taches, de carrefour et de changer le menu de droite. Le menu de réglage de route n'étant plus nécessaire dans le nouveau système de choix de route, celui-ci a été retiré. Une fois l'interface en place, il a fallu changer le script en charge de l'ancienne interface pour la nouvelle.

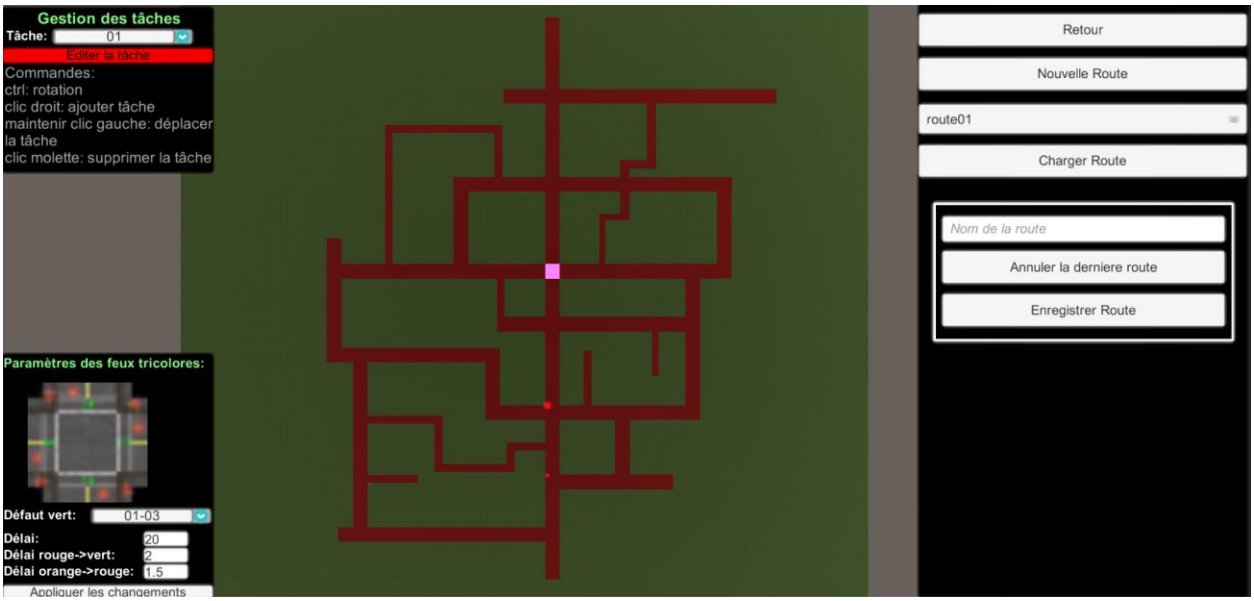

Figure 2.2.3.1 : Nouvelle interface de traçage de route

4 - Algorithme de choix de route

Comme dit précédemment, le système de choix de routes présente des limites dans son fonctionnement et son utilisation n'était pas des plus intuitives, notamment le fait de devoir choisir pour chaque route le coté emprunté. Afin de contrer cela, il a été choisi de dessiner la route en choisissant les intersections de la route que l'on souhaite emprunter. Lorsque l'on commence une nouvelle route, des sphères apparaissent au-dessus de toutes les intersections, ces sphères représentent tous les points de départs possibles pour une route. Pour sélectionner son point de départ, il suffit de cliquer dessus.

