Projet final - POO C++ - Synthèse d'Images 2

IMACRUN 3D

À rendre avant le : [À DÉFINIR] (veille de la soutenance)

Contacts : sylvain.cherrier@univ-eiffel.fr/ venceslas.biri@univ-eiffel.fr/ vincent.nozick@univ-eiffel.fr/ clement.chomicki@univ-eiffel.fr

0 Introduction

L'objectif de ce projet est de développer un jeu de type « Temple Run » en 3D en C++/OpenGL 3+. Voici un exemple de jeu jouable en ligne :

https://www.plonga.com/game/Temple-Run-2-Online

Capture d'écran du jeu Temple Run 2 (Imangi Studios - 2013).

La suite de ce document donne les spécifications nécessaires au développement. Comme vous le verrez, certains points sont précis et devront donc être respectés scrupuleusement, d'autres vous laissent totalement libres, et d'autres sont un peu ambigus... Et oui ! C'est souvent le problème quand on développe un programme pour un client. « Welcome to the real world ! »

1 Règles du jeu

Vous incarnez un explorateur qui a volé un trésor dans un temple et qui tente d'échapper aux singes démoniaques le poursuivant pour le dévorer et récupérer le trésor.

Dans le jeu original, le parcours est infini et le jeu se termine lorsque vous tombez du temple ou lorsque les singes-démons vous ont rattrapé. Le chemin est semé d'embûches ! Attention aux obstacles, aux pièges, aux trous... L'objectif est de ramasser un maximum de pièces pour obtenir le plus de points possible.

Pour (un peu) plus d'infos : https://en.wikipedia.org/wiki/Temple_Run

Plateau de jeu

Le plateau de jeu est constitué d'éléments statiques :
— le sol, les murs et les obstacles du parcours ;
— les pièces (de différentes valeurs) ;
— le décor ;
— des bonus, si vous souhaitez en intégrer,
et d'éléments dynamiques :
— l'explorateur, guidé par le joueur ;
— les singes-démons qui suivent et essayent de rattraper l'explorateur.

Début de partie

Au début d'une partie, le joueur a 0 point. La position initiale de l'explorateur et des différents éléments est précisée dans un fichier de description du niveau.

Déroulement d'une partie

Les déplacements de l'explorateur s'effectuent latéralement (de gauche à droite) sur le chemin du parcours et sont contrôlés par le joueur. Tout au long du parcours, l'explorateur croisera des trous (par dessus lesquels il devra sauter), des obstacles (qu'il devra éviter), des arches (sous lesquelles il devra passer)...

Les singes-démons suivent l'explorateur en permanence en essayant de le rattraper.

Lorsque l'explorateur passe sur une pièce, celle-ci disparaît et le score du joueur est augmenté de la valeur de la pièce (que vous déterminerez).

Si l'explorateur tombe du parcours ou s'il est rattrapé par les singes-démons (lorsqu'il n'évite pas un obstacle), la partie se termine et le score est sauvegardé (s'il fait partie des X meilleurs).

2 Spécifications

Nom du jeu

Si vous n'aimez pas « IMACRUN 3D », n'hésitez pas à nommer votre jeu autrement !

Le parcours

Le parcours est représenté par une grille dont chaque cellule est identifiée par un système de coordonnées classique. Une cellule contient un unique élément du plateau (L'explorateur, les singes, les murs, le sol, le vide, *etc.*)

Dans le jeu initial, le parcours est infini. Ici, vous créerez un parcours fini (*i.e.* avec un départ et une arrivée). Une fois cette version fonctionnelle, vous pourrez éventuellement travailler sur une génération aléatoire d'un parcours infini (*cf.* Section 3).

Il doit être décrit à l'aide d'un fichier (chargement et sauvegarde). Le format de fichier n'est pas imposé. Par exemple, vous pouvez utiliser JSON, XML, ou encore un format binaire créé par vos soins.

