Rapport de Projet : Développement d'un Jeu 2D

CRYPT RAIDER

Supervisé par :

Christopher Leturc

Stéphane Jeannin

Préparé par :

Abdellah BOUALAM

Achour DJERADA

Naoufal AMIRAOUI

Section 1. Introduction

Dans le cadre de ce projet, nous avons conçu un jeu 2D captivant à l'aide de la bibliothèque LibGDX et de l'outil Tiled. Ce jeu, qui plonge le joueur dans une aventure palpitante, s'inspire des mystères de l'Égypte antique et des défis intellectuels associés aux pyramides.

Le concept du jeu repose sur un espion intrépide, chargé de décrypter des énigmes complexes cachées dans les profondeurs des pyramides égyptiennes. L'objectif principal est de récupérer des boules magiques, indispensables pour débloquer l'accès aux niveaux suivants. Tout au long de son parcours, le joueur doit faire preuve d'habileté pour :

Éviter les ennemis, qui patrouillent les lieux et mettent en péril la mission.

Échapper aux bombes, disséminées dans les couloirs de la pyramide.

Résoudre des énigmes, qui ajoutent une dimension intellectuelle au jeu et renforcent l'immersion dans cet univers mystérieux.

Ce projet a nécessité la combinaison de plusieurs outils et techniques. Tiled a été utilisé pour concevoir les cartes du jeu, offrant une personnalisation et une modularité optimales. LibGDX, une bibliothèque puissante dédiée au développement de jeux, a permis de gérer efficacement les mécaniques de jeu, les collisions, les animations et l'interaction entre les différents éléments.

Ce rapport se propose de détailler les étapes de conception, les choix techniques, ainsi que les défis rencontrés lors du développement. Il met également en lumière les mécaniques de jeu, l'implémentation des fonctionnalités principales, et les solutions adoptées pour garantir une expérience ludique et immersive.

Section 2. Présentation du projet

➢ Technologies et Outils Utilisés

Le développement du jeu 2D a été réalisé en utilisant une combinaison de technologies et d'outils permettant de gérer efficacement les différentes phases du projet, de la programmation à la conception graphique.

1. LibGDX

- Utilisé comme framework principal pour le développement du moteur de jeu.
- Fournit des fonctionnalités robustes pour gérer les graphiques, les collisions et l'interaction utilisateur.

2. IntelliJ IDEA

- Environnement de développement intégré (IDE) utilisé pour écrire et déboguer le code.
- Offre des outils puissants pour la gestion de projets Java, facilitant le développement.

3. Tiled

- Logiciel utilisé pour la création et la configuration des cartes du jeu.
- A permis de concevoir des niveaux détaillés avec des éléments interactifs et esthétiques.

4. Photoshop

- Utilisé pour créer et éditer les images du jeu, notamment les sprites et les arrière-plans.
- A permis de personnaliser les visuels pour une meilleure immersion.

5. Git

- Système de gestion de versions utilisé pour suivre l'évolution du projet et collaborer efficacement.
- Permet de sauvegarder les itérations du code et de gérer les contributions des membres.

6. Meet

- Plateforme de visioconférence utilisée pour la communication en ligne et les discussions d'équipe.
- A permis d'assurer une collaboration à distance fluide et efficace

7. UML-Diagrams.net

- Outil utilisé pour créer des diagrammes UML permettant de modéliser les différentes parties du système et de visualiser l'architecture du jeu.
- Le site est accessible à l'adresse suivante : https://app.diagrams.net

➢ Fonctionnalités Implémentées

1. Gestion des collections

- Les objets collectables, tels que les clés, sont intégrés au jeu. Les clés permettent d'ouvrir des portes, facilitant ainsi la progression du joueur à travers les niveaux.

2. Gestion des niveaux

- Une structure de niveaux a été mise en place, permettant une progression séquentielle ou conditionnelle dans le jeu. Chaque niveau possède des objectifs spécifiques à atteindre avant de passer au suivant.

