Développement
en environnement de base de données

Projet 2

Table des matières

Énoncé

Vous devez concevoir et réaliser un jeu vidéo similaire au légendaire Blockade de type Snake game.

Objectif général

Ce deuxième laboratoire vise la réalisation d'un projet complet visant la pratique des éléments suivants :

- programmation orientée objets;
- développement modulaire;
- programmation en C++ moderne;

Objectifs spécifiques

Plus spécifiquement, ce projet vise ces considérations :

- conception orientée objets mettant de l'avant la notion d'encapsulation;
- développement favorisant la modularité et la réutilisabilité;
- utilisation de 2 patrons de conceptions :
    - patron de séquençage,
    - singleton,
    - automate fini;
- mise en place d'une expérience utilisateur satisfaisante;
- programmation moderne en C++.

Présentation générale du projet

Le logiciel réalisé doit offrir :

- un environnement de jeu simple, mais complet;
- des interfaces utilisateurs permettant de bien guider le joueur dans les différentes sections du logiciel;
- un jeu qui soit visuellement agréable sans tomber dans le piège de considérations esthétiques;
- une expérience de jeu satisfaisante considérant ce type de jeu.

Vous avez plusieurs contraintes techniques à respecter, mais vous avez tout de même une grande liberté pour donner la saveur que vous désirez au jeu.

Contraintes

Vous devez respecter ces contraintes:

- Vous devez travailler en équipe (3 équipes de 4 étudiants et 2 équipes de 5 étudiants). La charge de travail doit correspondre à la taille de l'équipe. Aucun étudiant ne peut travailler seul!
- Il est très important que tous les membres de l'équipe travaillent équitablement, car l'évaluation finale tient compte de plusieurs critères, dont la répartition de la tâche de travail.
- Vous devez utiliser :
    - le langage C++;
    - Visual Studio sous Windows.

- Vous devez faire une conception orientée objets soignée de votre projet.
- L'utilisation des patrons de conception présentée est obligatoire :
    - le déroulement de la partie doit utiliser le patron par séquençage (jeu vidéo);
    - la logique interne de l'état du logiciel doit être réalisée par un automate fini (états);
    - un outil de fichier journal doit être réalisé par un *singleton*.
- Vous devez faire un jeu en mode console. Sans y être contraint, vous pouvez utiliser la petite librairie fournie pour faciliter l'utilisation de la console.
- La date de remise est non négociable.

Présentation détaillée du jeu

Jeu de type « snake game »

Concepts fondamentaux

Il existe plusieurs variantes de ce type de jeu, mais ils présentent tous un même concept basé autour d'un *serpent*[1]. Voici les éléments communs :

- le(s) joueur(s) contrôle(nt) un serpent;
- un serpent avance inexorablement et sans possibilité de s'arrêter;
- un serpent peut se déplacer :
    - à l'avant,
    - à sa gauche,
    - à sa droite;
- la partie termine lorsqu'un serpent touche :
    - les bords de l'aréna de jeu;
    - son propre corps ou le corps d'un autre serpent

Scénario original

Dans sa version originale (*Blockade*), le jeu se déroule ainsi :

- deux joueurs jouent simultanément et contrôlent chacun un serpent;
- progressivement, les serpents s'allongent avec le temps;
- le joueur gagnant est celui qui reste le plus longtemps sans toucher à l'autre où aux bords de l'aréna de jeu.

Scénario le plus répandu

Dans sa version la plus fondamentale, la partie se déroule ainsi :

- un seul joueur contrôle un seul serpent;
- une pastille apparait à l'écran et le serpent est invité à la manger;
- lorsque la pastille est mangée, le serpent s'allonge d'un élément et le joueur accumule un point supplémentaire;
- la partie termine lorsque le serpent touche à son corps où aux bords de l'aréna de jeu;
- l'objectif étant de faire le maximum de points et de mettre son nom sur le tableau d'honneur.

[1] Dans le texte qui suit, on utilisera le terme *serpent* au sens générique.

Votre scénario

Pour ce projet, vous pouvez faire le scénario que vous voulez.