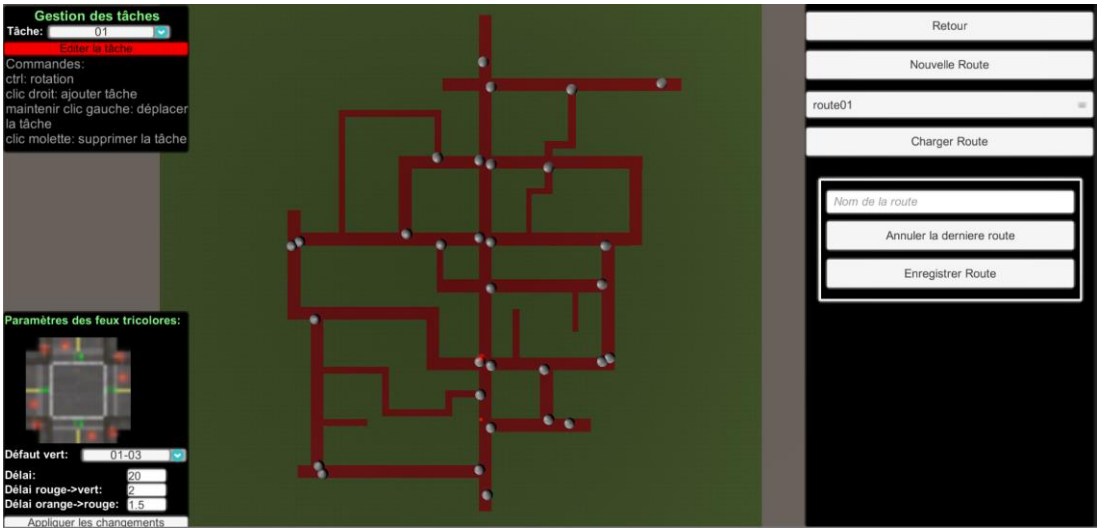

Figure 2.2.4.1 : Sélection d'une nouvelle route

À ce moment-là, seul les points de sorties de la route restent affiché à l'écran, il suffit donc de cliquer dessus pour continuer de tracer la route. Une ligne va apparaitre au fur et à mesure que l'utilisateur trace la route, afin d'avoir un repère visuel de celle-ci. Une fois la route terminée, il suffit simplement de la sauvegarder.

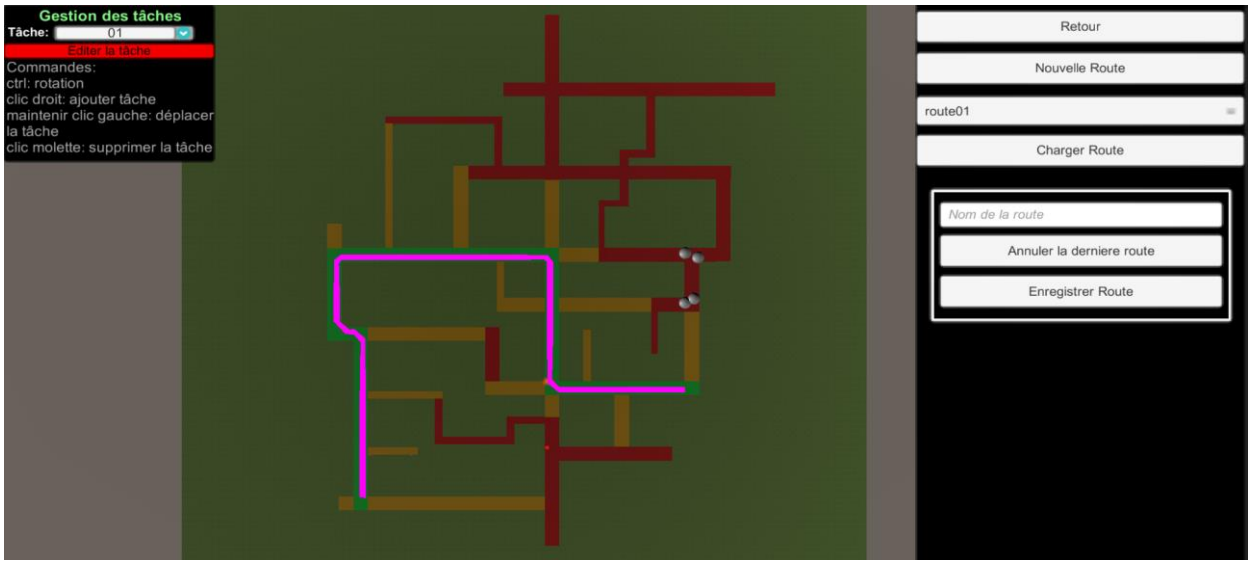

Figure 2.2.4.2 : Exemple de trajet

Le système de configuration de la tâche et des carrefours et toujours présent et fonctionne exactement comme avec l'ancienne interface.
Il est également toujours possible de bloquer ou non certaines routes grâce au clic droit. Même principe qu'avant, les routes vertes et rouges sont les routes praticables, et les routes oranges sont des routes présentant des obstacles à leurs extrémités. Pour finir, une fonction "annuler" a été implémentée, afin d'annuler les derniers points sélectionnés.

