INF1015 - Programmation orientée objet avancée
Projet

Tous les chapitres.

Objectifs : Introduire l'étudiant à la conception de programmes orientés objet en faisant des choix par rapport aux différents concepts vus dans ce cours.

Durée : Jusqu'à la fin de la session.

Livrables intermédiaires :

1 – Avant le 23h30 le lundi 12 avril 2021.

2 – Avant le 23h30 le jeudi 22 avril 2021.

Remise du travail final : Avant 23h30 le dimanche 9 mai 2021.

Documents à remettre : sur le site Moodle des travaux pratiques, vous remettrez la solution Visual Studio ou le dossier Visual Studio Code, **nettoyés***, avec l'ensemble des fichiers .cpp et .hpp compressés dans un fichier .zip en suivant la procédure de remise des TDs.

Directives particulières

- Vous devez suivre les bons principes de programmations appris dans le cours.
- Le côté artistique visuel n'est pas dans les critères d'évaluation, mais l'implémentation correcte de l'interface graphique et son utilisabilité l'est. Vous pouvez utiliser des images légalement prises sur Internet en spécifiant la source.
- Vous pouvez récupérer une partie des points perdus dans un livrable si vous corrigez les problèmes pour la remise finale. Pour cette raison, aucun solutionnaire ne sera donné mais vous aurez des commentaires sur ce que vous avez fait.
- Les deux premiers livrables seront évalués principalement sur leur conception même si pas tout est fonctionnel, sauf la partie liée au TD6 (pour laquelle vous pouvez éliminer les parties non fonctionnelles du projet). Le travail aura une bonne proportion de l'évaluation sur le fonctionnement.
- Vous devez éliminer ou expliquer tout avertissement de « build » donné par le compilateur (avec /W4).
- Respecter le guide de codage, les points pertinents pour ce travail sont donnés en annexe à la fin.
- N'oubliez pas de mettre les entêtes de fichiers (guide point 33).
- ***** Le nettoyage pour la remise : il ne faut pas remettre les dossiers générés lors de la compilation (.vs, Release, Debug, x64, build).

Description du projet :

Nous voulons un jeu de type « échecs » (https://fr.wikipedia.org/wiki/%C3%89checs), et plus particulièrement les fins de parties.

Pour information (on ne fera pas l'implémentation complète de tous les mouvements ni de toutes les règles) :

Les règles : https://www.iechecs.com/regles.htm

Les mouvements des pièces : https://www.iechecs.com/pieces.htm

Avec plus de dessins pour les mouvements : https://www.chessvariants.com/d.chess/echec.html

Permet de déplacer les pièces à l'écran : https://www.iechecs.com/iechecs.htm?app,a=lecavalier

Les cas intéressants de fin de partie : https://en.wikipedia.org/wiki/Pawnless_chess_endgame

Les fins de parties ont un roi de chaque côté et quelques autres pièces de chaque côté (ils parlent de cas de une à quatre dans le lien précédent).

Le même pointeur de souris servira à déplacer les pièces des deux joueurs, il n'y a pas de composante réseau ni intelligence artificielle dans la matière de ce cours.

Livrable 1 : La modélisation du des déplacement des pièces (pas de dessin ni d'interaction avec l'usager)
Vous devez avoir un roi et deux autres pièces de types différents, par exemple tour et cavalier.
Votre programme devrait être conçu pour pouvoir facilement ajouter les autres types de pièces.
Les pièces doivent savoir quels mouvements sont valides, sans tenir compte de la mise en échec de notre propre roi (dont la vérification demande de regarder toutes les pièces du jeu), pour pouvoir faire un mouvement ou dire que le mouvement n'est pas valide.
Des tests préliminaires pour vérifier qu'on peut bien bouger les pièces seulement pour des mouvements valides.

Livrable 2 : Complétion du modèle et début d'interface graphique Qt
La vérification s'il y a échec : si une pièce peut atteindre le roi adverse et ne pas permettre un mouvement qui nous met nous-même en échec.
Affichage de l'échiquier (grille de 8×8 cases) dans une fenêtre pour y mettre les pièces, en utilisant la bibliothèque Qt. Les pièces peuvent être des mots/lettres dans les cases.
On veut pouvoir indiquer avec le curseur de la souris quelle pièce déplacer à quel endroit et avoir une rétroaction visuelle si le mouvement n'est pas permis.

