RAPPORT DE PROJET

Projet flight arena

mr cube :

Vincent Rospini-Clerici,

Guillaume Rebut

chef de projet : Arthur Remaud

16 juin 2015

Sommaire

Introduction

Le groupe mr cube, composé de Guillaume Rebut, Vincent Rospini Clerici et Arthur Remaud, est fier de vous présenter son projet : *flight arena*.

Ce projet répond à une obligation de l'école EPITA de faire un projet libre par groupe de quatre. Nous avons donc choisi de faire un jeu de vaisseau dirigé à la troisième personne dans lequel le but est d'éliminer ses ennemis. Le but est de nous apprendre à travailler en équipe, tout en gérant un projet du début à la fin, comportant obligatoirement un cahier des charges, un rapport de projet et plusieurs soutenances.

Nous avions pour ce projet l'obligation d'utiliser les langages *Caml* ou C#, avec le moteur Unity pour faire le jeu.

Nous étions au départ avec Nikolas Miletic dans le groupe, mais celui-ci est parti en S1# au cours du mois de janvier. Nous avons alors réparti les tâches de cette manières : Vincent s'occupera principalement des graphismes et du site internet, Guillaume du gameplay et des cartes, et Arthur du code du jeu et des menus.

1 Graphismes et son

1.1 Graphismes

1.2 Son

2 Contrôles du vaisseau

Les contrôles du vaisseau sont les premières choses implémentées dans le jeu. En effet nous avions besoin de déplacements pour faire par la suite pour faire la première carte du jeu, afin d'espacer correctement les obstacles pour que le vaisseau puisse faire des manœuvres, et des tirs pour pouvoir gérer les dégâts et la mort des vaisseaux.

2.1 Déplacement

Dans un premier temps, Arthur a codé les déplacements des vaisseaux. Pour prendre les touches que le joueur entre, on utilise la fonction *Input.GetKey()* de Unity. Nous avons choisi comme touches pour le départ les même contrôles que dans les jeux *Battlefield*.

Les vaisseaux ont trois types de mouvements : le roulis, rotation du vaisseau selon l'axe longitudinale, le tangage , rotation du vaisseau sur son axe transversal, et le lacet, rotation du vaisseau selon l'axe vertical. Les vitesses de rotations s'inspirent aussi des valeurs des avions : le tangage est plus rapide que le roulis qui est bien plus rapide que le lacet. Nous avons aussi choisi de prendre ces valeurs pour des raisons de gameplay. En effet, il est plus facile de tourner avec le lacet plutôt que d'utiliser la combinaison tangage plus roulis. Il est donc logique de rendre cette dernière manœuvre plus rapide en exécution pour récompenser les joueurs les plus talentueux. Nous avons aussi placé la caméra de manière à inciter le joueur à utiliser cette dernière manœuvre : le vaisseau n'est pas représenté au milieu de l'écran mais en bas pour donner plus de visibilité. Toujours dans une optique de réalisme et de difficulté, le vaisseau a de l'inertie, lorsqu'il avance et aussi sur les trois types de mouvements, et il est impossible de reculer avec le vaisseau, il faut faire demi-tour.

Pour avancer, on utilise la touche Z, pour aller vers la gauche on utilise Q, vers

la droite c'est D, pour faire une rotation sur la gauche on utilise la flèche gauche, une rotation vers la droite c'est la flèche droite, et enfin pour aller vers le haut on utilise la flèche bas et vers le bas on utilise la flèche haut. Le haut et le bas sont inversés dans les contrôles pour faire comme dans les jeux utilisant un joystick.

Le vaisseau doit donc avancer et tourner sur lui-même pour parcourir les niveaux. Pour cela, nous utilisions tout d'abord respectivement les fonctions *transform.Translate()* et *transform.Rotate()*. L'inertie du vaisseau était gérée par une variable de déplacement qui augmentait à force d'appuyer sur la touche d'accélération, et diminuait dans le cas contraire.

