Sommaire

Introduction

Le projet "MAZE GAME" est un jeu interactif où un joueur doit naviguer dans un labyrinthe pour atteindre la sortie tout en échappant à un minotaure. Ce projet combine différents aspects de la programmation : la conception d'algorithmes, l'implémentation d'interfaces graphiques, et l'exploration des capacités de l'intelligence artificielle (IA).

C'est pour cette raison que notre licence nous a demandé de réaliser un projet consistant à créer un site, une application ou même un jeu vidéo à l'aide du code informatique. Cette démarche nous permettra d'apprendre de manière différente, et peut-être plus efficace.

Le projet que nous avons choisi vise à conceptualiser et développer un jeu vidéo. À travers ce projet, nous explorerons les différentes étapes de sa création, depuis la phase initiale de recherche et d'idéation jusqu'au déploiement final. Nous aborderons également les défis potentiels rencontrés en cours de route ainsi que les solutions innovantes mises en œuvre pour les surmonter.

Enfin, ce projet nous offrira une expérience pratique précieuse dans le domaine du développement, en mobilisant des compétences techniques, créatives et en gestion de projet. Notre objectif ultime est de produire un jeu fonctionnel, esthétique et adapté aux attentes de son public cible. C'est dans cette optique que nous entamons ce passionnant voyage vers la création de ce jeu vidéo.

Ce document présente une analyse détaillée du processus de développement, des outils utilisés, et des défis rencontrés. Bien que nous ayons tenté d'intégrer une IA pour générer les labyrinthes, l'intégration complète de cette fonctionnalité n'a pas été réussie. Cependant, ces tentatives ont permis de mieux comprendre les possibilités offertes par les modèles IA et les contraintes techniques liées à leur utilisation.

Idée De Projet

Trouver l'Idée du Projet:

La première étape dans le développement de notre projet consiste à trouver une idée pertinente et intéressante, qui nous permette d'apprendre le plus possible tout en étant la plus complète possible.

Nous avons commencé par organiser des sessions de débat où chaque membre de l'équipe a eu l'occasion de présenter ses idées pour le projet. Ces idées ont été discutées ouvertement, en mettant en avant leurs avantages et leurs inconvénients potentiels.

Une fois plusieurs idées prometteuses générées, nous avons évalué leur viabilité technique. Nous avons analysé les outils nécessaires, ainsi que le niveau de difficulté associé à chaque proposition.

Après des discussions approfondies et une analyse minutieuse, nous avons finalement décidé de nous lancer dans le développement d'un jeu de labyrinthe. Cette décision repose sur plusieurs facteurs, notamment la faisabilité technique élevée du projet et notre confiance dans nos capacités à le mener à bien avec succès.

Le jeu vise à :

- Offrir une expérience ludique et immersive où le joueur doit utiliser stratégie et réflexion pour réussir.
- Expérimenter l'intégration de l'IA dans un jeu pour créer des labyrinthes adaptés.
- Proposer un jeu évolutif avec des niveaux de difficulté.

Contexte et Inspiration

Les labyrinthes ont été une source d'inspiration pour les jeux depuis l'Antiquité. En combinant ce concept classique avec des mécanismes modernes, tels que les graphismes interactifs et les règles adaptatives, ce projet vise à proposer une version numérique unique et engageante.

Fonctionnalités Clés

1. Labyrinthes dynamiques : Génération de labyrinthes différents à chaque partie.

2. Ennemi stratégique : Un minotaure qui poursuit activement le joueur.

3. Modes de difficulté : Facile, moyen, difficile et personnalisé.

4. Interface intuitive : Contrôles simples et visuels attractifs.

Nous avons opté pour ce choix pour plusieurs raisons. Tout d'abord, ce sujet de projet nous intéresse particulièrement, et il représente une opportunité d'explorer comment nous pouvons concrétiser une telle idée.

Ensuite, nous avons estimé que sa réalisation était à notre portée, sans présenter de difficultés excessives tout en offrant un défi stimulant.

Le Commencement Du Projet

Le début de notre projet de création du MAZE GAME a été précédé par une analyse approfondie des outils et des technologies disponibles afin d'atteindre nos objectifs. Nous avons sélectionné les applications et les langages de programmation en fonction de leur adéquation avec les besoins du projet et des compétences de notre équipe. Voici un aperçu des outils que nous avons choisis et de la manière dont nous prévoyons de les utiliser :

Architecture du Jeu

L'architecture repose sur trois modules principaux :

1. Board (Plateau)

- Représente le labyrinthe sous forme de graphe avec des cases, des murs et des emplacements-clés (départ, sortie, position du minotaure).
- Implémente des algorithmes de validation pour assurer la navigabilité.

2. File IO

- Gère la création de labyrinthes, soit par génération aléatoire, soit via des requêtes à l'API OLLAMA.

