POO & C++ : Examen sur machine

EISE4 2014—2015

Pablo Rauzy

rauzy@enst·fr

pablo.rauzy.name/teaching.html#epu-cpp

14 novembre 2014

N'oubliez pas :

- **Lisez attentivement les consignes !**
- *Ce TP doit impérativement être rendus par courriel à `rauzy@enst·fr` au plus tard à 12h30, avec `[EISE4]` suivi de votre nom dans le sujet du mail.*
- Le code rendu doit être propre !
- Le TP doit être rendu dans une archive tar gzippée : `tar czvf NOM.tgz NOM` où `NOM` est le nom du répertoire dans lequel il y a votre code, qui doit correspondre à votre nom de famille ; attention à ne pas rendre un répertoire avec des fichiers temporaires dus à la compilation (pensez à faire un `make clean` avant `tar`).
- Quand un exercice demande des réponses qui ne sont pas du code, vous les mettrez dans un fichier texte `reponses.txt`.
- Le code est de préférence en anglais, avec des commentaires en français ou anglais, en restant cohérent.
- Le code doit être proprement indenté.
- Respectez les conventions de nommage :
  - variables : `nom_explicite` (*e.g.*, `game_type`),
  - fonctions : `verbeAction` (*e.g.*, `launchNewGame`),
  - classes : `NomExplicite` (*e.g.*, `BoardGame`),
  - membres privés ou protégés : comme variable ou fonction + `_` (*e.g.* `max_number_of_player_`, ou `updateState_`),
  - accesseurs d'affectation : `set_` + nom de l'attribut sans le `_` final (*e.g.*, `set_max_number_of_player`),
  - accesseurs de consultation : nom de l'attribut sans le `_` final (*e.g.*, `max_number_of_player`),
  - constantes : `NOM_EXPLICITE` (*e.g.*, `VERSION_NUMBER`).
- **Pensez à utiliser les outils de développement comme le debugger `gdb` ou le profiler `valgrind`.**
- **N'hésitez jamais à chercher de la documentation par vous-mêmes sur le net !**
  Par exemple la STL est documentée ici : `http://en.cppreference.com/w/cpp`,
  et la SFML l'est ici : `http://sfml-dev.org/documentation/1.6/`.

Dans ce TP :

- Jeu de TicTacToe (morpion) graphique.

« Enfin ! Le projet du cours de M. Rauzy est bouclé ! » se dit un étudiant d'EISE4. Ce petit jeu de TicTacToe n'est vraiment pas grand chose, même pour un projet seul, mais ça a le mérite de bien fonctionner. Il est tout de même assez fier de son menu, qui est simple et sobre, mais extensible et facilement réutilisable pour d'autres projets. Il n'a plus qu'à écrire le rapport en insistant là dessus avant de pouvoir rendre tout ça et en avoir enfin fini avec ce cours. Mais avant, il mérite bien une petite pause, et puis il a besoin d'un café avant de continuer...

Cinq minutes plus tard, il est de retour dans la salle informatique. « Tiens, on dirait que j'ai oublié de verrouiller ma session, étrange. » pense-t-il. Il ouvre son éditeur de texte pour commencer à rédiger le rapport : « Bon, commençons par la partie sur l'implémentation du menu du jeu, vu que c'est ça qui m'a le plus amusé dans ce projet. ». Il commence à rédiger puis souhaite faire référence à quelques lignes de code en particulier dans son rapport. Pour en retrouver les numéros exacts, il ouvre son fichier `menu.cpp`.

HORREUR. Le contenu de toutes les fonctions du fichier à disparu !

Il ouvre un autre fichier, et là aussi, le contenu des fonctions a disparu. Il en va de même pour tous les `.cpp` de son projet : à part sa fonction `main`, le code de chacune des fonctions qui l'a écrite se résume a une ligne vide. Les fichiers d'en-tête ne semblent pas avoir été altérés, maigre consolation, mais c'est déjà ça...

Qui est l'abruti qui a bien pu lui faire cette blague stupide pendant qu'il cherchait son café ? Si il le trouve...

Mais bref, il doit rendre le projet dans quelques heures, et vu le temps qu'il estime que lui prendra l'écriture du rapport, il calcul qu'il lui reste un peu moins de 4h pour tout refaire !

Il est conseillé de lire intégralement l'énoncé avant de commencer. Vous n'êtes pas obligé de suivre l'ordre des questions qui sont là pour vous servir de guide.

Exercice 0.

Récupération des fichiers nécessaires.

1. Pensez à organiser correctement votre espace de travail, par exemple tout ce qui se passe dans ce TP devrait être dans `~/cpp/NOM/`.

