EMT – INF3A – Atelier

Amongus-Jumper

Rapport de Projet

Table des matières

1 Préambule

Dans le cadre de l'atelier de **31-Programmation OO** enseigné par Jérôme Connus nous devons réaliser un rapport de projet, qui doit être compréhensible par un informaticien ne connaissant pas *Qt*. Il sera par la suite utilisé pour l'évaluation de la maitrise de l'Atelier.

2 Biographie

Je suis Léo Küttel, j'ai 19 ans et je suis né dans le canton de Neuchâtel. J'ai déménagé à l'âge de 4 ans dans le Jura pour y passer près de 15 ans de ma vie. Actuellement, je suis en apprentissage de troisième année d'informaticien d'entreprise. Je suis un grand fan de musique et de cinéma. Je suis de nature curieuse et j'aime bien en apprendre plus sur des sujets du quotidien qu'on a tendance à négliger et qui peuvent pourtant devenir une source d'inspiration très surprenante.

3 But

Le but de ce projet est de réaliser un jeu vidéo avec tout ce qu'il englobe en passant par le développement (C++), jusqu'à la documentation de celui-ci. Cela permettra de tester nos compétences et d'accroitre nos connaissances de programmation orientée objet.

4 Glossaire

Sprite	Élément graphique affiché dans la fenêtre du jeu.
SpriteSheet	Dans le domaine graphique du jeu vidéo, une SpriteSheet est définie comme un fichier d'image bitmap composé de nombreuses images plus petites intégrées dans une formation de grille en mosaïque. Cette combinaison de plusieurs sprites en un seul fichier permet à votre application de lire en une seul et unique fois le fichier, au lieu d'en lire plusieurs à la suite.
Gameplay	Le gameplay ou jouabilité en français est un terme caractérisant des éléments d'une expérience vidéoludique.
Tick	Cadence du jeu.
Plate-former	Un Platformer est un sous-genre du jeux vidéo d'action qui consiste à contrôler un personnage qui doit sauter sur des plateformes dans les airs et éviter des obstacles.
super Footmongus	Personnage incarné par le joueur
Bulio	Antagoniste principal du jeu
Entités	Type de Sprite du jeu ayant des fonctionnalités particulières des autres sprites.
Data	Donnée définit pour différentier un élément du jeu d'un autre.

5 Étapes du projet

5.1 Choix du projet

Quand j'ai appris que nous allions faire dans le prochain bloque d'atelier un jeu vidéo, j'étais aux anges. J'ai tout de suite su ce que je voulais faire, c'est-à-dire un platformer. Dans ma tendre enfance, je passais mes journées à jouer à des jeux comme Mario 64, new super Mario bros, Rayman Origin etc... Tous ces jeux ont contribué à cette envie de faire un jeu du genre pour comprendre l'envers du décor. Mon rêve de réaliser un platformer seul se réalise enfin.

5.2 GameFramework

M. Connus nous a mis à disposition un Game Framework, qui a pour but d'être utilisé comme support de base pour développer notre jeu ainsi que des documentations pour mieux l'appréhender. J'ai rencontré quelques difficultés malgré ça car je n'avais pas suffisamment d'informations utiles à la programmation de mon jeu. Par l'aide de mes camarades et de nombreuses recherches, j'ai su trouver des solutions pour passer outre.

5.3 Implémentation du joueur

Vient ensuite l'élément essentiel au jeu, l'implémentation de ***super footmongus***. Le personnage (géré par la classe `Character`) repose entièrement sur ses déplacements. Plus précisément sur la gestion de la gravité et des collisions avec les éléments du décor. Cette partie du développement fut modifiée et améliorée continuellement durant le projet. Le Joueur se déplace selon sa vélocité, qui est en grande partie gérée par le `Gamecore`. Après une série de test, le joueur est déplacé par le `tick()` du `Gamecore`.

5.4 Entity

Par la suite de mon projet, je me suis rendu compte qu'à part une série de plateformes qui se succédaient, quelques caisses déplaçables et des pièges par-ci par-là, je n'avais rien autre. J'ai donc entrepris d'ajouter des ennemis. Je me suis rendu compte que le mieux était de créer directement une hiérarchie de classe pour faciliter le développement et éviter de la répétition dans mon code. J'ai donc créé la classe `Entity` qui m'a permis de gérer tous les éléments dit « vivants » dans le jeu.