3. Game

- Supervise la logique du jeu : mouvements, conditions de victoire, et affichage graphique.

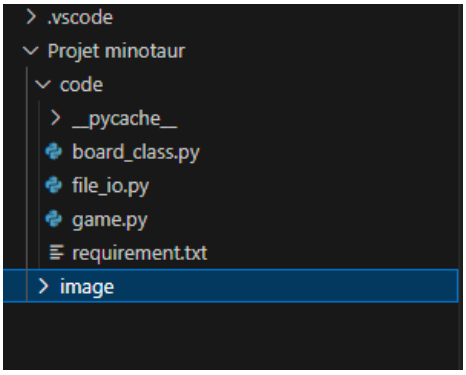
Outils et Bibliothèques

- Python : Langage principal.
- Pygame : Gestion des graphiques et animations.
- NetworkX : Manipulation des graphes pour représenter les labyrinthes.
- OLLAMA API : Génération de labyrinthes adaptés à l'aide d'un modèle IA.
- JSON : Format standard pour la représentation des labyrinthes.

Diagramme des Classes

Le jeu repose sur trois classes principales :

- Board : Crée et valide les labyrinthes.
- Player et Minotaur : Représentent respectivement le joueur et l'ennemi avec des mécanismes de déplacement précis.
- Game : Gère les interactions entre les éléments du jeu.



Apprentissage et exploration continue :

Tout au long du projet, nous encouragerons l'apprentissage continu et l'exploration de nouvelles idées et technologies. Pour approfondir notre compréhension des outils utilisés, nous tirerons parti de ressources en ligne telles que des tutoriels, des documentations officielles, l'aide de notre professeur lors des cours magistraux, ainsi que le soutien d'outils d'IA, comme ChatGPT.

Pour faciliter la collaboration au sein de l'équipe et la communication des idées, nous utiliserons divers outils de travail collaboratif en ligne. En complément des réunions régulières à la faculté, nous prévoyons d'utiliser des plateformes d'appel comme Discord pour organiser des sessions de brainstorming en temps réel, ainsi que des outils de messagerie instantanée pour des échanges rapides et informels.

Developpement

Le développement du projet " MAZE GAME " a impliqué plusieurs phases distinctes allant de la conception initiale du code à l'implémentation des fonctionnalités et à l'optimisation des performances. Cette section présente une analyse approfondie des étapes de développement, en mettant l'accent sur les choix techniques et les méthodes adoptées.

Conception du Code

La structure du projet a été conçue pour assurer une modularité et une extensibilité maximale.

Le code a été organisé autour de trois modules principaux :

1. Board (Plateau) :

- Représente le labyrinthe sous forme de graphe. Chaque nœud correspond à une case et les arêtes définissent les chemins possibles.

- Fournit des méthodes pour valider la structure du labyrinthe et gérer les interactions entre les éléments du jeu.

La création des différentes interfaces de la messagerie s'est faite en utilisant le langage PHP, qui a servi non seulement à la connexion entre la base de données et la messagerie, mais également à la majeure partie du code, combiné avec le langage HTML. PHP a été largement utilisé pour récupérer les données saisies par les utilisateurs, les faire interagir avec la base de données et les autres parties du site. Le code de la messagerie est divisé en différentes actions ou tâches nécessaires, et PHP permet de les lier ensemble, comme illustré ci-dessous :

Exemple :

```
def check_win_condition(self):
    """
    Check the win condition of the game.
    """
    if self.graph.graph["mino_location"] == self.graph.graph["player_location"]:
        return True, False
    elif self.graph.graph["player_location"] == self.graph.graph["goal"] and self.graph.graph["mino_location"] != self.graph.graph["player_location"]:
        return True, True
    else:
        return False, False
```

