INF1015 - Programmation orientée objet avancée

Projet A2021

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

Dates de remise : Livrable 1: 13 décembre; Livrable 2: 22 décembre.

Directives particulières

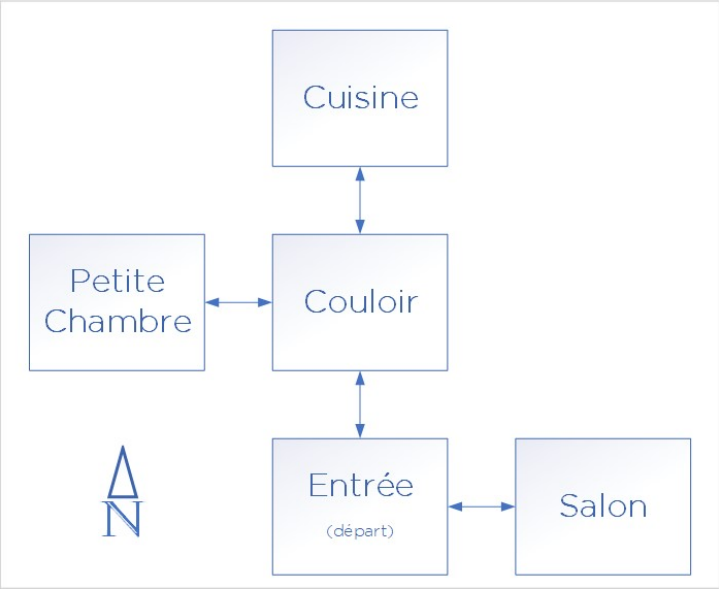
- Vous devez suivre les bons principes de programmations appris dans le cours.
- Le côté artistique visuel n'est pas dans les critères d'évaluation, mais l'implémentation correcte d'une interface en ligne de commande et son utilisabilité le sont.
- Vous pouvez récupérer une partie des points perdus dans 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.
- Le premier livrable est évalué principalement sur la conception et la fonctionnalité de base. Le travail aura une bonne proportion de l'évaluation sur le fonctionnement.
- Respectez le guide de codage, les points pertinents pour ce travail sont les mêmes que pour les TD.
- N'oubliez pas de mettre les entêtes de fichiers (guide point 33).
- Vous avez le droit d'utiliser des bibliothèques tierces telles que boost, cppitertools, gsl et tclap, mais seulement si leur utilisation n'affecte pas les objectifs pédagogiques du projet. Par exemple, si dans le projet on vous demande de développer un module d'opérations matricielles, vous ne pouvez pas utiliser une bibliothèque d'algèbre linéaire à la place. Toutefois, vous pourriez utiliser boost-format ou boost-algorithm pour faire de l'affichage formaté, vu qu'on ne vous évalue pas particulièrement sur votre capacité à manipuler des string à la main pour formater et que c'est complémentaire à ce qui est déjà présent dans la librairie standard de C++.
- Vous pouvez vous baser sur le « Projet VS/VSCode "vide" » disponible sur Moodle, mais vous n'êtes pas obligés de vous en servir. Vous gérez vous-même votre environnement de développement.
- Un code "mauvais" qui répond aux exigences fonctionnelles du livrable 1, mais nullement aux exigences de bonne conception, est fourni sur Moodle.

Description du projet de jeu textuel

Nous voulons un jeu d'aventure textuelle, un peu comme les *Zork* de ce monde. Le jeu entier est en ligne de commande et l'utilisateur interagit avec le jeu en entrant du texte dans la console. On veut pouvoir se promener dans le monde du jeu et effectuer des actions dans celui-ci. Plusieurs exemples d'utilisation vous seront donnés dans les sections suivantes.

Monde du jeu

Dans ce jeu, le monde est réparti en des zones, cellules ou cases qui peuvent être connectées entre elles dans les quatre directions cardinales (nord, sud, est, ouest). Chaque zone peut donc au maximum être connectée à 4 autres zones. Par exemple, on pourrait avoir la carte du monde suivant représentant une maison :



À partir du couloir, on pourrait aller au Nord vers la cuisine, au sud vers l'entrée ou à l'ouest vers la chambre, par exemple. Dans votre code, le monde devrait donc ressembler à des cases connaissant leurs voisines.

