ESIR1 – Projet de programmation

1. Présentation du projet

Ce projet vous permettra de mettre en application ce que vous avez appris au cours de l'année en programmation orientée objet (Java / C++).

Vous allez devoir réaliser en groupe de 4 personnes **un jeu de type tour par tour** dans un monde 2D. Vous pouvez vous inspirer des différents jeux de type *JRPG* ou *deck builder.* Vous aurez la liberté de créer la carte de jeu de votre choix en utilisant un fichier texte. Cette carte pourra inclure plusieurs types de décors en fonction de l'univers que vous avez imaginé.

Votre jeu sera **mono-joueur**. Le personnage principal pourra réaliser diverses actions en fonction du type de jeu choisi (se déplacer, gérer des objets, etc.). Vous devez également prendre en compte la présence d'ennemis sur la carte (attention, ils peuvent être par exemple invisible ...) et déclencher un combat lors de leur rencontre avec votre personnage principal. Lors de ces combats, vous allez devoir gérer une barre de vie, ou une barre d'action/mana qui évoluera au cours du jeu. Les points perdus peuvent être récupérable de différentes manières.

Vous allez devoir également offrir la possibilité de faire évoluer votre personnage (via des objets interactifs, un gain d'expérience, etc.). Vous pourrez ainsi par exemple augmenter les caractéristiques de vos chaussures turbofusées, changer de classe, etc.

Vous allez devoir également trouver un moyen de terminer le jeu. Il faut que votre jeu ait une fin bien définie et atteignable.

Vous êtes libres de définir l'univers de jeu vous souhaitez réaliser ainsi que le design dans ce projet. Vous devez cependant respecter les différentes consignes données dans la suite du sujet.

Attention, ce projet est bien un **projet de programmation**. Le design de votre jeu peut être simple. Vous serez notés sur les fonctionnalités du jeu et non sur son rendu graphique.

Sur Moodle, des fichiers de bases vous sont fournis pour débuter le jeu. Vous pouvez les utiliser, vous en inspirer ou commencer de zéro.

Vous avez accès à :

- Un fichier texte représentant une carte du jeu possible.
- Des fichiers de codes Java ou C++ permettant de charger une carte du jeu et de l'afficher en 2D (*tilemap*).

1.1. Organisation

Ce projet est à réaliser par groupe de **4 personnes**. Vous avez dû saisir les membres de votre groupe sur Moodle. Veuillez vérifier les informations saisies, elles vont permettre d'organiser le planning des soutenances de projet.

Ce projet est à réaliser sur **deux jours**. Vous allez devoir réfléchir au sujet de votre jeu, à la répartition et à la réalisation des tâches ainsi qu'à la préparation de la soutenance. Ne soyez pas trop ambitieux lors de la rédaction de votre cahier des charges. Commencez par une version simple du jeu et apportez des améliorations au fur et à mesure. **Nous souhaitons avoir une version stable à la fin du projet.**

Ce projet de programmation doit être réalisé obligatoirement soit en **Java**, soit en **C++**. **Aucun autre langage ne sera accepté**. Veuillez également n'utiliser aucun moteur graphique dans votre jeu (Unity, Unreal engine, etc.).

L'utilisation d'outils de gestion de travail collaboratif vous est **fortement recommandé** (gestion des versions de votre code avec git, gestion de projet avec Trello par exemple).
Lien GitLab de l'ESIR/ISTIC : https://gitlab2.istic.univ-rennes1.fr

1.2. Objectifs obligatoires

Le but du projet ESIR 1 est de développer un jeu vidéo **tour par tour** en 2 dimensions réalisé avec le langage de programmation de votre choix (JAVA ou C++). Le jeu devra se soumettre aux contraintes suivantes :

- Avoir un jeu **complètement fonctionnel**.
- Créer un monde continu ou discontinu.
- Mise en place d'une fenêtre de fin pour votre jeu.
- Avoir un personnage principal se déplaçant dans le jeu.
- Présence d'ennemis (2 types différents au minimum).
- Gestion des combats avec gestion de la vie du perso (barre de vie, points d'actions, mana, etc.).
- Possibilité d'évolution du personnage (via des objets ou du gain d'expérience).

