|  |
| --- |
| EMT – INF3A – Atelier |
| **Amongus-Jumper** |
| **Rapport de Projet** |

Table des matières

[1 Préambule 2](#_Toc93495423)

[2 Biograhpie 2](#_Toc93495424)

[3 But 2](#_Toc93495425)

[4 Glossaire 2](#_Toc93495426)

[5 Étapes du projet 3](#_Toc93495427)

[5.1 Choix du projet 3](#_Toc93495428)

[5.2 GameFrameWork 3](#_Toc93495429)

[5.3 Implémentation du joueur 3](#_Toc93495430)

[5.4 Entity 5](#_Toc93495431)

[6 Etat du Projet 5](#_Toc93495432)

[6.1 Entités 5](#_Toc93495433)

[6.1.1 Animation 5](#_Toc93495434)

[6.2 Gestion des collisions 6](#_Toc93495435)

[6.2.1 Collision par anticipation 6](#_Toc93495436)

[6.2.2 Collision actuelle 6](#_Toc93495437)

[6.2.3 Collision ciblée 6](#_Toc93495438)

[6.3 Grounds 7](#_Toc93495439)

[6.4 Problèmes rencontré 7](#_Toc93495440)

[6.4.1 SpriteSheet 7](#_Toc93495441)

[6.4.2 Gestion de la gravité 7](#_Toc93495442)

[6.4.3 Gestion des collisions 7](#_Toc93495443)

[7 Amélioration possible 8](#_Toc93495444)

[7.1 Design 8](#_Toc93495445)

[7.2 Niveau 8](#_Toc93495446)

[7.3 Ennemie et pièges 8](#_Toc93495447)

[7.4 Element de gameplay 9](#_Toc93495448)

[8 Ce que j’ai appris 10](#_Toc93495449)

[9 Conclusion 10](#_Toc93495450)

[9.1 Avis personnel 10](#_Toc93495451)

[10 Annexes 11](#_Toc93495452)

[10.1 Supports de developments 11](#_Toc93495453)

[10.2 Logiciel 11](#_Toc93495454)

[10.3 Typographie 11](#_Toc93495455)

# 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 connaissance pas *Qt*. Il sera par la suite utilisé pour l’évaluation de la maitrise de l’Atelier.

# Biograhpie

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 au moment où j’écris. 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 pour tant devenir une source d’inspiration très surprenante.

# But

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

# Glossaire

|  |  |
| --- | --- |
| Sprite | Élément graphique affichée dans la fenêtre du jeu. |
| SpriteSheet | Dans le domaine de graphique du jeu vidéo, une SpriteSheet est définie comme un fichier d'image bitmap composé de nombreux image plus petits intégrés dans une formation de grille en mosaïque. Cette combinaison de plusieurs sprites en un seul fichier. Vous permettez de les utiliser pour des animations et diverses autres applications en chargeant simplement un seul fichier. |
| Gameplay | Le gameplay ou jouabilité en français est un terme caractérisant des éléments d'une expérience vidéoludique. |
| Tick et TickHandler |  |
| Programation orienté objet |  |
| Plate-former |  |
| super Footmongus | Personnage incarné par le joueur |
| Bulio | Antagonist principal du jeu |
| Entités | Type de Sprite du jeu est-y-en des fonctionnalités particulières des autres sprites. |
| Énuméré |  |
| Hiérarchie de class |  |
| Data | C’est une donnée |
|  |  |

# Étapes du projet

## Choix du projet

Quand j’ai appris que nous allions faire dans le prochain bloque d’atelier un jeu vidéo, j’étais au ange. J’ai tout suite su se que je voulais faire c’est-à-dire une plate-forme. Dans ma tendre enfance je passais mes journée à joué à des jeux comme Mario 64 , new super Mario bros, Rayman Origin etc.. Tout ces jeux on contribué à cette envie de faire un jeu du genre pour comprendre l’envers du décor. Mon rêve de réaliser un plate-formeur seul, se réalise enfin.