2. Récupérez les fichiers nécessaires au TP sur la page du cours.

3. Une fois que vous l'avez extrait de l'archive (`tar xzf tictactoe.tgz`), renommez le répertoire `tictactoe` en votre nom (par exemple `Dupond`). Si vous ne le faites pas tout de suite, pensez à le faire avant de rendre votre TP.

Exercice 1.

Découverte du projet.

1. Il reste toujours une version compilé du jeu dans le répertoire.

→ Lancez là pour voir ce que vous devez (à peu près) reproduire.

2. Prenez connaissance des fichiers sources disponibles. Les modules sont les suivants :

- *Main.* Point d'entrée du programme. Lance simplement le jeu en appelant la méthode `launch` d'une instance de la classe `TicTacToe`. L'en-tête défini des constantes qui seront utilisées un peu partout.
- *TicTacToe.* Classe principale du jeu. Créé la fenêtre graphique et gère la boucle d'évènement et l'affichage en faisant appelle aux autres modules du projet.
- *Menu.* Classe permettant de gérer le menu du jeu. Est notamment capable de gérer autant de bouton que souhaité en les ajoutant simplement à une liste.
- *Button.* Représente les boutons utilisés par le menu.
- *Board.* Gère le plateau de jeu, c'est à dire les 9 cellules de la grille 3×3.
- *Cell.* Représente les cellules de la grille de jeu.

→ Lisez le code source des fichiers, notamment des fichiers en-tête afin d'essayer de comprendre au mieux l'organisation du code.

Pour la suite du TP, n'hésitez pas faire les modifications nécessaires par exemple du `Makefile` afin de pouvoir régulièrement compiler ce que vous avez déjà fait pour vérifier que cela fonctionne.

Exercice 2.

Boucle d'affichage et d'évènements.

1. → Commencez par remplir le constructeur et le destructeur de la classe `TicTacToe`. Attention, on souhaite que la fenêtre du jeu ne soit pas redimensionnable.

2. → Créez une boucle d'affichage et d'évènements basique (affichage d'une fenêtre blanche, gestion de la fermeture de la fenêtre) dans la fonction `launch`.

Exercice 3.

Les boutons du menu.

1. Le constructeur de `Button` prend trois arguments : le texte du bouton, le nom de l'action associée, et sa position dans le menu (1 pour premier, 2 pour deuxième, etc.).

Les éléments du menu utilisent la police par défaut en noir dans la taille par défaut (30px). Il doivent être centrés et s'afficher les uns au dessus des autres dans la moitié basse de la fenêtre.
Fonctions utiles de la SFML : `sf::String::GetRect` et `sf::String::SetPosition`.

Une fois positionné, vous pouvez récupérer le rectangle dans lequel un bouton est placé en réappelant `GetRect`. Cela pourra être utile plus tard pour détecter si la souris est sur le bouton ou non.

→ Écrivez le constructeur de `Button`.

2. → Écrivez la fonction `contains`, qui renvoie vrai si les coordonnées qu'on lui passe en argument sont sur le bouton, faux sinon.

3. → Écrivez la fonction `getAction`.

4. → Écrivez les fonctions `display` et `displayActive`. La seconde affiche un rectangle de couleur avant d'appeler la première qui se contente d'afficher le texte.

Exercice 4.

Le menu, première partie.

1. → Écrivez le constructeur de `Menu`. Celui-ci doit entre autre initialiser le texte de titre du jeu (dans l'exemple, c'est la police par défaut en noir avec une taille de 42px), et le positionner de manière à ce qu'il soit centré. Il doit aussi créer les deux boutons "New Game" et "Exit" et les mettre dans sa liste de boutons.

2. → Écrivez la fonction `displayButtons` qui affiche les boutons en appelant leur méthode `display` sauf pour le bouton actif (celui dont l'action est égale à `current_button_`) pour lequel vous appellerez `displayActive`.

3. → Écrivez la fonction `display` qui affiche le titre puis affiche les boutons.

4. → Écrivez la fonction `setCurrentButton` qui prend en argument les coordonnées de la souris et rend actif le bouton sur lequel elle se trouve, ou aucun, le cas échéant.

5. → Écrivez la fonction `getAction`.

Exercice 5.

Affichage et gestion du menu.

1. → Créez une instance de `Menu` dans la fonction `launch`, et écrire le code qui va permette de l'afficher.

2. → Lors d'un mouvement de la souris, mettez à jour le bouton actif du menu.

3. → Lors d'un clic de souris récupérez l'action correspondante au bouton actif du menu et agissez en conséquence : ne rien faire si c'est `NoAction`, fermer le jeu si c'est `Exit`, et on s'occupera plus tard de l'action `NewGame`.

Exercice 6.

Les cellules de la grille de jeu.