Fenêtre de jeu

La fenêtre de jeu est constituée d'au moins deux éléments principaux :
— le champ de jeu (le parcours, *etc.*) ;
— les informations relatives à la partie en cours (score actuel, *etc.*).
Pour afficher du texte sur la fenêtre de jeu, vous pourrez utiliser SDL_ttf. Voici une page d'OpenClassrooms pour vous aider : ici.

Menu

Lorsque le joueur appuie sur Echap, le jeu est mis en pause et un menu pop-up s'affiche. Ce menu permet de recommencer le niveau et de consulter les meilleurs scores. **Pour le(s) groupe(s) de 4 :** Le menu doit aussi permettre de sauvegarder la partie en cours et charger une partie sauvegardée.

Déplacement des personnages

L'explorateur avance à vitesse constante. En ligne droite, il peut se déplacer latéralement sans pouvoir traverser les murs du parcours. Il peut sauter pour éviter les trous et se baisser pour éviter les obstacles en hauteur. En cas de virage, l'explorateur peut aussi tourner (il est fort hein ?) Le joueur contrôle sa direction à l'aide des touches :
— Z pour sauter ;
— Q pour se déplacer ou tourner à gauche ;
— S pour se baisser ;
— D pour se déplacer ou tourner à droite.
Vous devrez donc déterminer à tout moment si le déplacement demandé par l'utilisateur est possible (*i.e.* s'il n'y a pas de mur/d'obstacles sur le passage).

Les singes-démons se déplacent tout seuls en suivant l'explorateur en permanence.

Contrôle de la caméra

Le jeu doit contenir deux types de caméra :
— une caméra centrée sur l'explorateur, qui permet de tourner autour et de zoomer à l'aide de la souris ;
— une caméra permettant de voir à travers les yeux de l'explorateur et offrant la possibilité de tourner la tête à l'aide de la souris (attention, nous ne sommes pas dans l'Exorciste, les déplacements de la tête doivent être limités).

À tout moment, le joueur peut changer de vue en appuyant sur la touche C et bloquer la caméra dans une configuration idéale pour jouer avec la touche L.

Apparence du jeu

Vous êtes libres de choisir le style général du jeu. Par exemple, les murs peuvent être des haies, l'explorateur un chat, les singes démons des chiens, et les pièces des souris. Pour charger un modèle 3D (par exemple un .obj), ne codez pas la lecture du fichier, utilisez une bibliothèque telle que Assimp (http://www.assimp.org/).

Nous n'attendons pas que le personnage (quel qu'il soit) soit animé.

Vos seules obligations sont :
- utiliser plusieurs textures ;
- utiliser plusieurs lumières (par exemple, les pièces peuvent produire de la lumière) ;
- la scène doit être englobée dans une boîte texturée de votre choix (*e.g.* un ciel étoilé) ;
- le jeu doit être jouable sans faire perdre 5 points à chaque œil du joueur :-).

3 Extensions possibles

Une fois l'ensemble des fonctionnalités implémentées, testées et validées, vous pouvez améliorer votre programme en proposant par exemple (attention, certaines extensions sont beaucoup plus complexes que d'autres) :
- plusieurs niveaux de difficultés (*e.g.* plusieurs vitesses de course) ;
- ajout de bonus sur le parcours (*e.g.* ralentissement temporaire de la course, points doubles) ;
- génération aléatoire d'un parcours infini (avec augmentation constante de la vitesse) ;
- niveaux sur plusieurs étages, accessibles par des échelles (ou autres) ;
- sauvegarde de la partie en cours et chargement d'une partie sauvegardée (**uniquement pour le(s) groupe(s) de 4**) ;
- animation des éléments visuels (*e.g.* les pièces tournent sur elles-mêmes) ou du personnage ;
- ajout d'effets visuels (*e.g.* du brouillard, « trainée » de particules suivant les déplacements...) ;
- rendu des ombres en utilisant des shadow maps (attention ce n'est pas trivial).

Si vous avez d'autres idées, surtout, n'hésitez pas ! Par contre, pensez à nous soumettre votre idée avant de vous y mettre. Ce serait dommage de travailler sur quelque chose que nous trouvons sans intérêt...