Néanmoins, il est obligatoire de considérer ces aspects :

- vous devez respecter les points indiqués dans la section concepts fondamentaux plus haut;
- vous devez faire grandir le serpent;
- vous devez faire accélérer le serpent.

Ajout personnel

Comme pour le laboratoire précédent, 20% du projet est attribué à l'ajout d'une fonctionnalité supplémentaire.

Voici quelques suggestions :

- faire rapetisser le serpent;
- décélérer le serpent;
- vous pouvez ajouter d'autres items participants :
    - des obstacles,
    - des éléments décoratifs,
    - d'autres entités « intelligentes »;
- le faire passer au-dessus ou au-dessous de la surface de jeu pour éviter une collision avec un autre serpent;
- vous pouvez le faire téléporter dans la section opposée de l'aréna de jeu;
- ...

Les points sont attribués selon ces trois critères :

- pertinence;
- niveau de difficulté;
- qualité de la réalisation.

Il est fortement encouragé d'en discuter avec l'enseignant.

Navigation dans les différentes parties du jeu

Au démarrage du logiciel, vous devez prendre en charge l'utilisateur et, à travers plusieurs invites, lui offrir plusieurs possibilités. Vous devez offrir minimalement ces neuf activités :

- Accueil :
    - contextualiser le joueur;
    - offre 3 actions possibles :
        - Options,
        - Démarrer,
        - Quitter;
- Options :
    - permets une paramétrisation du logiciel;
    - vous n'êtes pas obligé d'avoir des options (c'est optionnel) toutefois, vous devez offrir la possibilité de consulter les instructions du jeu

    - offre 2 actions possibles :
        - Accueil;
        - Instructions;
- Instructions :
    - permets de voir les règles du jeu;
    - offre 1 action possible :
        - Options;
- Quitter :
    - demande une confirmation de quitter le jeu;
    - offre 2 actions possibles :
        - Accueil
        - Fin
- Démarrer :
    - demande au joueur d'appuyer sur une touche lorsqu'il est prêt pour jouer;
    - offre 1 action possible :
        - Jouer
- Jouer :
    - une partie est en cours et le joueur joue au jeu;
    - offre plusieurs actions possibles :
        - toutes les actions liées au jeu;
        - lorsqu'une partie est terminée, va à l'activité PartieTerminée
        - Pause
- Pause :
    - le joueur est en train de jouer, mais la partie est momentanément arrêtée;
    - offre 3 actions possibles :
        - Jouer (retour au jeu en cours)
        - Accueil
        - Quitter
- PartieTerminée :
    - affiche les informations relatives à la fin de la partie;
    - offre 2 actions possibles :
        - Accueil
        - Quitter
- Fin
    - le logiciel termine et quitte.

Chacune de ces activités doit offrir une interface usager adéquate. La modularité de votre code est essentielle ici.

Patrons de conception

Singleton

Le singleton est l'un des patrons de conception les plus simples. Il consiste à garantir une seule instance d'une classe pour toute une application.

La librairie `Console` en offre un exemple. En effet, la programmation Windows en mode console possède une contrainte importante : il ne peut y avoir qu'une seule console par application. Aussi, la classe `Console` donnée représente un « superviseur » de la console. Dans ce contexte, il serait fautif d'avoir plusieurs « superviseurs ».

