Conception Logiciel
Rapport sur le projet

RAMDANI Adam
OUAFI Rabah
MAKHLOUF Feras
MANSOUS Younes

25 avril 2022

Projet : Jeu de labyrinthe à gravité

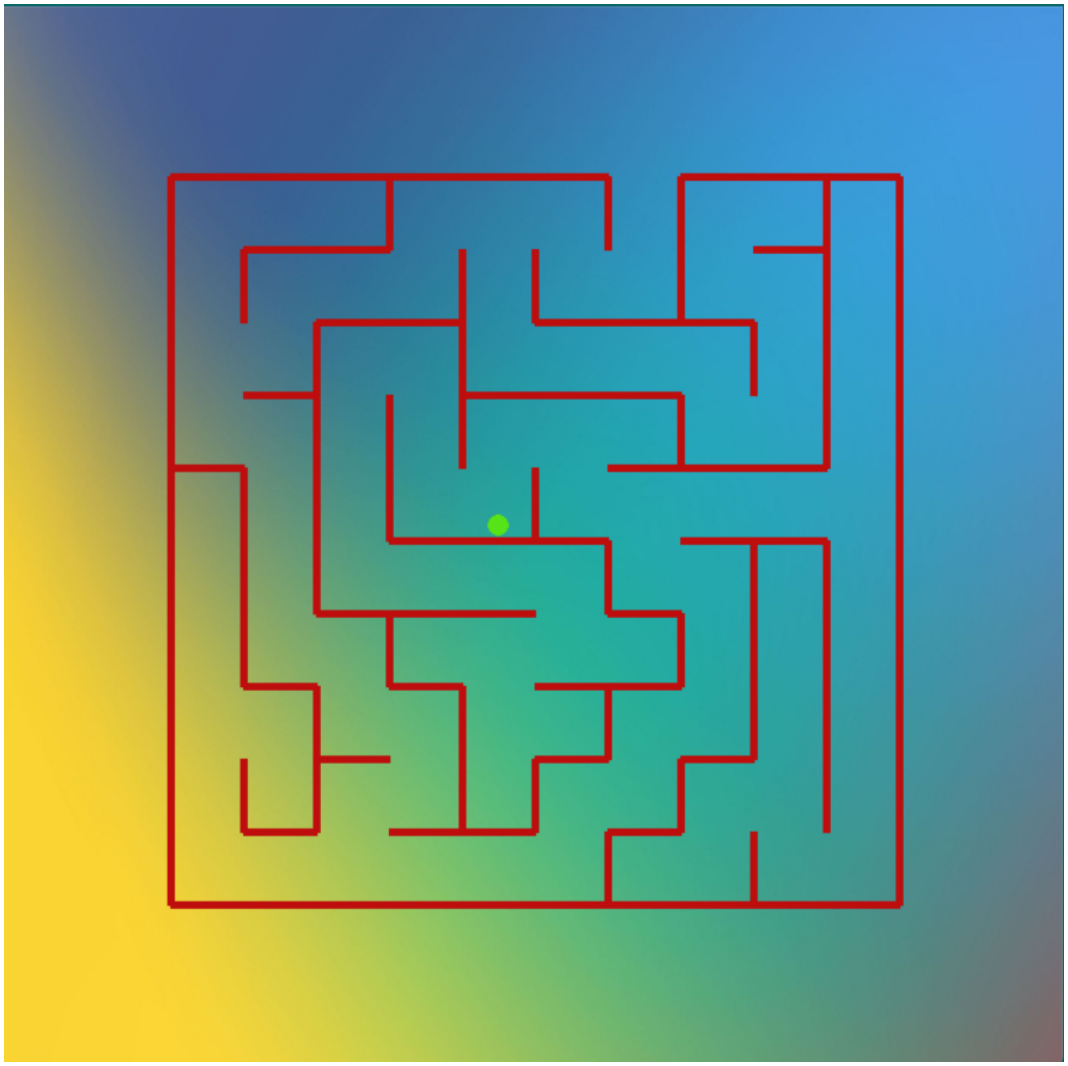

Table des matières

1 Introduction **3**
1.1 Description générale du projet . . . 3
1.2 Présentation du plan du rapport . . . 3

2 Objectifs du projet et organisation **3**
2.1 Objectifs . . . 3
2.2 Organisation du Groupe . . . 3

3 Fonctionnalités **4**
3.1 Fonctionnalités principales . . . 4
3.1.1 Une génération aléatoire du labyrinthe . . . 4
3.1.2 Une rotation . . . 4
3.1.3 Un moteur physique dynamique . . . 4
3.2 Fonctionnalités secondaires . . . 4