4 Contraintes techniques

- Groupe de 3 ou 4 étudiants
- Programmation en C++
- Rendu avec OpenGL 3+ (utilisez les shaders, pas de pipeline fixe)
- Fonctionne au moins sous Linux : programme qui compile avec `g++` et exécutable sur les machines de la fac (typiquement celles de la salle 1B110N)
- Gestion de la fenêtre et des évènements avec la SDL
- Bonne architecture logicielle (*e.g.* séparez bien le côté « Moteur de jeu » du côté « Moteur de rendu »)
- Utiliser un outil de gestion de versions (type Git) pour partager votre code.

5 Spécifications liées au C++

Ce projet est à réaliser en C++. Il est largement conseillé d'utiliser une version moderne du C++ (11, 14 ou 17).

5.1 Compilation

Vous devrez fournir les fichiers permettant de gérer la compilation avec CMake (au moins la version 3.13). Vous devrez également spécifier les options `-W` , `-Wall` et `-Werror` pour compiler votre projet.

5.2 Documentation

Nous attendons un code lisible, bien organisé et auto-documenté. Bien organisé signifie que vous répartirez judicieusement votre code sur plusieurs fichiers, et que les fonctions ou classes composants chaque fichier seront elles aussi bien réparties. Par ailleurs, il est nécessaire de commenter votre code en gardant à l'esprit qu'il doit être compréhensible même sans ces commentaires : essayez de nommer vos variables et vos fonctions explicitement (utilisez '`objectCount`' au lieu de '`n`' par exemple) et n'hésitez pas découper de longs algorithmes en plus petites fonctions.

5.3 Côté technique

Voici ce qu'il serait bon de rencontrer dans votre projet :
- classes
- classes et fonctions template
- polymorphisme
- encapsulation
- usage massif d'outils de la STL
- espaces de nommage
- exceptions pour traiter les erreurs
- messages d'erreurs pour traiter les erreurs utilisateurs (fichiers inexistants, entrées invalides, etc.)
- asserts pour détecter et prévenir les erreurs de programmation (division par zéro, valeur nulle non attendue, etc.)
- fonctions lambdas
- héritage (seulement dans les cas où c'est justifié)
- (bonus) des fonctions et des variables constexpr
- (bonus) fonctions variadics
- (bonus) code documenté avec Doxygen

Dans votre rapport, vous fournirez un tableau décrivant pour chacun de ces points (et d'autres le cas échéant) si vous les utilisez, et si oui, dans quels fichiers.

Voici ce qu'il serait regrettable de rencontrer dans votre projet :
- une mauvaise organisation de vos fichiers et de vos classes
- de la redondance de code
- des variables, des fonctions ou des types mal nommés
- une utilisation inappropriée de `auto`
- des références non constantes alors qu'elles devraient l'être
- des passages par copies non justifiés
- des erreurs de segmentation
- des fuites de mémoires

— des bugs
— du code de debug non retiré
— du code mort (= des portions de code non utilisés par le programme)
— du code illisible d'un point de vue sémantique (des `for` dans des `for` dans des `if` dans des `for` dans des `else` par exemple)
— du code illisible d'un point de vue esthétique (indentation irrégulière, mélange camelCase / snake_case, mélange tabs / spaces)
— des mega fonctions de plus de 30 lignes
— des mega fichiers de plus de 500 lignes
— des variables globales non constexpr
— des crashs lorsque l'utilisateur effectue une opération non prévue

6 Design pattern

Listez les Design Patterns que vous avez utilisés pour votre projet, et quelques endroits où ils apparaissent.

Choisissez-en un que vous trouvez très pertinent, et expliquez pourquoi vous l'avez choisi et ce qu'il permet de faire.