Acceuil et description des cases

Le jeu devrait commencer par afficher une petite bannière donnant le nom du jeu. Vous donnez au jeu le nom que vous voulez, mais il faut l'afficher au démarrage. On affiche ensuite le nom et la description de la case de départ (dans la maison ci-dessus ce serait l'entrée), puis les noms et directions des cases connexes. Enfin, on laisse à l'utilisateur un espace pour taper la commande qu'il veut exécuter.

On dit souvent dans le cours qu'il faut afficher un message d'invite avant de lire du clavier, du genre « Veuillez entrer votre commande : ». Dans le cas de notre jeu où l'utilisateur est toujours en train d'entrer des commandes, on pourrait simplement mettre un caractère spécial comme `$` ou `>`, de la même façon que les terminaux sur vos machines.

Voici un exemple d'exécution de cette partie :

```
>>>>> INF1015 DUNGEON CRAWLER 2021 <<<<<
> > > > GAME OF THE YEAR EDITION < < < <

-- Foyer --
This is the entrance of the house. There is a sturdy carpet on the floor.
Main Hallway is to the North (N)
Living Room is to the East (E)

> ▬
```

Dans ce cas, *Foyer* est le nom de la case dans laquelle le joueur se trouve au début du jeu, donc la case de départ. On affiche ensuite la description détaillée de la case (ce que le joueur observerait dans un jeu graphique). On peut aussi voir que, tout comme sur la carte ci-dessus, on a le couloir (*Main Hall*) au nord et le salon (*Living Room*) à l'est.

Navigation

La navigation dans le jeu, en supposant qu'on commence dans l'entrée, pourrait avoir l'air de ceci :

```
>>>>> INF1015 DUNGEON CRAWLER 2021 <<<<<
> > > > GAME OF THE YEAR EDITION < < < <

-- Foyer --
This is the entrance of the house. There is a sturdy carpet on the floor.
Main Hallway is to the North (N)
Living Room is to the East (E)

> N
Going North

-- Main Hallway --
This is the main hallway. There is a bunch of boxes against the wall.
Kitchen is to the North (N)
Foyer is to the South (S)
Small Bedroom is to the West (W)

> E
Cannot go there.

> W
Going West

-- Small Bedroom --
This is the small bedroom. It is not particularly clean or well organised. There is a small window.
Main Hallway is to the East (E)

> ▬
```

Ici on voit que l'utilisateur a entré une lettre pour aller au nord (*N* pour *North*), puis à l'ouest (*W* pour *West*). On voit aussi qu'il y a un petit feedback donné à l'utilisateur sur son action de navigation (« Going North » ou « Cannot go there »). En effet, il n'y a rien à l'est du couloir, donc on le dit à l'utilisateur et le joueur ne bouge pas. Quand on change de case, on affiche le nom et la description de la nouvelle case dans laquelle le joueur se trouve.

Traitement des commandes de base

Dans le cadre du livrable 1, on veut une commande qui permet de réafficher le nom et la description de la case dans laquelle se trouve le joueur (par exemple `look`). Il faut aussi reconnaître les commandes de navigation, qui pourraient être tout simplement le nom ou l'abréviation (dans l'exemple la lettre) de la direction. Les commandes non reconnues devraient afficher un bref message qui l'indique. À chaque exécution d'une commande, on réaffiche l'invite et on attend l'entrée du clavier de l'utilisateur.

```
>>>>> INF1015 DUNGEON CRAWLER 2021 <<<<<
> > > > GAME OF THE YEAR EDITION < < < <

-- Foyer --
This is the entrance of the house. There is a sturdy carpet on the floor.
Main Hallway is to the North (N)
Living Room is to the East (E)

> hello
I do not know that.

> S
Cannot go there.

> look
-- Foyer --
This is the entrance of the house. There is a sturdy carpet on the floor.
Main Hallway is to the North (N)
Living Room is to the East (E)

> N
Going North

-- Main Hallway --
This is the main hallway. There is a bunch of boxes against the wall.
Kitchen is to the North (N)
Foyer is to the South (S)
Small Bedroom is to the West (W)

> ▬
```

Description du livrable 1

Vous devez implémenter toutes les fonctionnalités présentées dans la description du projet ci-dessus et avoir une version fonctionnelle de votre jeu dans lequel on peut naviguer à travers la carte du monde. Vous devriez avoir au moins 5 cases dans votre carte, dont une qui est connectée à plusieurs cases à la fois, et votre monde devrait avoir une case de départ prédéterminée. Il devrait être possible d'avoir une boucle dans la carte. Vous n'êtes pas obligés d'avoir exactement les mêmes noms de commandes que dans l'exemple précédent, mais vous devez avoir les mêmes commandes et donner du feedback au joueur par rapport aux commandes qu'il entre.