4 Éléments techniques **5**
4.1 Génération aléatoire du labyrinthe parfait . . . 5
4.2 Le système de rotation . . . 5
4.3 Implémentation d'une physique dynamique . . . 6
4.3.1 Bibliothèque *Pymunk* . . . 7
4.4 Affichage graphique . . . 7
4.4.1 Affichage du menu . . . 7
4.4.2 Affichage du labyrinthe et de la balle . . . 7
4.5 Les différents niveaux de difficultés . . . 7

5 Architecture du projet **8**
5.1 Diagrammes des modules et des classes . . . 8
5.2 Chaînes de traitement . . . 9

6 Conclusion **10**
6.1 Récapitulatif . . . 10
6.2 Problèmes rencontrés . . . 10
6.3 Propositions d'améliorations . . . 10

1 Introduction

1.1 Description générale du projet

Ce projet est un programme réalisé en python. Il s'agit d'un jeu de *labyrinthe à gravité* dans lequel une balle doit se déplacer depuis le centre du labyrinthe jusqu'à une sortie *généré automatiquement* sur l'un des quatre murs extérieurs. La balle étant soumise à la gravité peut se déplacer en *tournant* le labyrinthe. Le choix de ce jeu est fait grâce à la compatibilité des connaissances et des capacités des membres du groupe avec ce que demande la réalisation de ce projet.

1.2 Présentation du plan du rapport

Ce rapport a pour but d'expliquer les différentes étapes suivies afin de réaliser ce jeu. Il porte sur une description des grandes étapes et des points clés du projet ainsi que les éléments implémentés et son organisation.
Il consiste aussi à identifier les structures de données utilisées, les algorithmes et l'architecture du projet (Diagrammes des modules et des classes)
Enfin, en guise de conclusion, un récapitulatif général du résultat finale, ainsi que les améliorations possible à faire et les problèmes rencontrés.

2 Objectifs du projet et organisation

2.1 Objectifs

Pour que le jeu soit opérationnel, il nous a fallu mettre en place trois fonctionnalités principales qui sont :

- La génération aléatoire du labyrinthe.
- La rotation.
- Un moteur physique dynamique.

Une fois ces trois objectifs atteint, plusieurs suggestions ont été proposé par les membres du groupe pour l'améliorer et l'enrichir comme :

- L'implémentation d'un menu.
- L'ajout de plusieurs niveaux de difficultés (facile, moyen, difficile)
- La personnalisation des couleurs utilisées.
- L'ajout de objets utilitaires comme des télé-porteurs, des points-bonus.
- L'implémentation d'un chronomètre.

2.2 Organisation du Groupe

Pour mener à bien ces objectifs et pouvoir les atteindre, notre groupe s'est équitablement répartie les tâches, tout en s'entraidant et en échangeant les idées et les

connaissances entre les membres.

L'implémentation du moteur physique a été majoritairement traité par OUAFI Rabah, en ce qui concerne la génération automatique du labyrinthe MANSOUS Younes s'en est occupé. La rotation de la balle et du labyrinthe a été réalisé par RAMDANI Adam, et enfin la balle et le menu ont été géré par MAKHLOUF Feras.

3 Fonctionnalités

3.1 Fonctionnalités principales

On a réussi à réaliser tout les objectifs principaux. En effet, notre programme comporte actuellement :

3.1.1 Une génération aléatoire du labyrinthe

Comme le titre l'indique, à chaque nouvelle partie un nouveau labyrinthe différent du dernier est généré, cela est possible en suivant un certain algorithme spécialisé dans la génération des labyrinthes parfaits.

3.1.2 Une rotation

En utilisant des règles mathématiques et trigonométriques, on a réussi à simuler une rotation du labyrinthe par rapport au centre de la fenêtre, selon un certain degré de rotation paramétrable, tout ça en utilisant les deux touches du clavier « flèche gauche »et « flèche droite ».