7 Aide à la gestion de projet

1. Avant de vous lancer tête baissée dans le code, vous devez réfléchir ensemble à l'architecture de votre programme. Cette étape est essentielle à la réussite du projet : une architecture mal réfléchie au départ entraînera des modifications par la suite et donc un retard possiblement conséquent dans l'avancement du projet. Néanmoins, ne passez pas trop de temps sur cette réflexion, l'objectif est d'avoir une architecture simple, répondant aux attentes des spécifications. Il n'est pas nécessaire que votre programme soit ultra-modulable.
2. Optez pour de petits cycles de développement (type analyse, spécification, développement, test, validation).
3. Découpez le projet en tâches courtes, intégrables et testables rapidement, pour pouvoir mieux jauger l'avancement du projet.
4. Définissez, en amont, le rôle de chacun en fonction de ses compétences et ses affinités.
5. Utilisez un outil de gestion de versions type GIT, par exemple GitHub (https://github.com/) ou Bitbucket (https://bitbucket.org/).
6. Utilisez un outil de répartition et suivi de tâches, par exemple Trello (https://trello.com/) ou Azendoo (https://app.azendoo.com)
7. Codez une surcouche à OpenGL pour faciliter le développement. Elle pourrait contenir des classes simplifiant la gestion des ressources (VBO, textures, *etc.*) ou encapsulant des techniques utilisées régulièrement.
8. Ne passez pas les trois quarts du projet à modéliser de beaux objets. Vous aurez une meilleure note si le programme fonctionne bien et que les éléments sont de simples cubes ou sphères, qu'avec de beaux assets et un programme tout buggué...
9. Suivez les instructions du guide de survie donné sur e-learning.

8 Livrables

La veille de la soutenance (*i.e.* le [À DÉFINIR]), vous devrez nous fournir par mail (venceslas.biri@univ-eiffel.fr/ sylvain.cherrier@univ-eiffel.fr/ vincent.nozick@univ-eiffel.fr/ clement.chomicki@univ-eiffel.fr) :

- le code source : donnez nous accès à votre dépôt Git ;
- la documentation Doxygen ;
- un rapport en PDF (minimum 4 pages, hors page de garde, tables des matières, *etc.*), contenant au moins :
  - un mode d'emploi ;
  - une description simplifiée de l'architecture du programme (pas le détail de toutes les classes, concentrez vous sur l'essentiel) ;
  - la justification de vos choix ;
  - une présentation de la gestion de projet (qui a fait quoi, ce qui a fonctionné ou non, *etc.*) ;
  - les résultats.

Attention ! Pour le rapport, nous ne voulons pas que vous suiviez à la lettre le plan ci-dessus, pour la simple et bonne raison qu'il ne s'agit pas d'un plan mais simplement des notions qui doivent être contenues dans le rapport. Par exemple, une section *Justification des choix* n'aurait aucun sens. Vous justifierez vos choix au fur et à mesure que vous les présenterez !

9 Soutenance

Vous devrez présenter votre projet lors d'une soutenance de **15 minutes** (+ 5 minutes de questions), devant vos camarades et nous. Votre présentation devra comporter :

- une démonstration de votre jeu, en présentant les diverses fonctionnalités (spécialement les fonctionnalités supplémentaires). L'objectif ici est de « vendre » votre jeu. Imaginez que nous sommes des éditeurs et que vous cherchez à éditer votre jeu : vous devez nous en mettre plein la vue !
- une présentation succincte de l'architecture de votre programme et des bibliothèques utilisées (pas la STL, ni OpenGL, ni SDL, on connait et vous êtes obligés de les utiliser) ;
- un résumé de votre gestion de projet (qui a fait quoi, difficultés rencontrées, outils utilisés, si c'était à refaire vous changeriez quoi ?, *etc.*).

Nous attendons de vous une soutenance claire et fluide. Vous devez donc bien la travailler et faire plusieurs répétitions avant le jour J (et pas avant la minute M). La présentation orale de ses travaux devant un public n'est pas une tâche triviale ! Voyez cette soutenance comme un entraînement car, dans la suite de vos études et dans vos futurs emplois, vous serez souvent amenés à effectuer ce type d'exercice.