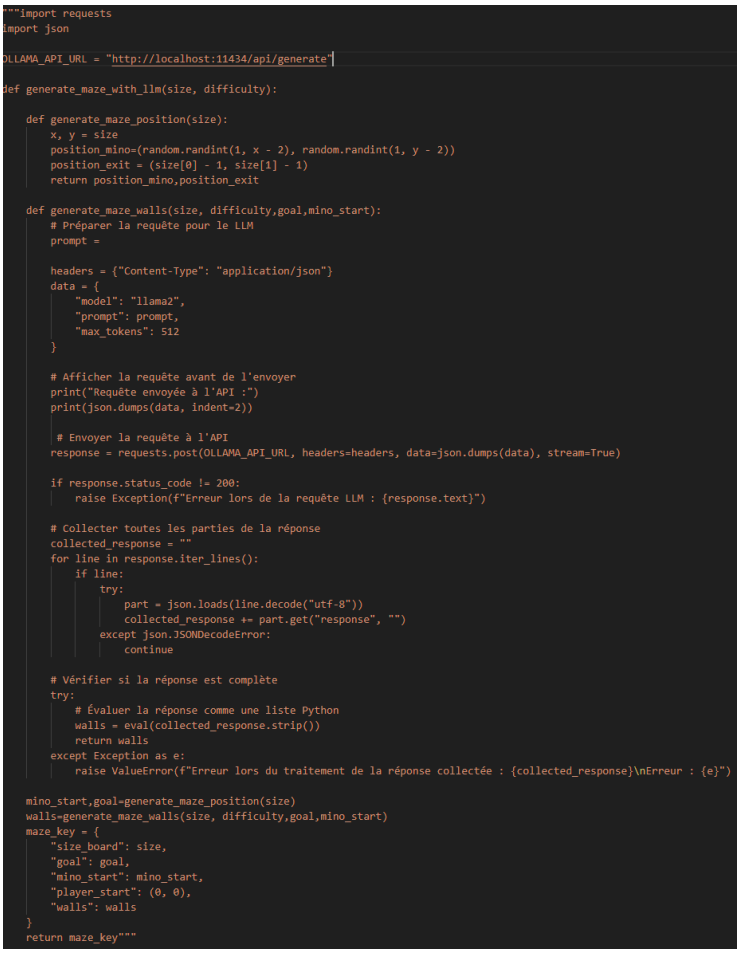
```python
def get_move_options(self, node=None):
    """
    Get the move options for the player and minotaur.
    """
    move_options = {}
    if node:
        move_options["player"] = self.graph.nodes[node]["options"]
        move_options["mino"] = self.graph.nodes[node]["options"][1:]
        return move_options

    player_location = self.get_player_location()
    mino_location = self.get_minotaur_location()
    move_options["player"] = self.graph.nodes[player_location]["options"]
    move_options["mino"] = self.graph.nodes[mino_location]["options"][1:]

    return move_options
```

2. File IO :

- Permet de sauvegarder ou charger des labyrinthes en format JSON pour simplifier les tests et la configuration.
- Gère la création des labyrinthes aléatoires ou générés par une IA via l'API OLLAMA.



```
"""import requests
import json

OLLAMA_API_URL = "http://localhost:11434/api/generate"

def generate_maze_with_llm(size, difficulty):

    def generate_maze_position(size):
        x, y = size
        position_mino=(random.randint(1, x - 2), random.randint(1, y - 2))
        position_exit = (size[0] - 1, size[1] - 1)
        return position_mino,position_exit

    def generate_maze_walls(size, difficulty,goal,mino_start):
        # Préparer la requête pour le LLM
        prompt =

        headers = {"Content-Type": "application/json"}
        data = {
            "model": "llama2",
            "prompt": prompt,
            "max_tokens": 512
        }

        # Afficher la requête avant de l'envoyer
        print("Requête envoyée à l'API :")
        print(json.dumps(data, indent=2))

         # Envoyer la requête à l'API
        response = requests.post(OLLAMA_API_URL, headers=headers, data=json.dumps(data), stream=True)

        if response.status_code != 200:
            raise Exception(f"Erreur lors de la requête LLM : {response.text}")

        # Collecter toutes les parties de la réponse
        collected_response = ""
        for line in response.iter_lines():
            if line:
                try:
                    part = json.loads(line.decode("utf-8"))
                    collected_response += part.get("response", "")
                except json.JSONDecodeError:
                    continue

        # Vérifier si la réponse est complète
        try:
            # Évaluer la réponse comme une liste Python
            walls = eval(collected_response.strip())
            return walls
        except Exception as e:
            raise ValueError(f"Erreur lors du traitement de la réponse collectée : {collected_response}\nErreur : {e}")

    mino_start,goal=generate_maze_position(size)
    walls=generate_maze_walls(size, difficulty,goal,mino_start)
    maze_key = {
        "size_board": size,
        "goal": goal,
        "mino_start": mino_start,
        "player_start": (0, 0),
        "walls": walls
    }
    return maze_key"""
```

3. Game (Gestion du Jeu) :

- Contrôle la logique du jeu, les mouvements des personnages, et les conditions de victoire ou défaite.

```python
def adjust_wall_coordinates(wall):
    """
    Adjust the coordinates of walls to fit within the scaled tile size.
    """
    x1, y1, x2, y2 = wall[0][0], wall[0][1], wall[1][0], wall[1][1]
    if y2 - y1 == 0:
        if x1 > x2:
            x1, x2, y1, y2 = x2, x1, y2, y1
        return (adjust_token_coordinates((x1 + 0.5, y1 - 0.5)), adjust_token_coordinates((x2 - 0.5, y2 + 0.5)))
    elif x2 - x1 == 0:
        if y1 > y2:
            x1, x2, y1, y2 = x2, x1, y2, y1
        return (adjust_token_coordinates((x1 - 0.5, y1 + 0.5)), adjust_token_coordinates((x2 + 0.5, y2 - 0.5)))
```