3.1.3 Un moteur physique dynamique

Certainement la fonctionnalité la plus difficile à implémenter, on a tout de même réussi à le faire en important la bibliothèque *Pymunk*, qui nous a facilité la simulation de la gravité et de la collision entre la balle et les éléments du décors.

3.2 Fonctionnalités secondaires

Quant aux objectifs secondaires, la plupart d'entre eux on été atteint, notamment :

- Un écran d'accueil qui fait office de menu.
- Trois niveaux de difficulté qui se caractérisent en :
  - Facile : tout le labyrinthe est affiché, ce qui facilite la recherche du chemin de sortie.
  - Moyen : Le champ du vision est restreint et centré sur la balle.
  - Difficile : Le champ de vision est restreint encore plus ce qui rend la sortie difficile à trouver.
- Changement des couleurs utilisées sur la balle et le labyrinthe à chaque nouvelle partie de façon aléatoire.

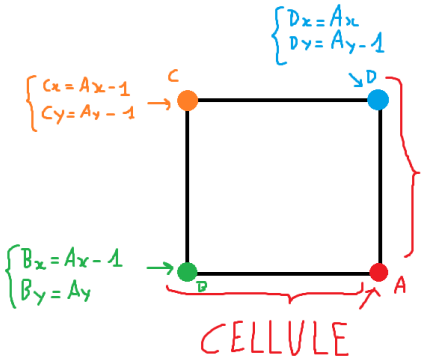

FIGURE 1 – Représentation d'une cellule

4 Éléments techniques

4.1 Génération aléatoire du labyrinthe parfait

Pour réussir cela, on a d'abord commencé par générer tout les points de tout les segments du labyrinthe (une grille entière) grâce à la méthode *Generation* dans la classe labyrinthe, s'en suit la génération des segments horizontaux et verticaux avec les deux méthodes *SegmentVerticalt* *SegmentHorizontal*.

Une fois cela fait, on obtient une grille avec plusieurs cellules selon la taille choisie au préalable, mais comme notre code ne comporte pas de classe cellule (difficile de les représenter), on a eu l'ingéniosité de représenter chaque cellule avec son point en bas à droite, ce dernier va la définir comme si c'était son index, puis on stock tout les points de ces cellules dans une liste "points".

Pour conclure, On fait appel à notre algorithme qui lui, va suivre les étapes suivantes :

- Choisir un point aléatoire depuis la liste des points qui représentent les cellules comme point de départ et l'ajouter à une liste des *Points visités*.
- vérifier quelles sont les cellules voisines *disponibles* (qui ne sont pas encore visitée).
- S'il y a au moins une possibilité, on en choisi une au hasard, et on supprime le segment qui les sépare puis on recommence toutes ces étapes depuis le début avec la nouvelle cellule.
- S'il n'y en pas, on revient à la case précédente et on recommence.

L'algorithme s'arrête quand la taille de la liste des points visitées est égale au carré de la taille de la grille (carré du nombre de cellules).

4.2 Le système de rotation

Pour être exacte, la rotation se fait non pas sur le labyrinthe mais plutôt sur les points qui le composent, de même pour la balle.

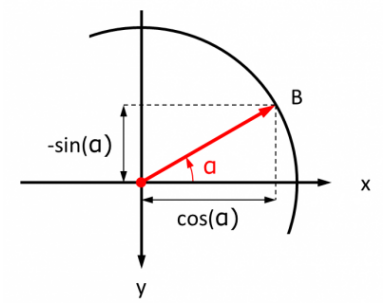

FIGURE 2 – Méthode de rotation

Le calcul pour la rotation des points se fait avec des calculs de trigonométrie, on va d'abord de calculer l'image d'un point par rotation d'un angle définie au préalable autour du centre de la fenêtre du jeu puis on remplace ses anciennes coordonnées par les nouvelles.

Voici les équations utilisées : (La hauteur et la largeur sont celle de la fenêtre)

$$x' = (x - largeur/2) * \cos\theta - (y - hauteur/2) * \sin\theta + largeur/2 \quad (1)$$

$$y' = (x - largeur/2) * \sin\theta + (y - hauteur/2) * \cos\theta + hauteur/2 \quad (2)$$

La direction de rotation est contrôlée avec les deux touches du clavier, flèche droite et flèche gauche.

4.3 Implémentation d'une physique dynamique

Sans doute la partie la plus délicate du projet, car si la simulation de la gravité est assez facile, gérer la collision entre la balle et les murs du labyrinthe l'est beaucoup moins.

