420-V31-SF
PROGRAMMATION DE JEUX VIDÉO III
TRAVAIL PRATIQUE #2
JEU DE PLATEFORME AVEC NAVIGATION
PONDÉRATION : 10%

OBJECTIFS PÉDAGOGIQUES

Ce travail vise à familiariser l'étudiante ou l'étudiant avec les objectifs suivants :

- Appliquer une approche de développement par objets (compétences 016T)
- Apporter des améliorations fonctionnelles à une application (compétence 0176)
- Concevoir et développer une application dans un environnement graphique (compétence 017C)

De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de documentation et de programmation.

MISE EN CONTEXTE ET MANDAT

Ce travail pratique a pour but premier de créer un mini-jeu de plateforme en 2D (Affichage en sprite), entouré d'une architecture de menus qui permettra de faire une gestion de compte utilisateur. L'étudiant devra donc, suite aux spécifications de l'application, définir une architecture orientée objet, où l'héritage, le polymorphisme, ainsi que certains design patterns seront prépondérants. Il pourra par la suite utiliser des éléments de code fournis afin d'accélérer le développement de l'application.

Par la suite, dans un environnement C++ / SFML, l'étudiant devra développer un jeu complet et en assurer la qualité à la fois par des tests unitaires, assertions et tests d'utilisation

SPÉCIFICATIONS DE L'APPLICATION DE BASE

Vous devez créer un jeu de plateformes, du moins un niveau de jeu de plateforme classique : il faut traverser le niveau d'un bout à l'autre, sauter sur des plateformes pour pouvoir progresser, vaincre et/ou éviter des ennemis et ramasser en chemin des "collectibles". Outre les contraintes qui vous seront présentés plus loin, vous êtes libre d'implémenter le projet selon le thème de votre choix.

Ce sera une bonne occasion de mettre en pratique les concepts d'héritage et de polymorphisme, d'utiliser les premiers design patterns que nous avons exploré en classe et, éventuellement, d'utiliser des animations de sprite (bien que ce dernier point ne soit vraiment pas prioritaire)

Il y aura quatre parties assez di strictes dans votre projet :

1- Le jeu de plateforme

2- Les différents menus, la navigation dans ceux-ci et les données qu'ils vont manipuler (selon le modèle MVC)

3- Les tests unitaires sur le contrôleur et le modèle

4- Le document de projet.

PROJET ET ASSETS DE DÉPART

Un projet de départ vous est fourni, comprenant quelques assets utilisables. Vous n'êtes nullement tenu d'utiliser ceux-ci et pouvez utiliser les assets de votre choix. Les assets fourni viennent de la démo de plateformer pour XNA 4 (C#).

Ce même projet original vous est fourni également, pour fin d'observation (pour installer XNA c'est ici : https://mxa.codeplex.com/releases).

Ceci dit, il y a de nombreux jeux commerciaux dont pratiquement toutes les spritesheet se trouvent sur le web. Alors si vous souhaitez cloner un jeu existant, ne vous gênez pas!

1- LE JEU DE PLATEFORME

Consignes

1- À la base le jeu peut rester très simple. Pensez à Mario Bros (pas Super) ou Donkey Kong (l'original). Ceci dit, outre les contraintes qui suivent vous avez un certain degré de liberté sur ce que vous voulez produire.

Ayez au moins un objectif clair sur comment compléter le niveau (traverser le niveau, vaincre tous les ennemis, ramasser tous les collectibles, survivre le plus longtemps possible, etc.) car vous devrez implémenter un système de score dans votre jeu (voir plus bas).

2- Vous n'êtes pas obligé d'utiliser les sprites fournis, mais au moins un acteur doit être animé, et il doit avoir au moins deux "sets" d'animation. Pas obligé que l'acteur en question soit le personnage principal (même si c'est ce qui ferait le plus de sens).

3- Votre personnage doit pouvoir sauter sur des plateformes. Donc le niveau doit avoir des plateformes.

4- Vous devez implémenter au moins trois types d'ennemis. Tous les ennemis hériteront de la même super-classe. Dans votre code, vous aurez une collection d'objets de la super-classe et vous la parcourerez à chaque update pour gérer le mouvement et les attaques des ennemis.

5- Un des ennemis devra pouvoir faire une action qu'aucun autre ne pourra faire. Il faudra donc vérifier son type pour qu'il puisse utiliser cette action correctement.

