Programmation Orientée Objet

Projet 2024-2025 – **Falling Sand Simulation**

1 Consignes générales

1.1 Objectif du projet

L'objectif est de renforcer et de mettre en pratique les concepts fondamentaux de la Programmation Orientée Objet (POO) à travers le développement d'un projet en C++. A l'issue de ce projet vous devrez être capable de :

- concevoir les classes d'une application simple en utilisant les relations d'agrégation, d'association et d'héritage,
- coder en C++ et tester les différentes classes dans une démarche ascendante pour arriver à la réalisation d'une application fonctionnelle, interactive et conviviale,
- utiliser à bon escient les structures de données de la STL `string`, `vector` et `list`.

1.2 Travail demandé

Le travail demandé concerne la conception et le développement d'un jeu de simulation graphique qui sera décrit plus en détail dans la partie 2 de ce document. Le développement devra être réalisé en C++ et devra mettre en pratique les concepts de la POO vus en cours. Ainsi, le code devra être structuré en **plusieurs classes** spécifiques au projet avec des **relations** permettant de les organiser et de les faire collaborer (agrégation, héritage et association). Il devra également privilégier l'utilisation des **structures de données** de la STL vues en cours.

Vous devrez rendre **un rapport** au format PDF ainsi qu'une archive ZIP contenant tous **les codes sources** c'est à dire les fichiers `.h`, `.cpp`, makefile ou solution et éventuels fichiers supplémentaires nécessaires à la compilation.

Le rapport devra comporter :

- 1 page maximum de spécifications présentant l'interface utilisateur et décrivant les principales interactions possibles avec l'utilisateur,
- 2 pages maximum de conception présentant le diagramme des classes et une présentation synthétique du rôle des principales classes et du choix des relations que vous avez utilisées,
- 1 page maximum indiquant la répartition des rôles dans le binôme et précisant les classes ou méthodes dont chacun s'est occupé de réaliser.

Les codes sources devront :

- être correctement découpés c'est à dire un fichier `.h` et un fichier `.cpp` par classe,
- comporter du **code lisible** respectant les **conventions de codage** vues en cours,
- être **soigneusement commentés** notamment lors de la déclaration des classes et des méthodes,
- comporter le **nom de l'auteur** pour chaque fichier ou le cas échéant pour chaque méthode.

1.3 Evaluation

La note du projet est individuelle et correspond à 60% de la note du module POO. Elle sera combinée avec la note des 2 QCM déjà effectués (comptant pour 20% chacun). L'évaluation du projet est notée sur 20. Elle est calculée à partir des éléments suivants :

- qualité de la conception objet (25%) – présence de plusieurs classes avec des relations cohérentes et une bonne répartition des rôles entre les classes ;
- qualité du code source (25%) – code lisible et commenté, bien structuré en cohérence avec la conception, respectant les conventions de codage, utilisant a bon escient la STL ;

- fonctionnalités implémentées et état de finalisation de l'application (50%) – fonctionnement du code avec une bonne fiabilité, fluidité et une bonne expérience utilisateur.

Les fonctionnalités et la finalisation seront évaluées pour chaque binôme lors de la dernière séance de TP qui aura lieu **12 mai**. Lors de cette séance, il faudra présenter chacune des **4 fonctionnalités** décrites dans le cahier des charges (cf. paragraphe 2.3). L'enseignant vous posera également à chacun **quelques questions** portant sur la conception ou sur les algorithmes mis en œuvre dans l'application.

1.4 Remise du projet

Le projet (rapport + sources) est à remettre au plus tard le **9 mai** à 23 :59 :59. Il devra être déposé sur le portail MOOTSE, dans l'espace de dépôt correspondant à votre groupe de TP (sur le compte de l'un des deux membres du binôme).

1.5 Organisation

La réalisation du projet doit se faire impérativement en binôme. Lorsque le nombre d'étudiants d'un groupe de TD est impair, un étudiant doit travailler seul (on ne peut avoir qu'un seul étudiant travaillant seul). La constitution des binômes est laissée au choix des étudiants.

