Projet C++

23 janvier 2024

Le but de ce projet est de réaliser un **jeu multi-joueur** (plusieurs joueurs sur un même écran) **en C++**. Votre programme doit être structuré de telle sorte qu'il soit possible de remplacer un ou plusieurs joueurs par une « intelligence artificielle » gérée par l'ordinateur. On ne vous demande pas (nécessairement) de programmer cette IA, mais votre modélisation doit rendre facilement possible l'écriture d'une IA en modifiant peu votre code par ailleurs. Vous pouvez vous inspirer de jeux existants en ce qui concerne la thématique et la mécanique de jeu, et vous aurez un légér bonus si le jeu implique des robots. Votre travail sera évalué sur la qualité de la modélisation (extensions possibles, élégance, généricité, bon usage de design patterns si approprié), et du code (concision, efficacité, robustesse, propreté mémoire, clarté, bon usage des possibilités offertes par le langage). L'évaluation porte aussi sur votre capacité à viser quelque chose d'atteignable : on préfère un jeu fonctionnel mais pas trop ambitieux qu'un jeu très ambitieux mais en chantier. Les principales contraintes sont les suivantes :

- Votre programme doit compiler et fonctionner sur les ordinateurs de l'école sous GNU/Linux.
- Un Makefile ou un CMake doit être fourni. il doit être être possible de compiler le projet par la commande make sans avoir à utiliser d'autre build system ou d'IDE.
- Il doit utiliser au moins 6 classes (c'est un minimum) dont au moins une abstraite.
- L'héritage doit être utilisé (de façon utile et pertinente) au moins une fois.
- Le polymorphisme doit être utilisé (de façon utile et pertinente) au moins une fois.
- Toutes les classes concrètes doivent avoir des tests (Vous pouvez par exemple utiliser Boost.Test)
- Votre programme doit fonctionner sans erreurs majeures 3 avec Valgrind.

Vous pouvez utilisez pour votre projet des bibliothèques et outils existants, à conditions qu'ils :

- Soient disponibles en ligne sous licence libre ;
- Soient compatibles avec l'environnement cible (= utilisables facilement sur les machines de l'école) ;
- Ne simplifie pas la tâche au point de rendre l'exercice sans intérêt : vous pouvez utiliser des bibliothèques diverses et variées pour répondre à vos besoins logiciels (affichage, son, physique, sérialisation, réseau, IA, algorithmique, etc.) mais pas utiliser un moteur de jeu existant (type Unreal Engine), un gros framework qui fait toute l'intégration à votre place (type Oxygine) ou un "squelette" de projet qui définit déjà toute la hiérarchie de classes. Si vous avez le moindre doute sur ce que vous pouvez utiliser ou non, **posez la question à vos encadrant.e.s** !

Pour l'affichage, il vous est fortement recommandé d'utiliser les bibliothèques SFML ou SDL2 ; vous trouverez en ligne de très nombreux exemples et tutorials, par exemple `https://www.parallelrealities.co.uk/tutorials/` pour SDL2 et `https://www.sfml-dev.org/tutorials/2.5` pour SFML. D'autres options sont possibles (e.g. GTK+) mais elles tendent à être plus complexes à prendre en main et moins adapté aux jeux. **Il n'est pas autorisé d'utiliser QT.**