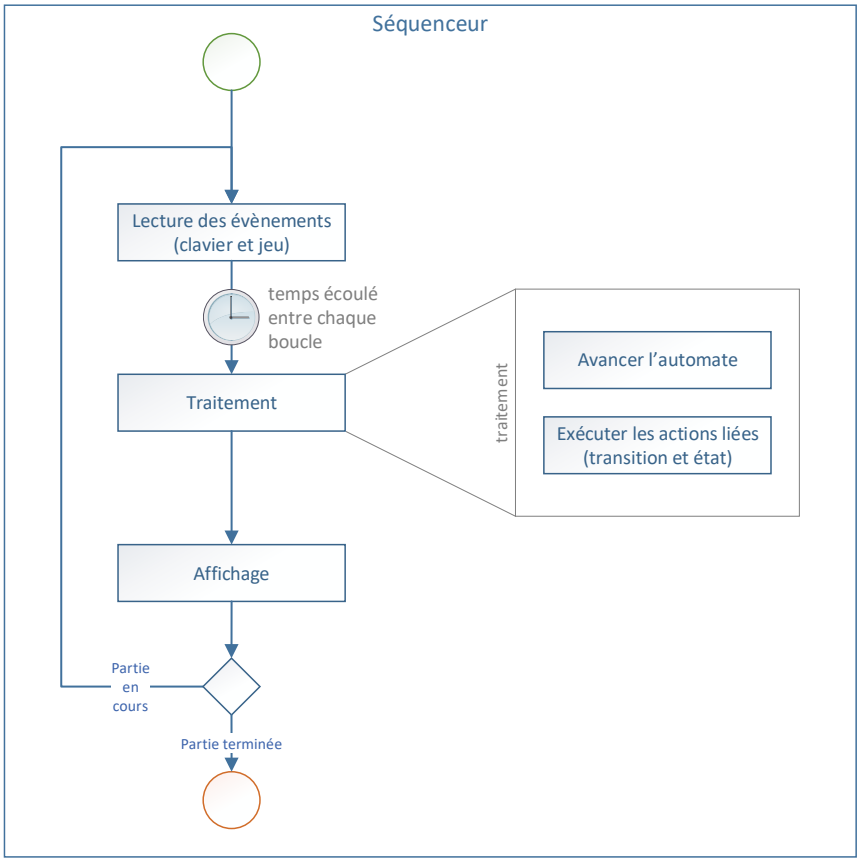
Ainsi, la réalisation d'un singleton garantit que l'usage de cette classe passera par une instance exclusive tout au long de la durée de vie du logiciel.

Pour ce deuxième projet, vous devez produire votre propre singleton visant la gestion d'un outil de fichier de journal décris plus loin.

Patron de séquençage

Le patron de séquence est utilisé dans plusieurs domaines où il est important de maintenir une mise à jour constante entre des intrants, un modèle et sa vue. Le patron n'est pas lié au MVC en soit mais peut y être relié.

Comme pour tous les patrons de conception, il existe plusieurs variantes plus ou moins sophistiquées. Dans le cadre de ce laboratoire, vous devez réaliser une infrastructure permettant la stratégie présentée sur le schéma suivant.



L'idée est relativement simple :

- on tourne en boucle tant que le logiciel s'exécute;

- la lecture des évènements consiste à faire l'acquisition de tous les états du système via l'interface utilisateur (en entrée) – clavier et les états du jeu lorsqu'il est en cours;
- le traitement réalise un pas d'avancement du jeu (un « tic »);
- l'affichage met à jour l'interface utilisateur (en sortie).

L'horloge sert à déterminer le temps qui s'est écoulé entre chaque boucle. Cette information est cruciale pour la tâche de mise à jour de la simulation. Cette stratégie permet, via un mécanisme simple, de tenir compte des grandes variabilités qui peuvent exister entre chaque opération du logiciel. C'est la tâche de traitement qui doit recevoir et traiter cette information adéquatement.

Note : Ce patron de séquençage est très simple, mais n'est pas optimal. Néanmoins, c'est celui que vous devez implémenter sans tenter d'en réaliser un autre plus sophistiqué.

Automate fini

Un automate fini ou un automate à états finis (ou « *finite states machine* » en anglais) est une modélisation mathématique permettant de simplifier grandement la résolution de problème récurrent en informatique. Il permet de modéliser un système par un ensemble d'états et de transitions. Chaque état est relié à des transitions conditionnelles. Un état courant passe d'un état à l'autre suivant la conformité des critères des transitions liées à l'état courant.

Il existe plusieurs modèles d'automate. Nous utiliserons un modèle simple auquel on ajoute la possibilité de faire une action pour chaque état et pour chaque transition.