6- Vos ennemis doivent être générés par des fabriques. L'idéal serait d'avoir des spawn points (des genres de cabanes à ennemi). Mais si vous n'en avez pas, utilisez tout de même des fabriques pour créer vos ennemis, question de montrer que vous maîtrisez le concept. Au moins une fabrique standard et une statique doivent être implémentées. NOTEZ BIEN : si un autre acteur qu'un ennemi se prête à l'utilisation de fabriques, celui-ci peut être celui qui les utilise, mais ça suppose un jeu complexe.

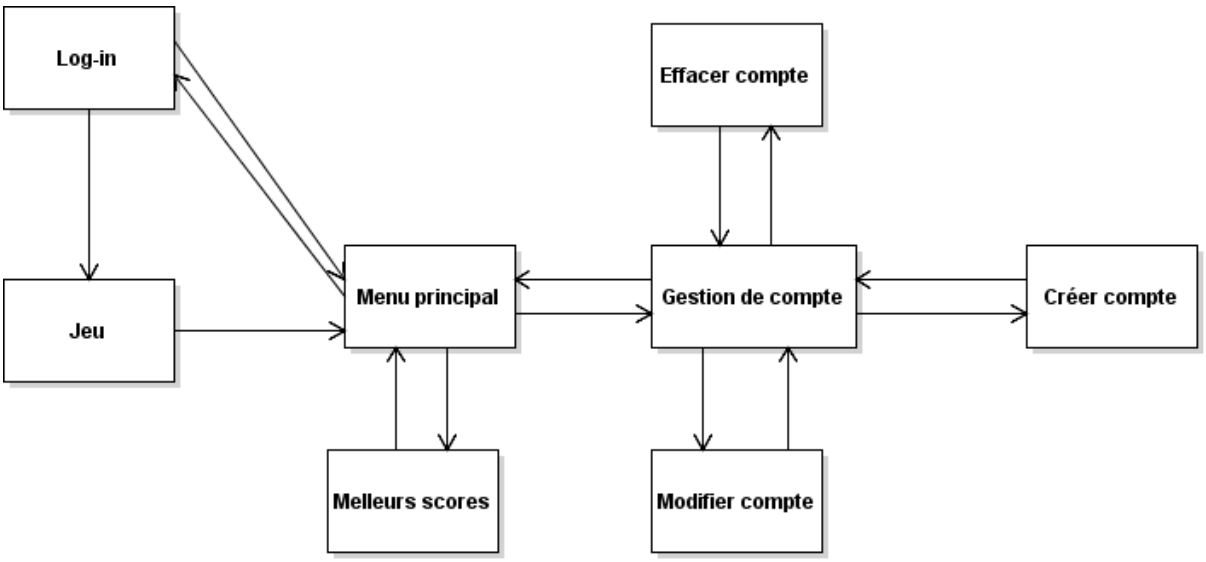
7- Vous devez avoir au moins un objet qui doit être implémenté selon le patron de conception du flyweight. Un collectible serait un bon choix.

8- Votre système doit avoir un système de score. À vous d'implémenter ses mécaniques. Mais à la fin de la partie, le score sera sauvegardé par le compte utilisateur (voir plus loin).

9- Pour le reste les choses sont à votre choix. Votre personnage n'est même pas obligé de pouvoir s'attaquer aux ennemis (juste traverser le niveau, collectionner des objets ou survivre le plus longtemps possible), ou alors il peut leur sauter sur la tête, leur tirer dessus ou vous pouvez même simplement cliquer sur un ennemi avec la souris pour le vaincre suivant certaines conditions. Mais bref, une règle à suivre : **gardez les choses simples!**

2 – LES MENUS ET LE MVC

SFML n'étant pas Winforms, une textbox vous est fourni, les validations et choix de menu se feront strictement avec le clavier. Précisez à l'écran sur quelle touche du clavier il faut appuyer pour passer d'un écran à l'autre.



Menu principal

Écran de titre. Aucun textbox. On affiche les instructions pour commencer le jeu, aller à l'écran de gestion de compte ou à l'écran d'affichage des meilleurs scores.

Log-in

Pour entrer dans le jeu, il faut d,abord passer par l'écran de log-in. On doit alors entrer le bon nickname et le bon mot de passe. Si cela est fait, on se dirige au jeu. On peut revenir au menu principal sinon. On en "retient" pas le fait que l'utilisateur est connecté; il doit entrer son info à chaque partie.

Jeu

Sortir du jeu nous ramène au menu principal

Meilleurs scores

On peut entrer un nickname et un score.

- Si on entre rien, on nous donne les 10 meilleurs scores au total (ou moins si le système en a moins).

- Si on entre un nickname, on obtient les 10 meilleurs scores de ce joueur (ou moins)

- Si entre un score, on obtient les 10 meilleurs scores (ou moins) égaux ou au-dessus de ce score (ou moins)

- Si on entre les deux, on obtient les 10 meilleurs scores (ou moins) égaux ou au-dessus de ce score pour ce joueur précis.