- Gère l'interface graphique et les interactions utilisateur via Pygame.

```python
def draw_game_over_menu(win_message):
    window = pygame.display.set_mode((600, 400))
    pygame.display.set_caption("Menu Principal")

    font = pygame.font.Font('freesansbold.ttf', 32)
    play_text = font.render("Rejouer", True, (255, 255, 255))
    quit_text = font.render("Quitter", True, (255, 255, 255))

    play_rect = play_text.get_rect(center=(300, 150))
    quit_rect = quit_text.get_rect(center=(300, 250))

    while True:
        window.fill((20, 20, 20))
        window.blit(play_text, play_rect)
        window.blit(quit_text, quit_rect)
        pygame.display.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if play_rect.collidepoint(event.pos):
                    return "play"
                if quit_rect.collidepoint(event.pos):
                    pygame.quit()
                    sys.exit()
```

Implémentation des Fonctionnalités Clés

1. Génération des Labyrinthes La génération des labyrinthes repose sur deux approches principales :

- Algorithm DFS (Depth-First Search) :
- Utilisé pour créer des labyrinthes navigables.
- Cet algorithme assure que chaque chemin a une solution unique.

```python
def adjust_wall_coordinates(wall):
    """
    Adjust the coordinates of walls to fit within the scaled tile size.
    """
    x1, y1, x2, y2 = wall[0][0], wall[0][1], wall[1][0], wall[1][1]
    if y2 - y1 == 0:
        if x1 > x2:
            x1, x2, y1, y2 = x2, x1, y2, y1
        return (adjust_token_coordinates((x1 + 0.5, y1 - 0.5)), adjust_token_coordinates((x2 - 0.5, y2 + 0.5)))
    elif x2 - x1 == 0:
        if y1 > y2:
            x1, x2, y1, y2 = x2, x1, y2, y1
        return (adjust_token_coordinates((x1 - 0.5, y1 + 0.5)), adjust_token_coordinates((x2 + 0.5, y2 - 0.5)))
```

- Utilisation de l'IA (OLLAMA API) :
- Nous avons intégré des appels à l'API pour générer des labyrinthes adaptés en fonction de la difficulté.
- Toutefois, des limitations dans l'implémentation et des problèmes de validation des labyrinthes ont freiné cette fonctionnalité.

2. Mécanismes de Jeu
Le cœur du jeu repose sur les interactions entre le joueur et le minotaure :

- Mouvements du Joueur :
- Le joueur peut se déplacer dans quatre directions (haut, bas, gauche, droite) en fonction des chemins ouverts.
- Des vérifications sont effectuées pour empêcher les mouvements invalides (ex. collision avec un mur).
- Comportement du Minotaure :
- Le minotaure poursuit activement le joueur en suivant des heuristiques simples pour minimiser la distance entre eux.
- Il peut effectuer deux mouvements consécutifs pour augmenter le niveau de difficulté.

```python
class Minotaur:
    def __init__(self, maze):
        """
        Initialize the minotaur with the maze.
        """
        self.maze = maze
        self.location = maze.graph.graph["mino_location"]

    def move(self):
        """
        Move the minotaur based on the movement ruleset.
        """
        mino_location = self.maze.graph.graph["mino_location"]
        player_location = self.maze.graph.graph["player_location"]

        remaining_moves = 1
        move_to = []

        while remaining_moves > 0:
            move_options = self.maze.get_move_options()["mino"]

            if "right" in move_options and player_location[0] > mino_location[0]:
                mino_location = (mino_location[0] + 1, mino_location[1] + 0)
                move_to.append(mino_location)
            elif "left" in move_options and player_location[0] < mino_location[0]:
                mino_location = (mino_location[0] - 1, mino_location[1] + 0)
                move_to.append(mino_location)
            elif "up" in move_options and player_location[1] < mino_location[1]:
                mino_location = (mino_location[0] + 0, mino_location[1] - 1)
                move_to.append(mino_location)
            elif "down" in move_options and player_location[1] > mino_location[1]:
                mino_location = (mino_location[0] + 0, mino_location[1] + 1)
                move_to.append(mino_location)
            else:
                move_to.append(mino_location)

            remaining_moves -= 1
            self.maze.graph.graph["mino_location"] = mino_location

        self.location = mino_location
        return mino_location, move_to
```

3. Interface Graphique

L'interface utilisateur a été développée avec Pygame pour offrir une expérience interactive :

- Affichage des Labyrinthes : Chaque case est représentée graphiquement avec des différents symboles pour le joueur, le minotaure et la sortie.
- Animations : Les mouvements des personnages sont animés pour renforcer l'immersion.
- Menus Intuitifs : Le jeu inclut un menu principal, un écran de game over, et des options de redémarrage ou d'affichage de la solution.