Considérant tous ces éléments, voici la définition d'un automate fini :

- un automate possède :
  - n_s états — n_s est le nombre d'états et $n_s > 0$
  - 1 état initial, e_0 — au démarrage de l'automate, l'état courant est l'état initial, $e_0 \neq$ nul
  - 1 état courant, e_c — l'état courant indique l'état actuel de l'automate, $e_c \neq$ nul
  - une fonction d'initialisation — permet de réinitialiser l'automate
    ```
    void initialize();
    ```
  - une fonction d'avance — fait la résolution d'un pas de simulation à partir d'un évènement qui est évalué
    ```
    void process(Event & event);
    ```
- un état possède :
  - n_t transitions — n_t est le nombre de transitions et $n_t > 0$
  - une fonction évaluant si une transition est effective
    ```
    Transition* isTransiting(Event & event);
    ```
  - une action optionnelle
    ```
    void (*action)();
    ```
- une transition possède :
  - une fonction évaluant la condition de transition
    ```
    bool isTransiting(Event const & event);
    ```
  - un état suivant, e_s — cet état indique vers quel état la transition a lieu
  - une action optionnelle
    ```
    void (*action)();
    ```

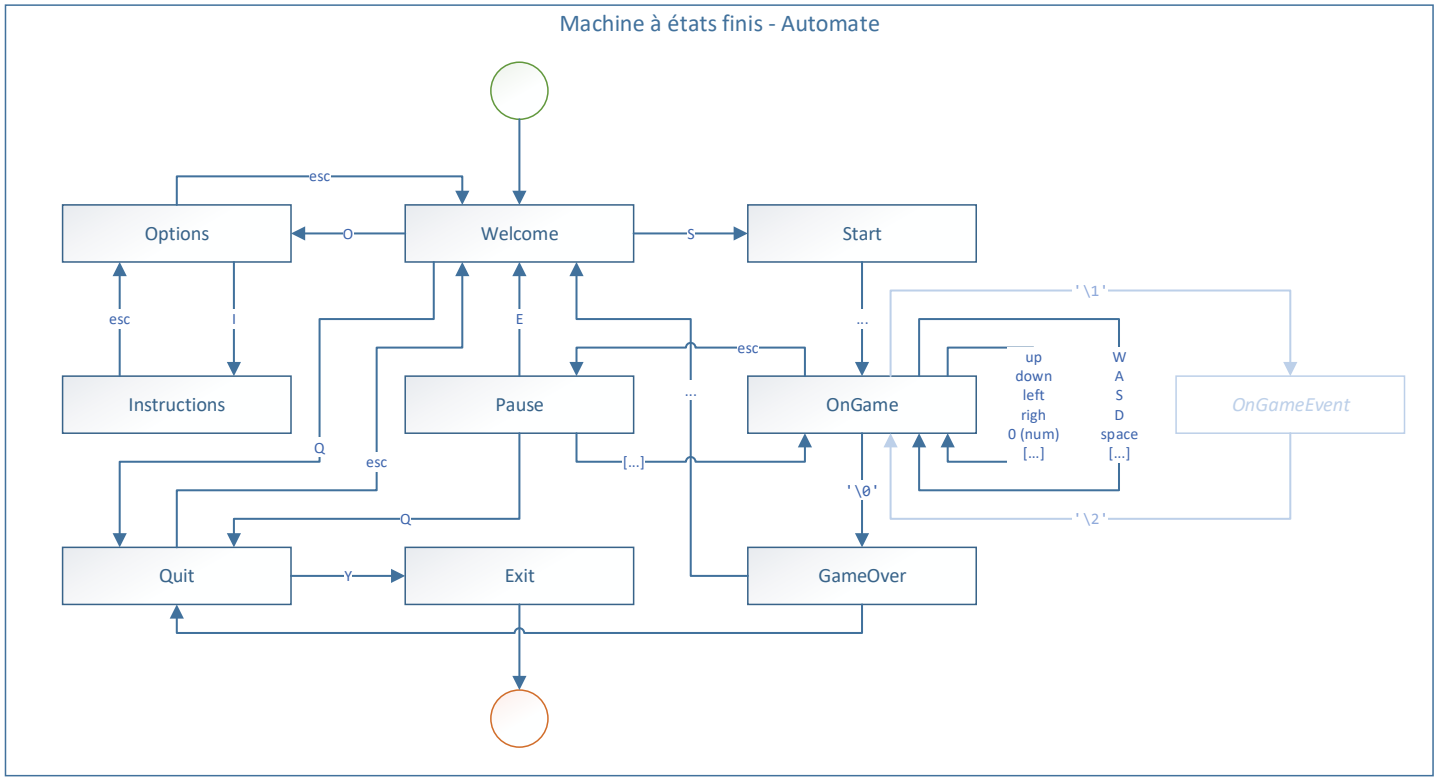
Voici le pseudo code des fonctions fondamentales `State::isTransiting`, `FSM::init` et `FSM::process` (`FSM` est l'acronyme pour « *Finite State Machine* ») :

```
// Fonction recherchant si un état transite selon un évènement donné
Transition* State::isTransiting(Event & event)
{
	for (toutes les transitions) {
		if (transition.isTransiting(event)) {
			return transition;
		}
	}

	return nullptr;
}

// Fonction initialisant l'automate
void FSM::initialize()
{
	ec = e0;
}

// Fonction réalisant un tic de simulation
void FSM::process(Event & event)
{
	// on recherche dans toutes ses transitions la première
	// qui accepte le event
	Transition * t = ec.isTransiting(event);

	// s'il y a une transition :
	//    1. on effectue le changement d'état
	//    2. on exécute l'action de la transition
	if (t != nullptr) {
		ec = t->es;
		t->action();
	}

	// on execute l'action de l'état
	ec->action();
}
```