En fonction du type de jeu que vous allez créer, vous allez devoir avoir les contraintes suivantes :

- Gestion simple des collisions (murs, objets, ennemis, etc.) entre le personnage principal et les éléments de la carte.
- Avoir un inventaire (afin de gérer les cartes, les objets de combats ou autre).
- Gérer une équipe lors des combats (plusieurs personnages ou plusieurs deck de cartes)
- Gérer différents sorts ou objets en combats.

Dans votre jeu, vous pouvez utiliser un des modes de fonctionnement suivant (minimum 2 cartes/niveaux) :

- Charger une grande carte et zoomer sur une partie (gestion des mouvements de la caméra).
- Ou alors avoir plusieurs fichiers textes qui permettent de charger au fur et à mesure les cartes (gestion des transitions entre deux cartes).

Votre code doit être structuré en différentes classes et interfaces. Pensez à commenter votre code.

1.3. Objectifs secondaires

Vous avez la possibilité d'intégrer au moins une des fonctionnalités suivantes dans votre projet :

- Création d'un menu (*retry, exit*, etc.).
- Gestion avancée des déplacements avec animation du personnage.
- Gestion du son.

2. Rendu du projet

Un espace dédié sur Moodle est disponible pour rendre le projet. Le rendu du code de votre projet est fixé pour le **mardi 27 mai 2025 à 14h00**. Vous le déposerez sous le format d'un dossier zippé (contenant tout votre code source) ou sous le format d'un fichier texte avec le lien de votre dépôt git. Si vous choisissez la deuxième option, vos évaluateurs doivent avoir accès à votre code. Veuillez bien indiquer le **nom de chaque étudiant** du groupe.

Vous fournirez également un document contenant les dépendances de votre projet (ou fichier readme).

3. Soutenance

Une soutenance clôturera le projet de programmation le **mardi 27/05/2025 de 14h00 à 16h00**.

La soutenance sera sous le format d'un pitch : vous aurez uniquement **5 minutes** pour nous présenter votre projet.

Vous devez aller à l'essentiel : nous présenter votre projet et ses points clés, vos choix techniques et votre organisation (gestion du projet et les différents rôles au sein de l'équipe). Il est fortement recommandé d'avoir **01 ou 02 diapos par projet**. Une de vos slides devra présenter la liste des objectifs demandés. Vous indiquerez ceux que vous avez réalisé.

Le reste du temps devra être consacré à la présentation du jeu sous forme de **démo**. Vous devez préparer votre démo suivant un scénario en montrant les objectifs réalisés. **Veuillez bien la préparer pour ne pas dépasser le créneau**.

L'ordre de passage des groupes sera disponible prochainement sur Moodle. Respectez bien le créneau. Si vous n'êtes pas présent lors de votre créneau, cela comptera comme une absence injustifiée.

4. Notation

Une note de groupe commune doit être normalement donnée au projet de programmation. Le respect des consignes sera pris en compte : objectifs obligatoires, travail de groupe, rendu des projets et soutenance. Pour rappel, le design du jeu n'est pas considéré comme le critère principal de notation.

Mais attention, si certaines personnes du groupe ne s'investissent pas dans le projet, **les notes pourront être différentes**.

Le rendu final du projet doit être le résultat du travail personnel de votre groupe.

Il est strictement interdit de plagier ou de copier une partie (ou la totalité) du code d'un autre groupe. Tout travail issu directement d'internet, d'intelligence artificielle (chatgpt ou autre) ne sera également pas pris en compte. **Reprendre du code dont vous n'êtes pas auteur n'est pas acceptable**. **Nous voulons évaluer votre capacité à produire une solution par vous-même.**

Cependant, l'entraide entre groupes n'est pas interdite mais elle doit se limiter à des conseils ponctuels.

