Plat-formeur

Rapport de Projet

# Préambule

Dans le cadre de l’atelier de **31-Programmation OO**enseigné par Jérôme Connus nous devons réaliser un 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 troisièmes 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.

# But

Le bute 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. |
| Qt | Logiciel utilisé pour la programation |
| Piskel | Logiciel de dessin en spécilisé dans le pixel art. |
| Tick et TickHandler |  |
| Programation orienté objet |  |
| Plate-former |  |
| super Footmongus | Personnage incarné par le joueur |
| Bulio | Antagoniste principal du jeu |
| Entités | Type de Sprite du jeu est-y-en des fonctionnalités particulières des autres sprites. |
| **Énuméré** |  |
| **Hirarchie de class** |  |
| data |  |
|  |  |

# 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 un 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 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.

## Grounds

Pour la création du sol en cours de développement je me suis rendu compte que implémenté le sol à la main premet inutiliement de la place dans le code et était très répétitif jai donc créée une classe Ground premettant de généré automatiquement des blocs de sol en précisent juste l’emplacement du premier bloque et le nombre de colonne et de ligne.

## 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

## Etat des 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 tout les classe découlant de la classe entité utilise toutes ces fonctionnalités.

### 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

Les collisions actuelles sont les collisions qui sont directement en contact avec l’entité. Par exemple pour savoir si une caisse doit se déplacer, On vérifie s’il touche la caisse et si oui de quel côté le joueur la touche. Pour cela on test la collision avec la BoundingBox ou Shape selon la précision demandé.

### Collision ciblée

Dans ce jeu on a besoin de savoir de quel côté une entité entre en contact avec un sprite, ainsi que son type.

Pour se faire tout les sprites est-y-en une mécanique particulière ont une **data** intégrer pour pouvoir les différencies.

## Problèmes rencontré

### SpriteSheet

### Gestion de la gravité

### Gestion des collisions

#### Entité

#### Enemie

#### Caisse

#### Joueur

# Amélioration possible

## Disigne

## Niveau

## Gestion des collision

#### Enemie

#### Caisse

## Element de gameplay

# Ce que j’ai appris

# Conclusion

## Avis personnel



* 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.

Shape

Bonding box

Le joueur va à droiteddewe

# 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