On a d'abord essayé de détecter cette collision en tentant de prédire si les bords de la balle se trouve à l'intérieur d'un mur, ce qui revient à vérifier si un point fait partie d'un segment en suivant ces deux étapes :

1. Vérifier si les deux points du segment et le bord de la balle sont sur la même ligne.
2. Si la première condition est vraie, vérifier si la distance de la balle est inférieur à la longueur du segment.

Mais cette technique est assez lente et pas très fiable (beaucoup d'erreurs de calculs à cause des nombres flottants).

Après plusieurs recherches, on a fini par trouver une bibliothèque Python qui fait exactement ce qu'on veut, c'est à dire la simulation d'une gravité et la détection de collision avec les éléments du décors, cette bibliothèque est *Pymunk*

4.3.1 Bibliothèque *Pymunk*

Pymunk est une bibliothèque de physique 2D en python facile à utiliser, qui simule les interactions physiques entre les objets. Elle possède plusieurs paramètres

à intégrer et à définir, comme le sens et la force de gravité, un espace, une masse pour les objets dynamique comme la balle.

Son fonctionnement consiste à créer un espace de travaille « space », donner des corps « body »en précisant si c'est un corps dynamique comme pour la balle ou un corps statique comme pour les murs. On l'a implémenter dans la classe Balle et la classe Labyrinthe. Une fois le labyrinthe créé, elle s'occupe de donner à tout les segments en utilisant leur points qui les définissent un corps « body »statique.

4.4 Affichage graphique

L'affichage de l'interface graphique est effectué directement avec la bibliothèque *Pygame* et ses différentes méthodes.

4.4.1 Affichage du menu

L'affichage du menu était assez simple à faire, on a nous mêmes créé les « sprites »qui vont représenter les différents boutons du menu tel que « PLAY, Hard, Normal, Easy, QUIT », mais aussi la bannière du jeu.

Ensuite, on implémente tout en utilisant la méthode *Pygame.surface.blit*. Enfin en récupérant les évènements de la souris depuis *Pygame.events* on a pu attribué à chaque bouton son rôle et son fonctionnement.

4.4.2 Affichage du labyrinthe et de la balle

Une fois le labyrinthe créé, son affichage consiste uniquement à :

1. Parcourir ses segments et récupérer leur coordonnées.
2. Dessiner des lignes avec *Pygame.draw.line* en utilisant les coordonnées récupérées pour représenter ses murs.
3. Dessiner la balle avec *Pygame.draw.circle* selon ses coordonnées actuelles.
4. Actualiser l'affichage à chaque tour dans la boucle du jeu avec *Pygame.draw.update*.

4.5 Les différents niveaux de difficultés

En voulant rendre le jeu plus attractif, on a décidé que l'ajout de différents niveaux de difficultés était nécessaire, et pour cela on s'est directement attaqué au rayon de vision du joueur de cette façon :

- **Difficulté facile** : Le rayon de vision n'est pas altéré et l'affichage de labyrinthe est effectué dans sa totalité.
- **Difficulté moyenne** : Le rayon de vision est réduit, en conséquence l'affichage des différents murs est soumis à une condition de distance, on affichera que les segments qui sont à un maximum de 200 unités de distance de la balle.

- **Difficulté : élevée** : Le rayon de vision est réduit encore plus, jusqu'à un maximum de 125 unités de distances depuis la balle.

Voici pour illustrer, les deux niveaux de difficulté, moyenne et élevée :

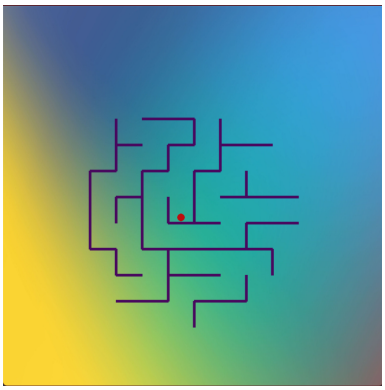

FIGURE 3 – Difficulté : moyenne

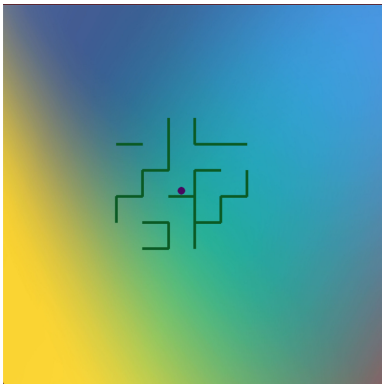

FIGURE 4 – Difficulté : élevée

5 Architecture du projet

5.1 Diagrammes des modules et des classes

Pour illustrer la structure des classes utilisées dans notre code, on a choisi de construire un diagramme *UML* qui représente les différentes classes utilisées et leurs attributs.