Cependant, les mouvements n'étaient pas très réalistes et il nous avait été suggéré à la fin de la première soutenance d'utiliser les quaternions que propose Unity pour donner une meilleure inertie et donc des déplacements plus crédibles. Cela fut rajouté dans la semaine qui suivit, grâce aux nombreux tutoriels et documentations trouvables sur internet.

Le problème qui se posa avec les quaternions fut en rapport avec les collisions. En effet avant de les ajouter, nous avions mis le *rigidbody* des vaisseaux, soit l'élément qui gère les collisions avec les rebonds et l'inertie engendrée, à zéro. Dans le cas contraire, dès que le vaisseau touchait un obstacle, il rebondissait et partait en vrille, sans que le joueur ne puisse y remédier.

2.2 Tirs

Le joueur peut tirer grâce à la touche espace, elle aussi détectée par la fonction *Input.GetKey()*. Unity instancie alors une balle qui est stockée dans un prefab grâce à la fonction qui s'appelle, logiquement, *Instantiate()*. La balle part droit devant elle si elle ne touche aucun obstacle, elle disparait d'elle-même au bout de trois secondes avec la fonction *DestroyObject()*, ce qui lui laisse le temps de

traverser la carte, afin qu'il n 'y ait pas trop d'objets en même temps à gérer par Unity, ce qui pourrait ralentir le jeu.

La balle contient un *trigger*, ce qui veut dire qu'elle déclenche un événement lorsqu'elle rencontre un objet qui contient un détecteur particulier. En effet, si un vaisseau détecte une collision avec une balle grâce à la fonction *OnTriggerEnter()*, alors la balle est détruite et le vaisseau perd un point de vie. De plus, si le vaisseau n'a plus de vie, il est alors détruit avec une animation d'explosion. Des sons ont été intégrés à chaque étape : au tir, à la collision et à l'explosion.

3 Gameplay

4 Création de niveaux

5 Interfaces utilisateur

5.1 Les menus

Tout jeu d'aujourd'hui possède un menu, afin de choisir le mode de jeu, si on veut jouer seul ou à plusieurs, pour modifier les options de son, de graphisme ou autre. Unity possède de nombreuses aides pour faire un menu décent, le tout principalement stocké dans la classe GUI (pour *Graphical User Interface*). Ainsi, la fonction *GUI.Button()* crée un bouton qui prend en paramètre le rectangle de sa localisation et ce qu'il contient, soit du texte ou une image. Cette fonction est mis dans une condition, ce qui fait que dès que le joueur clique sur le bouton, on rentre dans la condition et le programme fait les instructions qui suivent.

Cet exemple simple permet de montrer basiquement comment Unity gère ses interfaces. Il y a aussi des instructions pour afficher du texte (*GUI.Label*), d'autres pour faire des curseurs que l'on utilise pour le son (*HorizontalSlider()* ou *VerticalSlider()*) et d'autres que nous n'avons pas utilisées.

En fonction de ses choix dans le menu, le joueur provoque donc l'ouverture de différentes scènes en fonction du mode de jeu qu'il veut. En effet, les modes de jeu sont dans différentes scènes pour faciliter la lecture des scripts. Mais dans le menu des options, le joueur peut aussi modifier les paramètres de la qualité de l'image, du volume du son et peut changer ses touches de contrôle.

Mais en plus, ces paramètres sont sauvegardés par le jeu et ne se réinitialise pas lorsque l'on redémarre le jeu par la suite. Cela est rendu possible grâce à la classe *PlayerPrefs()* de Unity. En effet, cette classe permet de sauvegarder des entier (volume), des chaines de caractères (contrôles) et des flottants (nous n'utilisons pas cette dernière option dans ce projet). On charge ainsi les données de ses sauvegarde au démarrage du jeu pour que le joueur garde ses paramètres habituelles. Cette classe nous permet aussi de sauvegarder le vaisseau qu'a choisi le

joueur de la scène du menu à la scène de la carte. Ainsi quand la partie commence, le jeu regarde quelle valeur est sauvegardée pour attribuer au joueur le vaisseau correspondant.