3. Gestion de la fin de partie

- Une fonctionnalité de fin de partie a été implémentée pour marquer la victoire lorsque les objectifs sont atteints. À la fin d'un niveau, le jeu charge directement le niveau suivant, sans animations supplémentaires.

4. Gestion du Game Over

- En cas d'échec (par exemple, collision avec un ennemi), le jeu redirige le joueur vers l'écran d'accueil, signalant la fin de la partie et permettant de relancer le jeu.

5. Déplacement et intelligence des ennemis

- Les ennemis disposent d'un système de déplacement contrôlé par une intelligence artificielle. Leur comportement est conçu pour interagir dynamiquement avec le joueur et les éléments du jeu.

6. Déplacement des objets dans le jeu

- Tous les objets interactifs du jeu, tels que :
  - **Boule**,
  - **Rochers**,
  - **Bombes**,

  sont programmés pour se déplacer, enrichissant le dynamisme et la complexité du gameplay.

7. Objets collectables

- Les objets collectables, tels que les clés, jouent un rôle crucial. Ils permettent d'ouvrir des portes pour débloquer l'accès à de nouvelles zones.

8. Explosion et chute des objets déplaçables

- Les objets déplaçables, comme les bombes, disposent d'une fonctionnalité d'explosion, interagissant avec l'environnement et modifiant l'état du jeu.
- De plus, les objets déplaçables ont été programmés pour chuter naturellement sous l'effet de la gravité, ajoutant un élément de réalisme et de stratégie au gameplay.

➢ Configuration et Ajout de Contenu avec Tiled

Dans ce projet, Tiled a été utilisé pour créer les niveaux du jeu en structurant les différents éléments interactifs et statiques dans des calques bien définis. Chaque calque contient des objets associés à un type spécifique, et chaque type est lié à une classe correspondante dans le moteur du jeu. Cette organisation permet une gestion modulaire et extensible des niveaux.

1. Structure des Calques dans Tiled

Les calques (ou layers) dans Tiled sont utilisés pour organiser les différents types d'éléments du jeu. Voici la structure adoptée et leurs rôles respectifs :

- Calque "Murs"

Contient les objets représentant les murs dans le jeu.

Chaque mur possède une propriété type définissant son comportement :

Destructible : Un mur pouvant être détruit, associé à la classe MursDestructible.

Indestructible : Un mur qui ne peut pas être détruit, associé à la classe MursIndestructible.

Mangeable : Un mur que le joueur peut "manger", associé à la classe MursMangeable.

- b. Calque "Cle-Port"

Contient les objets interactifs :

Cle : Une clé que le joueur doit ramasser pour ouvrir une porte, associée à la classe Cle.

Porte : Une porte verrouillée que le joueur peut ouvrir avec une clé, associée à la classe Porte.

- c. Calque "Deplacable"

Contient les objets mobiles ou manipulables :

BouleMagique : Une boule que le joueur doit collecter, associée à la classe Boule.

Rocher : Un objet déplaçable, associé à la classe Rocher.

Bombe : Un objet dangereux à éviter, associé à la classe Bombe.

- d. Calque "Fin"

Contient l'objet PorteFin :

PorteFin : Une porte indiquant la fin du niveau, associée à la classe PorteFin.

- e. Calque "Personnage"

Contient les objets représentant les entités du jeu :

Joueur : La position de départ du joueur, associée à la classe Joueur.

Enemie : La position des ennemis, associée à la classe Enemie.

2. Chargement des Cartes dans leMoteur de Jeu

Les cartes créées dans Tiled sont exportées au format .tmx et chargées dynamiquement dans le moteur de jeu. Voici comment le processus fonctionne :

- Chargement de la Carte

La carte est chargée dans le moteur à l'aide de TmxMapLoader, et chaque calque est traité individuellement. Exemple :

```
map = new TmxMapLoader().load("level1.tmx");
renderer = new OrthogonalTiledMapRenderer(map);
camera = new OrthographicCamera();
camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
```

- b. Traitement des Calques et des Objets

Chaque calque est parcouru pour identifier et instancier les objets en fonction de leur type. Voici comment cela est réalisé pour chaque calque :