En C++, on implémente souvent l'état initial et l'état courant par des pointeurs. Ainsi, il est possible de connaître l'état courant de l'automate par le pointeur courant en tout temps.

Note : La réalisation d'un automate générique comme celui présenté demande de la réflexion et du travail. Néanmoins, une conception adéquate pourra en faire une petite librairie très utile dans plusieurs projets. Même s'il est beaucoup plus facile de faire un automate avec des switch/case, vous devez réaliser un modèle d'automate comme celui qui vous est présenté.

Le schéma suivant présente la logique à mettre en place :



Fichier de journalisation

Un outil de journalisation (ou simplement un « log file ») permet de sauvegarder des informations lors du déroulement d'une application. Cet outil essentiel utilisé partout dans l'industrie permet de créer des données persistantes sur les activités réalisées par l'application. Voici les fonctionnalités que vous devez mettre en place :

- La classe ne peut pas être instanciée, un objet est uniquement accessible via un singleton.
- La conception de cette classe est votre responsabilité, mais il est attendu que son usage soit des plus simples et efficace tout est considérant son usage potentiel dans n'importe quel projet.
- Votre solution doit offrir les fonctions utilitaires suivantes :
    - configuration
        - détermine le fichier de sortie
        - détermine si les éléments suivants sont ajoutés à chaque évènement :
            - date
            - heure
    - inscription au journal (log)
        - inscription d'un texte en spécifiant l'une de ces catégories :
            - info
              représente un évènement naturel et l'évolution du système
            - trace
              représente des évènements importants non-système (souvent lié à des actions utilisateurs)

- debug
  représente des évènements importants système (souvent lié à des tests internes du système)
- warning
  représente une situation inattendue du système, mais laquelle l'application peut continuer
- error
  représente une erreur du système, mais laquelle l'application peut continuer
- fatal
  représente une erreur sévère du système et pour laquelle le comportement de l'application est dorénavant indéterminé

Voici un exemple de fichier pouvant avoir été produit avec votre outil :

```
[2020.07.25 13:27:45] INFO    : Démarrage de l'application
[2020.07.25 13:28:23] WARNING : Fichier ressource manquant (imageTitle.jpg)
[2020.07.25 13:29:07] INFO    : Image par défaut utilisée
[2020.07.25 13:29:54] DEBUG   : arraySize = 32
[2020.07.25 13:30:15] INFO    : Connexion à la base de données
[2020.07.25 13:30:41] DEBUG   : arraySize = 78
[2020.07.25 13:32:27] INFO    : L'application quitte
```

