INF1010

Programmation Orientée-Objet

Travail pratique #6
Interfaces Utilisateurs et exceptions

Objectifs :	Permettre à l'étudiant de se familiariser avec les concepts d'interfaces utilisateurs, d'exceptions et de variables de classe en C++.
Remise du travail :	Mardi 6 Décembre 2016, 8h
Références :	Notes de cours sur Moodle + http://doc.qt.io/
Documents à remettre :	La solution ainsi que les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires. Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Travail à réaliser

Pour votre dernière tache, les responsables design et gameplay vous demandent d'apporter la touche finale au jeu. Jusque-là nous ne n'avions que des affichages à la console. Cependant en ayant connaissances de vos super capacités, il vous a été demandé d'ajouter une interface graphique à tout ce que vous avez fait auparavant.

Au cours des laboratoires précédents, vous avez développé un jeu comportant des combats entre différentes créatures. Mais voir ces combats dans la console est un peu limité. Votre dernière tache va être d'apporter la touche finale au jeu, il est maintenant temps d'y intégrer une interface graphique ☺.

Une interface simple permettant de visualiser le magnifique jeu que vous avez développé tout au long de la session est presque terminée. Il ne manque plus que de rajouter quelques éléments manquant au programme. Votre travail consistera principalement à la gestion de **quelques signaux d'événements et d'exceptions**. L'interface fonctionnelle doit ressembler aux images ci-dessous. Sauf avis contraire, toutes les classes et fonctions sont fournies et terminées (Ne vous amusez pas à les modifier pour rien). Les sections à faire ou à modifier sont clairement indiquées dans le fichier fournis par le commentaire **« !!!!!! A COMPLETER !!!!!! »**.

Pour l'affichage des différents éléments, vous pouvez vous référer aux différentes images qui vous sont fournies dans la partie « Exemples d'exécution ».

Dans la liste des fichiers fournis, le fichier .pro est le projet Qt, il suffit de l'ouvrir avec Qt Creator.

IMPORTANT : N'oubliez pas d'exécuter « qmake » après l'ouverture du projet Qt et après l'ajout de nouveaux fichiers.

Note : Vous pourrez remarquer que l'encapsulation a été brisé sur certaines classes graphiques, cela a été fait pour un gain de temps et une implémentation plus rapide. Ce n'est pas à reproduire !

Attention : Vous devez aller chercher dans les differents fichiers les bons attributs a utiliser(notamment pour les classes graphiques)

Classe *ExceptionAttaqueEchouee*

Écrivez une classe d'exception simple ExceptionAttaqueEchouee qui hérite de l'exception standard *std ::runtime_error.* Cette exception personnalisée sera lancée lors de l'échec d'une attaque à cause du manque d'énergie de la créature

Classe Exception*EchecCapture*

Écrivez une classe d'exception simple ExceptionEchecCreature qui hérite de l'exception standard *std ::runtime_error.* Cette exception personnalisée sera lancée lorsque l'utilisateur essaye d'attraper une créature qu'il possède déjà.

Classe *ExceptionCreatureMorte*

Écrire une classe d'exception *ExceptionCreatureMorte* qui hérite de l'exception standard *std::runtime_error*. Cette exception personnalisée sera lancée lors de l'échec de l'attaque sur une créature car celle-ci n'a plus de point de vie.

Cette classe doit contenir l'attribut suivant :

- *compteur_* : un entier (int) statique privé permettant de compter le nombre de fois que l'exception *ExceptioCreatureMorte* est lancée.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètre recevant une chaîne de caractères (string) contenant le message de l'erreur. Le constructeur doit incrémenter la valeur du compteur.
- La méthode statique public *obtenirValeurCompteur()* qui retourne la valeur de *compteur_*.