Calque "Murs" :

```
for (MapObject object : map.getLayers().get("Murs").getObjects()) {
    if (object instanceof RectangleMapObject) {
        Rectangle rect = ((RectangleMapObject) object).getRectangle();
        String type = (String) object.getProperties().get("type");
        if ("Destructible".equals(type)) {
            gameObjects.add(new MursDestructible(texture, rect.x, rect.y));
        } else if ("Indestructible".equals(type)) {
            gameObjects.add(new MursIndestructible(texture, rect.x, rect.y));
```

```
        } else if ("Mangeable".equals(type)) {
            gameObjects.add(new MursMangeable(texture, rect.x, rect.y));
        }
    }
}
```

3. Ajout de Nouveaux Niveaux

Pour ajouter un nouveau niveau ou modifier un existant, voici les étapes :

Créer une nouvelle carte dans Tiled :

Définir les dimensions à 15x9 tuiles avec chaque tuile de 80x80 pixels.

Ajouter les objets dans les calques appropriés.

Configurer les propriétés des objets :

Spécifier le type pour chaque objet en fonction de son rôle dans le jeu.

Exporter la carte au format .tmx :

Ajouter le fichier dans le dossier des ressources du jeu.

Modifier le code pour charger la nouvelle carte :

Exemple :

```
map = new TmxMapLoader().load("level2.tmx");
```

➢ Compilation et Exécution

Cette section explique comment exécuter le projet à partir du fichier fourni, sans nécessiter d'autres installations préalables que celles décrites ci-dessous.

- *Prérequis*

Avant de lancer le projet, assurez-vous que :

1. **Java Development Kit (JDK)** version 8 ou supérieure est installé sur votre machine.
2. Le fichier ZIP du projet a été téléchargé depuis la plateforme Moodle.

- *Étapes pour exécuter le projet*

1. **Télécharger et décompresser le fichier ZIP**
    - Téléchargez le fichier `projet-jeu.zip` depuis Moodle.
    - Décompressez le fichier dans un répertoire de votre choix.

2. **Ouvrir l'invite de commande (CMD)**
   - Ouvrez l'invite de commande de votre système d'exploitation.
   - Naviguez jusqu'au dossier où le projet a été décompressé à l'aide de la commande :

     ```
     cd chemin\vers\le\dossier\projet
     ```

3. **Exécuter le jeu**
   - Une fois dans le dossier du projet, exécutez la commande suivante pour lancer le jeu :

     ```
     gradlew.bat lwjgl3:run      (Windows)
     ./gradlew lwjgl3:run        (Linux)
     ```

   - Le jeu devrait démarrer automatiquement.

Section 3. Présentation technique du projet et contributions

➢ *Architecture Générale du moteur de jeu*

GameObject (Classe de base pour tous les objets du jeu) :

Description : Définit les propriétés et comportements de base pour tout objet du jeu. Inclut des méthodes pour la collision, le dessin et d'autres propriétés générales.

Implémente : Collision.

Sous-classes directes :

Entity (Classe abstraite pour les entités interactives avec comportement dynamique).

Murs (Classe pour les types de murs du jeu).

Cle (Classe pour les clés permettant d'ouvrir les portes).

Porte (Classe pour les portes verrouillées).

PorteFin (Classe pour la porte qui termine un niveau).

RollingEntity (Classe abstraite pour les objets dynamiques déplaçables ou soumis à la gravité).

Entity (Hérite de GameObject) :

Description : Classe abstraite représentant les entités dynamiques et interactives dans le jeu.

Sous-classes directes :

Joueur (Classe représentant le joueur contrôlé par l'utilisateur).

Enemie (Classe représentant les ennemis qui poursuivent le joueur ou interagissent avec les objets).

RollingEntity (Hérite de GameObject) :

Description : Classe abstraite pour les objets soumis à des mouvements dynamiques comme les chutes ou les déplacements.

Sous-classes directes :

Boule (Classe pour les boules magiques nécessaires à la progression).

Bombe (Classe pour les bombes dangereuses pour le joueur et les ennemis).