## GameFrameWork

M. Connus nous a mis à disposition un Game Framework, qui a pour bute d’être utilisé comme support de base pour développer notre jeu. Ainsi que des documentations pour mieux l’appréhender. J’ai rencontré quelque difficulté malgré ça car je n’avais pas suffisamment d’information utile à la programmation de mon jeu. Par l’aide de mes camarades et de nombreuse recherche j’ai su trouver des solution pour passer outre.

## Implémentation du joueur

Venue suite, l’élément essential 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 modifié et amélioré continuellement durant le projet. Le Joueur se déplace selon sa vélocité, qui est en grande partie géré par le Gamecore. Après une série de test le joueur est déplacé par le tick() du Gamecore.

## Entity

Par la suite de mon projet je me suis rendu compte qu’a par une série de plateforme qui se succédait, quelque caisse déplaçable et des pièges par-ci par-là, je n’avais rien autre. J’ai donc entrepris d’ajouté des ennemies je me suis rendu compte que le mieux était de créée directement hiérarchie de classe pour faciliter le développement et évité de la répétitions dans mon code. J’ai donc créé la classe Entity qui m’a permis de gérer tous les éléments « vivant » dans le jeu.

# Etat du Projet

## Entités

Dans le chapitre Entity je parle de gérer des éléments dit vivant. En effet dans ce jeu les Spirites est y une vélocité, pouvant être détruit ou soumis à la gravité son considéré comme vivant. Actuellement il existe trois classes qui découlant de la classe Entity la classe Bulio, Character et CaisseAmovible.

### Animation

Le joueur ainsi que les Bulio ont des animations de déplacement. Ces entités géré leur animations grâce à la fonction configureAnimation() qui prend en paramètre un **énuméré** qui permettra de récupéré le SpriteSheet et de l’affiché dans la scène.

#### SpriteSheet

Frame 2

Frame 1

Le SpriteSheet consiste à récupérer une image et de la découpé pour afficher des morceaux de celle-ci, qui correspond aux frames du Sprite. Le joueur utilise ce procédé pour ses déplacements latérale (même chose pour les Bulio). Les autres animations sont des simples images appelé par la même fonction, qui ne demande pas de découpage pour les afficher.

Frame 4

Frame 3

## Gestion des collisions

Les collisions sont détectées et utilisées par les **entités** du jeu. La plus par des entités comme les caisses sont géré leur collision par leur TickHandler (qui est activé dans le tick du gamecore). Le TickHandler va géré les collision (selon le type d’entité) par **anticipation** ou par collision **actuel**.

### Collision par anticipation

Les collision par anticipation est utilisé essentiellement pour la gestion de la gravité. Effectivement, on en a besoin pour savoir à l’avance à quel moment l’entité devras être attiré par le bas ou non pour se 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.

### Collision actuelle



Le joueur va à droiteddewe

* la Bounding Box du joueur entre en contact avec celle de la caisse à droite et le joueur va dans sa direction. Elle est donc poussée.

Bonding box

Shape

Ce type de collision sont directement en contact avec 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 test la collision avec la* **bounding box** *ou Shape selon la précision demandée.*

### Collision ciblée

Dans ce jeu on a besoin de savoir de quel côté une entité entre en contact avec un sprite. Pour se faire les entité utilise la méthode getCollisionLocate().

Qui récupère les valeurs de la fonction intersected() mise à dispositions par Qt. Celle-ci renvoie l'intersection de ce rectangle et du rectangle donné. En l’occurrence on utilise la **bouding Box** des deux sprites pour ensuite localisé la zone en contact.

*Tous les sprites est-y-en une mécanique particulière ont une* ***data*** *intégrer qui les différencies.*

## Grounds

En cours de développement je me suis rendu compte qu’implémenté 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 précisent simplement l’emplacement du premier bloc, le nombre de colonne et de ligne. Cette classe s’occupe de la gestion de leur orientation des blocs, en utilisant une sprite sheet. Actuellement elle ne peut pas générer des figures complexe. Elle fait uniquement des rectangles et des carrés.

# Problèmes rencontré

## SpriteSheet

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

**Résolution :** Le nombre frame et le nombres de colonne, pour parcourir la sprite sheet était inversé. La boucle doit parcourir le nombre de frame et se positionner sur la bonne colonne au moment de la copie.