6 Etat du Projet

6.1 Entités

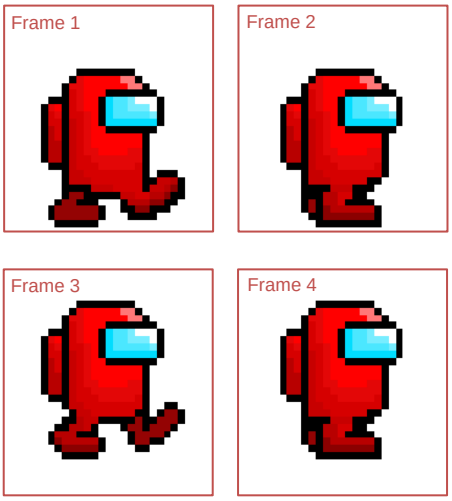
Dans le chapitre Entity, je parle de gérer des éléments vivants. En effet dans ce jeu, les Sprites ayant une vélocité, pouvant être détruit ou étant soumis à la gravité sont considérés comme vivants. Actuellement, il existe trois classes qui découlent de la classe `Entity.` `Les` classes `Bulio`, `Character` et `CaisseAmovible` gérant respectivement les ennemis, le joueur et les caisses en bois.

6.1.1 Animation

Le joueur ainsi que les Bulios ont des animations de déplacement. Ces entités gèrent leurs animations grâce à la fonction `configureAnimation()` qui prend en paramètre un **énuméré** qui permettra de récupérer le SpriteSheet et de l'afficher dans la scène.

6.1.1.1 SpriteSheet

Le SpriteSheet consiste à récupérer une image et à la découper pour afficher des morceaux de celle-ci qui correspondent aux frames du Sprite. Le joueur utilise ce procédé pour ses déplacements latéraux (même chose pour les Bulios). Les autres animations sont de simples images appelées par la même fonction qui ne demandent pas de découpage pour les afficher.



6.2 Gestion des collisions

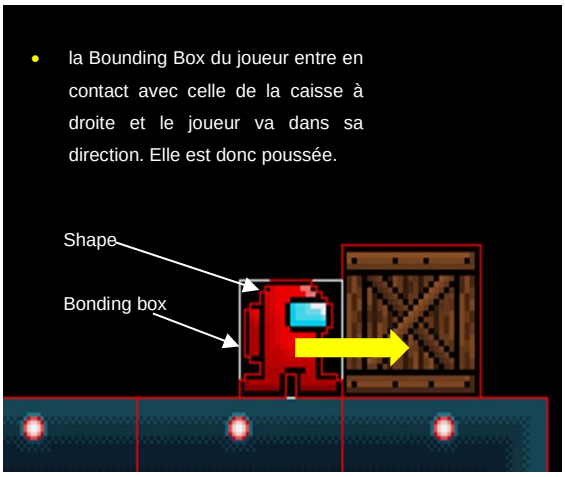
Les collisions sont détectées et utilisées par les **entités** du jeu. La plupart des entités comme les caisses gèrent leur collision par leur `TickHandler()` (qui est activé dans le `tick()` du `gamecore`). Le `TickHandler()` va s'occuper des collisions (selon le type d'entité) par **anticipation** ou par collision **actuelle**.

6.2.1 Collision par anticipation

La collision par anticipation est utilisée essentiellement pour la gestion de la gravité. Effectivement, on en a besoin pour savoir à l'avance à quel moment l'entité devra être attirée par le bas ou non. Pour ce faire, on utilise la vélocité de l'entité pour savoir si elle va entrer en collision avec la scène ou un élément de celle-ci.
On utilise la fonction `nextCollision()` du `TickHandlerEntity`.

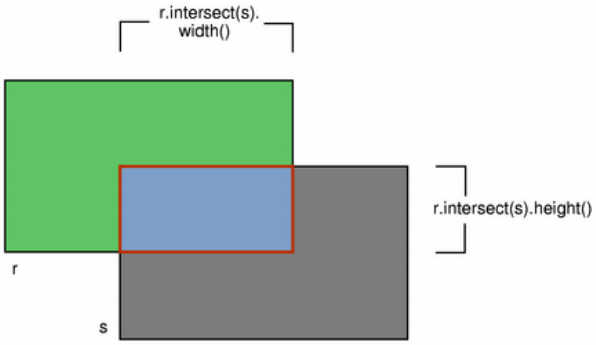
6.2.2 Collision actuelle

Ce type de collision se produit lors d'un contact direct entre l'entité et un autre sprite. Par exemple, pour savoir si une caisse doit se déplacer, on vérifie si le joueur touche la caisse.
Pour cela, on teste la collision avec la **bounding box** *ou Shape selon la précision demandée.*