1. Le constructeur de `Cell` prend en argument la position (i, j) sur la grille de 3×3 de la cellule et créé en fonction de ça un rectangle de couleur à afficher (`sprite_`) et un `sf::IntRect` permettant de facilement vérifier si la souris est sur ce rectangle, comme pour les boutons du menu. N'oubliez pas d'initialiser `player_`.

→ Écrivez le code du constructeur de `Cell`.

2. → Écrivez la fonction `contains`.

3. → Écrivez la fonction `getCenter`, qui retourne un `sf::Vector2i` contenant les coordonnées du centre de la cellule.

4. → Écrivez les fonction `play` et `getPlayer`, qui sont les accesseurs du membre `player_`.

5. → Écrivez la méthode `display`.

Exercice 7.

La grille de jeu, première partie.

1. Le constructeur de `Board` créé les 9 cellules de la grille de jeu et initialise `current_cell_` à une valeur impossible, par exemple `-1`.

→ Écrivez le constructeur et le destructeur de `Board`.

2. La fonction `setCurrentCell` doit mettre à jour `current_cell_` pour être l'indice de la cellule sur laquelle la souris se trouve, mais seulement si celle-ci est encore jouable.

→ Écrivez cette fonction.

3. La fonction `play` joue un coup du joueur qu'elle reçoit en argument, mais seulement si `current_cell_` est valide. Elle renvoit un booléen qui indique si un coup a été joué ou pas.

→ Écrivez cette fonction.

4. Les fonctions `drawX` et `drawO` servent respectivement à tracer une croix ou un cercle qui soit centré sur le point dont les coordonnées sont fournies en argument. Le dernier argument est l'opacité de la couleur, il servira pour afficher de manière semi-transparente les coups pas encore joués au survol des cellules.

→ Écrivez ces fonctions.

5. La fonction `displayGrid` affiche la grille de jeu et les coups déjà joués.

→ Écrivez cette fonction.

6. La fonction `showNextMove` affiche en semi-transparent le possible prochain coup du joueur courant, si la souris est sur une case jouable.

→ Écrivez cette fonction.

Exercice 8.

Affichage et gestion de la grille de jeu.

1. Dans la fonction `launch`, il y a besoin de connaître : le joueur courant, si un joueur a gagné et si oui lequel, le nombre de coups qui ont été joués. Il y a aussi bien sûr besoin d'une instance de `Board`.

→ Dans le cas d'une action `NewGame` réinitialisez toutes ces variables afin de mettre le jeu dans un état de début de partie.

2. → Lors d'un mouvement de la souris, si une partie est en cours, alors mettez à jour la cellule active plutôt que le bouton du menu.

3. → Dans le cas un clic de souris, si une partie est en cours, alors faites jouer le joueur si la souris est bien sur une case jouable, et mettez à jour l'état du jeu.

4. → Au niveau de l'affichage, si une partie est en cours, alors affichez la grille de jeu et le possible prochain coup plutôt que le menu.

Exercice 9.

La fin de partie.

1. → Écrivez la fonction `Board::getWinner` qui renvoie le joueur gagnant. Prenez soin de le faire intelligemment.

2. → Écrivez la fonction `Menu::displayWinner` qui affiche le gagnant ou précise si il y a match nul dans la moitié supérieure de la fenêtre et affiche le menu dans la moitié inférieure.

3. → Dans la fonction `TicTacToe::launch`, appelez la fonction `displayWinner` si il y a un gagnant ou si la partie est terminée.

Exercice 10.

Perspectives.

1. On voit bien que l'étudiant a eu une bonne idée pour son menu qu'il dit "réutilisable". En pratique, son menu est certes facilement adaptable à d'autres utilisations, mais il n'est pas réutilisable tel quel.

Essayez d'imaginer comment vous feriez pour rendre le menu vraiment générique. On voudrait pouvoir personnaliser l'affichage des boutons et les gérer (ajout, suppression, ordre d'affichage), sans jamais avoir à modifier la classe menu : on veut se placer en tant qu'utilisateur de la classe, comme si il s'agissait d'une bibliothèque telle que la SFML.

→ Décrivez comment vous développeriez une telle bibliothèque de menu pour les applications basées sur la SFML.

2. → Sans nécessairement donner la stratégie d'une intelligence artificielle pour le TicTacToe, expliquez comment vous l'implémenteriez et quelles modifications du code du projet seraient nécessaires (combien de classes utiliseriez vous, quels seraient leurs rôles, avec quelles classes existantes ces classes interagiraient, etc.).

3. Bonus.

→ Conservez un compteur de score entre les deux joueurs et affichez le en même temps que le menu (+3 points par victoire et +1 point chacun en cas d'égalité).