Aide : Faites attention à l'initialisation d'un attribut statique

Classe *choixAttaque*

Partie à compléter :

Vous devez aussi compléter l'affichage des différentes attaques de votre créature :

- Vous devez positionner correctement les boutons attaques dans un GridLayout et le placer correctement pour respecter l'affichage suivant : *(Pour plus d'informations voir commentaires dans le code).* Pour cela vous devez compléter le constructeur de la classe *ChoixAttaque.*

voir la documentation

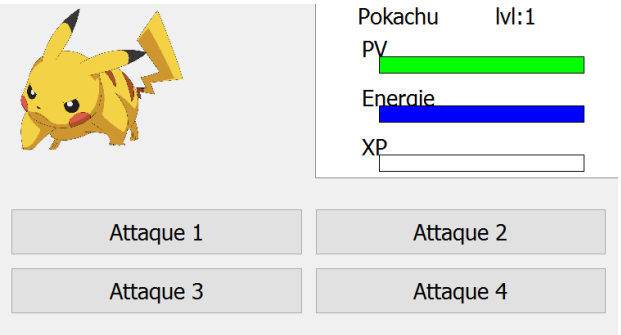

Figure 1. Affichage des attaques

Classe *Gamebay*

La classe *Gamebay* est la fenêtre principale de l'application. Elle hérite de la classe *QMainWindow* et comprend les attributs suivants, vous n'avez pas à modifier ces attributs :

- Quatres QLabel correspondant aux flèches de navigation
- Un widget Menu permettant l'affichage d'un menu avec différents QPushButton et QListWidget
- 2 QPushButton correspondant aux boutons centraux Select et Start
- Un widget choixAttaques_ qui va contenir les 4 boutons d'attaque de votre créature
- 2 Widgets AffichageInformationCreature qui vont permette d'afficher les données sur les créatures se combattant
  2 QLabel pour afficher les images des créatures se combattant
- Un booléen pour savoir si un combat est en cours
- Un pointeur sur Polyland
- Un pointeur sur votre créature en train de se battre
- Un pointeur sur la créature adverse

Seules les méthodes avec un crochet √ doivent être modifiées :

- Un constructeur par paramètres, recevant entre autres un pointeur Polyland, et qui initialise l'interface.
- √ Une méthode privée *setUI()* qui crée les différents éléments de l'interface. Vous devez modifier cette fonction pour ajouter la partie qui va permettre de sélectionner les attaques que votre créature va lancer. Veuillez-vous référer à la figure explicative (figure 2) et aux commentaires dans la méthode.
- √ Une méthode privée *setConnections()* qui connecte les évènements des éléments de l'interface aux méthodes *private slots*. Vous devez compléter les connexions suivantes :

- Un clic sur le bouton Start (*boutonStart_*) affiche le menu, un slot privé a déjà été implémenté pour vous (*gestionDuMenu()*)
- Un clic sur le bouton Attaques affiche les différentes attaques de votre créature lorsqu'un combat a lieu. Vous devez implémenter pour cela une slot privé *afficherAttaques(),* qui fera appel à au slot public *afficherAttaques(Creature* creature)* de la classe choixAttaque
- Une connexion entre le clique de chaque bouton attaque de *choixAttaque_* et la méthode *attaquerCreatureAdverse()* de la classe *Gamebay*. Cela va permettre de mettre à jour les points de vie de la créature recevant l'attaque et l'énergie de la créature lançant l'attaque.
- Un clic sur le bouton « Vos Creatures » (*boutonAffichageCreaturesDresseur_*) va afficher la liste des créatures que vous possédez (de l'attribut *hero_* de polyland). Le slot privé qui sera appelé ici est *afficherCreaturesDresseur().*