De même l'utilisation **d'intelligence artificiel pour le code est interdite**. Ce n'est pas chatGpt qui est noté mais bien votre groupe. Son utilisation entrainera un retrait de point. Vous pouvez cependant l'utiliser uniquement pour la génération d'éléments multimédias (assets, son, etc.).

5. Annexes

Sur moodle des projets de bases sont donnés. Il s'agit de projet réalisé sous eclipse (comme pour les modules PROG1 ou PROG2). Vous pouvez normalement les importer directement. Attention, le projet en C++ utilise la SDL2. Veuillez penser à l'installer sur vos ordinateurs (SDL2, SDL2-gfx, SDL2-image, SDL-ttf).

Les deux projets reposent sur le même principe.

Ci-dessous une rapide explication du projet Java.

Le projet est conçu en orienté objet, tel que chaque classe désigne un des concepts du jeu. Une classe principale appelée “main” est exécutée afin que le jeu démarre. Cette classe qui instancie une JFrame ainsi qu’un JPanel sur java affiche la fenêtre principale (dans notre cas, elle lance directement la map). Dans cette même classe, plusieurs instances d’autres classes sont créées afin de gérer différents aspects du jeu comme suit :

1. **GamePanel :** la classe centrale du jeu, et qui affiche la fenêtre du jeu. Afin que le jeu fonctionne, un Thread *GamePanel* est instancié, celui-ci lance une boucle while qui est rafraîchie 60 fois par seconde (vous pouvez modifier le FPS). À l’intérieur de cette boucle, deux opérations sont appelées à chaque fois, l’opération *update*() qui permet de mettre à jour les différentes valeurs dans le jeu (joueur, ennemis, etc.), et une autre opération *repaint*() qui a pour rôle de redessiner l’intégralité des graphismes à chaque rafraîchissement.
2. **Player :** cette classe constituée d’un constructeur, des opérations suivantes : *getPlayerImage* pour retourner l’image du héros (une seule image dans notre cas), *draw*() pour dessiner le personnage dans le jeu, et *update*() pour mettre à jour différentes valeurs à chaque rafraîchissement. Elle hérite de la classe abstraite *Entity*.
3. **Tile et TileManager :** la classe *TileManager* qui hérite de la classe *Tile* gère l’aspect graphique des maps. Les différentes méthodes de la classe *TileManager* permettent de charger les différentes images constituant la carte du jeu (*getTileImage*()), de dessiner la carte du monde à travers la méthode *draw*(), et de charger la carte du jeu avec la méthode *loadMap*().
4. **KeyHandler :** gère les évènements d’appui et de relâchement des touches du jeu. Celle-ci est fortement couplée à la classe Player et utilisée principalement par cette classe.

Le programme C++ contient un Makefile vous permettant de compiler votre programme. Sa structure est similaire au programme Java et est basé sur quelques classes du TP3 de Prog2.

1. **main.cpp** : ce fichier est le point d'entrée du programme. Il initialise et lance une instance de la classe *Game*.
2. **graphics** : ce dossier contient les classes liées à l'affichage de éléments dans la fenêtre de jeu. La classe *Renderer*, similaire à celle utilisée dans le TP fourmis, permet de créer la fenêtre et de dessiner dessus via SDL. Le chargement des textures du programme est géré par la classe *TextureManager*.
3. **game** : ce dossier contient la classe *Game* permettant de créer la fenêtre du jeu, de charger la carte et un personnage. La boucle principale du jeu, dépendante du nombre de FPS (que vous pouvez modifier), permet de gérer les évènements clavier, de mettre à jour la simulation, et

de redessiner l'intégralité des graphismes. La classe *Map* permet de charger la carte à partir de fichier texte.

4. **theme.h** : ce fichier permet de regrouper les textures de la carte utilisées dans le programme.

Les deux codes fournis vous servent de point de départ du projet. C'est une base pour votre projet, vous allez devoir modifier la structure de certaines classes. Vous pouvez également recommencer le projet depuis le départ en vous inspirant de la boucle principale du jeu et des chargements de textures.