VALIDATION : Le nickname doit exister dans le système et le score doit être un entier positif.

Gestion de compte

Écran de transition. Permet de naviguer vers un des trois écrans concrets de gestion de compte, ou de retourner vers l'écran principal.

Créer compte

Permet d'ajouter un utilisateur celui-ci doit inscrire les informations suivantes avec les validations suivantes.

Nickname : De 3 à 25 caractères de long. Ne doit pas déjà exister dans le système.

Mot de passe : 5 à 15 caractères de long. Doit contenir au moins une lettre minuscule, une majuscule, une chiffre et un caractère qui n'est ni un chiffre ni une lettre.

Nom: 2 à 25 caractères de long: ne doit contenir que des lettres, des points et des traits d'union

Prénom : Exactement comme le nom

Courriel : validation comme l'exercice 5 (courriel format Outlook).

Si toute cette info est inscrite correctement, le compte est ajouté au système.

Modifier compte

On doit entrer un nickname valide ainsi que le mot de passe. Si les deux sont entrées correctement, le nom, le prénom et le courriel apparaissent. L'utilisateur peut modifier ses informations, mais les règles de validation sont les mêmes que pour la création de compte.

Effacer compte

L'utilisateur entre un nickname valide et le mot de passe. Si ceux-ci sont valides, on demande une confirmation. S'il y a confirmation. Le compte est effacé, et les scores associés aussi.

CONTRÔLEUR

Toutes les validations sur la "forme" des informations transmises devront se faire par un contrôleur qui sera aussi un singleton. Même les demandes de transitions de scène se feront par lui (une scène envoie une « demande de transition » et le contrôleur renvoie l'enum de la scène où il faut se rendre.

Évidemment, afin de savoir si un nickname existe et un password est valide, le contrôleur devra interroger la vue pour le savoir.

MODÈLE

Toute l'information reçue par le modèle sera considéré comme valide dans sa forme (c'était le travail du contrôleur). Vous êtes libres de créer l'architecture de votre choix pour la partie modèle : une seule classe ou plusieurs, selon ce que vous analyserez. Quand une requête du contrôleur fonctionne, le plus simple serait de renvoyer une instance de classe contenant toute l'information demandée.

Évidemment, un pareil système demanderait de la persistance (un fichier, une base de données, un système et sérialisation, etc.). Ce TP étant déjà assez chargé, on va "mocker" (simuler) cette partie; votre modèle aura une fonction de mock qui, une fois le modèle créé, aura pour fonction de remplir ce même modèle des informations sur au moins 8 comptes utilisateurs, ainsi que quelques parties qu'ils auraient pu jouer, comme si le modèe venait de les charger d'une base de données.

3 – LES TESTS UNITAIRES

Toutes les méthodes du modèle et du contrôleur devront être testées unitairement (sauf le chargement "mock"). Si vous avez besoin du modèle, vous pouvez utiliser votre mock de persistance par défaut ou en créer un autre, plus léger et plus adapté aux tests, avec moins d'éléments.

4 – LE JOURNAL DE PROJET

Vous devrez remettre un journal de projet sous format word comprenant les informations suivantes.

1- Page titre

2- Table des matières

3- Document de design préliminaire, dûment rempli.

4- Une page d'instruction, pour expliquer votre concept final ainsi que toutes les commandes que le joueur doit utiliser pour jouer à votre jeu et naviguer dans vos différents menus.

5- Le schéma UML de base, illustrant l'architecture de votre projet. N'y mettez que les classes, relations et cardinalités.

6- Un log de tâche : à chaque fois qu'un des membres de l'équipe travail sur le projet durant une journée donné, il doit inscrire le temps de travail total de ce sur quoi il a travaillé. Donnez un peu de détails sur ce qui a été fait (une phrase ou deux) et s'il y a eu des difficultés, vous devez les mentionner. Un tableau Excel vous sera fourni pour vous aider à maintenir la chose. Voici les infos à entrer.

- Qui l'a réalisé (ça peut-être les deux membres à la fois – si c'est le cas, chaque membre l'inscrit à son log)

- Date

- Temps travaillé.

- Tache effectuée (donnez un peu de détails)

- Si le travail s'est effectué sans problème ou si des difficultés se sont posées. Dans le second cas, veuillez les préciser.

Ce journal est absolument obligatoire. Le botcher donnera une note de 0 sur cette partie. Ne pas le remettre entraine une pénalité de 10%.

Faites-le à mesure que le projet avance.

On répète, faites-le à mesure.