6.2.3 Collision ciblée

Dans ce jeu, on a besoin de savoir de quel côté une entité entre en contact avec un sprite. Pour ce faire, les entités utilisent la méthode `getCollisionLocate()` qui récupère les valeurs de la fonction `intersected()` mise à dispositions par Qt. Celle-ci renvoie l'intersection entre les deux rectangles donnés. En l'occurrence, on utilise la **bouding Box** des deux sprites pour ensuite localiser la zone en contact.



Tous les sprites ayant une mécanique particulière ont une ***data*** *intégrée qui les différencie.*

6.3 Grounds

En cours de développement, je me suis rendu compte qu'implémenter le sol à la main me faisait perdre inutilement du temps et rendait le code très répétitif. J'ai donc créé une classe `Ground` permettant de générer automatiquement des groupes de sol en précisant simplement l'emplacement du premier bloc, le nombre de colonnes et de lignes. Cette classe s'occupe de la gestion de l'orientation des blocs, en utilisant une sprite sheet. Actuellement, elle ne peut pas générer des figures complexes. Elle fait uniquement des rectangles et des carrés.

7 Problèmes rencontrés

7.1 SpriteSheet

Constatation : La découpe de la sprite sheet ne se faisait pas correctement.

Résolution : Le nombre frames et le nombre de colonnes pour parcourir la sprite sheet étaient inversés. La boucle doit parcourir le nombre de frames et se positionner sur la bonne colonne au moment de la copie.

7.2 Gestion de la gravité

Constatation : L'accélération de chute ne fonctionnait pas.

Résolution : Le calcul fut modifié pour correspondre à celui-ci :

- v = vélocité
- g = gravité
- T = vitesse du tick

$$\mathrm{v} + g \times \frac{T}{100}$$

7.3 Gestion des collisions

7.3.1 Bugs persistants

7.3.1.1 Bug lié au tick

Constatation : Actuellement il existe de légers bugs. Parfois, lorsque le jeu rencontre une perte de ms, l'entité passe à travers les sprites car ils n'ont pas le temps de vérifier s'il touche un sprite.
Supposition : Le bug serait lié par l'optimisation du `tick()` du `gamecore` ou à cause du taux de rafraîchissement de la scène que je n'ai pas eu le temps de modifier pour l'adapter au jeu.

7.3.1.2 Bug lié aux caisses en bois

Constatation : Le joueur peut se déplacer dans les airs avec une caisse s'il continue d'avancer en poussant la caisse vers le vide et qu'il attend pendant une certaine période qui varie de manière encore inconnue, le joueur peut se déplacer dans les airs avec la caisse.
Supposition : Vu que le joueur pousse la caisse, il repositionne la caisse et ralentit sa chute. Comme le joueur se trouve légèrement à l'intérieur de la caisse, il est considéré comme sur le sol ce qui désactive la force de gravité.

7.3.2 Fantôme

Constatation : Le fantôme entre en collision avec les entités.
Résolution : Ajouter une data au fantôme pour que les entités qui collisionnent avec son sprite le reconnaissent et évitent de le considérer comme un sprite quelconque.

7.3.3 Entité

7.3.3.1 Problème 1

Constatation : Quand une entité entrait en collision avec un bloc de sol sur sa surface, il entrait en collision avec les côtés de ce même bloc et ne pouvait pas se déplacer sur les côtés.
Résolution : Il fallait simplement ajouter une intersection minimale lors de la collision ciblée pour éviter de prendre en compte des micro-collisions inutiles.

7.3.3.2 Problème 2

Constatation : Les entités avaient une partie dans le sprite collisionné ce qui bloquait où appelait d'autres fonctions qui ne devaient pas être appelées.
Résolution : l'intersection minimale était trop élevée, la solution était de la diminuer.