Rocher (Classe pour les rochers déplaçables par le joueur).

Murs (Hérite de GameObject) :

Description : Classe de base pour représenter les différents types de murs dans le jeu.

Sous-classes directes :

MursDestructible (Classe pour les murs pouvant être détruits, par exemple avec une explosion).

MursIndestructible (Classe pour les murs permanents, non modifiables).

MursMangeable (Classe pour les murs qui disparaissent après interaction avec le joueur).

Interface Collision (Implémentée par GameObject) :

Description : Décrit les comportements que tout objet interactif doit posséder, notamment en ce qui concerne les collisions.

Méthodes :

void onCollision(Joueur player) : Définit les actions lors d'une collision avec un joueur.

boolean isPassable() : Indique si l'objet est traversable.

boolean isDestructible() : Indique si l'objet peut être détruit.

GameObject (implements Collision)

| ___ Entity (abstract)

| |___ Joueur

| |___ Enemie

| ___ RollingEntity (abstract)

| |___ Boule

| |___Bombe

| |___ Rocher

| ___ Murs (abstract)

| |___MursDestructible

| |___ MursIndestructible

| |___ MursMangeable

| ___ Cle

| ___ Porte

| ___PorteFin

GameObject : La classe centrale à partir de laquelle toutes les entités et objets sont dérivés. Elle fournit les propriétés générales (comme les coordonnées, les textures, et les interactions).

Entity : Utilisée pour des objets interactifs dotés de comportements dynamiques (comme le joueur ou les ennemis).

RollingEntity : Une extension spécifique de GameObject pour les objets pouvant être déplacés, comme les boules ou les rochers.

Murs : Regroupe les types de murs, chaque sous-classe ayant des comportements distincts.

1.Relation entre Main, MainMenu et Play

Classe Main (Classe Principale du Jeu)

Rôle :

Point d'entrée principal du jeu.

Gestion des écrans (menu principal et gameplay).

Contrôle de la transition entre le menu principal et les niveaux.

Gère la progression des niveaux et la sauvegarde.

Relations :

Avec MainMenu :

Main initialise le menu principal (MainMenu) au démarrage du jeu via setScreen(new MainMenu(this)).

MainMenu utilise une instance de Main pour démarrer un niveau avec startGame(int startingLevel).

Avec Play :

Main crée et affiche l'écran de jeu (Play) en fonction du niveau courant via startGame().

Vérifie la progression du joueur en utilisant la méthode getEND() de la classe Play.

Passe au niveau suivant après avoir sauvegardé la progression (play.saveLevel(level)).

Classe MainMenu (Menu Principal du Jeu)

Rôle :

Fournit une interface utilisateur avec des options comme "Jouer" ou "Continuer".

Permet à l'utilisateur de charger un niveau sauvegardé ou de démarrer une nouvelle partie.

Relations :

Avec Main :

Communique avec Main pour démarrer le jeu au niveau spécifié (game.startGame(level)).

Utilise FileManager pour lire un fichier de sauvegarde et déterminer le niveau à charger.

Classe Play (Écran de Jeu)

Rôle :

Gère l'affichage du niveau, le chargement des ressources, et les interactions entre les objets du jeu.

Met à jour les objets du jeu (joueur, ennemis, murs, etc.) et contrôle la logique du niveau.

Vérifie si le niveau est terminé (getEND()) ou si le joueur a perdu.

Fournit un mécanisme pour revenir au menu principal.

Relations :

Avec Main :

Reçoit une instance de Main pour permettre la transition entre le gameplay et le menu.

Notifie Main lorsqu'un niveau est terminé en réglant END et en appelant saveLevel() pour sauvegarder la progression.

Avec FileManager :

Utilise FileManager pour sauvegarder et charger les niveaux.

2. Gestion des Niveaux et du Gameplay

Chargement des Niveaux

Play utilise Tiled pour charger et interpréter les fichiers .tmx (cartes des niveaux).

Les couches dans Tiled (par exemple, Murs, Cle-Porte, Deplacable) définissent la disposition des objets dans le jeu.