Juste pour être absolument certain : FAITES-LE À MESURE

CONTRAINTES GÉNÉRALES AU NIVEAU DE LA PROGRAMMATION

- Organisez vos classes et assets de manière sensée dans le projet à l'aide de filtres.
  - Effacez tout asset "de départ" qui ne serait pas utilisé.
- Respectez les standards de programmation dans la nomenclature et dans l'encapsulation.
- Faites des fonctions courtes et concises.
- Utilisation correcte du C++. Exploitation judicieuse des possibilités du langage (pointeurs et références, fichiers .h et .cpp)
- Utilisez les constantes autant que possible. Laissez le moins possible de "nombres libres"
- Utilisez les méthodes et attributs constantes autant que possible (on sera sévère cette fois-ci).
- Chargez le moins de textures possibles (pensez à les mettre static à une classe au besoin)
- **Documentez votre code**

BONUS

Puisque le projet permet une certaine liberté au niveau du design, il en sera de même au niveau des bonus. Pensez consulter votre professeur pour savoir si telle ou telle fonctionnalité que vous souhaiteriez ajouter pourrait apporter un bonus.

Au départ, une véritable persistance, que ce soit par fichier ou sérialisation serait l'élément bonus le plus évident à implémenter. Voici une librairie de sérialisation qui pourrait vous aider en ce sens : http://uscilab.github.io/cereal/

Aussi, un niveau de production plus élevé que demandé (autant sur les menus que sur le jeu) ainsi que certains éléments de base spécialement bien réalisés pourraient valoir un bonus.

Mais pour être candidat aux bonus, le travail doit avoir une note de base de 80%. Faites une bonne base, vous ajouterez des "condiments" à la fin.

CONTEXTE DE RÉALISATION ET DÉMARCHE DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **en équipe de 2.** (Notez bien que votre équipe de 2 devra être différente de celle du TP3).

Vous aurez accès à un dépôt de données SVN pour développer votre projet. Son utilisation ne sera pas notée, mais il est fortement recommandé de l'utiliser puisque cette même utilisation sera notée lors du TP3. Aussi bien vous pratiquer maintenant.

Bien livrable 1 :

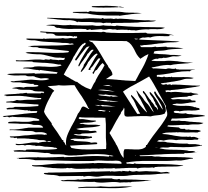
- Document de design préliminaire.
- Sur LEA avant le cours **du mercredi 25 octobre (Gr01) / du jeudi 26 octobre (Gr02).**
- Seront consultés et vérifiés en classe durant ce cours. Ne pas la remettre cette journée-là entrainera une pénalité de 2% sur la note sur TP.

Bien livrable 2 :

- Projet avec code source de l'application.
- Application fonctionnelle.
- Journal de projet SURTOUT avec le log de tâches.
- Sur LEA pour **le jeudi 9 novembre à minuit (Gr01) / vendredi 10 novembre à minuit (Gr02)**

MODALITÉS D'ÉVALUATION

- Tous les **biens livrables** devront être **remis à temps** et selon les modalités spécifiées.
- Dans l'éventualité où vous récupériez du code existant ailleurs (internet, MSDN, etc), vous devez clairement indiquer la source ainsi que la section de code en question. Tout travail plagié ou tout code récupéré d'une source externe et non mentionnée peut entraîner la note zéro (0) pour l'ensemble du travail.



Grille d'évaluation du travail pratique #2

Critères d'évaluation	
Code C++ correct et respectant les standards. Bonne division du code. Respect des contraintes générales de programmation (voir plus haut). Utilisation adéquate des concepts d'héritage et de polymorphisme.	**20%**
Jeu sans bug, respectant les contraintes demandées.	**20%**
Système de menu sans bug, respectant les contraintes demandées	**15%**
Tests unitaires sur le système de menus	**10%**
Utilisation adéquate des design patterns suivant : Singleton, factories, Flyweight	**10%**
Utilisation adéquate du design pattern MVC	**10%**
Animation correcte d'un acteur	**5%**
Journal de projet (zéro automatique si considéré comme "botché", -10% si non-remis). **Faites-le à mesure que le projet avance.**	**10%**
TOTAL	**100%**

ATTENTION CERTAINES ERREURS POURRAIENT SE REFLÉTER DANS **LES DEUX ENTRÉES EN GRAS.** EXEMPLE : SI AUCUN PERSONNAGE N'EST ANIMÉ ALORS C'EST UNE CONTRAINTE DU JEU QUI N'EST PAS RESPECTÉE.

