# Documentation du Projet de Moteurs de jeux

## M2 IMAGINA

**Jérémy DAFFIX**
**Arnault LEMMEL**
**Fatma MILADI**

https://github.com/jeremydaffix/hmin317-projet

**Mise en route**

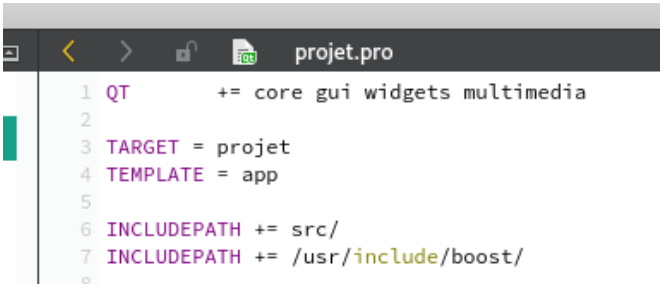
Pour notre projet, nous avons utilisé l'IDE QT Creator sous GNU/Linux.

En plus des bibliothèques nécessaires à la création d'une application OpenGL de base avec QT, nous avons utilisé le module *multimedia* de QT (pour jouer les sons et la musique, avec QMediaPlayer pour la musique, et QSoundEffect pour les effets sonores "basse latence").

*-> Il faudra donc installer, pour pouvoir compiler et tester, les packages ou les fichiers de développement nécessaires au module multimedia de QT (peut dépendre selon le système d'exploitation / la distribution Linux).*

De plus, nous avons intégré une (petite) bibliothèque externe pour la partie pathfinding, qui nécessite Boost, une collection de bibliothèques C++ générique et très utilisée.

*-> Même chose, il faudra également installer les packages et / ou fichiers de développement nécessaires sur votre système.*
*De plus, il sera probablement nécessaire de modifier le fichier projet.pro pour que le compilateur trouve les headers et bibliothèques statiques Boost dont il a besoin.*
*En particulier, la variable INCLUDEPATH pour les headers, et éventuellement la variable LIBS pour les bibliothèques statiques (nous n'en avons pas eu besoin sur Linux, mais cela pourrait être nécessaire sur Windows). Exemple :*




**Liste des ressources utilisées**

Sprites (tiles et personnages animés) :

http://www.reinerstilesets.de/graphics/2d-grafiken/2d-humans/ (public domain)
https://kenney.nl/assets/medieval-rts (CC0 1.0 Universal)

Sons & Musiques :

https://opengameart.org/content/battle-sound-effects (CC0)
https://opengameart.org/content/battlecry (CC By 3.0)
https://opengameart.org/content/rpg-sound-pack (CC0)
https://opengameart.org/content/impact (CC0)
https://opengameart.org/content/ghost (CC0)
https://opengameart.org/content/rumbleexplosion (CC By 3.0)

https://opengameart.org/content/battle-theme-a (CC0)

Code :

https://github.com/quantumelixir/pathfinding (bibliothèque de pathfinding très légère)

**Rappel du Gameplay**

Chaque joueur est d'un côté de la map, et possède 2 bâtiments qui génèrent des soldats toutes les 5 secondes.
Le joueur peut bouger ses "targets" (une par bâtiment) case par case pour changer la destination des soldats générés, qui ne se déplacent plus une fois arrivés à leur destination initiale (mais peuvent toujours se battre).
Il peut également transformer chaque bâtiment (en archerie, cabane de fées ou caserne) pour changer le type d'unités générées (chaque type ayant un bonus d'attaque contre un autre).

Les fantassins sont forts contre les archers, les archers sont forts contre les fées, les fées sont fortes contre les fantassins.
Lorsque deux soldats ennemis se croisent, ils se battent, puis l'éventuel survivant reprend sa route.
Déplacements : utilisation d'un algorithme A* sur un tableau de cases pour définir un trajet en évitant les obstacles (arbres, buissons,...).
Lorsqu'un autre soldat allié se trouve sur son chemin, on s'arrête (création de "files").
Possibilités stratégiques : accumuler les soldats en défense, attaquer avec le bon type d'unités pour percer les défenses ennemies, profiter d'un bâtiment pas assez protégé, etc.
Il est possible de créer ses propres maps grâce à un fichier JSON, ce qui peut également influer sur les choix stratégiques (par exemple, l'accès à un bâtiment peut être difficile à cause des obstacles de la carte).

But = détruire un des bâtiments ennemis.

Contrôles :

Touches fléchées : bouger Target bâtiment 1 (case par case).
ZQSD : bouger Target bâtiment 2.
NumPad 1 et 2 : bâtiment suivant.

**Fichiers importants du Moteur de jeu**

Répertoire src/engine/.
Fichiers : nom de la classe + .cpp / .h.

- GameObject : objet de base du graphe de scène, contient notamment les méthodes à redéfinir et les transformations de repère.
- GameScene : root object du graphe de scène (singleton). Calcul de la pose redéfini pour simuler une caméra.
- Model3D : classe (abstraite) d'un GameObject affichable.
- Game : classe générique d'un jeu (abstraite).
- Sprite : objet sprite (Plane avec une texture et des constructeurs 2D).
- Component : "interface" pour les composants attachables aux GameObject. Méthodes à redéfinir par les enfants : update() et fixedUpdate().
- SpriteAnimationComponent : composant d'animation. On ajoute une liste de textures et on les fait défiler à la vitesse donnée, sur le Sprite auquel est associé le component.
- ResourcesManager : chargement / récupération des ressources (sons, textures,...). Méthodes pour charger une ressource dans un "dictionnaire", et méthodes pour les récupérer à partir de leur nom.

**Fichiers importants du Jeu**

Répertoire src/game/.

- ImaginaWars : hérite de Game (~ MainWidget), initialisation (jeu, shaders, textures, sons,...), création objets,... Événements clavier et souris.
- Soldier : classe d'un soldat, hérite de Sprite, est associé à des components pour IA et gestion vie.
- BuildingComponent : création unités, gestion vie et destruction des bâtiments.
- GameMap : GameObject, chargement (JSON) et création map. Gestion des obstacles. Wrap la lib de pathfinding.
- GamePlayer : joueur (humain ou IA) + components.
- SoldierAIComponent : IA du soldat, partie combat.
- WalkPathFindingComponent : pathfinding et mouvement du soldat.
- EnemyAIComponent : IA du joueur PC (simple).

Petit script de génération de code :

tools/generate-component.sh

Shaders :

resources/shaders/fshader_dark.glsl pour assombrir la map
resources/shaders/fshader_toon.glsl pour "simplifier" les couleurs des soldats et supprimer leurs ombres

Maps (format JSON) :

resources/game/

Textures et images :

resources/textures/

Sons et musiques :

resources/sounds/

Nous avons essayé de suffisamment commenter le code : dans les .h pour les explications générales, et à l'intérieur des .cpp pour les choses plus contextuelles.