5 - Fusion avec l'existant

L'élaboration de l'algorithme pour tracer la route nous prit quelque jours, mais la partie la plus complexe fut l'intégration de celle-ci au reste de l'application, de manière à ce que l'on puisse sauvegarder les routes, les carrefours, les tâches et les obstacles, mais surtout que l'on puisse après lire ces sauvegardes et les traduire en un rail que va suivre le joueur dans le reste de l'application. Pour cela, l'ancien système de sauvegarde a été conservé, mais légèrement modifié, car les données de la route tracées sont différentes de celles qui étaient sauvegardées avant. Les données étant différentes, il a fallu refaire la fonction de pathfinding une nouvelle fois, mais la fonction étant très proche dans son fonctionnement de celle traçant la ligne dans l'éditeur de route, il nous a fallu très peu de temps pour mettre à jour cette fonction. Une fois ces dernières modifications apportées, le nouveau système de choix de route fut pleinement fonctionnel.

Partie 3 : Ajout de nouvelles IHM

1) Volant Logitech G27

1 - C’est quoi ?

1 - Description

Le volant Logitech G27 est une interface permettant, sous la forme d’un volant, de contrôler diverses applications. Ce volant permet une meilleure immersion et un contrôle plus précis dans les applications et jeux de conduite.

Figure 3.1.1.1 : Volant Logitech G27

1 - Caractéristiques techniques

Le volant Logitech possède les caractéristiques suivantes :

- Mécanisme de retour de force puissant à deux moteurs avec engrenage hélicoïdal;
- Levier six vitesses avec enclenchement de la marche arrière par simple pression;
- Témoins de changement de vitesse/tachymètre;
- Volant réaliste de 28 cm de diamètre à revêtement en cuir;
- Pédales d'accélérateur, de frein et d'embrayage en acier.

2 - Comment est-ce utilisé ?

Lors de son intégration, deux choix se sont offerts à nous, le premier était d'utiliser la librairie fournie afin d'intégrer au mieux le volant et de profiter de toutes ses capacités, tel que le retour de force ou le levier de vitesse, mais cela limité notre application qu'à ce volant précis. Le second choix d'intégration était d'intégrer le volant comme un joystick ou une manette, cela permet d'avoir les fonctions de base du volant et ceci avec n'importe quel type de volant. Cette dernière solution a était retenue car elle est beaucoup plus simple à mettre en place et ne limite pas l'application à un volant précis. De plus, elle permet également d'utilisé une manette sur l'application.

2) Oculus Rift

1 - C'est quoi ?

1 - Description

L'oculus rift est un casque de réalité virtuelle permettant une meilleure immersion de l'utilisateur dans l'univers dans lequel il évolue. Ce casque se présente comme un masque recouvrant les yeux, attaché au visage de l'utilisateur par des sangles. Au sein de ce casque ce situe deux lentilles braquées sur un écran plat où sont projetées deux images à destinations de chaque œil de l'utilisateur. A ce système de vision stéréoscopique, ce couple un système de positionnement dans l'espace à l'aide de gyroscope plus accéléromètre pour les rotations de la tête, et d'une caméra infrarouge, pour le positionnement de la tête.

Figure 3.2.1.1 : Oculus Rift

2 - Caractéristiques techniques

L'Oculus Rift possède les caractéristiques suivantes :

- Ecran OLED 1920*1080 (par œil: 960*1080);
- Gyroscope 3 axes, Accéléromètre, Magnétomètre;
- Caméra infrarouge.

2 - Comment il devait être utilisé ?

L'oculus n'est pas intégré au sein de l'application, mais pourrait être un plus pour celle-ci. Cela apporterait de l'immersion au joueur qui pourrait se sentir plus investit dans sa tâche. Mais cela pourrait gêner l'expérimentateur, car il n'est pas possible d'enregistrer les données d'occulométrie avec un tel dispositif sur le visage, seuls les mouvements de têtes pourraient être enregistré.

3) Occulomètre

1 - C'est quoi ?

L'occulomètre est un dispositif permettant d'enregistrer et mesurer les mouvements, la vitesse et la position des yeux par rapport à un écran. Ce dispositif se présente sous différentes formes, comme des lunettes ou uns barre de caméras à placer sous l'écran. Il est généralement constitué d'un ou plusieurs couples, capteurs infrarouge plus caméra, qui scrutent en permanence le mouvement des yeux et les enregistrent.

Figure 3.3.1.1 : Lunette occulométrique (gauche) & Barre occulométrique (droite)

2 - Comment ça devait être utilisé ?