6 Intelligences artificielles

Dans notre jeu, nous voulions que le joueur puisse jouer seul face à d'autres vaisseaux pilotés par l'ordinateur. Il fallait donc créer une intelligence artificielle, c'est-à-dire un programme qui, en fonction de la situation, va changer de comportement pour donner l'illusion que l'on joue contre un vrai joueur. Nous avons fait ce programme en deux parties : l'algorithme pour éviter les obstacles que l'on a présenté pour la deuxième soutenance, et l'algorithme pour suivre les ennemis que nous avons fait après.

6.1 Éviter les obstacles

C'est Arthur qui s'est occupé de faire l'intelligence artificielle. Dans un premier temps, nous voulions faire un algorithme de *pathfiding*. C'est un algorithme qui permet de calculer la trajectoire la plus courte d'un point A à un point B en évitant les obstacles, et donc permet à une intelligence artificielle de se déplacer facilement.

Nous avons alors recherché des éléments qui seraient intégrés à Unity pour faire ce genre de programme. Nous avons ainsi découvert les *NavMesh* qui permettent de définir une surface sur laquelle un personnage géré une intelligence artificielle peut se déplacer. Cependant, ils étaient inutiles dans notre jeu, car les vaisseaux ne se déplace pas sur une surface comme le ferait un personnage de jeu de rôle. Il semble qu'il n'existe pas d'aide pour faire de la recherche de trajectoire dans Unity qui intègre aussi la troisième dimension qu'est la hauteur. Nous avons donc commencé à le faire nous-même.

Il existe des algorithmes de *pathfiding* qui reposent sur des tableaux contenant le terrain en disant si le personnage peut passer dans telle ou telle case. Cependant c'est assez compliqué à faire en 3D, sans parler du fait que les bâtiments ne sont pas alignés, peuvent être pivotés et qu'il faudrait refaire un tableau pour chaque

carte. Nous avons donc essayé de faire autrement.

On voulait faire en sorte que les vaisseaux ne rentrent pas en collision avec les immeubles ou le sol. Nous avons donc placé à l'avant des vaisseaux pilotés par l'ordinateur des objets invisibles qui servent à détecter les collisions qu'aura le vaisseau s'il ne les évite pas. Ils contiennent des *trigger* qui, dès qu'ils rentrent en collisions, le signal au vaisseau qui ralentit et tourne en conséquence. Il y a 4 objets : un en haut, un en bas, un à gauche et un à droite, tous avancés par rapport au vaisseau. De manière logique, si l'objet en haut détecte une collision, le vaisseau pivote vers le bas, si c'est celui du bas, le vaisseau monte, si c'est l'objet droit qui rencontre un obstacle, le vaisseau tourne à gauche, si c'est le gauche, le vaisseau va à droite.

Nous avons remarqué que les collisions n'étaient pas détectées pour les objets qui avait pour collider un Mesh Collider. Nous avons donc rajouté des Box Collider à tous les bâtiments pour qu'il ne fonce pas dedans, tout en veillant à ce qu'il ne crée pas de collision avec les autres vaisseaux. Le vaisseau peut ainsi slalomer entre les immeubles sans les percuter, mais il ne cherche pas à attaquer les autres vaisseaux.

6.2 Poursuivre les ennemis

Dès le début de la partie, les vaisseaux qui ont une intelligence artificielle contiennent les informations de tous les autres vaisseaux, de sorte à savoir à tout moment leur localisation. Grâce à la géométrie dans l'espace, on calcule des équations de plan par rapport au vaisseau, de sorte à le découper sur les trois axes. Ainsi, grâce à ses équations et aux coordonnées des vaisseaux ennemis, on peut savoir de quel côté du plan ces vaisseaux se trouvent. Ainsi, on peut savoir à tout moment si un vaisseau est derrière ou devant, s'il est à gauche ou à droite, et enfin s'il est au-dessus ou en dessous. En fonction de ces résultat, le vaisseau va donc

poursuivre de préférence un vaisseau qui est devant lui et tournera pour le suivre. Cependant, l'algorithme fait pour éviter les obstacles décrit précédemment est prioritaire, car le principal, avant d'attaquer les adversaires, est de ne pas rentrer dans tous les gratte-ciels qui passent.