UN PROJET QUI NE COMPILERAIT PAS À LA REMISE POURRAIT SE VOIR DÉCERNER DES PÉNALITÉS SUPPLÉMENTAIRES**.**

APPENDICE : CONSEILS DE DÉVELOPPEMENT

LE JEU :

1- Vous pouvez utiliser l'algorithme traditionnel des plateformers pour gérer la gravité et les chutes. L'exemple XNA l'utilise par ailleurs. Libre à vous de vous en inspirer. En attendant en voici un résumé.

- Si le joueur est au sol, au niveau du "plancher" on ne fait aucune vérification.

- Si on est sur une plateforme, on est aussi considéré comme étant au sol. On ne test pas la gravité, mais on vérifie s'il y a bien un bloc sous nos pieds. On peut restreindre la recherche à partir des "positions sous nos pieds"

- Dans les deux premiers cas ci-dessus, on vérifie quand même les collisions avec les blocs à l'horizontale.

- Si le joueur saute, alors on se crée un système de gravité. On teste les collisions avec les blocs à proximité. Si une collision se produit sous nos pieds, alors on est considéré à nouveau au sol. Un bloc au-dessus arrête immédiatement la progression.

- Vous pouvez mettre des plateformes qui n'arrêtent pas la progression vers le haut, à la Ice Climber, ou comme dans certains blocs dans l'exemple.

- Si vous les souhaitez, vous pouvez implémenter des changements de directions dans les airs à la Super Mario. Dans ce cas, la vitesse de déplacement serait la moitié de la vitesse de déplacement en courant, et la vitesse maximum n'est pas changée pour autant.

- Vous pouvez aussi y allez à la Super Meat Boy. Vitesse dans les airs très élevée et saut possible sur les murs.

2- Gardez vos ennemis simples. Ils peuvent être des personnages qui font un aller-retour sur un étage, des "fantômes" qui passent au travers des plateformes, etc, etc, etc. La seule contrainte est la structure d'héritage.

3- Si vous utilisez les assets du personnage fourni, vous remarquez qu'une "case" d'animation est plus grande que son visuel. Vous pourriez facilement lui coller un IntRect de 32 de longueur total en x, qui serait beaucoup plus approprié pour tester les collisions.

4- Vous avez déjà un système qui "charge des blocs" à la grandeur du niveau. Le niveau fourni peut contenir 20 blocs par 15, un espace vide de bloc est un pointeur null. À vous de vous servir intelligemment de ce tableau.

LES MENUS ET LE MVC :

1- Vous constaterez dans le projet de départ que la boucle de jeu est maintenant gérée au niveau de la scène. Game ayant à présent comme tâche de charger la bonne scène. Dans votre système, la game enverra au contrôleur une "touche pressée" et celui-ci lui renverra la bonne scène à charger. Un des paramètres de la demande demande de changement de scène devrait être le nom de la scène qui fait la demande; cela ajoute à la qualité de la validation.

2- Normalement la touche Enter valide l'entrée des informations, Escape permet de revenir à la scène précédente. Pour les autres touches affichez précisément lesquelles vous voulez que l'utilisateur appuie pour obtenir tel ou tel résultat.

3- Vous n'êtes pas obligé de donner tous les messages d'erreur si plusieurs champs ne sont pas valides en même temps. Le plus simple est de signaler une erreur dès que le contrôleur en trouve une.

4- Normalement, chaque écran de menu devrait être une scène. Oui, Création de Compte et Modification de Compte sont excessivement semblables, mais d'avoir une exception et d'avoir une scène qui est modifiables suivants certains paramètres est une source d'erreur. Gardez les choses simples.

5- Pour ce que modèle renvoie vers le contrôleur, qui le renverra vers la vue. Le plus simple c'est que votre modèle ai une ou plusieurs classes "reader" qui aurait comme pour simple rôle de lire les bonne infos attaché à tel interface. En cas d'action correcte effectuée par le modèle, celui-ci renverrai un pointeur vers une instanciation de cette classe reader.

LES TESTS UNITAIRES

1- Liez rapidement le contrôleur et toutes les classes du modèle aux tests unitaires et faites un test bison question de bien lier le tout dès le départ.

2- N'utilisez pas d'instances d'écran de menu pour les tests unitaires. Après tout le contrôleur a besoin de certaines infos venant de la vue ; pas besoin d'implémenter la vue au complet pour ça.

LE JOURNAL DE PROJET

1- Je le répète, faites votre log de tâche **à mesure.**

2-NE VOUS SPÉCIALISEZ PAS. Chaque membre devrait travailler sur le jeu, sur les menus et sur les tests. Vous spécialiser n'est pas rendre service dans votre carrière. Chaque membre devrait être à l'aise avec tout le code produit.

3- Vérifiez votre code mutuellement (code review)