

Ecocity



Un Jeu par :
Stein Nicolas & Roussarie Thomas

Description de l'application

Pour illustrer le thème du projet de C++ (there is no planet B), nous avons décidé de coder un jeu de type city builder (comme SimCity ou City Skyline). Le but principal étant de créer une ville la plus écologique possible.



Dans cette optique, nous avons eu pour objectif que ce jeu soit un jeu de gestion, où nous pourrons retrouver la gestion des différentes zones (résidentielle, commerciale, industrielle). Ainsi que la gestion de l'argent, de l'électricité, de l'eau, des habitants, des demandes et surtout de la pollution.

Pour rentrer un peu plus dans les détails, notre jeu commence sur un écran d'accueil où l'on peut régler quelques options. Une fois la partie commencée nous retrouvons l'interface du jeu avec notamment en bas dans le premier onglet, les valeurs indispensables à tout jeu de gestion, c'est à dire l'argent disponible, le nombre d'habitants, la demande résidentielle, commerciale et industrielle, mais encore la consommation électrique et la pollution. Dans le second onglet nous retrouvons les constructions qui se répartissent en différentes catégories. Premièrement nous avons les routes, leur placement permet par la suite de placer les zones qui sont très importantes pour la gestion. Et enfin des bâtiments utilitaires, pour l'électricité par exemple. (Pour toute aide concernant le jeu se référer au "Guide de jeu")



Et le plus important au milieu, votre terrain où vous pourrez construire votre ville, générer de manière aléatoire de façon à ce que chaque expérience soit unique. Ce terrain alternera entre prairies verdoyantes, montagnes, lacs, rivières ou même océan avec ses magnifiques plages.

Vous retrouverez bien évidemment différents types de résidences, d'industries, ou de commerces choisis de manière aléatoire et en fonction de leur taille, générant chacun un certain nombre de place habitable ou bien d'emplois.

Il faudra bien évidemment gérer au mieux les demandes et les consommations des habitants sous peine de banqueroute ou de perte de résidents qui irait vivre dans une ville mieux entretenu. Bien évidemment la pollution doit aussi être limitée au maximum en jonglant entre les énergies renouvelables et les énergies fossiles.

Il faudra savoir prendre son temps afin d'aménager sa ville, mais pas de panique de belles musiques seront là pour vous accompagner dans votre quête de la ville parfaite.

Le code du projet est disponible sur Github à l'adresse suivante:



<https://github.com/nicolas-stein/EcoCity>

Mise en valeur de l'utilisation des contraintes

8 classes

Notre jeu est plutôt gros, il contient de nombreuses lignes de code, et par conséquent bien plus de 8 classes. On n'a donc largement eu l'occasion de s'entraîner à la réalisation de tous types de classe.

3 niveaux de hiérarchie

De même, notre jeu est organisé sur bien plus de 3 niveaux de hiérarchie. Étant donné le grand nombre d'éléments à gérer, il est normal d'avoir des hierarchie sur beaucoup de niveaux. Néanmoins cela permet d'avoir un diagramme UML bien plus compréhensible.

2 fonctions virtuelles différentes et utilisé à bon escient

Les fonctions virtuelles sont plutôt utiles ici en effet dû au grand nombre de niveaux de hiérarchie, il est pratique de pouvoir adapter certaines méthodes dans des classes enfant très éloignées du parent.

2 conteneurs différents de la STL

Nous avons utilisé Qt pour créer notre jeu, par conséquent nous n'avons pas utilisé directement de conteneurs de la STL mais des conteneurs de Qt qui s'en apparentent.
(Ex : QList au lieu de std::vector, QMap au lieu de std::map)

Diagramme de classe UML complet

Comme on peut le voir plus bas, le diagramme UML permet d'avoir une bonne vue du code pour mieux comprendre son fonctionnement et son organisation. Ce diagramme est à la fois complet et complexe dû au grand nombre de classes et de méthodes

Commentaire du code

Afin de pouvoir se retrouver dans le code un minimum de commentaire est impératif, de plus ils pourront permettre à quelqu'un qui ne connaît pas le code de mieux comprendre son fonctionnement.

Pas d'erreurs avec Valgrind

Évidemment il faut que le code puisse s'exécuter sans polluer la mémoire c'est pour cela que le code ne doit pas avoir d'erreur avec valgrind, ce qui est le cas.

A noter que valgrind peut signaler des potentielles fuites mémoire mais celles-ci sont dues aux modules Qt et non pas à notre programme et ne sont donc pas correctible (on ne modifie pas le code source de Qt)

Rendu par dépôt git, adresse à envoyer par mail avec dans le sujet le motif

[EISE/MAIN 4 C++ Projet]

Comme précisé plus haut le lien du Github est <https://github.com/nicolas-stein/EcoCity>, le dépôt git permet de gérer les versions de son code pour toujours en garder une trace.