Chaque type d'objet dans une couche est associé à une classe Java correspondante (ex. : MursDestructible, Boule, Joueur).

Gestion des Objets de Jeu

Play maintient une liste de tous les objets du jeu via List<GameObject> gameObjects.

Pendant chaque frame (render), Play :

Parcourt cette liste.

Met à jour chaque objet actif en appelant des méthodes spécifiques à sa classe.

Supprime les objets inactifs ou détruits via objectsToRemoveGlobal.

Progression et Fin du Niveau

Play vérifie si le joueur a accompli les objectifs du niveau (par exemple, collecter toutes les boules magiques).

Si les conditions de fin sont remplies, Play règle END sur true, indiquant à Main que le joueur peut passer au niveau suivant.

Interactions et Collisions

Play gère les interactions entre les objets grâce à la méthode handleInput pour le joueur et des méthodes spécifiques pour d'autres objets.

Les collisions sont vérifiées via des méthodes comme onCollision(Joueur) implémentées dans les classes d'objets.

3. Relations Clés entre les Classes

Classe Relation

Main - Lance le jeu avec MainMenu.

- Charge et affiche les niveaux via Play.

- Contrôle la progression des niveaux et la sauvegarde avec FileManager.

MainMenu - Interagit avec Main pour démarrer un niveau ou continuer une partie.

- Utilise FileManager pour lire le fichier de sauvegarde.

Play - Gère les objets du jeu (GameObject et ses dérivés).

- Utilise les fichiers .tmx pour charger les niveaux.

- Notifie Main de la fin d'un niveau (END).

FileManager - Fournit des méthodes utilitaires pour lire et écrire dans le fichier de sauvegarde (niveau du joueur).

Résumé

Main : Responsable des écrans du jeu et de la progression des niveaux.

MainMenu : Interface utilisateur pour démarrer ou continuer une partie.

Play : Coeur de la logique de gameplay, gère les interactions et le rendu des niveaux.

FileManager : Gère la persistance des données (sauvegarde des niveaux)

➢ *Utiliser et étendre la librairie du moteur de jeu*

1. Utilisation de la librairie

Chargement et exécution du jeu

Pour démarrer un jeu avec la librairie :

Lancer le jeu :

Utilisez la classe Lwjgl3Launcher pour démarrer l'application.

Cette classe configure les paramètres de la fenêtre (taille, titre, FPS) et lance l'instance principale Main.

Création et navigation entre niveaux :

Les niveaux sont définis dans des fichiers Tiled (.tmx) avec des couches pour chaque type d'objet (Murs, Personnage, Deplacable, etc.).

La classe Play charge ces fichiers, interprète les couches, et gère les interactions entre objets.

Gestion des états de jeu :

MainMenu permet de naviguer entre les options du menu principal (Jouer, Continuer).

Play gère le gameplay actif, les transitions entre niveaux, et les événements comme "Game Over".

Structure de base

Les objets du jeu héritent généralement de la classe abstraite GameObject. Les classes spécialisées (Joueur, Enemie, Boule, Porte) implémentent des comportements spécifiques via des interfaces comme Collision.

2. Étendre la librairie

Un utilisateur extérieur peut facilement étendre la librairie pour ajouter de nouveaux éléments, personnages ou mécanismes. Voici des exemples concrets.

Ajouter un nouveau type de personnage

Étendre la classe Entity :

Entity est la classe de base pour les personnages. Créez une sous-classe pour définir un nouveau personnage.

```
public class Robot extends Entity {
    public Robot(Texture texture, float x, float y, float speed) {
        super(texture, x, y, speed);
    }

    @Override
    public void move(float delta) {
        // Définir un comportement unique, par exemple, se déplacer de manière aléatoire
    }
}
```

Ajout au niveau Tiled :

Ajoutez un objet dans la couche Personnage avec le type "Robot".

La méthode show() de Play doit être modifiée pour inclure la logique de chargement du Robot.

Ajouter un nouveau type d'obstacle

Étendre RollingEntity ou Murs :

Pour des obstacles interactifs (comme des bombes ou rochers), utilisez RollingEntity.