Optimisations Techniques

Pour garantir une bonne performance, plusieurs optimisations ont été mises en place :

1. Gestion des Graphes :

- Utilisation de NetworkX pour représenter efficacement les labyrinthes et calculer les chemins.
- Les murs sont stockés sous forme d'arêtes avec des poids pour simplifier les calculs.

2. Animation et Rafraîchissement :

- Limitation du nombre d'images par seconde (FPS) à 60 pour préserver les ressources CPU.
- Optimisation des redessinages : seuls les éléments modifiés sont réactualisés.

```python
def get_animation_position(before, after, current_frame, max_frames):
    """
    Calculate the position of the token during animation.
    """
    x = before[0] + (current_frame / max_frames) * (after[0] - before[0])
    y = before[1] + (current_frame / max_frames) * (after[1] - before[1])
    return (x, y)
```

3. Validation des Labyrinthes :

- Implémentation d'une fonction de validation pour s'assurer que tous les labyrinthes générés sont résolvables.

```python
def generate_mazetest(size, difficulty):
    x, y = size
    maze_key = {
        'size_board': size,
        'goal': (x - 1, y - 1),
        'mino_start': (random.randint(1, x - 2), random.randint(1, y - 2)),
        'player_start': (0, 0),
        'walls': []
    }

    # Create a grid to represent the maze
    grid = [[0] * y for _ in range(x)]

    # Directions for moving in the grid (right, down, left, up)
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    def in_bounds(x, y):
        # Check if the coordinates are within the grid bounds
        return 0 <= x < len(grid) and 0 <= y < len(grid[0])

    def dfs(x, y):
        # Mark the current cell as visited
        grid[x][y] = 1
        # Shuffle the directions to randomize the maze generation
        random.shuffle(directions)
        for dx, dy in directions:
            nx, ny = x + dx * 2, y + dy * 2
            if in_bounds(nx, ny) and grid[nx][ny] == 0:
                # Mark the wall between the current cell and the next cell as visited
                grid[x + dx][y + dy] = 1
                dfs(nx, ny)

    # Start DFS from the player start position
    dfs(0, 0)

    # Convert the grid to a list of walls
    for i in range(x):
        for j in range(y):
            if grid[i][j] == 0:
                if i > 0 and grid[i - 1][j] == 1:
                    maze_key['walls'].append(((i, j), (i - 1, j)))
                if j > 0 and grid[i][j - 1] == 1:
                    maze_key['walls'].append(((i, j), (i, j - 1)))

    # Remove some walls based on difficulty
    if difficulty == 'easy':
        num_walls_to_remove = int(len(maze_key['walls']) * 0.6)
    elif difficulty == 'medium':
        num_walls_to_remove = int(len(maze_key['walls']) * 0.3)
    elif difficulty == 'hard':
        num_walls_to_remove = int(len(maze_key['walls']) * 0)
    else:
        num_walls_to_remove = int(len(maze_key['walls']) * random.uniform(0, 0.6))

    walls_to_remove = random.sample(maze_key['walls'], num_walls_to_remove)
    maze_key['walls'] = [wall for wall in maze_key['walls'] if wall not in walls_to_remove]

    return maze_key
```

En résumé, le développement du jeu "MAZE GAME" a exigé une attention particulière aux détails pour garantir une expérience de jeu fluide et immersive. Les choix techniques adoptés ont permis de relever la plupart des défis, bien que certaines fonctionnalités, comme l'intégration de l'IA, restent à améliorer dans de futures versions. La robustesse du code et sa modularité offrent une base solide pour des évolutions futures, telles que des labyrinthes en 3D ou des comportements IA plus avancés.

Les Problèmes Rencontrés Et Leurs Solutions

Le développement du projet "MAZE GAME" a présenté plusieurs défis techniques, organisationnels et liés à l'intégration d'outils externes. Cette section détaille les principaux problèmes rencontrés et les solutions mises en place pour les résoudre.

Difficulté

1. Gestion des Dépendances

- Problème : Assurer que toutes les dépendances nécessaires (comme pygame, networkx, etc.) sont installées et correctement configurées.
- Solution : Utiliser un fichier requirements.txt pour gérer les dépendances et s'assurer que l'environnement de développement est correctement configuré.

2. Calcul des Coordonnées

- Problème : Calculer les coordonnées correctes pour les tokens et les murs en fonction de la taille de la fenêtre et de la taille des tuiles.
- Solution : Utiliser des fonctions comme $adjust_token_coordinates$ et $adjust_wall_coordinates$ pour ajuster les coordonnées.

3. Animation des Mouvements

- Problème : Animer les mouvements des tokens de manière fluide.
- Solution : Utiliser des interpolations linéaires pour calculer les positions intermédiaires des tokens pendant les animations.