Pas de méthodes/fonctions de plus de 30 lignes (hors commentaires, lignes vides et assert)

Les méthodes et fonctions utilisées ont été faites de sorte à ne pas être trop longues afin d'avoir une meilleure compréhension code, néanmoins certaines fonctions peuvent être un peu plus longues car cela n'aurait pas de sens de les raccourcir et que cela complexifierait encore plus le code, notamment pour initialisation les textures.

Utilisation d'un Makefile avec une règle "all" et une règle "clean" ou autre outil de build automatique

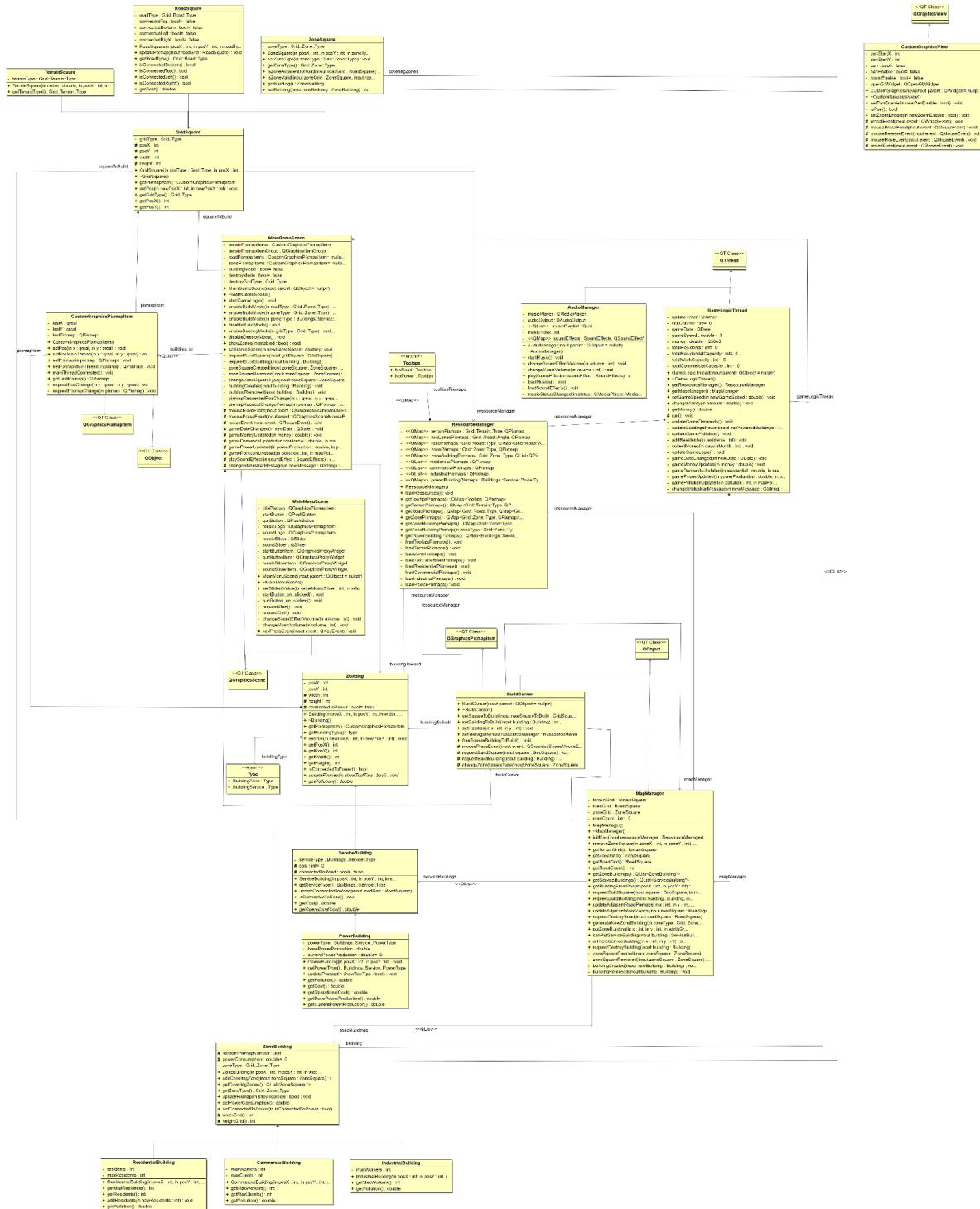
Évidemment il faut pouvoir compiler le code, ce qui se fait très simplement sur QT, et néanmoins nous avons mis à disposition toute les ressources pour pouvoir construire l'exécutable (script make, bibliothèques...), voir le ReadMe du dépôt git.