Pour des obstacles fixes (comme des murs), héritez de Murs.

```
public class Lava extends Murs {
    public Lava(Texture texture, float x, float y) {
        super(texture, x, y);
    }

    @Override
    public void onCollision(Joueur player) {
        player.setEnvie(false); // Le joueur meurt s'il touche la lave
    }
}
```

Intégration dans Tiled :

Ajoutez un objet dans la couche Murs avec le type "Lava" et associez une texture de lave.

Ajouter un nouveau type d'objet

Créer une classe héritant de GameObject:

```
public class Treasure extends GameObject {
    public Treasure(Texture texture, float x, float y) {
        super(texture, x, y);
    }

    @Override
    public void onCollision(Joueur player) {
        System.out.println("Trésor collecté !");
        player.incrementScore(100); // Ajouter des points au joueur
        deactivate();
    }
```

}

Chargement via Tiled :

Ajoutez l’objet dans une nouvelle couche ou une couche existante comme Cle-Porte.

Mettez à jour Play pour inclure ce type d’objet.

Ajouter un nouveau type de niveau

Créer une nouvelle carte dans Tiled :

Ajoutez des couches (comme Murs, Personnage, etc.) en suivant la structure existante.

Sauvegardez le fichier .tmx sous un nom unique (levelX.tmx).

Configurer le chargement du niveau :

La logique dans Play chargera automatiquement le fichier basé sur le numéro de niveau (levelX.tmx).

3. Points d’extension pour utilisateurs avancés

Personnaliser les interactions :

Implémentez l'interface Collision pour gérer de nouvelles interactions spécifiques entre objets.

```
public interface Collision {
  void onCollision(Joueur player);
  boolean isPassable();
  boolean isDestructible();
}
```

Créer des objets explosifs :

Héritez de RollingEntity ou implémentez l’interface ObjetExplosif pour définir des comportements d’explosion.

Étendre la gestion des niveaux :

Modifiez la classe Play pour intégrer des mécanismes uniques à certains niveaux (par exemple, des puzzles).

➢ *Répartition des Taches*

1. Boualam ABDELLAH :

Déplacement du joueur :

Implémentation des méthodes pour permettre au joueur de se déplacer selon les entrées clavier.

Affichage des objets :

Gestion de l'affichage dynamique des objets à l'écran, y compris les interactions visuelles.

Effet de gravité des objets déplaçables :

Développement de la mécanique de chute pour les objets comme les rochers et les boules magiques.

Gestion des collisions avec les murs :

Mise en place des règles empêchant le joueur ou les objets de traverser les murs.

Classe Porte et Clé :

Implémentation des interactions entre les clés et les portes (ouvrir une porte après avoir récupéré une clé).

Création des cartes avec Tiled :

Conception des niveaux à l'aide de Tiled, en ajoutant les objets et propriétés nécessaires pour chaque calque.

2. Achour DJERADA :

Conception de l'architecture générale des classes :

Organisation des classes principales et définition de leurs relations (hiérarchie et interactions).

Système de gestion du jeu :

Mise en place de la gestion des écrans (menu principal, écran de jeu, écran de fin de partie).

Gestion de la logique de fin de partie et du game over.

Gestion des niveaux :

Chargement des cartes Tiled en fonction du niveau et gestion de la transition entre niveaux.

Classe Enemie :

Implémentation des déplacements et de l'IA des ennemis.

Gestion des collisions entre les ennemis et les autres objets (joueur, obstacles, etc.).

Gestion des explosions :

Mise en œuvre des mécaniques liées aux explosions des bombes et leurs interactions avec les objets environnants.

3. Travail en commun :

Achour DJERADA & Boualam ABDELLAH & Naoufal :

Rédaction du rapport :

Contribution à la rédaction des sections techniques, conceptuelles et analytiques du rapport.

Achour DJERADA & Boualam ABDELLAH :

Conception UML :

Élaboration des diagrammes UML pour représenter l'architecture générale du projet, les interactions entre les classes et les relations principales.