Conception

Vous devez concevoir la structure et le contenu de votre code selon une formule orientée-objet propre et facile à lire. Ci-dessous se trouvent quelques points nécessaires à prendre en compte dans votre code. Vous devez aussi respecter le guide de codage des TD et tout ce que ça implique.

- Gestion de mémoire, de construction et de destruction propre et efficace

Assurez-vous que la possession et la responsabilité de la gestion des ressources sont claires dans vos classes.

Vous devez utiliser de la mémoire allouée dynamiquement pour éviter les copies profondes là où c'est préférable. Il ne doit y avoir aucune fuite de mémoire. Vous avez appris à vous servir des pointeurs intelligents et des vecteurs, donc servez-vous-en (aucun `new` et `delete` à la main dans votre code).

- Fonctionnalités bien regroupées dans des classes et fichiers différents

Il est tout à fait possible de coder le jeu demandé au complet dans le `main()` avec quelques fonctions à-côté. Ce serait toutefois une conception bien pauvre et difficile à maintenir et à améliorer pour le livrable 2. On veut une conception orientée-objet où les fonctionnalités et les composantes sont regroupées dans des classes. Par exemple, vous devriez probablement avoir une classe qui représente une case du monde (son nom et sa description) et une autre classe qui représente la carte du jeu (toutes les cases). Vous pourriez décider d'avoir une classe représentant une connexion entre deux cases, ou mettre les connexions directement dans les cases. C'est un exemple des choix de conception que vous devez faire et justifier.ha

L'état du jeu, qui est pour l'instant pas mal seulement la case dans laquelle se trouve le joueur, devrait être aussi probablement gardé dans une classe, plutôt que dans des variables locales dans une fonction quelconque, dans le `main()` ou pire encore, dans des variables globales.

- Séparation claire entre le contenu du monde et la logique du jeu

Pour éviter le code spaghetti et l'interdépendance inutile entre les éléments du programme, il devrait y avoir une séparation très claire et facilement identifiable entre les classes qui gèrent le contenu du monde et celles qui gèrent la logique et l'état du jeu et l'interaction avec le joueur. Par contenu du monde, on veut dire les cases, les connexions entre les cases, la case de départ et la carte du monde en général. Par logique de jeu, on parle du *gameplay* en soi, comme le formatage des affichages, l'analyse des commandes du joueur et la logique appliquée pour faire progresser le jeu.

Ceci étant dit, pour garder le code du projet d'une taille raisonnable, vous pouvez mettre directement dans votre code source les données des cases et des connexions plutôt que de les charger à partir d'un fichier. Cette partie de données *hard-codées* doit toutefois être assez bien isolée dans ses propres méthodes des classes de gestion de l'exécution du jeu.

- Éviter la répétition et le code superflu, prioriser l'encapsulation

Votre code devrait être facile à utiliser et ne pas contenir trop de code superflu. Il devrait être facile de modifier certaines fonctionnalités spécifiques sans avoir à toucher à toutes les classes du projet. Les opérations devraient être faciles à faire. Par exemple, créer une nouvelle connexion entre deux cases devrait se faire en l'appel d'une ou deux méthodes.

Livrable 2

Objets interactifs

En plus d'un nom et d'une description, une zone de la carte peut contenir des objets avec lesquelles le joueur peut interagir. Un objet interactif possède un nom et une description. Lorsqu'on affiche la description d'une zone, on affiche aussi le nom des objets interactifs présents dans la zone. En regardant un objet avec la même commande que pour décrire une zone (`look` dans notre exemple), on obtient sa description. On peut utiliser l'objet avec une autre commande (`use` dans notre exemple), dont le message retourné dépend de l'objet.