4. Génération du Labyrinthe

- Problème : Générer un labyrinthe valide avec des murs et des chemins.
- (tenté avec l ia mais pas réussi)
- Solution : Utiliser des algorithmes de génération de labyrinthe comme l'algorithme de parcours en profondeur (DFS) pour créer des chemins et des murs.

5. Validation du Labyrinthe

- Problème : S'assurer que le labyrinthe généré est valide et ne contient pas de sections fermées.

- Solution : Utiliser networkx pour vérifier la connectivité du graphe du labyrinthe.

6. Gestion des Mouvements du Minotaure

- Problème : Implémenter les règles de mouvement du minotaure pour qu'il suive le joueur.

- Solution : Utiliser des règles de mouvement basées sur la position relative du joueur et du minotaure.

7. Optimisation des Performances

- Problème : Assurer que le jeu fonctionne de manière fluide sans ralentissements.

- Solution : Optimiser le code pour minimiser les calculs inutiles et utiliser des techniques de programmation efficaces.

8. Tests et Débogage

- Problème : Tester et déboguer le code pour s'assurer qu'il fonctionne correctement.

- Solution : Utiliser des outils de débogage et écrire des tests unitaires pour vérifier le bon fonctionnement du code.

9. Gestion des Ressources

- Problème : Gérer les ressources (mémoire, CPU) de manière efficace.

- Solution : Utiliser des techniques de gestion de la mémoire et optimiser les calculs pour minimiser l'utilisation des ressources.

10. Compatibilité

- Problème : Assurer la compatibilité du jeu avec différentes plateformes et configurations.

- Solution : Tester le jeu sur différentes plateformes et configurations pour s'assurer qu'il fonctionne correctement.

Test du Projet et Options Ajoutées

Les tests ont constitué une étape essentielle pour garantir la qualité et la jouabilité du jeu de labyrinthe. Nous avons mis en place un plan de test structuré qui couvre plusieurs aspects, notamment la fonctionnalité, les performances, et l'expérience utilisateur. Voici un compte rendu détaillé des différents scénarios de test, des résultats obtenus, et des options ajoutées en réponse aux retours des tests. Objectifs des Tests

- Validation Fonctionnelle : Vérifier que toutes les fonctionnalités principales du jeu fonctionnent comme prévu.
- Détection des Bugs : Identifier et corriger les anomalies logicielles.
- Évaluation des Performances : Mesurer la fluidité du jeu sur différentes configurations matérielles.
- Amélioration de l'Expérience Utilisateur : Recueillir les retours des utilisateurs pour affiner le gameplay.

Scénarios de Test

1. Tests Fonctionnels

Objectif : Valider les fonctionnalités principales du jeu, notamment :

- Génération des labyrinthes.
- Déplacements du joueur et du minotaure.
- Conditions de victoire et de défaite.

Résultats :

- Les labyrinthes générés sont correctement navigables et respectent les paramètres définis (taille, difficulté).
- Les mouvements du joueur et du minotaure sont fluides et conformes aux règles définies.
- Les conditions de victoire (atteindre la sortie) et de défaite (capture par le minotaure) fonctionnent correctement.

2. Tests de Performance Objectif : Mesurer l'impact des labyrinthes de grande taille et des animations sur les performances. Résultats :

- Les performances restent stables pour des labyrinthes jusqu'à 50x50 cases sur la plupart des machines.

- Pour des labyrinthes de très grande taille (¿100x100 cases), une baisse de performance est observée sur les configurations matérielles limitées.

- Les optimisations implémentées, telles que la réduction dynamique des dimensions et l'optimisation des redessinages, ont significativement amélioré la fluidité.

3. Tests Graphiques

Objectif : Vérifier la qualité des graphismes et des animations sur différentes résolutions. Résultats :

- L'affichage des labyrinthes est net et adapté à toutes les résolutions d'écran testées (HD, Full HD, 4K).

- Les animations des mouvements du joueur et du minotaure sont fluides, même sur des configurations matérielles modestes.

- Les ajustements automatiques des dimensions garantissent une expérience visuelle cohérente.

5. Tests Utilisateurs

Objectif : Recueillir des retours qualitatifs de la part des utilisateurs pour améliorer le gameplay.
Résultats :

- Les utilisateurs ont trouvé le jeu intuitif et immersif.

- Des suggestions ont été faites pour ajouter des options telles que :

- Un mode tutoriel pour les nouveaux joueurs.
- Des niveaux supplémentaires avec des labyrinthes plus complexes.

Options Ajoutées

En réponse aux tests, plusieurs options ont été intégrées pour enrichir l'expérience de jeu :

Optimisation de l'Interface :

- Amélioration de l'ergonomie des menus et des options de navigation.