Section 4. Conclusion et Perspectives

Le projet a permis de concevoir un jeu en 2D mettant en œuvre une architecture modulaire basée sur LibGDX et Tiled. Les principaux objectifs suivants ont été atteints :

Création d'un gameplay interactif : Un joueur doit récupérer des boules magiques tout en évitant des obstacles tels que des bombes et des ennemis.

Gestion des niveaux : Utilisation de fichiers .tmx pour définir les niveaux, avec des couches distinctes pour les différents éléments du jeu (murs, personnages, objets déplaçables, etc.).

Gestion des collisions et des interactions : Mise en place d'un système de collision robuste pour gérer les interactions entre les différents objets du jeu (ex : destruction, traversabilité, explosions).

IA des ennemis : Implémentation d'une intelligence artificielle de base permettant aux ennemis de se déplacer vers le joueur.

Gestion des objets dynamiques : Mise en œuvre de mécanismes comme la suppression d'objets en cours de jeu sans provoquer d'erreurs de concurrence liées à l'utilisation simultanée d'itérateurs.

Le projet a permis de répondre aux attentes initiales tout en posant les bases pour des fonctionnalités plus avancées.

Défis rencontrés

Conception du projet :

Définir une architecture claire et extensible pour le moteur de jeu a nécessité une réflexion approfondie sur la modularité et les interactions entre les classes.

L'utilisation des fichiers Tiled a nécessité une coordination entre les données de la carte et le moteur de jeu.

Gestion des objets dynamiques :

Supprimer des objets du jeu sans provoquer d'erreurs de concurrence lors des itérations a été un défi technique résolu par une approche qui marque les objets pour suppression, puis les traite après l'itération principale.

Intelligence artificielle des ennemis :

Implémenter une IA permettant aux ennemis de suivre le joueur de manière cohérente tout en évitant les obstacles s'est révélé complexe. Cela a nécessité une gestion efficace des mouvements et des priorités dans les choix des directions.

Gestion des collisions et des explosions :

Développer un système de collision capable de gérer des interactions multiples (exemple : une explosion détruit plusieurs objets dans un rayon) a demandé une attention particulière à l'optimisation et à la clarté du code.

Perspectives et améliorations futures

Le projet actuel constitue une base solide, mais plusieurs pistes d'amélioration et d'extensions peuvent être envisagées pour enrichir l'expérience utilisateur et augmenter la modularité du moteur de jeu.

Effets sonores et animations :

Ajout de sons pour les collisions, explosions, et interactions entre objets.

Implémentation d'animations visuelles (explosions dynamiques, mouvements fluides des personnages, transitions entre niveaux).

Système de gestion de niveaux :

Ajout d’un système de progression plus structuré avec des niveaux verrouillés, déverrouillés en fonction de la progression du joueur.

Extension des ennemis :

Création d’une super-classe SuperEnemie pour centraliser les comportements de base des ennemis, facilitant ainsi l’ajout de nouveaux types avec des comportements spécifiques (ex : ennemis volants, ennemis statiques qui tirent des projectiles).

Système de sauvegarde avancé :

Permettre une sauvegarde des niveaux non seulement par numéro, mais aussi avec l’état exact des objets sur la carte, permettant une reprise précise.

Optimisation du moteur :

Réduction de la consommation de mémoire et amélioration des performances pour des cartes plus grandes ou des environnements avec de nombreux objets actifs simultanément.

Ajout de nouveaux objets et mécaniques :

Intégration d’éléments comme des portails de téléportation, des pièges cachés, ou des puzzles dynamiques nécessitant une réflexion pour progresser.

Conclusion

Le projet a permis de développer un jeu fonctionnel et captivant, basé sur une architecture claire et extensible. Malgré les défis techniques rencontrés, les solutions mises en œuvre ont permis de poser une base robuste pour un moteur de jeu 2D. Les perspectives d’amélioration identifiées ouvrent la voie à un enrichissement significatif du gameplay et de l’expérience utilisateur, tout en renforçant la flexibilité et la modularité du moteur.