7 Multijoueurs

Dans notre jeu, nous voulions absolument qu'il y ait du multijoueur, car le jeu serait ennuyeux si on ne pouvait jouer que contre des ordinateurs. En effet pour un jeu de combat aérien en arène fermée, il fallait pouvoir jouer avec ou contre des amis. Nous avons implémenté deux sortes de multijoueur différents : le multijoueur en écran séparé, et le multijoueur en réseau local.

7.1 Multijoueur en écran séparé

7.2 Multijoueur en réseau

Le multijoueur en réseau LAN (pour Local Area Network) fut commencé après la première soutenance. Le but était de pouvoir jouer à plusieurs sur plusieurs ordinateurs différents reliés par un réseau local. L'avantage par rapport à l'écran séparé est que le joueur a tout l'écran donc peu mieux voir le jeu, et il n'y a pas de problème de répartition de touches de commandes pour les deux joueurs. C'est Arthur qui s'est occupé de cette partie.

Il a d'abord essayé de faire un protocole UDP pour relier en LAN (Local Area Network) en s'inspirant du TP que nous avions fait dans les cours habituels lorsque nous avions travaillé sur le protocole TCP, avant de se rendre compte qu'il existait la classe *Network* sur Unity qui simplifie grandement l'élaboration d'un mode multijoueur sur un jeu. Plusieurs tutoriels existent à ce sujet sur internet, Arthur s'en est donc inspiré pour faire un réseau.

Un des joueurs héberge la partie, et les autres doivent le rejoindre en utilisant son adresse IP locale. A travers le réseau, il fallait envoyer les données des vaisseaux, leurs déplacements, s'ils tiraient et autre. Les coordonnées vaisseaux sont transmises grâce à l'option *Network view* que l'on peut affecter à un objet. Ainsi, tous les joueurs connectés à cette partie voient le vaisseau généré par le joueur

qui se connecte.

L'un des problèmes qui se posa au début, fut que les joueurs ne contrôlaient le vaisseau de l'autre joueur et donc devait regarder sur l'autre écran pour jouer. De plus, à trois joueurs, les deux premiers connectés voyaient un seul et même vaisseau qu'il contrôlaient tous les deux pendant que le troisième joueur en pilotait un autre. Le dernier vaisseau quant à lui n'était contrôlé par personne et restait immobile. Au final, ce problème venait de l'assignation des caméras et des scripts aux joueurs lorsqu'ils instanciaient un nouveau vaisseau en arrivant.

Un autre problème fut que nous n'arrivions pas à faire disparaitre le vaisseau d'un joueur lorsqu'il se déconnectait de la partie en ligne. En effet la fonction *Network.Destroy()* ne s'appliquait pas au prefab tout entier car il ne contenait pas de networkView. Finalement, nous avons intégré cette fonction dans les scripts qui contrôle les déplacements du vaisseau et de la caméra.

Lorsque l'on démarre le mode réseau, l'antivirus des ordinateurs peut bloquer le jeu, ou tout au moins demander l'autorisation à l'utilisateur de laisser libre la connexion. Cela n'est pas une très grande gêne car une fois que l'on désactive les pare-feu, tout revient dans l'ordre, mais nous ne pouvons pas remédier à ce problème définitivement. Cela ne nous empêche cependant pas de jouer.

8 Site internet

Conclusion

Nous avons adorer faire ce projet tout au long du second semestre. Cela nous a apporté beaucoup de choses, notamment des connaissances des logiciels utilisés comme Unity, Blender . . .Mais au delà de l'aspect technique, cela nous a appris à travailler en équipe, à gérer les emplois du temps, à savoir répartir le travail aux différents membres du groupe en fonction de leurs compétences . . .etc.