- √ Une méthode privée *afficherCreaturesDresseur ()* qui remplit l'objet *QListWidget* avec la liste des créatures que vous possédez (*Aide* : Vous pouvez vous aider de ce qui a déjà été implementee dans la Classe Menu.)
- Une méthode privée *afficherCreatures ()* qui remplit l'objet *QListWidget listeCreatures_* avec la liste des créatures que vous pouvez rencontrer dans Polyland
- √ La connexion du signal creatureAdverseVaincue() qui permettra d'afficher un bouton pour capturer la creature
- √ Une méthode (slot) privée *attaquerCreatureAdverse ()* qui va vous permettre d'attaquer la creature adverse. *(voir Figure 3)*
  - Cette méthode est sujette à deux types d'exceptions ! Vous devez gérer ces deux exceptions. **Voir les indications dans les commentaires de la méthode**.
  - Lorsque l'attaque échoué par faute d'énergie, une exception de type *ExceptionAttaqueEchouee* est lancée.
  - Lorsque l'attaque échoué car la créature adverse est déjà morte, une exception de type *ExceptionCreatureMorte* est lancée
  - Vous devez traiter uniquement ces deux exceptions et afficher des messages en conséquent, pour cela aider vous des figures 3
  - Cette méthode doit émettre un signal *creatureAdverseVaincue()* lorsque la créature adverse n'a plus de point de vie suite à l'attaque de votre créature
  - Aide : Pour afficher une fenetre de message, utilisez la classe *QMessageBox*
- √ Le signal *creatureVaincue()* doit être émis lorsque vous sélectionnez une créature n'ayant plus de point de vie dans le slot *changerCreature()*.
- √ Le signal *creatureAdverseVaincue()* vaincue qui devra être émis lorsque la créature adverse n'a plus de point de vie suite à l'attaque de votre créature
- √ Le slot *attraperCreatureAdverse()* qui doit permettre de capturer la créature adverse une fois vaincue si le dresseur(*hero_)* ne possède pas la créature, sinon cette méthode lève une exception que vous devez gérer comme sur la *figure5.*

Classe Dresseur

La méthode suivante doit être modifiée :

- La méthode *ajouterCreature* doit prendre en compte le lancement d'exception lorsque c'est nécessaire

Classe Creature

La méthode suivante doit être modifiée :

- La méthode attaquer doit prendre en compte le lancement d'exception :
  - Si la créature attaquant n'a plus d'énergie, une exception de type ExceptionAttaqueEchouee est lancée
  - Si la créature adverse n'a plus de point de vie, une exception de type *ExceptionCreatureMorte*

Exemple d'exécution

Voici quelques captures d'écran vous permettant de voir le comportement des différents éléments de la GUI.

Le résultat final doit pouvoir avoir un comportement similaire à ce qui suit :