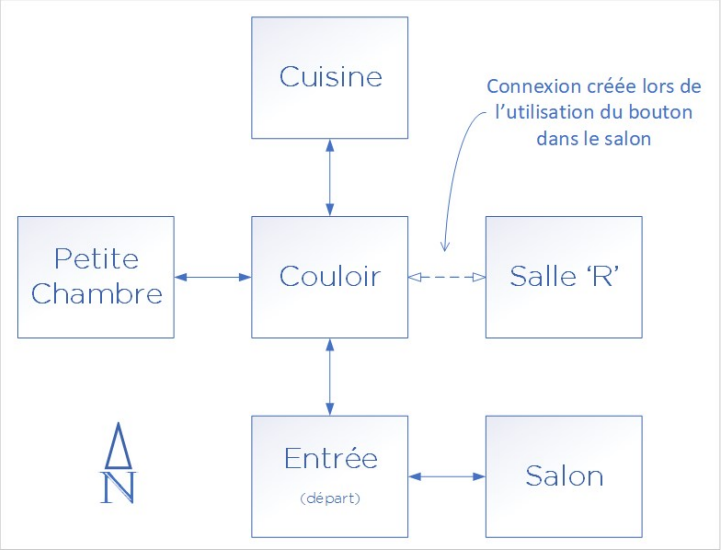
```
-- Living Room --
This is the living room. There is a computer desk with cameras and a bluescreen. Next to it is a TV and a couch.
You notice :
    A red button
    A cheap electric piano
Foyer is to the West (W)

> look piano
This is an old entry-level Yamaha with 61 keys. It looks like any cheap stuff from the late 90s.

> use piano
You play some chords on the piano. It does not sound very well.
```

Objets déverrouillant de nouvelles zones

En interagissant avec certains objets, on veut déverrouiller de nouvelles zones précédemment inaccessibles. Reprenons notre carte de l'exemple du livrable 1, mais en ajoutant une zone cachée (la *Salle R*) à l'est du couloir. Au départ, la salle n'est pas accessible à partir de ce dernier. En utilisant le bouton rouge qui se trouve dans le salon, la connexion se crée et l'accès à la salle est déverrouillé.



```
-- Main Hallway --
This is the main hallway. There is a bunch of boxes against the wall.
You notice :
    A light switch
Kitchen is to the North (N)
Foyer is to the South (S)
Small Bedroom is to the West (W)
```

<- Rien à l'est

On navigue jusqu'au salon...

```
-- Living Room --
This is the living room. There is a computer desk with cameras and a bluescreen. Next to it is a TV and a couch.
You notice :
    A red button
    A cheap electric piano
Foyer is to the West (W)

> use button
Main Hallway is now connected to Room R.
```

On navigue jusqu'au couloir...

```
-- Main Hallway --
This is the main hallway. There is a bunch of boxes against the wall.
You notice :
    A light switch
Kitchen is to the North (N)
Room R is to the East (E)
Foyer is to the South (S)
Small Bedroom is to the West (W)

> e
Going East.

-- Room R --
This is a strange room with red walls.
Main Hallway is to the West (W)
```

<- Maintenant connecté à la salle 'R'

Objets montrant des nouveaux objets

Comme pour ajouter l'accès à une zone (ci-dessus), interagir avec un objet pourrait montrer un objet qui n'était pas avant visible. Attention qu'interagir plusieurs fois ne devrait pas ajouter plusieurs fois l'objet. Noter que ceci n'est qu'un exemple, où le piano est modifié par rapport à l'exemple précédent: il y a un effet lorsqu'on l'utilise et un message différent à ce moment.

```
> use piano
A green key falls off while you play.
> look
This is the living room. There is a computer desk with cameras and a bluescreen. Next to it is a TV and a couch.
You notice :
    A red button
    A cheap electric piano
    A green key
> use piano
You play some chords on the piano. It does not sound very well.
> look
This is the living room. There is a computer desk with cameras and a bluescreen. Next to it is a TV and a couch.
You notice :
    A red button
    A cheap electric piano
    A green key
```

Traitement des commandes d'objets

Les commandes `look` et `use` permettent d'interagir avec des objets. Il faut toutefois spécifier quel objet on veut regarder ou utiliser. On veut que la commande soit suivie d'une chaîne de caractère (possiblement avec des espaces) qui contient comme sous chaîne un des mots importants associés à l'objet. Par exemple, l'objet avec le nom `A black leather jacket` pourrait avoir comme mots importants `leather` et `jacket`, et être observé en faisant `look black leather jacket`, `look jacket`, ou même `look leatherstuff` car ils contiennent une des sous chaînes importantes. Si plusieurs objets respectent le critère de recherche, on prend le premier trouvé. (Noter que cette règle est très simple, le traitement de la "langue naturelle" peut-être un sujet très complexe.)