Conclusion des Tests

Les tests ont permis de valider l'ensemble des fonctionnalités du jeu et d'identifier des axes d'amélioration. Grâce aux retours des utilisateurs et aux optimisations techniques, le jeu offre désormais une expérience fluide, immersive et adaptable à différents profils de joueurs. Ces efforts témoignent de notre engagement à fournir un produit de qualité et évolutif pour répondre aux attentes des utilisateurs.

Résumé du Développement

Le développement du jeu "MAZE GAME" a suivi une approche méthodique, visant à créer une application robuste et évolutive. Cette section récapitule les étapes principales et les efforts déployés pour concrétiser les objectifs fixés.

Points Forts du Développement

1. Modularité et Organisation du Code :

- La structure en modules indépendants (Board, File IO, Game) a permis une séparation claire des responsabilités, rendant le code plus facile à maintenir et à étendre.

- Chaque module a été conçu pour être réutilisable dans de futurs projets, notamment le générateur de labyrinthes et les algorithmes de validation.

2. Utilisation de l'Intelligence Artificielle :

- L'intégration de l'API OLLAMA pour générer des labyrinthes adaptatifs basés sur des paramètres de difficulté spécifiques a échoué. Bien que prometteuse, l'utilisation d'une API externe nécessite une compréhension approfondie de ses limites et une validation rigoureuse des données.

3. Graphismes et Interactivité :

- L'utilisation de Pygame a permis de créer une interface graphique immersive avec des animations fluides et des éléments visuellement distincts (joueur, minotaure, sortie).

- Les ajustements dynamiques aux résolutions d'écran garantissent une expérience cohérente pour tous les utilisateurs.

4. Tests et Itérations :

- Une attention particulière a été portée aux tests fonctionnels et de performance, aboutissant à un produit final stable.

- Les retours des utilisateurs ont été intégrés rapidement pour améliorer le gameplay et ajouter des fonctionnalités pertinentes (mode tutoriel, difficulté personnalisée).

Défis Majeurs et Solutions

1. Performance sur de Grands Labyrinthes :

- La gestion de labyrinthes de très grande taille (¿100x100) a initialement posé des problèmes de latence.

- L'optimisation des calculs, combinée à une limitation dynamique de la taille des labyrinthes, a permis de maintenir une fluidité acceptable.

2. Synchronisation des Animations :

   - Les mouvements simultanés du joueur et du minotaure ont nécessité une coordination minutieuse pour éviter les décalages visuels.

   - Un système basé sur des frames fixes a été implémenté pour assurer une synchronisation parfaite.

3. Validation des Labyrinthes :

   - Les labyrinthes générés aléatoirement ou par l'IA contenaient parfois des erreurs, comme des chemins bloqués.

   - Une fonction de validation automatique a été intégrée pour garantir la navigabilité avant l'affichage.

Innovations et Améliorations
Adaptabilité des Niveaux :

- La génération semi-aléatoire des labyrinthes a permis de proposer des défis variés tout en maintenant un niveau de difficulté équilibré.

Synthèse Le développement du projet a permis de relever des défis techniques complexes et de produire un jeu à la fois captivant et techniquement solide. Les efforts combinés en conception, implémentation et optimisation témoignent d'une approche rigoureuse et axée sur l'expérience utilisateur. Ce projet constitue une base solide pour des évolutions futures, notamment l'intégration d'éléments 3D ou de nouvelles mécaniques de jeu.

Conclusion et Perspectives

Le projet "MAZE GAME" illustre le potentiel de la collaboration entre programmation traditionnelle et utilisation de technologies modernes comme l'intelligence artificielle. En dépit des défis rencontrés, nous avons réussi à créer une application ludique, immersive et techniquement aboutie.

Accomplissements Principaux

- La modularité du code garantit une extensibilité pour de futures améliorations.
- L'utilisation de Pygame a permis de fournir une interface utilisateur fluide et attrayante.
- Les fonctionnalités supplémentaires, comme le menu ou l'aide aux contrôles enrichissent l'expérience utilisateur.

Leçons Apprises

- Optimisation des Performances : Les grands labyrinthes posent des défis uniques, soulignant l'importance de tests approfondis sur des configurations variées.
- Feedback Utilisateur : Intégrer les suggestions des testeurs a été crucial pour améliorer le produit final.

Perspectives d'Avenir

Le jeu "Jeu de Labyrinthe" constitue une base solide pour de futures améliorations:

1. Extension des Fonctionnalités :

- Introduction de nouveaux ennemis avec des comportements variés.
- Création d'un mode multijoueur permettant à plusieurs joueurs de collaborer ou de s'affronter.
- Création d'un site ou bien d'une application.

2. Intelligence Artificielle Avancée :

- Utilisation de modèles plus complexes pour générer des labyrinthes plus sophistiqués.
- Développement d'une IA capable d'apprendre et de s'adapter aux stratégies des joueurs.