Quatre séances de TP sont prévues pour apporter une aide au développement du projet, mais il est indispensable de travailler également en dehors de ces séances. Les séances sont découpées en :

- **séance 1** (1h30) : présentation du cahier des charges, constitution des binômes et premières réflexions sur la conception orientée objet,
- **séance 2** (1h30) : point d'avancement individuel sur la conception et le codage des classes (mi-projet),
- **séance 3** (1h30) : point d'avancement individuel sur le codage et la finalisation de l'application,
- **séance 4** (1h30) : évaluation des projets sous forme de démonstration.

Pour la séance 2, un bilan individuel de chaque groupe sera réalisé. Il faudra donc que chaque groupe présente un diagramme des classes qui sera discuté avec l'enseignant.

1.6 Pénalités

Retard pour le rendu – 1 point de pénalité par jour de retard sur la note finale (ou au prorata si moins d'1 jour de retard).

Copie – en cas de copiage ou plagia au niveau des codes, la note 0 sera attribuée à chacun des rapports/sources mis en cause.

2 Cahier des charges de l'application

2.1 Contexte

Il s'agit de réaliser un jeu de **simulation d'écoulement de sable**. L'interface se présente sous la forme d'une fenêtre graphique représentant l'espace de jeu dans lequel se déroule la simulation. Le joueur peut dessiner des éléments de différente nature dans cet espace à l'aide de sa souris. Cette interface sera réalisée grâce à la bibliothèque graphique SFML[1] (cf. paragraphe 3).

Le joueur peut ainsi dessiner un élément solide comme de la pierre ou du métal, ou un élément pouvant s'écouler ou se propager comme le sable, l'eau, l'acide ou les champignons (cf. figure 1). Lorsqu'un élément pouvant s'écouler est dessiné, il va évoluer dans la scène en fonction de règles qui lui sont propres (le sable s'écoule, l'eau s'écoule et se propage pour remplir les creux,...). Lorsqu'un élément solide est dessiné ou gommé, il va instantanément changer le comportement des éléments mobiles. Par exemple si un trou est percé dans le sol, le sable ou l'eau va s'écouler par le trou.

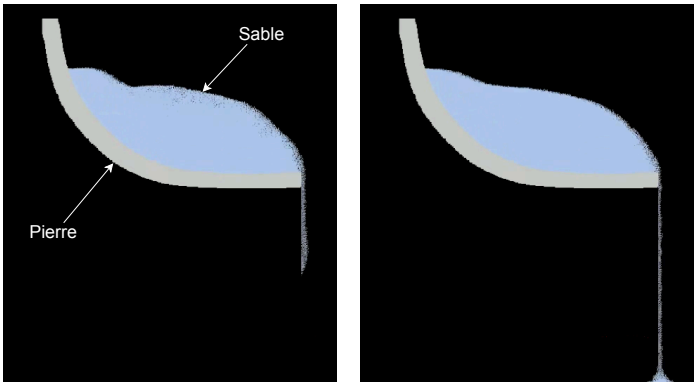

FIG. 1 : Scène de jeu avec une simulation d'écoulement de sable.

La scène de jeu est vide au départ et, à tout moment, l'utilisateur peut interagir avec la simulation en ajoutant des éléments à la souris (ou en gommant des parties). Pour changer le comportement de la souris l'utilisateur peut appuyer sur une touche du clavier. Il doit aussi pouvoir mettre la simulation en pause à tout moment.

2.2 Principe de la simulation

La simulation est réalisée à partir d'un algorithme d'automate cellulaire tel que celui du célèbre Jeu de la vie de Conway. Dans ce type d'algorithme, les éléments simulés sont organisés dans un tableau 2D représentant l'espace de jeu. Chaque cellule du tableau représente un carré élémentaire (ou cellule) de la scène de simulation. Dans notre simulation de sable, chaque cellule représente soit du vide, soit un matériau (pierre, sable, eau, ...). Ainsi, la scène de simulation peut être représentée graphiquement en dessinant toutes les cellules du tableau avec une couleur correspondant à la nature de la cellule.

Pour réaliser un pas de simulation, il suffit donc de parcourir le tableau et mettre à jour le type de chaque cellule en fonction des cellules voisines selon des règles d'évolution qui dépendent de la nature de la cellule. Dans un automate cellulaire, les règles d'évolution sont très simples et sont basées l'état des cellules directement adjacentes dans un voisinage (cf. figure 2. Pour plus d'information, vous pouvez consulter les blogs Making games with Falling Sand et Creating A Falling Sand Simulator.