Les mêmes règles de recherches s'appliquent pour `use` et pour `look`. Si on fait `look` sans rien après, on a le même comportement qu'au livrable 1, c'est-à-dire obtenir la description de la zone. `use` ne peut pas être utilisé sans argument.

Fermeture du jeu

Nous voulons aussi une commande qui permet de quitter le jeu en affichant un message quelconque; dans un vrai jeu, ou pourrait vouloir

sauvegarder la partie au moment de quitter, mais ce n'est pas ici demandé de faire une conception pour le permettre.

```
>>>>> INF1015 DUNGEON CRAWLER 2021 <<<<<
> > > > GAME OF THE YEAR EDITION < < < <

-- Foyer --
This is the entrance of the house. There is a sturdy carpet on the floor.
You notice :
    A pair of man's shoes
Main Hallway is to the North (N)
Living Room is to the East (E)

> exit

ok game done now, go away!
```

Description du livrable 2

Vous devez implémenter toutes les fonctionnalités présentées dans la description du projet ci-dessus (incluant celles du livrable 1) et avoir une version fonctionnelle de votre jeu dans lequel on peut naviguer à travers la carte du monde, interagir avec des objets et utiliser toutes les fonctionnalités décrites ci-dessus.
Comme dans le livrable 1, il faut donner à l'utilisateur une certaine rétroaction sur les commandes qu'il entre. Par exemple, utiliser la commande `use` sans arguments devrait afficher un message disant spécifiquement que la commande `use` doit être utilisée avec un nom d'objet ou un mot-clé.

Conception (suite)

Les éléments importants de conception du livrable 1 s'appliquent tous aussi au livrable 2.

- Orienté objet et héritage pour réduire les if/else/switch/case

On vous demande aussi maintenant d'avoir différents types d'objets interactifs dans les zones. Ils ont en commun le fait qu'ils ont un nom et une description et une action à appliquer. Le nom est affiché quand on liste les objets d'une zone et la description est donnée quand on observe l'objet (commande `look`). L'action est déclenchée par la commande `use` et l'effet dépend du type de l'objet. Le code de l'action devrait être associé au type d'objet (ou à l'objet lui-même), pas être dans une séries de `if/else` pour déterminer quoi appliquer pour l'objet.

- Déverrouillage de nouvelles zones

Quand vous déverrouillez une nouvelle zone en utilisant un objet, ça devrait seulement créer une nouvelle connexion, pas une nouvelle zone en soi. La zone cachée devrait être déjà créée comme les autres zones et présente dans la carte du jeu, mais elle est seulement inaccessible avant d'utiliser l'objet approprié.

- Aiguillage des "commandes" en temps constant

Au livrable 1 vous aviez probablement un `if/else` pour associer une action à une commande entrée. Continuer de cette manière ferait une recherche linéaire en fonction du nombre de commandes reconnues.

On veut maintenant que vous fassiez cette recherche en temps constant (amorti). Plutôt que d'avoir une série de conditions, utilisez un dictionnaire qui associe un nom de commande (string) à une action à effectuer (fonction/méthode/lambda).

Nous voulons que la recherche de commande soit en temps constant, mais pas nécessairement son exécution. Par exemple, si le joueur entre `look electric piano`, la recherche de l'action effectuée par `look` devrait être faite en temps constant, mais la recherche de l'item qui pourrait correspondre à `electric piano` n'a pas besoin de l'être.

Modifié le: mercredi 1 décembre 2021, 01:36

◄ Énoncé TD6

Aller à...

Code fonctionnel du livrable 1, très mal conçu ►

Liens utiles

- Polyfolio
- Dossier étudiant
- Les 12 qualités requises en génie
- Répertoire des cours
- Bibliothèque
- Courriel IMP

Obtenir l'app mobile

Politiques