Ce dispositif est une demande de la personne qui a commandé le projet. Son objectif est d'observer le comportement des personnes face à différents distracteur et d'étudier leur conduite, afin de déterminer si la distraction a impacté cette dernière. N'ayant pas eu accès à un occulomètre au cours du projet, nous n'avons pas été en mesure d'implémenter cette fonctionnalité.

Conclusion

Bilan du projet

Ce qui a été fait

Une nouvelle zone a entièrement été construite pour le projet : la zone de campagne. La construction de la zone inclut la génération du terrain, l'ajout de routes, et l'ajout d'éléments de décor tel que les maisons et la végétation. La refonte de l'interface de tracés de routes est complète, mais de petites améliorations pourraient encore être apportées pour faciliter son utilisation. Des erreurs provenant du projet initial ont également été corrigées.

Ce qui n'a pas été fait

Bien que la zone de campagne soit finie, certaines fonctionnalités du projet initial manquent encore tel que les menus, l'Intelligence Artificielle, et le rail. L'intégration de l'occulomètre n'a pas pu être réalisé, due à un manque de matériel et principalement à un manque de temps. Quelques indications d'utilisations de l'interface de tracés de routes avaient aussi été envisagé, mais n'ont pu être mis en place.

Ce qui peut être amélioré

D'après la personne en charge du projet avant nous, il aurait pu être possible de changer le type d'élément qui bloque les routes, mais par manque de temps, il n'avait pas pu l'intégrer. Des indications sur le fonctionnement de l'interface de tracé de route serait un plus pour l'expérience utilisateur.

Bilans personnels

Avis Paulin

Ce projet fut une bonne expérience, tout d'abord, ce fut le premier projet que j'ai effectué en groupe avec un *SVN*. De plus le projet étant sous *Unity 5*, cela m'a permis d'apprendre le fonctionnement des nouvelles interfaces mises en place dans cette version de l'éditeur. Ce projet m'a également montré l'importance d'avoir un projet structuré et clair, afin de facilité la reprise du code par d'autres personnes extérieur au projet.

Avis Pierre

Ce projet fut, pour moi, une expérience très intéressante. Il m'était déjà arrivé de travailler sur un projet Unity dans une équipe, mais jamais n'ai-je eu à utiliser de logiciel de *versionning* tel que *Git* de la sorte. De plus, ce projet fut une première expérience en *Level Design*. Pour la première fois j'ai commencé à me poser des questions quant à l'esthétique que doit avoir une scène dans un jeu vidéo. Mais plus que tout, ce projet s'est montré intéressant car nous avons eu l'occasion de collaborer avec des personnes extérieures à l'école afin d'améliorer le projet.

Si ce projet était à refaire, je pense que je changerais l'ordre dans lequel j'ai fait les choses. En effet, je me suis beaucoup concentré sur l'aspect graphique de la scène avant de tenter d'implémenter *l'Intelligence Artificielle*. Le souci est que lorsqu'un problème survient en milieu de projet, il peut encore être corrigé. Mais si des données sont perdues à la fin, comme c'est arrivé, il est beaucoup plus compliqué de réagir. Ce que je ferais à la place serait finir de créer les routes. Et une fois ceci fait, j'implémenterais le système d'intelligence artificielle.

Bibliographie

Sites Web:

- Projet de recherche PERCEIVE
  http://laris.univ-angers.fr/fr/activites-scientifiques/projets/projets-actuels/perceive.html
- Bruit de Perlin
  http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
- Occlusion Culling
  http://docs.unity3d.com/Manual/OcclusionCulling.html
- The definitive .gitignore for Unity projects – 6 Mai 2013 – Kleber LOPES
  http://kleber-swf.com/the-definitive-gitignore-for-unity-projects/

ANNEXE A : Tutoriel – Mettre un projet Unity sur Git

Dans le développement d'applications sur *Unity*, nous sommes souvent amenés à travailler en équipe. L'utilisation d'un outil tel que *Git* ou *Mercurial* peut ainsi se révéler particulièrement pratique afin de partager ses modifications avec le reste de l'équipe. En revanche, avant de gérer le *versioning* du projet, il y a quelques précautions à prendre qui pourraient rendre la vie plus facile par la suite.

Modifications sur le projet *Unity*

Avant de créer le répertoire *Git*, il y a une petite modification à effectuer :
1 - Dans *Unity*, aller dans *Edit* > *Project Settings* > *Editor*
2 - Mettre les options encadrées en rouge ci-contre comme indiqué.

