Rapport Projet Jeu : ATLIR5 – ATLC

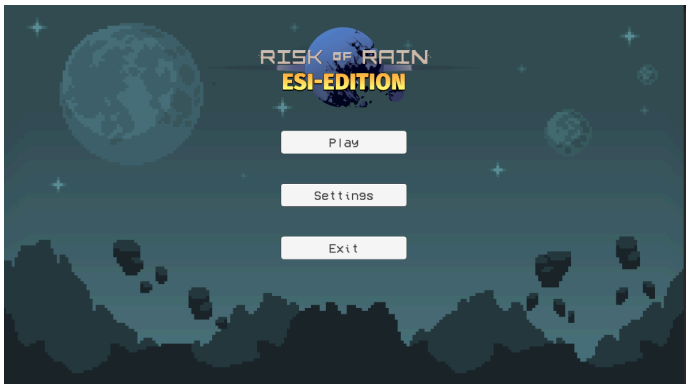
Description de l’application

Description technique

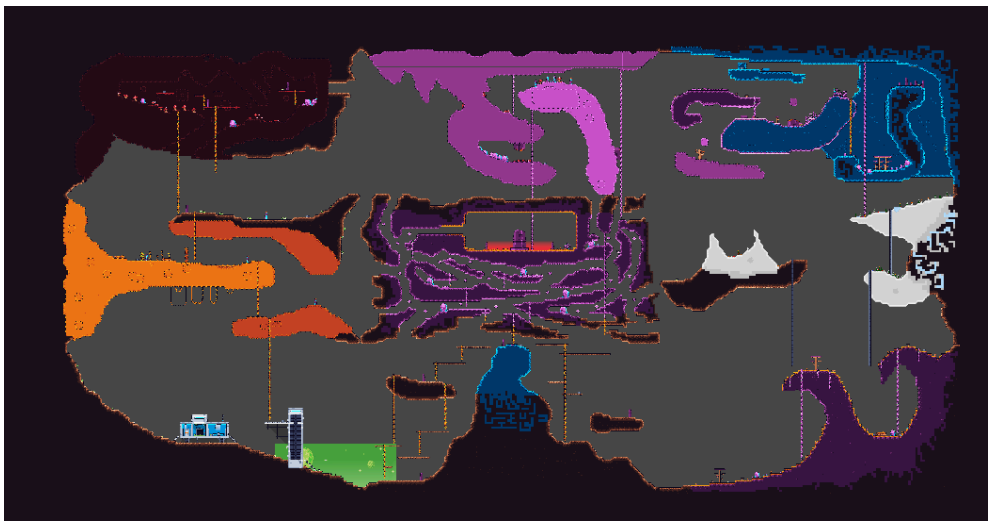
Notre application est un jeu Unity possèdant un model en C++.