En conclusion, ce projet marque une étape importante dans notre apprentissage de la conception de jeux vidéo. Il met en lumière les défis inhérents à

l'intégration de l'IA et à l'optimisation des performances, tout en ouvrant la voie à de nombreuses possibilités d'évolution. Nous espérons que cette expérience inspirera d'autres projets mêlant créativité, technique et innovation.

Bibliographie

1. Documentation de Pygame : cliquez ici

2. NetworkX pour les graphes : cliquez ici

3. API OLLAMA : Documentation interne de l'API utilisée pour générer les labyrinthes.

4. Algorithme DFS pour la génération des labyrinthes : Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009). Introduction to Algorithms.

5. JSON et manipulation de données : cliquez ici

6. Optimisation des performances graphiques : Références issues des forums Pygame et Stack Overflow.

7. Tests logiciels et méthodologies : Myers, G.J., Sandler, C., Badgett, T. (2011). The Art of Software Testing.

Webographie

1. Tutoriel sur l'utilisation de Pygame : Real Python - Getting Started with Pygame : cliquez ici

2. Documentation officielle de Python : cliquez ici

3. Forum de développeurs sur Stack Overflow (sections liées à Pygame et IA) : cliquez ici

4. Explications sur les algorithmes de graphes : GeeksforGeeks - Graph Algorithms : cliquez ici

5. Introduction à l'intelligence artificielle pour les jeux : Towards Data Science - Using AI to Generate Game Levels : cliquez ici

6. Exemples d'implémentations de labyrinthes : GitHub Repositories - Maze Generation Algorithms : cliquez ici

Annexe

Installation : Prérequis

- Python 3.x
- Pygame
- NetworkX
- Requests

Étapes d'installation :

1. Clonez le dépôt :
git clone https://github.com/Tibxla/Maze-game.git

2. Installez les dépendances :
pip install -r requirements.txt

Utilisation

Lancer le jeu : Pour lancer le jeu, exécutez le fichier principal main.py :

- python game.py

Options de ligne de commande : Vous pouvez spécifier la taille et la difficulté du labyrinthe en utilisant des arguments de ligne de commande (par défaut c'est random):

size : Taille du labyrinthe (small, medium, large, random).
difficulty : Difficulté du labyrinthe (easy, medium, hard, random).

Exemples :

- python game.py medium hard

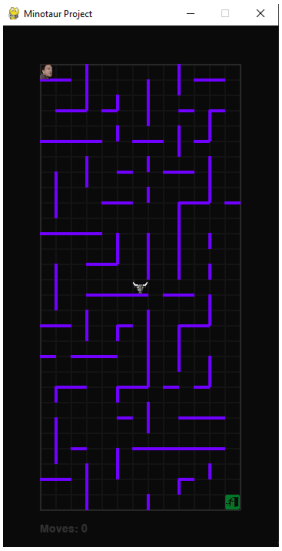
Contrôles :
Monter : Z ou Flèche Haut
Descendre : S ou Flèche Bas
Gauche : Q ou Flèche Gauche
Droite : D ou Flèche Droite
Passer : Espace
Réinitialiser : Retour Arrière
Annuler : Maj

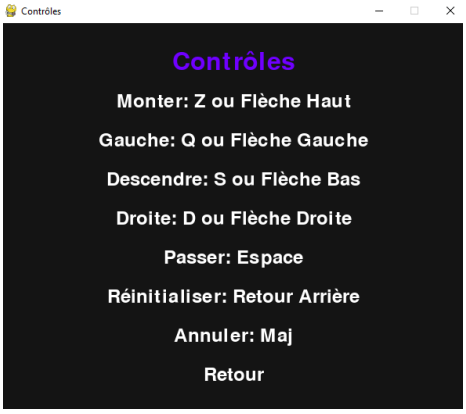
Exemple d'Exécution du Code :

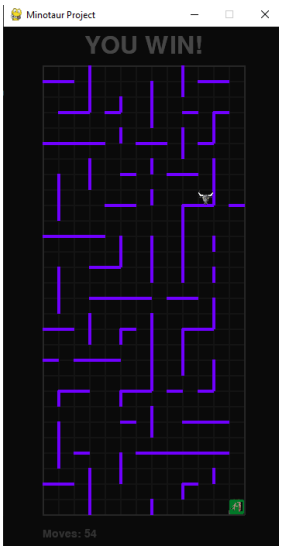
Voici un exemple d'exécution du code pour lancer le jeu avec un labyrinthe de taille random et de difficulté random :

- python game.py

Captures d'Écran :




Contrôles
Contrôles
Monter: Z ou Flèche Haut
Gauche: Q ou Flèche Gauche
Descendre: S ou Flèche Bas
Droite: D ou Flèche Droite
Passer: Espace
Réinitialiser: Retour Arrière
Annuler: Maj
Retour


Minotaur Project
YOU WIN!
Moves: 54