Conseils

- Ne perdez pas de temps sur la qualité des graphismes, ce n'est pas du tout le propos de l'exercice et ne compte pas dans la notation. Vous pouvez utiliser des graphismes très simples, ou des éléments disponibles en ligne sous license libre ;
- Prenez du temps pour préparer et concevoir votre projet, explorer les divers choix techniques possibles (modélisation, bibliothèques, etc.) et vous former sur les bibliothèques nécessaires (SFML ou SDL, etc.) ;
- Vous serez évalués sur la qualité technique de votre jeu, pas la profondeur ou la complexité de son gameplay, donc soyez raisonnables. Vous ne programmerez pas un Civilization ou un jeu de stratégie en temps réel complet en deux mois. Mieux vaut un jeu relativement simple, bien conçu, bien réalisé, bien documenté et testé, et accessoirement amusant, plutôt qu'un prototype qui fait un quart de ce que vous avez prévu et plante deux fois sur trois.
- Cela reste un exercice de POO, donc adhérez le plus possible aux principes de la programmation orientée objet. Ce point aura un rôle déterminant dans la notation, notamment la division du code et du travail entre classes de facon appropriée. Un programme qui réaliserait presque tout ses traitements dans une grande "boucle principale" et n'utiliserait l'objet que de manière anecdotique aura au mieux une note médiocre, indépendamment de sa taille ou de ses qualités ludiques.
- Pour cette raison, les versions informatiques de jeux de cartes ou de plateau sont généralement à éviter : ce genre de jeu est souvent assez simple en terme de structuration objet, tout en demandant un travail important sur des aspects non évalués ici comme l'interface (afficher correctement différents éléments de jeu, interagir avec les joueurs de façon souvent complexe, etc.) et la logique du jeu. Autrement dit, ce sont des projets intéressants en soi mais généralement pas adaptés au présent exercice.
- Utilisez les tests, validez ce que vous avez implémenté avant de passer à la suite.
- Idéalement, architecturez votre projet de façon à pouvoir rapidement avoir un prototype minimal qui tourne puis l'enrichir progressivement. Al faut mieux pouvoir se concentrer sur un (ou quelques) problème(s) à la fois que construire une grosse usine à gaz buggée et passer des semaines à (ne pas réussir à) la débugger ;
- N'hésitez pas à consulter vos encadrant.e.s pour obtenir un retour sur vos choix de conception et d'implémentation ; ils pourront vous donner des idées et vous éviter des écueils.
- — Les design patterns suivants peuvent être particulièrement utiles pour résoudre certaines problématiques typiques se posant dans la programmation d'un jeu :
  - **Observateur -** pour quand un évènement se produit qui peut affecter un ensemble d'objets (par exemple un personnage se déplace et on veut vérifier s'il entre en collision avec d'autres éléments du jeu) ;
  - **Strategie -** : pour quand on veut qu'un même type d'objet puisse avoir plusieurs comportements (par exemple le personnage du joueur qui peut être contrôlé par le clavier ou une IA, ou un élément du jeu qui peut avoir plusieurs types de mouvements), et/ou que ce comportement peut changer au cours de l'exécution du programme (par exemple un jeu de combat ou un ennemi changerait de mouvement après avoir été endommagé) ;
  - **Visiteur -** comme vu en TP3, si vous avez des traitements génériques multiples sur des types d'objets multiples ;
  - **Commande -** : pour quand un ou des objets doivent envoyer des commandes ou des ordres complexes à un autre objet ;

Avertissement

Il existe de nombreux jeux libres et tutorials de programmation de jeux en C++ sur Internet. Vous êtes bien sûr autorisés, et encouragés, à les consulter et vous en inspirer. Cependant, le code que vous rendez devra être uniquement votre propre travail. Une attention toute particulière sera portée au plagiat, et tout projet qui s'avérera être copié (depuis Internet, vos camarades actuels ou passés, etc.) recevra la note 0. Si vous avez un doute sur ce qui est autorisé ou non, demandez à vos encadrant.e.s !

Rendu

Votre projet sera rendu sous la forme d'un dépot git, dont vous déposerez le lien sur Moodle au plus tard le ... . Votre rendu doit comprendre :

- Un diagramme UML
- Le code source et de quoi le compiler
- Toutes les ressources graphiques nécessaires
- Un fichier README comportant
  - Le nom des auteurs du projet ;
  - La liste des bibliothèques, framework, outils externes etc. que vous avez utilisés ;
  - Les instructions sur comment compiler votre programme et si besoin les bibliothèques nécessaires ;
  - Un rapide manuel d'utilisation de votre jeu (but, comment jouer, commandes, etc.)

En plus de ce dépot git, vous aurez également une soutenance de 20min par min (10min de présentation + 10 min de question). Lors de cette soutenance vous présenterez votre programme, comment vous l'avez conçu, ce qui ne marche pas, ce qui vous parait intéressant, comment votre binome a fonctionné... Lors de la phase de question, il y aura une partie test où l'on tentera d'installer votre projet comme si nous ne vous avions pas suivi, ie comme des utilisateurs lambda. Appliquez vous donc bien sur la partie documentation du projet.

Barème indicatif

La grille suivante sera utilisée pour évaluer votre projet. Des points bonus pourront être attribués en cas de réalisations impressionnantes allant au delà des exigences du sujet.

- Structuration objet et bonne utilisation des principes de la programmation objet : 6 points
- Ampleur du projet et quantité de travail réalisé : 6 points
- Cadrage projet (capacité à rendre un jeu jouable et fini) : 2 points
- Respect des consignes (6 classes dont 1 abstraite, polymorphisme, etc.) : 3 points
- Tests : 1.5 points
- Documentation (inclue les commentaires et la documentation texte) : 1.5 points