7.3.4 Joueur

7.3.4.1 Problème 1

Constatation : Quand le joueur appuie sur la touche de saut, il arrive que parfois il ne puisse pas sauter.
Résolution : Le bug provenait de la nouvelle façon dont le joueur interagissait avec les autres sprites qui ne sont pas définis. Quand un sprite n'a pas une data définie, le joueur le considère comme un sol. Ceux-ci replacent le joueur à moins d'un pixel au-dessus de lui ce qui était considéré comme s'il ne touchait pas de sol. La solution est d'ajouter une marge d'erreur à la vélocité de l'axe Y du joueur, l'autorisant à sauter lors de petites chutes.

8 Améliorations possibles

8.1 Elément de gameplay

- Saut mural
  - Permetterait de rebondir sur les murs pour esacalader les surfaces raides pour ainsi dynamiser le gameplay et rendre les niveaux plus complexes et variés.

- Attaque
  - Mettre à disposition du joueur des attaques. Comme pouvoir tirer ou taper les ennemis. Entre autre donner la possibilité de rajouter des ennemis plus difficiles à battre.
- Possibilité de planer
  - Pour atteindre des platformes hors de portées par un simple saut ce qui faciliterait les mouvements du joueur et qui lui permettrait de se rattrapper lors d'un saut raté.

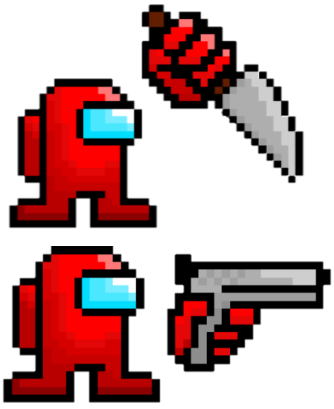


Figure 1 Concepts art des animations d'attaque.

Figure 2 Rayman Origins pouvoir de vol

8.2 Design

Ajouter un fond pour rajouter un décor au jeu ainsi que des éléments de décor au premier plan comme des tuyaux, des circuit etc… pour ajouter une ambiance générale au jeu.

8.3 Niveau

Créer plusieurs niveaux pour donner de la durée de vie au jeu ainsi que la possibilité d'en créer plus facilement comme un interpréteur d'image qui générait un niveau à partir des couleurs de celle-ci.

8.4 Ennemis et pièges

J'imaginais ajouter d'autres types de pièges que de simples piques. Comme de la lave et des laser qui s'activent et se désactivent ainsi qu'un ennemi (le Skrublutarux) se trouvant sur une toile se déplaçant dessus et fonçant sur le joueur quand celui-ci entre en contact avec la toile.

Figure 3 Design du Skrublutarux

9 Conclusion

9.1 Ce que j'ai appris

Durant cet atelier, j'ai pu tester mes capacités à développer une application graphique en C++ en programmation orientée objet, avec la bibliothèque Qt. Par la même occasion, j'ai pu reprendre un projet en cours de route et l'adapter aux besoins du mien par recherche et étude de celui-ci. J'ai dû apprendre à me gérer seul et définir mon projet de A à Z de la première ligne de code jusqu'au dernier point de mes documentations.

9.2 Avis personnel

Cet atelier m'a beaucoup plu. J'ai aimé le faire car la programmation est quelque chose qui m'a toujours intéressé, depuis le début de ma formation. Grâce à celle-ci, j'ai une meilleure vision des choses concernant mon avenir professionnel et je sais où m'orienter plus tard. Je reste cependant sur ma faim car j'ai trouvé que le tutoriel fourni, le **EMT-INF3-CR02_TutorielGameFramework.pdf,** manquait d'informations par rapport à mon projet que je m'étais fixé. Cela a malheureusement ralenti mon avancement. J'espère malgré tout que mon projet pourra être utile pour les prochains apprentis.

10 Annexes

10.1 Supports de développements

Définition (en anglais) de sprite sheet utilisée pour le glossaire :

http://www.spritesheeteditor.com/spritesheet.html

Documentation Qt Creator 4.8 :

https://doc.qt.io/archives/qt-4.8/

10.2 Logiciels

Qt Creator 4.9.1 :

IDE utilisée pour coder le jeu.

Piskel 0.14.0 :

Logiciel de dessin pour le design.

Github Desktop :

Utilisé pour commit le projet sur un dépôt (GitHub).

10.3 Typographie

La typographie utilisée dans cette documentation est adaptée pour faciliter la lecture des dyslexiques.

- Police : Arial
- Taille : 11
- Interligne : 1,5