## Gestion de la gravité

**Constatation :** L’accélération de chute ne fonctionnait pas.

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

* = vélocité
* = gravité
* = vitesse du tick

## Gestion des collisions

### Fantôme

**Constatation :** Le fantôme entre en collision avec les entités.

**Résolution :** Ajouté une data au fantôme pour que les entités qui collisionne avec elle. Pour

### Entité

#### Problème 1

**Constatation :** Quand une entité entrait en collision avec un blocs 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-collision inutile.

#### Problème 2

**Constatation :** Les entités avaient une partie dans le sprite collisionné ce qui bloquait où appeler d’autre fonction qui ne devrait pas être appelé.

**Résolution :** l’intersection minimale était trop élevée, la solution était de la diminuer.

### Joueur

#### 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 définie. Qu’en un sprite n’a pas une data définit 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 si il ne touchait pas de sol. La solution est d’ajouté une marge d’erreur la vélocité de l’axe Y du joueur. L’autorisant à sauter, lors de petites chutes.

# Amélioration possible

## Element de gameplay

* Une image contenant ciel, roue à vent, objet d’extérieur

  Description générée automatiquementSaut mural
  + Permetterai de rebondir sur les murs pour esacalader les surface raide. Ainsi dynamisé le gameplay et de rendre les niveaux plus complexes et varié.
* Attaque

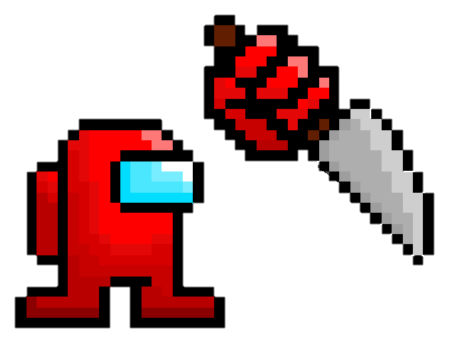
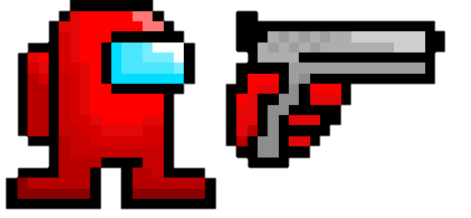


Figure 2 Concepts art des animations d’attaque.

* + Mettre à disposition du joueur des attaques. Comme pouvoir tirer ou taper les ennemies. Entre autre donné la possibilité de rajouté des ennemies plus difficile à battre.
* Possibilité Planer
  + Pour atteindre des platforme hort de porté pour un simple saut . Ce qui faciliterai les mouvements du joueur qui lui permettrai de se ratrappé lors de saut raté.



Figure 3 Rayman Origins pouvoir de vol

## Design

Ajouter un fond pour rajouter un décore au jeu.

## Niveau

Créé plusieurs niveaux pour donner de la durée de vie au jeu. Ainsi que la possibilité d’en créer plus facilement comme interpréteur d’image qui générai un niveau à partir des couleurs de celle-ci.

## Ennemie et pièges

J’imaginais ajouté d’autre type de piège que des simples piques. Comme de la lave et des laser qui s’active et se désactive. Ainsi qu’un ennemie (le Skrublutarux) se trouvant sur une toile, qui se déplacer dessus. Et fonce sur le joueur quand celui-ci entre en contact avec la toile.



Figure 1 Design du Skrublutarux

# Conclusion

## Ce que j’ai appris

Durant cet atelier j’ai pu tester mes capacités à développer une application graphique en C++ en orienté-objet, avec la bibliothèque Qt. Par la même occasion j’ai pu reprendre un projet en cours de route et l’adapté aux besoins du miens, 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 côté jusqu’au dernier point de mes documentation.

## Avis personnel

# Annexes

## Supports de developments

Definition (en anglais) de sprite sheet utilisé pour le glossaire :

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

## Logiciel

* 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).

## Typographie

La typographie utilisée dans cette documentations est adapté pour faciliter la lecture des dyslexiques.

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