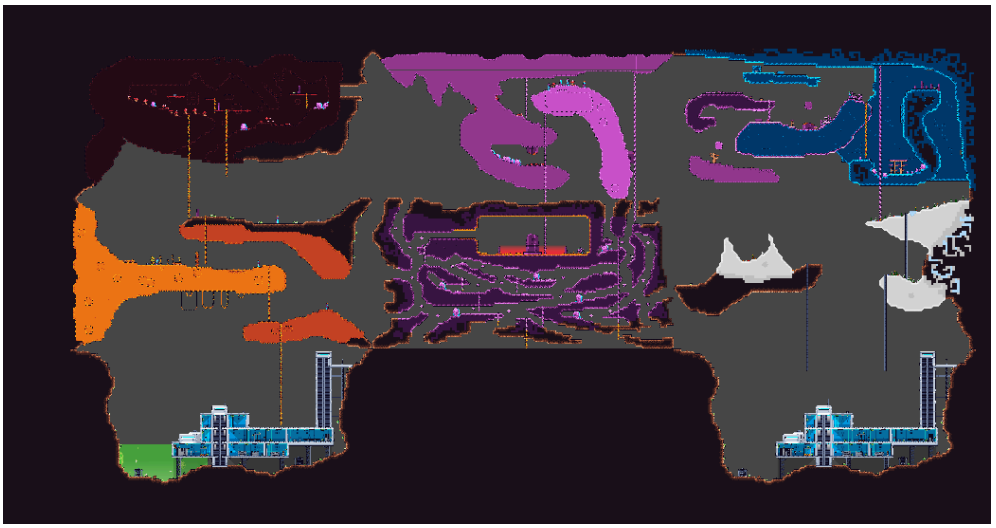
Description générales

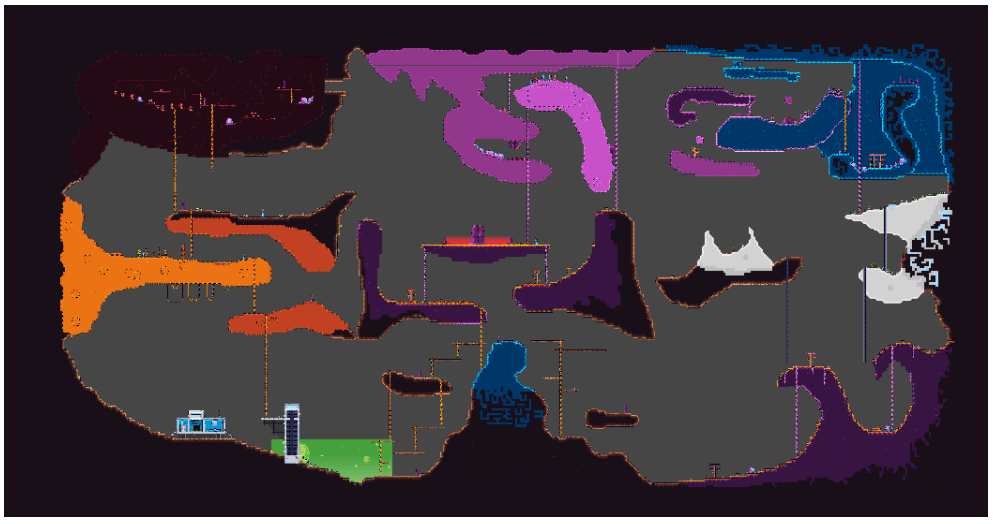
Risk of Rain ESI Edition est un jeu 2D inspiré de Risk of Rain ou le joueur incarne un survivant échoué sur une planète hostile.



C’est un rogue-lite car les niveaux sont générés de manière aléatoire et le joueur perd tout son équipement à chaque mort, recommençant à 0.





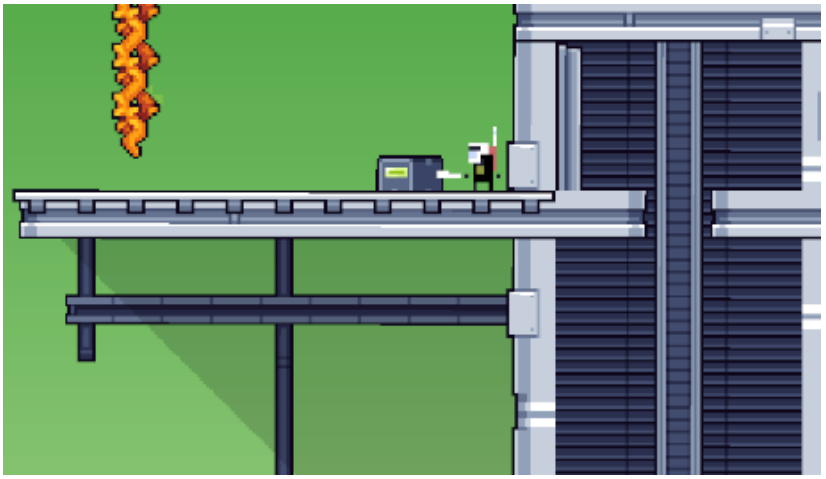


On peut voir ici que le monde n’est jamais exactement le même.
D’autres areas seront ajoutés dans les mises à jour futures pour ajouter de la diversité au monde.

Description du jeu

Avant le démarrage du jeu, le joueur pourra choisir son attaque légère et son attaque lourde (rapproché ou distance pour les deux).

Dans un premier temps, son objectif sera de trouver des coffres.



Ceux ci fourniront des items pour aider le personnage à devenir plus puissant

- **Kit de Soin:** permet de régénérer sa vie et utilisable à tout moments
- **Casque:** augmente les points de vie maximum
- **Bouteille:** augmente significativement sa vitesse
- **Epée:** augmente ses dégâts bruts
- **Fusée:** ajoute un saut supplémentaire.
- **Teddy:** permet d'avoir une faible chance d'être réanimé après sa mort.