Chaque groupe de projet aura des éléments spécifiques à simuler sur ce principe. Ils vous seront indiqués à la première séance de TP.

2.3 Fonctionnalités à implémenter

Les fonctionnalités suivantes doivent être obligatoirement implémentées dans le projet (en accord avec la description générale précédente). Elle feront l'objet d'une vérification avec l'enseignant lors de l'étape de démonstration :

[1] https://www.sfml-dev.org/

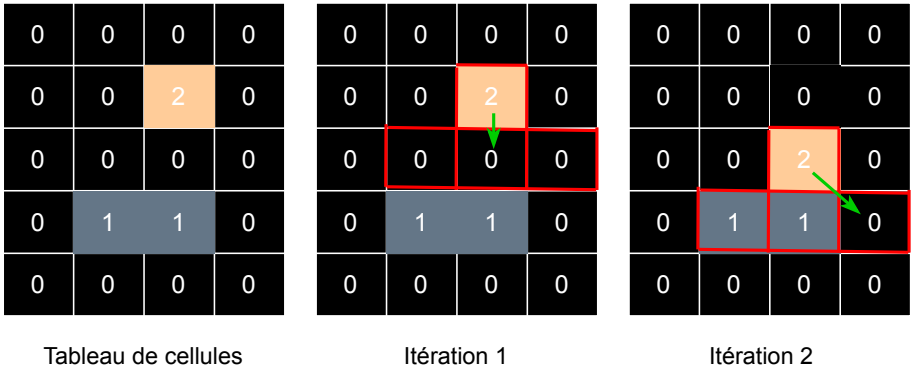

FIG. 2 : Principe de la simulation. Chaque valeur dans la grille correspond à un matériau différent. Ici 0 = vide, 1 = pierre, 2 = sable.

1. Affichage de l'espace de jeu dans une fenêtre graphique SFML. La taille de la grille de simulation devra pouvoir être paramétrée dans l'application et les différents matériaux seront dessinés avec des couleurs différentes.
2. L'utilisateur peut "dessiner" à la souris pour initialiser des cellules avec des matériaux de différente nature. Le changement de matériau se fait à l'aide de touches spéciales pré-définies. Le dessin peut avoir lieu à n'importe quel moment pendant la simulation pour que l'on puisse interagir avec la simulation.
3. La scène évolue avec les règles de simulation du matériau prévu. Les bords de la grille sont gérés correctement.
4. La simulation prends en compte les problèmes d'ordre de mise à jour et les différents paramètres demandés par l'enseignant de TP pour augmenter le réalisme de la simulation (caractère aléatoire, inertie des particules, interaction entre cellules,...).

3 Informations sur la bibliothèque SFML

SFML est une bibliothèque multimédia qui permet de manipuler sons et images en C++. La bibliothèque doit être téléchargée et installée depuis le site internet. Elle contient plusieurs module à inclure sans le projet en fonction des besoins : graphique, sons et réseau. Cette bibliothèque est portable (Windows, Unix, MacOS,...), légère et facile à utiliser.

Nous allons utiliser cette bibliothèque dans les mini-projets pour pouvoir gérer une petite interface graphique permettant de dessiner des éléments ou d'afficher des images.

3.1 Installation de la SFML

Il est recommandé d'utiliser la **version 2.6.x** qui est plus facile à utiliser que la version 3.x. Suivre les indications du tutoriel du site internet de SFML dans la rubrique "Démarrer" en choisissant la version correspondant à votre environnement.

Si vous êtes sous Windows avec le compilateur Visual Studio rendez-vous sur https://www.sfml-dev.org/tutorials/2.6/start-vc-fr.php. Si vous avez un autre compilateur, consulter le tutoriel correspondant sur https://www.sfml-dev.org/tutorials/2.6/. Dans tous les cas, bien choisir la version de la librairie correspondant exactement avec votre compilateur ou, à défaut, recompiler la librairie.

Code de test du tutoriel

Fichier `test-sfml.cpp`

```cpp
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        window.draw(shape);
        window.display();
    }

    return 0;
}
```

3.2 Utilisation de la bibliothèque

En guise d'introduction, vous pouvez commencer par étudier les exemples simples d'utilisation qui sont fournis sur Mootse dans l'archive `exemples-sfml.zip`.

Pour une présentation plus complète, vous pouvez vous référer aux tutoriels du site web de SFML, en cliquant ici.