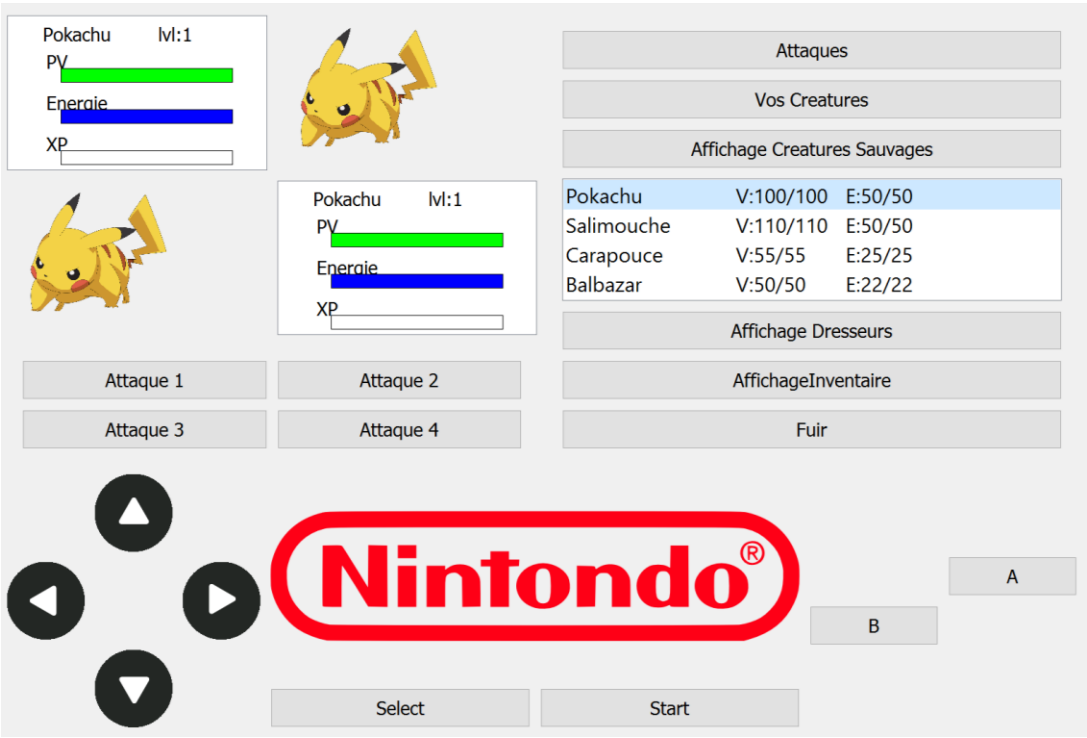

Figure 2. Affichage de la partie fenêtre

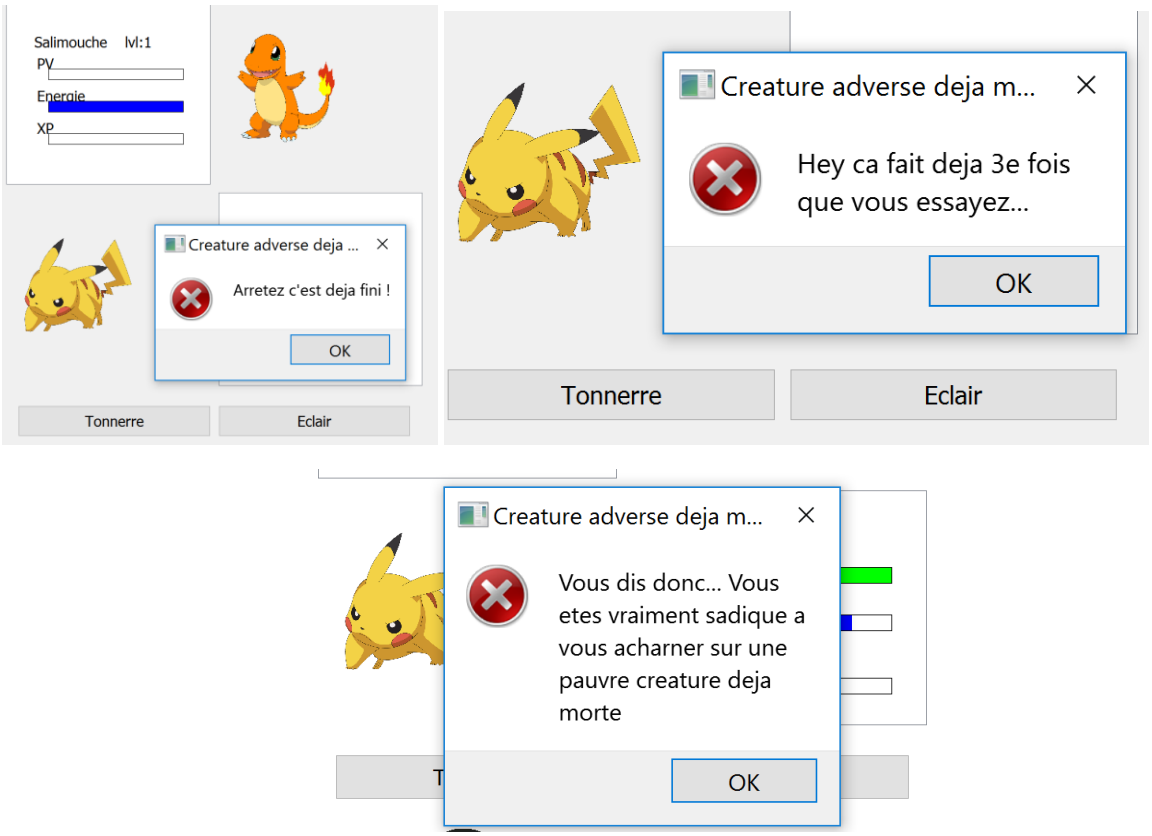

Figure 3. Affichage des différentes fenêtres en fonction du nombre d'exception

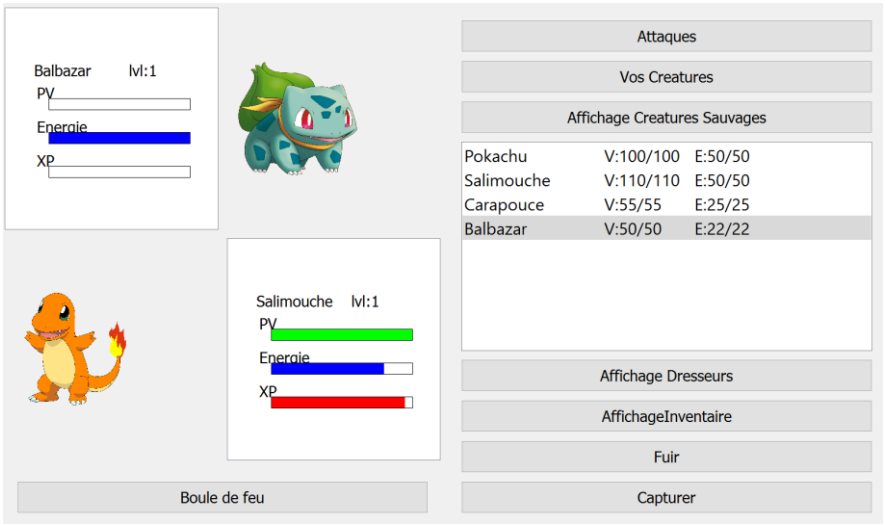

Figure 4. Exemple de la possibilité de capturer une créature vaincue

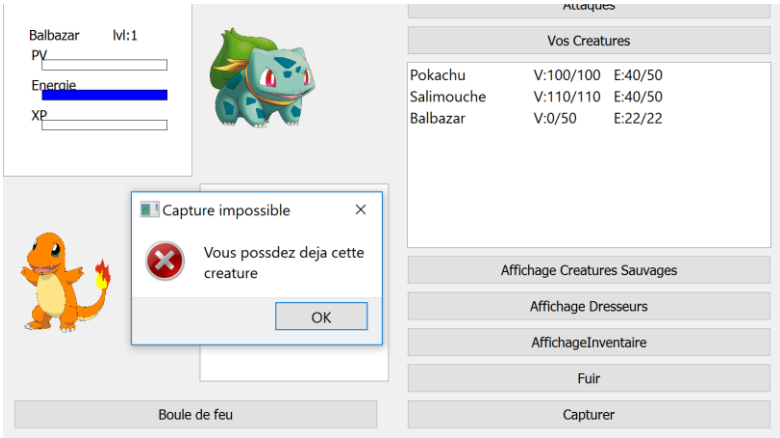

Figure 5.Exemple d'affichage pour une créature déjà capturée

Correction

La correction du TP6 se fera sur 20 points. Voici les détails de la correction :

- (3 points) Compilation du programme;
- (3 points) Exécution du programme;
- (4 points) Comportement exacte des méthodes et de l'interface;
- (1 point) Utilisation adéquate des Widgets Qt (boutons, boîtes de message)
- (4 points) Utilisation adéquate des signaux et des connexions en Qt;
- (3 points) Gestion adéquate des exceptions;
- (1 point) Utilisation adéquate des éléments statiques de classe;
- (1 point) Documentation du code;