Utilisation de tests unitaires

Sur un jeu tel que le nôtre, les tests unitaires sont assez difficile à réaliser car nous ne pouvons pas vraiment tester le visuel, nous sommes donc obligés de tester le jeu par nous même pour voir si tout fonctionne bien, et si il n'est pas trop dur ou trop facile.

Diagramme UML de l'application

Voici le diagramme UML du jeu, étant donné sa taille pour pouvoir observer plus précisément les différentes classes il est nécessaire d'ouvrir UML EcoCity.png .



Procédure d'installation (bibliothèques ...) et d'exécution du code

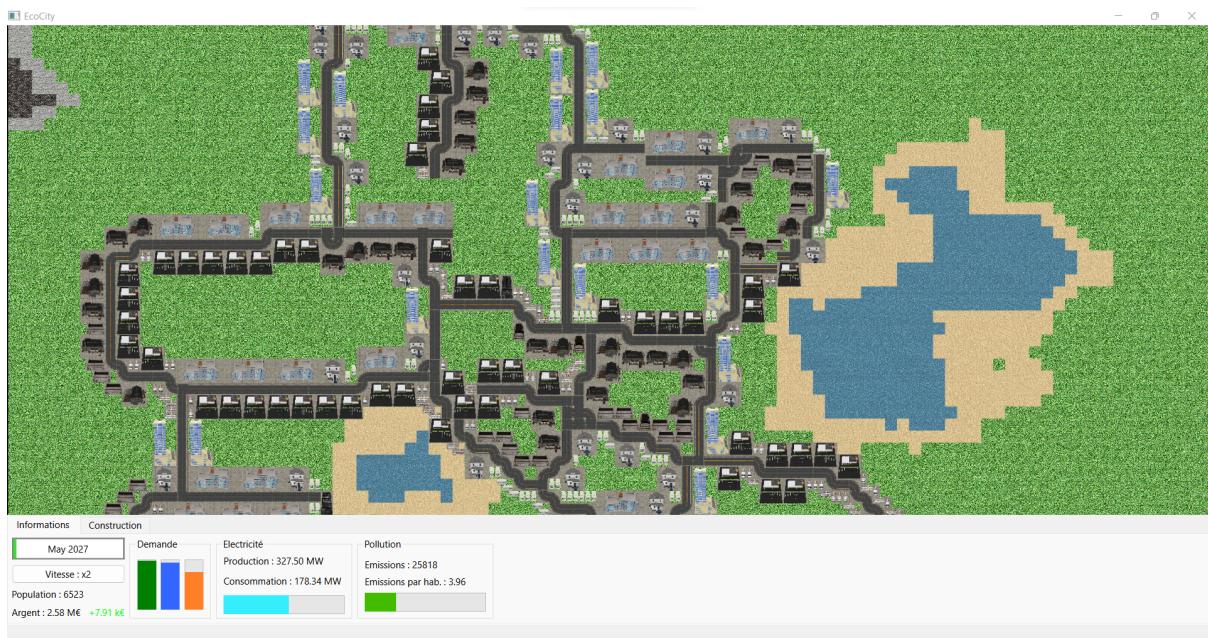
La procédure complète pour la compilation et l'installation du jeu est disponible dans le readme du [dépôt Github](#).

Les parties de l'implémentation dont vous êtes les plus fiers.

Le jeu est composé de beaucoup de classe, il y a donc beaucoup de choses à gérer en même temps, tout ceci conduit à une perte de fluidité, mais nous avons su contrer tout ceci, le jeu fonctionne donc très bien peu importe la situation. En utilisant différents thread, notamment un pour la partie graphique et un pour la gestion de la logique du jeu, et différentes techniques de cache intégrées à Qt, on peut maintenir un nombre d'images par secondes très plaisant.

De plus, chaque bâtiment, maison, magasin, industrie ou autre n'est pas là que pour la beauté, par exemple chaque maison ou immeuble a son propre nombre d'habitant qui peut évoluer au fil du temps. D'un autre côté équilibrer un jeu comme celui-ci est plutôt compliqué mais nous avons réussi à faire en sorte qu'il ne soit ni trop dur ni trop compliqué. De plus, avec toutes les variables prises en compte, notre jeu se rapproche beaucoup d'un jeu de gestion classique. Même les graphismes sont travaillés de sorte à ne pas être trop répétitif et plus réaliste que du pixel art, trouver des modèles 3D pour chaque bâtiment fut compliqué.

Nous pouvons dire que le jeu est très facile à prendre en main, et nous sommes fiers du résultat final.



Le but à atteindre.