Mais attention, il n’est pas seul sur cette planète, des aliens assoiffés de sang sont prêts à tout pour sauter sur notre jeune aventurier et lui faire la peau. Ceux-ci devenant de plus en plus forts au fur et à mesure que le temps avance.

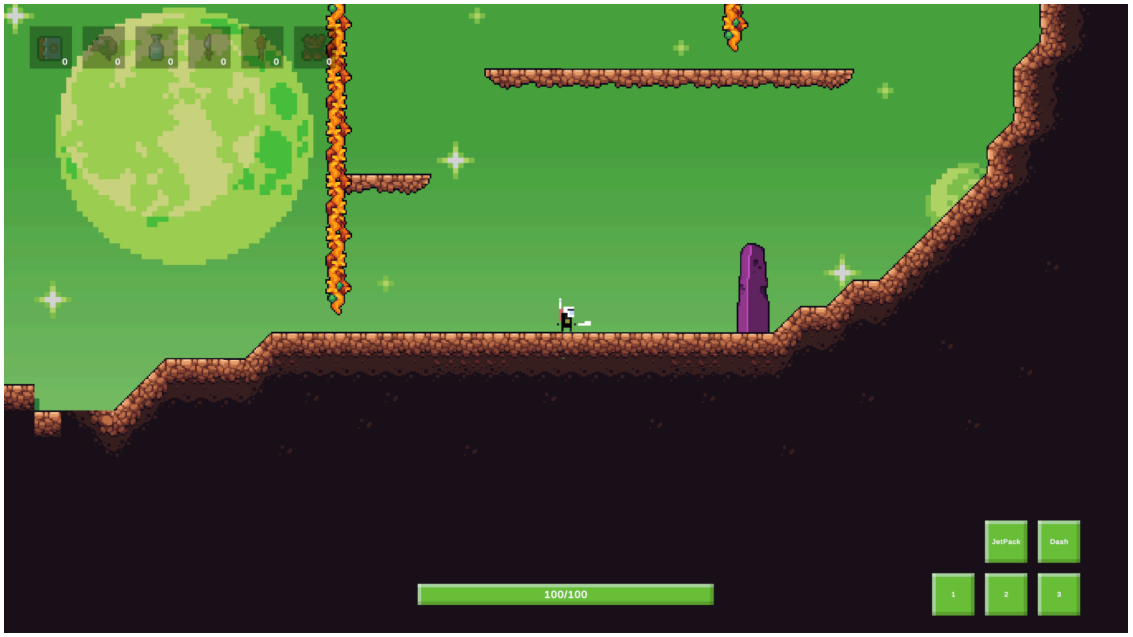
Il devra explorer le monde avec ses dashs, ses items et son jetpack à la recherche du boss.

Une fois le boss tué, il pourra passer au prochain niveau, en gardant ses items et sa progression dans le jeu.

Manuel d’utilisation

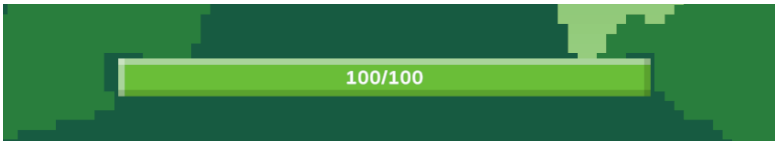
Interface

Durant une partie l’interface du joueur se présente comme suit :



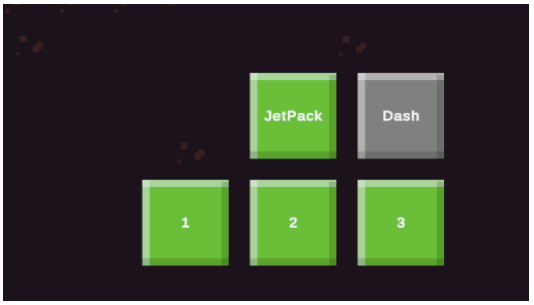
Celle-ci est composée de plusieurs parties

La barre de vie



Les capacitées / attaques

Ceux-ci se voient grisés après leur utilisation et persistent ainsi jusqu’à leur chargement.



Inventaire des items

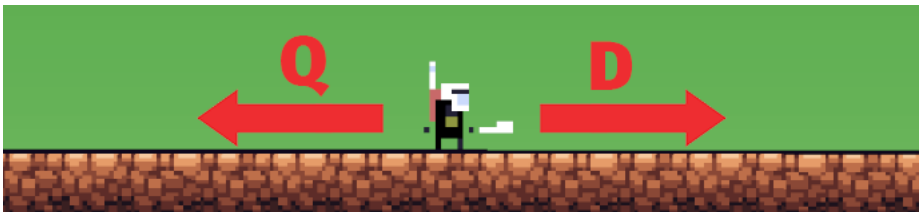
Lorsqu'un item se trouve dans l'inventaire, celui-ci n'est plus grisé et un numéro permet de savoir combien de ces items le joueur possède.

Interactions

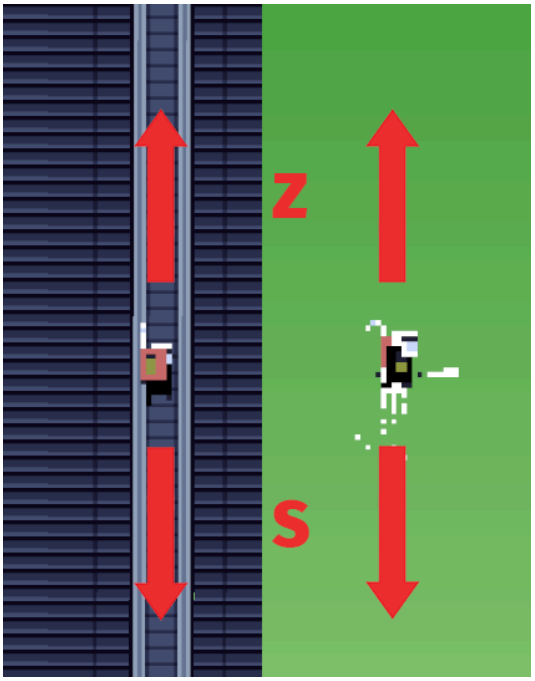
Lors d'une partie le joueur peut réaliser divers interactions.

Déplacement

Pour se déplacer le joueur peut utiliser les touches Q et D afin de réaliser un déplacement à gauche ou à droite.



Mais sur des échelles/lianes ou lors de la capacité "JetPack" celui-ci peut également utiliser les touches Z et S afin de se déplacer en haut et en bas.



Attaques et capacitées

Dans le jeu, le joueur dispose d'un éventail d'attaques et de capacités accessibles via des commandes :

- **Clic Molette** : *Attaque Primaire*
- **Clic Gauche** : *Attaque Secondaire*
- **Clic Droit** : *Attaque Ternaire*
- **Shift** : *Dash*
- **Ctrl Gauche** : *Jetpack*

Chaque action s'exécute instantanément à la pression de la touche correspondante, sans nécessiter de maintien prolongé. Une fois l'attaque ou la capacité terminée, elle entre automatiquement en phase de rechargement.

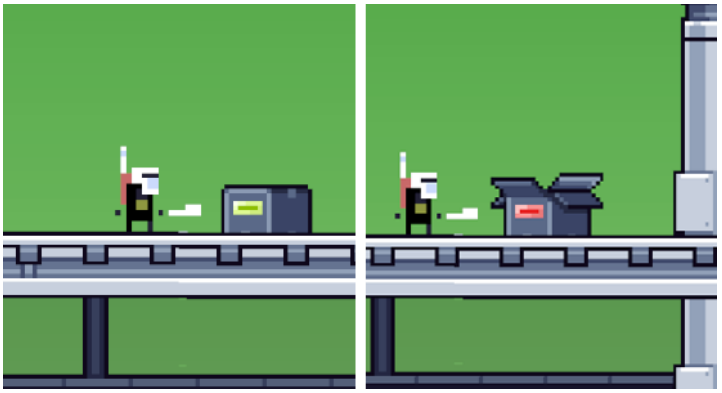
Cependant, une exception existe : l'utilisation du **Jetpack**.
Lorsqu'il est activé, le Jetpack modifie l'état du joueur, déverrouillant ainsi la possibilité de se déplacer verticalement. Ce mode spécial reste actif pendant une durée limitée avant de se désactiver automatiquement ou manuellement en appuyant de nouveau sur **Ctrl Gauche**. À l'extinction, un temps de rechargement s'enclenche.

Interactions avec l'environnement

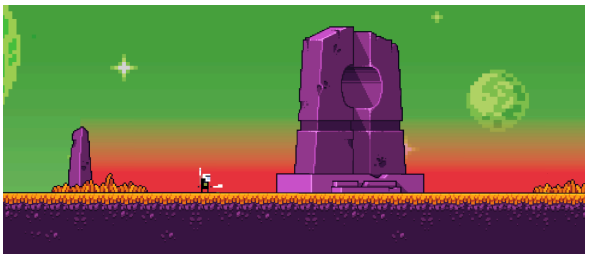
Le monde du jeu regorge d'éléments interactifs permettant au joueur d'explorer, de progresser, et de survivre. Ces interactions se divisent en plusieurs catégories :

Éléments interactifs principaux

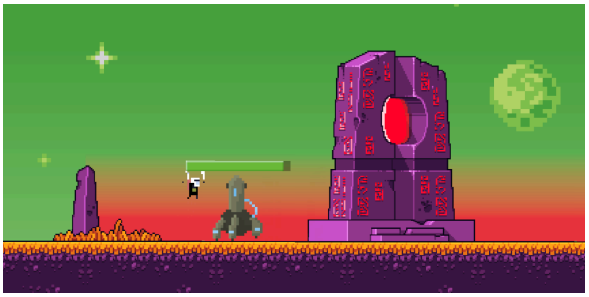
- **Coffres** : Pour ouvrir un coffre, le joueur doit s'approcher à proximité. Une simple pression sur **F** permet de l'ouvrir, révélant immédiatement l'objet qui sera automatiquement ajouté à l'inventaire.



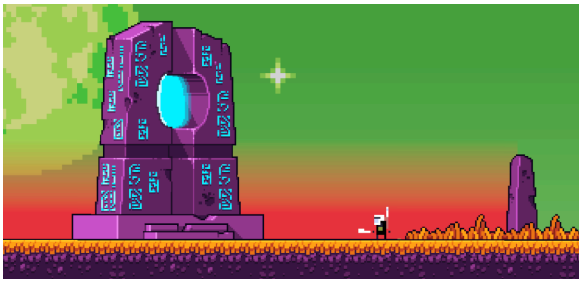
- **Portail du Boss** : L'activation du portail nécessite de se placer dans sa *zone rouge* et de presser **F**.



Une fois activé, le portail s'illumine d'un rouge, et le boss apparaît.
Après l'avoir vaincu, le portail change de couleur pour devenir bleu, signalant que la voie vers le prochain niveau est ouverte.



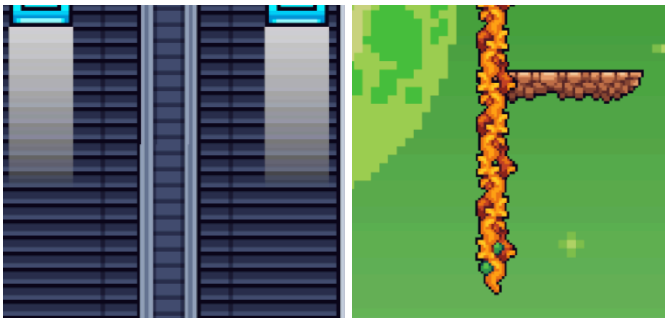
Le joueur peut alors interagir à nouveau avec le portail via **F**, toujours en restant dans sa zone, pour progresser vers le niveau suivant.



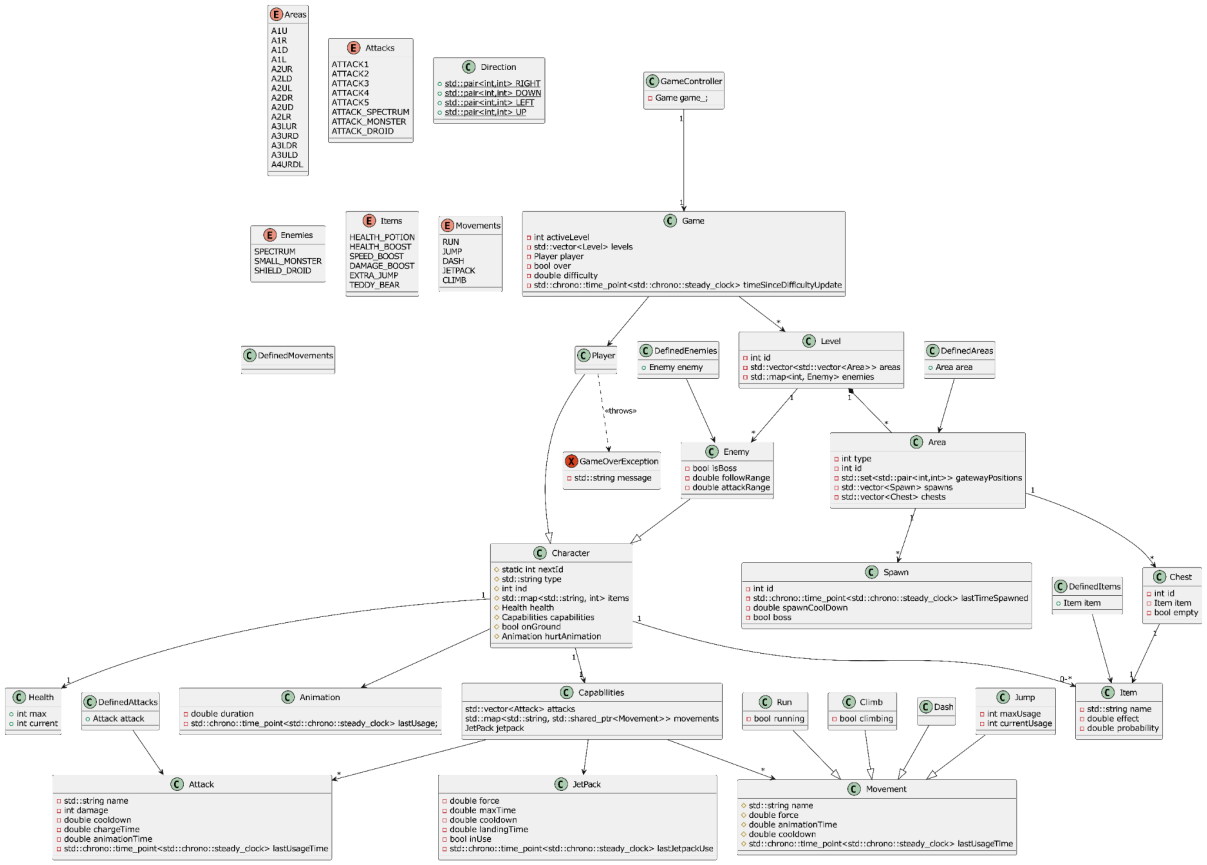
- **Trousse de soin** : Si le joueur possède une trousse de soin dans son inventaire, il peut se soigner à tout moment en pressant **E**.

Interactions passives

- **Échelles et Lianes** : Ces éléments permettent au joueur de grimper ou de descendre automatiquement. Dès qu'il entre en contact avec une échelle ou une liane, le joueur passe dans un *état de grimpe* où il peut ajuster sa direction pour monter ou descendre sans action supplémentaire.



Description des classes



Classes principales

GameController

GameController est l'interface principale avec Unity, jouant le rôle de façade pour connecter la logique du jeu à la vue. Cette classe contient une instance de la classe Game et expose les méthodes nécessaires à son fonctionnement. Toutes les fonctionnalités de la classe Game sont accessibles via GameController.

Game

Game est la classe centrale du système. Elle représente l'état global du jeu en regroupant les informations essentielles comme les niveaux, le joueur, et la difficulté des ennemis. Cette classe joue un rôle clé dans la gestion de la progression et des interactions au sein du jeu.

Level

La classe Level représente un niveau spécifique du jeu. Elle contient tous les ennemis présents dans le niveau, ainsi que les différentes zones préfabriquées (Area). Elle sert de conteneur logique pour organiser les éléments et la structure de chaque niveau.

Area

Area représente une zone préfabriquée d'un niveau. Elle est étroitement liée à son niveau parent et ne peut exister de manière autonome. Une zone est définie par :

- Un type (associé à une énumération Areas).
- Le nombre d'ouvertures permettant la connexion à d'autres zones.
- Un identifiant unique pour distinguer les zones du même type.

Elle contient également un ensemble de coffres (Chest) et des points d'apparition de monstres (Spawn).

Character

Character est la classe parente commune au joueur (Player) et aux ennemis (Enemy). Elle regroupe toute la logique nécessaire à leur fonctionnement, notamment :

- Les capacités (Capabilities).
- Les objets (inventaire).
- La gestion des points de vie (Health).

Player

La classe Player hérite de Character et redéfinit le comportement lié à la mort du joueur. Cela permet une gestion spécifique pour les actions à effectuer lorsque le joueur perd la partie.

Enemy
Enemy, également dérivé de Character, redéfinit la méthode d'utilisation des objets pour empêcher les ennemis de les utiliser, contrairement au joueur.

Gestion des capacités des personnages

Les capacités des personnages sont regroupées dans la classe Capabilities et divisées en trois catégories principales :

1. **Attaques** : Toutes les actions offensives disponibles.
2. **Mouvements** : Représentant les déplacements spécifiques.
3. **Jetpack** : Une capacité spéciale pour des déplacements avancés.

Ces capacités permettent de définir précisément ce qu'un personnage peut effectuer dans le jeu.

Movement

Movement est une classe générique utilisée pour modéliser les différents mouvements possibles. Des classes spécifiques héritent de Movement pour implémenter des comportements uniques :

- **Run** : Courir.
- **Climb** : Escalader.
- **Dash** : Effectuer une course rapide.
- **Jump** : Sauter.

Certains mouvements peuvent être utilisés continuellement, tandis que d'autres nécessitent un temps de recharge avant une nouvelle utilisation.

Gestion des éléments spécifiques

Spawn

Les points d'apparition des monstres (Spawn) sont définis dans chaque zone préfabriquée. Ils permettent de contrôler :

- Le délai entre les apparitions, choisi aléatoirement entre deux valeurs spécifiées lors de l'initialisation.
- L'apparition d'un boss lorsque le joueur remplit certaines conditions.

Chest

Les coffres (Chest) sont également définis dans chaque zone. Chaque coffre contient un seul objet, qui peut être :

- Une amélioration.

- Une potion de soin.
- Un objet spécial comme un ours en peluche.

Une fois ouvert, un coffre est définitivement vide pour ce niveau.

Gestion des exceptions

GameOverException

GameOverException est une exception personnalisée qui est déclenchée par le joueur lorsque la partie est terminée. Cette exception est récupérée par la classe Game, qui exécute alors la routine de fin de jeu.

Gestion des énumérations

De nombreux éléments du jeu sont modélisés à l'aide d'énumérations (entiers). Ces énumérations permettent de définir les valeurs associées à différents concepts :

- **Items** : Les objets disponibles dans le jeu.
- **Enemies** : Les types d'ennemis.
- **Movements** : Les types de mouvements.
- **Attacks** : Les différentes attaques utilisables.

Chaque énumération est complétée par une classe associée qui définit les caractéristiques spécifiques de chaque élément.

Description du travail réalisé:

1. Planning

Le projet a démarré début novembre avec une date limite fixée au 3 janvier. L'équipe a finalisé le gros du projet entre le 31 Décembre et le 1er Janvier, ce qui a permis de disposer de quelques jours pour affiner les dernières fonctionnalités et corriger les éventuels bugs.

- **Phases principales :**
    - **Définition des fonctionnalités :** Discussions initiales pour identifier les fonctionnalités clés du jeu et comprendre les contraintes techniques d'Unity et du modèle en C++.
    - **Apprentissage d'Unity :** Familiarisation avec l'environnement Unity pour les membres concernés par la partie graphique.
    - **Développement indépendant :** Chaque membre a commencé à coder sa partie (modèle ou graphique) individuellement.

    - **Collaboration via méthode Scrum :** L’équivalent d’une organisation agile a été mise en place, où l'équipe Unity définissait les fonctionnalités graphiques nécessaires et précisait les informations à récupérer depuis le modèle. L'équipe modèle adaptait ensuite son code en conséquence pour répondre aux besoins de l’autre équipe.
    - **Intégration et tests :** Une fois chaque fonctionnalité implémentée, elle était testée dans Unity, avec des retours pour ajustement.
    - **Finalisation :** Les derniers jours ont été consacrés à la correction des bugs, à l'ajustement des fonctionnalités et à la documentation.
- **Rythme de travail :**
    - Un début progressif pour la prise en main des outils.
    - Une forte intensification du travail en fin de projet pour respecter les délais.

2. Répartition du Travail

L'équipe était composée de quatre membres, chacun avec un rôle spécifique :

- **Unity :**
    - **Thomas :** Responsable des zones et niveaux préfabriqués.
    - **Marcel :** Réalisation des entitées du jeu (Animations, Scripts, Physique), des éléments d'environnement (Échelles, Spawners) et de l’interface utilisateur
- **Modèle C++ :**
    - **Enzo :** Responsable des contrôleurs principaux, notamment le `GameController et Game`, ainsi que de la génération des niveaux (classes `Level`, et `Area`).
    - **Julien :** Responsable de la gestion des personnages (`Character`, `Player`, `Enemy`), des objets (`Item`) et de la documentation technique du modèle.

Cependant, l’entraide et le travail d’équipe étaient de mise. Les équipes unity ou modèle n'hésitant pas à travailler ensemble sur une même problématique pour échanger et trouver des solutions et discuter de la meilleure façon de faire.

3. Tests Réalisés

a. Méthodologie de test

L'idée principale était de garantir un modèle fonctionnel et stable avant de le transmettre à l'équipe Unity. Les tests effectués comprenaient :

1. **Tests Unitaires :**
    - Réalisés avec Google Test (`gtest`) pour valider le comportement des classes du modèle.
    - Exemples de tests :
        - Validation des comportements de compatibilité dans les niveaux (tests sur la classe `Area`).
        - Vérification des délais et des conditions d'utilisation des mouvements (`Movement`).

        - Test des attaques pour s'assurer qu'elles respectent les règles de timing et de dégâts (`Attack`).
        - Vérification des classes de contrôleur (`GameController`) pour les interactions globales.
2. **Tests Fonctionnels :**
    - Réalisés après intégration avec Unity.
    - L'équipe Unity jouait des parties pour valider l'interaction entre le modèle et l'interface graphique.
3. **Processus de test intégré :**
    - Après chaque fonctionnalité ajoutée, des tests étaient réalisés par l'équipe Unity, qui revenait vers l'équipe modèle pour ajustements si nécessaire.
    - Cela a permis une itération constante et une amélioration continue.

b. Résultats des tests

- Le modèle a été jugé fonctionnel et répondant aux besoins exprimés par l'équipe Unity.
- Quelques ajustements mineurs ont été réalisés en cours de route pour des questions de cohérence globale.

4. Outils et Collaboration

- **Outils Utilisés :**
    - **Discord :** Pour la communication quotidienne, les discussions sur les problématiques rencontrées, et les retours entre les équipes.
    - **GitLab :** Chaque membre travaillait sur des branches spécifiques. Un système de merge request a été utilisé pour assurer la qualité et éviter les conflits. Enzo était chargé de valider les merges et de générer les fichiers `.dll` et `.dylib` pour l'équipe Unity.
- **Organisation Agile :**
    - Une organisation de type Scrum a été adoptée en cours de développement, permettant de travailler fonctionnalité par fonctionnalité en coordonnant les efforts des deux équipes.

5. Apprentissage et Bilan

Le projet a permis de remplir les objectifs fixés, bien que quelques ajustements mineurs aient été nécessaires pour des raisons de cohérence d'équipe. L'expérience a été enrichissante à plusieurs niveaux :

1. **Apprentissage technique :**
    - Prise en main d'Unity et renforcement des compétences en C++.
    - Utilisation avancée de Git pour la gestion collaborative du code.
2. **Collaboration :**
    - Travail en équipe élargie, nécessitant une coordination accrue entre deux parties distinctes (modèle et graphique).
    - Amélioration des capacités de communication pour transmettre clairement les besoins et contraintes.

3. **Leçons apprises :**
    - Un départ non structuré a entraîné une perte de temps initiale. L'adoption d'une méthode agile a corrigé cette faiblesse.
    - L'intégration de pratiques professionnelles comme les merge requests et les tests unitaires a contribué à un résultat final de qualité.