Outils fournis

Génération d'un texte représentant la date et l'heure

Le C++ est plutôt mal outillé pour produire des informations sur la date et d'heure. Par conséquent, voici une fonction vous permettant de retourner la date et l'heure courante.

```
#include <ctime>
#include <sstream>
#include <string>
#include <iomanip>

std::string timeStampString(std::string const & format = "%Y.%m.%d %H:%M:%S")
{
	std::stringstream stream;
	std::time_t time{ std::time(nullptr) };
	std::tm tm;
	localtime_s(&tm, &time);
	stream << std::put_time(&tm, format.c_str());
	return stream.str();
}
```

Librairie EWA

Cette petite librairie offre la possibilité de créer une application Windows en utilisant une fonction de rappel (« *callback function* »). La librairie offre plusieurs fonctions de dessin sur le canevas de l'application et la possibilité de connaître l'état du clavier. La souris n'est pas gérée avec cette librairie.

La librairie EWA possède sa propre documentation dans le fichier donné `EWA.zip`. Voir le fichier : `\dev\doc\html\index.html`.

Rapport

Vous devez produire un rapport correspondant à ceci :

1. Vous devez indiquer clairement qui sont les étudiants ayant travaillé sur le projet.
2. Vous devez répondre à ces questions :
    1. Patrons de conception :
        a. Quelles sont les classes qui s'occupent de faire la gestion de l'automate?
        b. Quelle est la classe implémentant l'outil de « log »?
    2. Quelle structure de données avez-vous utilisée pour maintenir le corps du serpent? Pourquoi?
    3. Avez-vous utilisé les notions d'héritage et de polymorphisme dans votre solution? Pourquoi? Si oui, où?
3. Finalement, vous devez présenter votre ajout personnel :
    1. Quelles étaient les intentions initiales, qu'est-ce que fonctionnent bien et qu'est-ce qui fonctionnent moins bien.
    2. Selon vous, quelle note (sur 10) méritez-vous pour ces trois critères :
        a. pertinence
        où 0 est insignifiant
        b. difficulté
        où 0 est trivial
        c. qualité de l'implémentation
        où 0 correspond à un travail dont vous ne parlerez pas en entrevu

Le fichier texte doit être déposé dans le dossier doc dans votre solution.

Stratégie d'évaluation

L'évaluation se fera en 2 parties. D'abord, l'enseignant évaluera le projet remis et assignera une note de groupe pour le travail. Ensuite, chaque équipe devra remettre un fichier Excel dans lequel sera soigneusement reportée une cote représentant la participation de chaque étudiant dans la réalisation du projet. Cette évaluation est faite en équipe et un consensus doit être trouvé.

Une pondération appliquée sur ces deux évaluations permettra d'assigner les notes finales individuelles.

Ce projet est long et difficile. Il est conçu pour être réalisé en équipe. L'objectif est que chacun prenne sa place et que chacun laisse de la place aux autres.

Ainsi, trois critères sont évalués :

- **participation** (présence en classe, participation active, laisse participer les autres, pas toujours en train d'être sur Facebook ou sur son téléphone, concentré sur le projet, pas en train de faire des travaux pour d'autres cours, …)
- **réalisation** (répartition du travail réalisé : conception, modélisation, rédaction de script, documentation, …)
- **impact** (débrouillardise, initiative, amène des solutions pertinentes, motivation d'équipe, …)

Remise

Vous devez créer un fichier de format `zip` dans lequel vous insérez :

- votre solution Visual Studio **bien nettoyée** `solution.zip`
- votre schéma de conception `schema.pdf`
- votre rapport `rapport.txt`
- le fichier Excel rempli sur la participation active des membres du groupe `evaluation.xlsx`

Vous devez remettre votre projet une seule fois sur Lea après avoir nommé votre fichier :

`NomPrenomEtudiant1_NomPrenomEtudiant2[_NomEtudiantN].zip`