La carte du monde (Map) :

La carte du monde est constituée de tuiles de taille égale (16x16 pixels chacune dans le cas échéant en fonction des fichiers de texture utilisés). Elle est stockée sous forme de matrice dans un fichier texte (une matrice de 16x12 dans le cas échéant), tel que chaque case (chiffre) correspond à une tuile en particulier comme décrit dans la figure 1. L'affichage de la carte doit respecter le même ordre du fichier texte. Le fichier peut avec comme caractère de séparation un espace, un point-virgule ou autre. Le code peut être modifié en fonction de votre type de fichier choisi.

```
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2 3 4 4 4 4 4 4 4 4 4 4 4 4 0 4
 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 4
 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
 6 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 4
 7 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 0
 8 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 0
 9 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 0
10 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig 1 : Représentation d'une map dans un fichier texte avec le numéro des lignes affiché.

Fig 2 : Quelques tuiles du jeu.

Pour afficher la map, trois étapes sont nécessaires :

1. La première consiste à charger les différentes tuiles. Sur l'exemple de la figure 1, six types de tuiles sont chargés (figure 2), où un nombre est attribué à chaque tuile (figure 2).

En Java, la méthode *getTileImage()* de la figure 3 résume l'étape du chargement des tuiles, tel que la tuile "grass.png" est la tuile 0.

```java
public void getTileImage() {
    try {

        tile[0] = new Tile();
        tile[0].image = ImageIO.read(getClass().getResourceAsStream("/tiles/grass.png"));

        tile[1] = new Tile();
        tile[1].image = ImageIO.read(getClass().getResourceAsStream("/tiles/wall.png"));
        tile[1].collision = true;

        tile[2] = new Tile();
        tile[2].image = ImageIO.read(getClass().getResourceAsStream("/tiles/water.png"));
        tile[2].collision = true;

        tile[3] = new Tile();
        tile[3].image = ImageIO.read(getClass().getResourceAsStream("/tiles/earth.png"));

        tile[4] = new Tile();
        tile[4].image = ImageIO.read(getClass().getResourceAsStream("/tiles/tree.png"));
        tile[4].collision = true;

        tile[5] = new Tile();
        tile[5].image = ImageIO.read(getClass().getResourceAsStream("/tiles/sand.png"));

        tile[6] = new Tile();
        tile[6].image = ImageIO.read(getClass().getResourceAsStream("/tiles/mushroom.png"));

    } catch(IOException e) {
        e.printStackTrace();
    }
}
```

Fig 3 : Java - méthode permettant de charger les différentes tuiles

En C++, le code correspondant se trouve dans le fichier *theme.h* (Figure 4).

```cpp
static void loadTextureMap() {
        //A compléter
        loadTextureMap("assets/tiles/grassCenter.png",0);
        loadTextureMap("assets/tiles/grassGreenCenter.png",1);
  }
```

Fig 4 : C++ - méthode permettant de charger les différentes tuiles

2. La deuxième étape consiste à lire le contenu du fichier texte et de récupérer ses valeurs pour les charger dans une matrice sur java.

3. La dernière étape consiste à parcourir la matrice obtenue de l'étape 2 et à afficher les tuiles une à une.

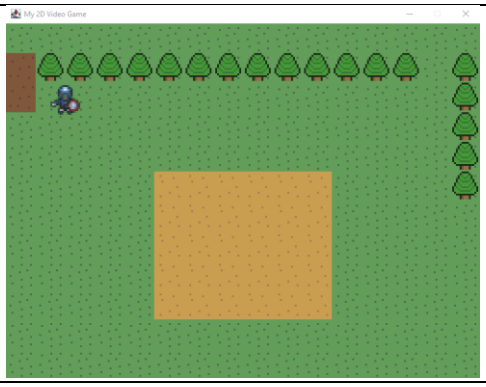

Fig 4 : Un exemple de map correspondant au fichier présenté dans la figure 1.

La taille de la carte est paramétrable, vous pouvez la modifier.