Mettre l'option *Asset Serialisation* à *Force Text* est particulièrement utile dans la gestion des conflits.

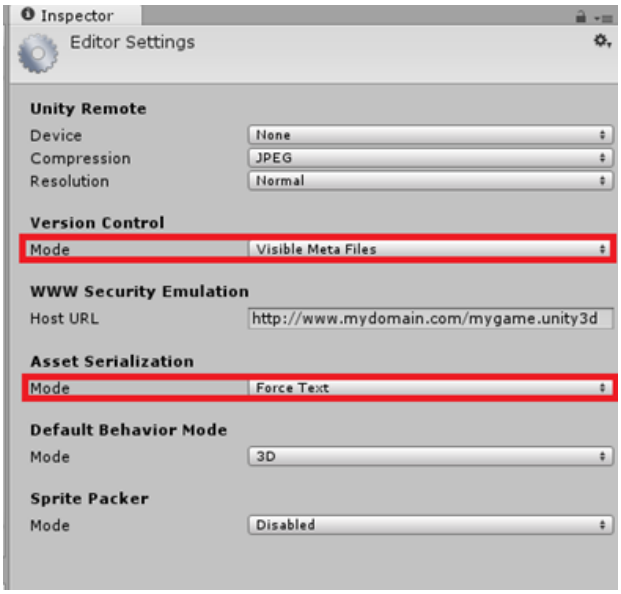

Figure A.1 : Paramètres à modifier sur Unity

Initialiser le Répertoire *Git*

1 - Créer le Répertoire *Git*
Les commandes ci-dessous permettent de créer et initialiser le répertoire *Git*.

```
mkdir <répertoire>
cd <répertoire>
git init
```

2 - Créer le fichier *.gitignore*

Le fichier .gitignore est un fichier listant tous les fichiers et dossiers qui ne seront pas mémorisés par le répertoire. Voici un fichier suggéré par *Kleber LOPES* :

```
# ============== #
# Unity generated #
# ============== #
[Tt]emp/
[Oo]bj/
[Bb]uild
[Ll]ibrary/
sysinfo.txt


# ==================================== #
# Visual Studio / MonoDevelop generated #
# ==================================== #
[Ee]xported[Oo]bj/
/*.userprefs
/*.csproj
/*.pidb
/*.suo
/*.sln*
/*.user
/*.unityproj
/*.booproj


# ============ #
# OS generated #
# ============ #
.DS_Store*
._*
.Spotlight-V100
.Trashes
Icon?
ehthumbs.db
[Tt]humbs.db
```

3. Copier les fichiers du projet dans le répertoire *Git*.
4. Ajouter les fichiers au *Git* et faire un *commit*.

Gérer les conflits

Pour la plupart des cas, la gestion des conflits lors de *merge* se fait de la même façon que pour n'importe quel autre type de projet. En revanche, dans le cas des fichiers *.unity*, il se peut que la gestion des conflits soit plus compliquée puisque les fichiers sont très longs et la moindre erreur peut corrompre le projet. Pour remédier à ce problème, il existe un outil nommé *UniMerge* qui permet d'avoir une approche visuelle à chaque choix possible dans le merge. Il est également conseillé de beaucoup utiliser les *prefabs* afin de préserver certaines configurations.

Résumé :

Dans le cadre de nos études à *l'ISTIA*, école d'ingénieurs d'Angers, il nous a été demandé de reprendre le développement d'un *simulateur de conduite* pour la *faculté de psychologie* et le *Cerema*. L'interêt de ce logiciel est d'évaluer le comportement des conducteurs sur la route. Nos principaux objectifs au cours de ce projet étaient de créer de nouveaux décors pour le simulateur, améliorer des fonctionnalités existantes, et implémenter l'usage de nouvelles *Interfaces Homme-Machine*.

Mots-clés : Unity, Simulateur, Conduite, Interface Homme-Machine

Summary:

As a part of our studies in *ISTIA*, engineering school in Angers, we were asked to continue the development of a *driving simulator* for the *psychology faculty* and the *Cerema*. The main goal of this project is to monitor the behaviour of the drivers when on the road. Our main objectives for this project were to create new landscapes for the simulator, improve existing functionalities, and implement new *Human-Machine Interfaces*.

Key words: Unity, Simulator, Driving, Human-Machine Interface