Remise finale : Complétion du jeu
Avoir une manière pour dire qu'on veut démarrer une nouvelle partie et choisir d'une liste une position de départ (vous pouvez prendre des positions classiques ou mettre n'importe quoi).
Les tests avec aucune ligne non couverte pour la partie modèle du jeu. Vous n'avez pas à automatiser les tests de l'interface graphique (pas matière à ce cours).
Avoir un jeu fonctionnel.
Vous pouvez ajouter des images à vos pièces.

ANNEXE 1 : Utilisation des outils de programmation et débogage.

Utilisation des avertissements :
Avec les TD précédents vous devriez déjà savoir comment utiliser la liste des avertissements. Pour voir la liste des erreurs et avertissements, sélectionner le menu Affichage > Liste d'erreurs et s'assurer de sélectionner les avertissements. Une recompilation (menu Générer > Compiler, ou Ctrl+F7) est nécessaire pour mettre à jour la liste des avertissements de « build ». Pour être certain de voir tous les avertissements, on peut « Regénérer la solution » (menu Générer > Regénérer la solution, ou Ctrl+Alt+F7), qui recompile tous les fichiers.

Votre programme ne devrait avoir aucun avertissement de « build » (les avertissements d'IntelliSense sont acceptés). Pour tout avertissement restant (s'il y en a) vous devez ajouter un commentaire dans votre code, à l'endroit concerné, pour indiquer pourquoi l'avertissement peut être ignoré.

Rapport sur les fuites de mémoire et la corruption autour des blocs alloués :
Le programme inclus des versions de débogage de « new » et « delete », qui permettent de détecter si un bloc n'a jamais été désalloué, et afficher à la fin de l'exécution la ligne du programme qui a fait l'allocation. L'allocation de mémoire est aussi configurée pour vérifier la corruption lors des désallocations, permettant d'intercepter des écritures hors bornes d'un tableau alloué.

Utilisation de la liste des choses à faire :
Le code contient des commentaires « TODO » que Visual Studio reconnaît. Pour afficher la liste, allez dans le menu Affichage, sous-menu Autres fenêtres, cliquez sur Liste des tâches (le raccourci devrait être « Ctrl \ t », les touches \ et t faites une après l'autre). Vous pouvez double-cliquer sur les « TODO » pour aller à l'endroit où il se trouve dans le code. Vous pouvez ajouter vos propres TODO en commentaire pendant que vous programmez, et les enlever lorsque la fonctionnalité est terminée.

Utilisation du débogueur :
Lorsqu'on a un pointeur « ptr » vers un tableau, et qu'on demande au débogueur d'afficher « ptr », lorsqu'on clique sur le + pour afficher les valeurs pointées il n'affiche qu'une valeur puisqu'il ne sait pas que c'est un tableau. Si on veut qu'il affiche par exemple 10 éléments, il faut lui demander d'afficher « ptr,10 » plutôt que « ptr ».

Utilisation de l'outil de vérification de couverture de code :
Suivez le document « Doc Couverture de code » sur le site Moodle.

Annexe 2 : Points du guide de codage à respecter

Les points du **guide de codage** à respecter **impérativement** pour ce TD sont :
(voir le guide de codage sur le site Moodle du cours pour la description détaillée de chacun de ces points)

Mêmes points que le TD3 :

2 : noms des types en `UpperCamelCase`
3 : noms des variables en `lowerCamelCase`
5 : noms des fonctions/méthodes en `lowerCamelCase`
7 : noms des types génériques, une lettre majuscule ou nom référant à un concept
8 : préférer le mot `typename` dans les template
15 : nom de classe ne devrait pas être dans le nom des méthodes
21 : pluriel pour les tableaux (`int nombres[];`)
22 : préfixe *n* pour désigner un nombre d'objets (`int nElements;`)
24 : variables d'itération `i`, `j`, `k` mais jamais `l`, pour les indexes
27 : éviter les abréviations (les acronymes communs doivent être gardés en acronymes)
29 : éviter la négation dans les noms
33 : entête de fichier
42 : `#include` au début
44,69 : ordonner les parties d'une classe public, protected, private
46 : initialiser à la déclaration
47 : pas plus d'une signification par variable
48 : aucune variable globale (les constantes globales sont tout à fait permises)
50 : mettre le & près du type
51 : test de 0 explicite (`if (nombre != 0)`)
52, 14 : variables vivantes le moins longtemps possible
53-54 : boucles `for` et `while`
58-61 : instructions conditionnelles
62 : pas de nombres magiques dans le code
67-78, 88 : indentation du code et commentaires
83-84 : aligner les variables lors des déclarations ainsi que les énoncés
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires