Dossier de Candidature MP2

Fatou Aïdara Ba Yari Spazzadeschi

1 Introduction

La structure fondamentale du jeu est restée inchangée, tout comme la plupart des éléments de l'énoncé. Le jeu débute dans l'aire *Spawn*. Le joueur traversera ensuite des zones générées procéduralement pour, à la fin, affronter et vaincre le boss.

Afin d'améliorer le gameplay, l'histoire et les graphismes, nous avons ajouté un nouveau système pour gérer la lumière et l'histoire avec des dialogues et des cinématiques. De plus, pour simplifier l'implémentation de nos extensions, nous avons amélioré des classes et des méthodes déjà existantes et en avons ajouté de nouvelles.

2 Moteur de Lumière (LightEngine)

2.1 Fonctionnement du système

Le système d'éclairage repose sur une analyse géométrique de l'environnement, fonctionnant selon les étapes suivantes :

1. **Détection des murs :** Le programme initie le processus avec l'`AreaGraph` de l'aire courante. Il utilise les coordonnées via la classe `Corners` pour déterminer les sommets de chaque `DiscreteCoordinate` représentant un obstacle (mur).
2. **Polygones :** Ces sommets sont stockés pour former un polygone (cas habituel) ou plusieurs (si le joueur détruit un mur).
3. **Lancer de rayons (Raycasting) :** Le programme trace des rayons vers chaque point et chaque intersection. La méthode `findIntersectionWithRay(currentPosition, ray)` est utilisée pour analyser le comportement du rayon.
4. **Stockage des intersections :** Des instances de `IntersectionSR` mémorisent les informations (existence de l'intersection, coordonnées). Pour chaque ensemble d'intersections sur un même rayon, seule la plus proche du centre de la source lumineuse est conservée.
5. **Gestion du vide :** Si aucune intersection n'existe dans le rayon de vision (`viewRadius`), de "fausses" intersections sont créées pour permettre l'illumination des espaces ouverts.
6. **Nettoyage :** Les intersections "vides" sont retirées et la géométrie est simplifiée en supprimant les points redondants (segments colinéaires).
7. **Tri :** Les points sont finalement ordonnés selon leur angle par rapport à l'origine pour permettre la construction correcte du polygone lumineux.

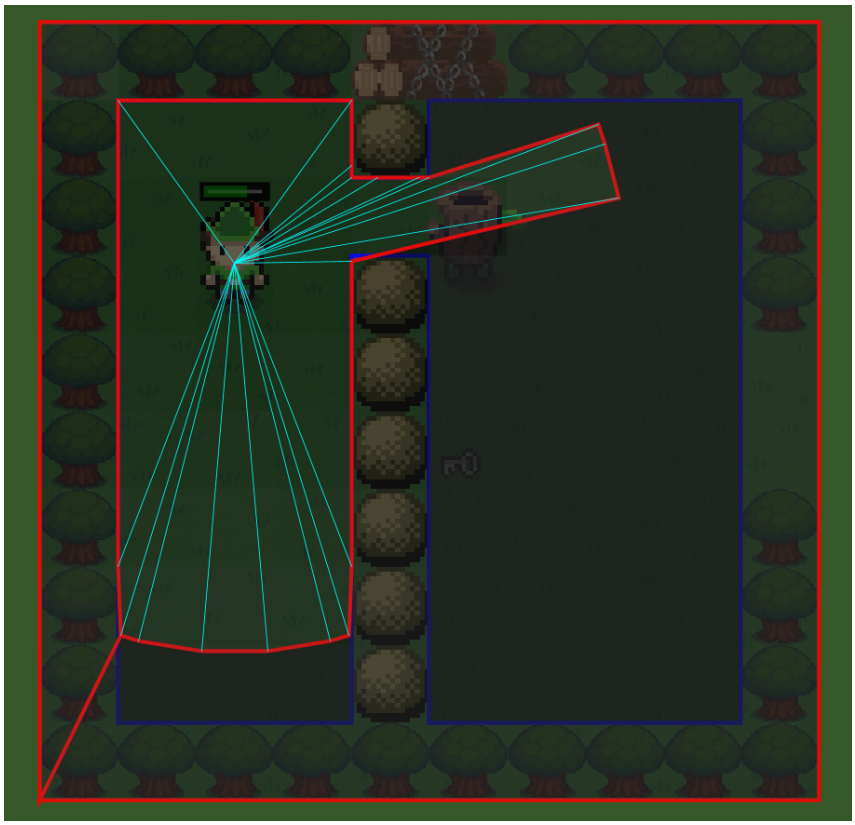


FIGURE 1 – Visualisation de débogage : la variable `debug` permet d'afficher les rayons (raycasting), le périmètre de l'`AreaGraph` et le périmètre de l'aire illuminée.

2.2 Rendu graphique et Ombres

En raison de contraintes liées au moteur de jeu (*GameEngine*) fourni et par choix stylistique, nous avons décidé de ne pas utiliser de masques alpha (*alpha masks*) pour dessiner les ombres.

Nous avons opté pour une approche alternative :

— Un carré est dessiné sur la surface de chaque `DiscreteCoordinate` avec une intensité variable selon la distance et les paramètres du jeu.

— Pour obtenir la zone d'ombre (le complément du polygone illuminé), nous partons du carré formé par les 4 sommets de l'aire globale. Nous relions un de ces sommets aux points d'intersection calculés, traçons le contour des intersections, puis revenons au sommet initial pour fermer la figure.

— Pour chaque `DiscreteCoordinate`, nous vérifions si elle se trouve dans le polygone illuminé en lançant un rayon vers la droite et en comptant les intersections. Si le point est à l'intérieur, il est considéré comme étant dans le champ de vision.

2.3 Cycle Jour/Nuit (SunlightOverlay)

Une valeur alpha (transparence) et des valeurs RGB sont attribuées à chaque coordonnée. Ces valeurs dépendent de la distance au centre, mais aussi des paramètres fournis par la classe `SunlightOverlay`. Celle-ci détermine l'ambiance lumineuse en fonction de la progression temporelle dans le jeu.

Le changement de couleur et d'intensité lumineuse s'effectue graduellement via une interpolation linéaire, permettant des transitions fluides entre les états : jour, coucher de soleil, nuit et éclipse.

FIGURE 2 – Exemple d'illumination avec l'overlay "SUNSET" (Coucher de soleil).

3 Optimisations et Fonctionnalités Avancées

3.1 Optimisations

Plusieurs efforts d'optimisation ont été réalisés. Par exemple, certains calculs lourds ne sont effectués que lorsque le centre de la lumière se déplace ou que l'`AreaGraph` subit des modifications (destruction de murs). Le `LightEngine` fonctionne correctement pour tout type d'aire et gère la plupart des cas limites.

Cette classe offre également des fonctionnalités similaires à `getFieldOfViewCells`, mais en tenant compte des obstacles physiques. Bien que la classe supporte déjà certaines méthodes pour gérer plusieurs sources de lumière, l'utilisation de masques alpha faciliterait une implémentation complète de cette fonctionnalité.

3.2 Le FlameSkull

La gestion de la lumière est utilisée par le nouvel acteur `FlameSkull`. Après la cinématique, le monde est plongé dans le noir à cause d'une éclipse provoquée par le boss. Le `FlameSkull` devient alors la source de lumière principale et guide le joueur vers la clé, puis vers la sortie.

Le système vérifie si le joueur se trouve dans le champ de vision du `FlameSkull`. Si ce n'est pas le cas après un certain temps (*cooldown*), le `FlameSkull` revient chercher le joueur. Il gère également son propre état lors des transitions entre zones déjà visitées, l'aire du boss, ou après la fin du jeu.

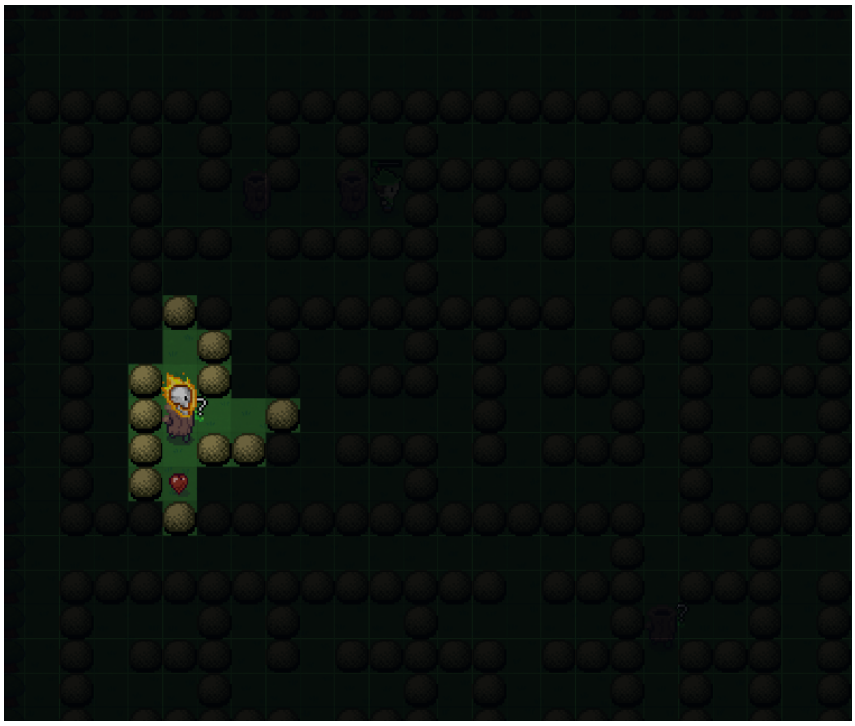


FIGURE 3 – L'acteur FlameSkull éclairant le joueur durant la phase "ECLIPSE".

4 Interface et Audio

4.1 Menus

Nous avons ajouté un gestionnaire de menus (`MenuManager`). Il permet au joueur de mettre le jeu en pause, affichant une interface dédiée avec une musique de fond. Un menu *GAME OVER* est également implémenté, apparaissant lors de la mort du joueur, accompagné de sa propre interface et musique.

(a) Menu Pause

(b) Menu Game Over

FIGURE 4 – Interfaces utilisateur ajoutées au jeu.

4.2 Effets Sonores et Musique

Pour renforcer l'immersion, nous avons intégré de nombreux effets sonores et musiques :
— Des sons différents se déclenchent lors des interactions avec les objets (collectibles), les ennemis ou les rochers.
— Pour rendre les déplacements plus réalistes, le bruit des pas est choisi aléatoirement parmi deux variantes.
— Des musiques spécifiques accompagnent le combat contre le boss et la séquence de victoire.

5 Cinématiques et Narration

5.1 Système de Cinématique

Nous avons créé deux classes et une interface pour implémenter les cinématiques. Vers le milieu du jeu, une séquence se lance où le joueur rencontre les personnages centraux et le boss. Durant cette phase, le joueur perd le contrôle de son personnage et les acteurs se déplacent automatiquement. Cette cinématique inclut également des dialogues. L'aire de la cinématique redevient une aire classique une fois la séquence terminée.

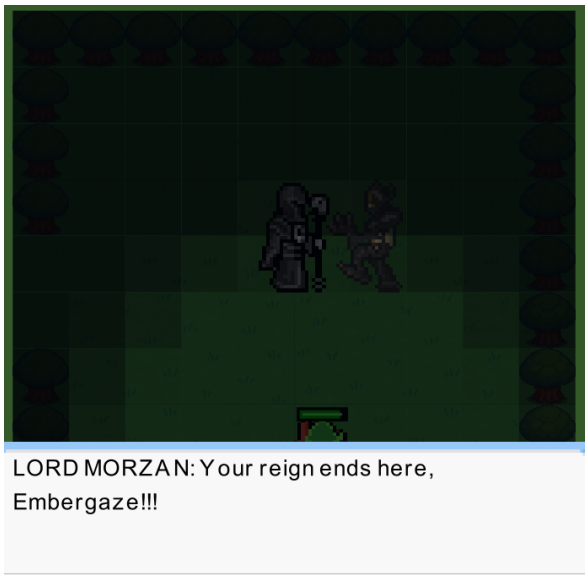
5.2 Histoire et Dialogues

Afin de rendre l'histoire compréhensible, des dialogues ont été ajoutés au début du jeu, pendant la cinématique, le combat de boss et la fin du jeu.

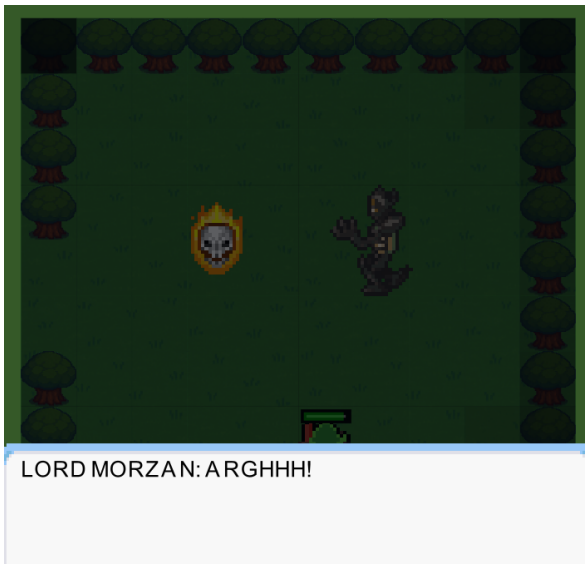
Synopsis : L'histoire débute par une confrontation. Lord Morzan, le plus grand sorcier de l'Empire, a tenté de vaincre le terrible monstre Embergaze (le boss du jeu) pour récupérer la Sainte Épée, un artéfact impérial volé. Ayant échoué, il a été maudit et transformé en un crâne flottant enflammé.

Profitant de son triomphe, Embergaze s'est enfui dans les profondeurs du labyrinthe. Le joueur, après avoir erré pendant des heures, arrive juste à temps pour assister à la scène. Plongés dans l'obscurité totale du labyrinthe, le joueur et Morzan concluent un marché : Morzan servira de torche et utilisera sa magie pour guider le joueur ; en retour, ce dernier l'aidera à se venger.

La mission est claire : traverser le labyrinthe, occire le monstre et récupérer la Sainte Épée pour restaurer l'honneur de l'Empire.



(a) Confrontation initiale



(b) Transformation de Lord Morzan

FIGURE 5 – Séquences de la cinématique narrative montrant les dialogues entre Lord Morzan et Embergaze.