Ce diagramme sert à représenter graphiquement les différentes classe utilisées et la relation existante entre elles mais aussi tout les aspects uniques de chacune des classes comme les coordonnées, la couleur, la taille.

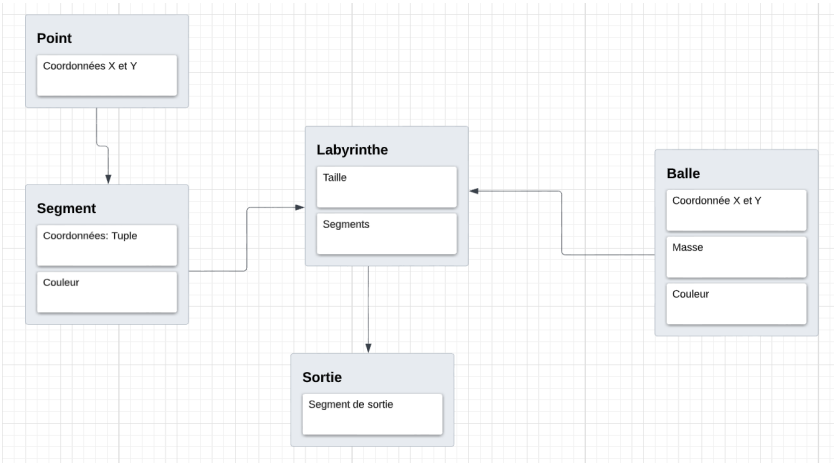

FIGURE 5 – Diagramme des classes

5.2 Chaînes de traitement

Chaque classe utilisée est définie par des attributs et des méthodes qui permettent de traiter les différentes étapes à la réalisation du jeu :

- **Classe Point** : c'est une classe qui sert à identifier les point du labyrinthe par rapport à l'axe des abscisses et celui des ordonnées. Elle prend comme attribut deux coordonnées x et y qui sont multipliées pour que ça soit aménagés vers les dimensions de la fenêtre de l'interface graphique.
  Et pour les méthodes, elle en a une seule qui est la fonction *rotation* ayant comme argument l'angle de rotation thêta (θ) et qui permet de faire tourner les points du labyrinthe à l'aide des équations vu précédemment.
- **Classe Segment** : c'est une classe qui sert à identifier les segments du labyrinthe en fonction des points identifiés par la *classe point*. Les segment sont sous forme de deux points reliés entre eux ; le point de départ et le point d'arrivée qui sont sous forme d'un tuple. En ce qui concerne les méthodes il y en a deux, la première est la fonction *dessiner* qui permet de dessiner les segments avec *pygame.draw.line*, la deuxième méthode est la fonction *rotation* qui se sert de la même fonction dans la classe point pour tourner les segments.
- **Classe Labyrinthe** : c'est la classe qui sert à générer automatiquement le labyrinthe et ses segments identifiés dans la *classe segment*.
  Cette classe est constituée de plusieurs méthodes qui permettent de gérer l'implémentation de l'algorithme exhaustive qu'on a expliqué ainsi que la méthode qui permet la rotation et celle de l'affichage.
- **Classe Ball** : la classe qui s'occupe de la création de la balle et son mouvement à l'intérieur du labyrinthe. Elle prend comme attribut ses coordonnées, la couleur et la masse.
  Cette classe est caractérisée par la méthode *afficher* qui permet de dessiner la balle à l'aide de la fonction *pygame.draw.circle*, en plus de la méthode *rotation* qui sert à tourner la balle avec les mêmes équations et relations citées auparavant.
- **Classe Control** : c'est la classe dédiée au contrôle de l'affichage et de la définition de notre balle et labyrinthe.
  Elle contient aussi la méthode *Update* qui est utilisé pour rafraîchir l'image et actualiser l'affichage à chaque fin de boucle qui fait tourner le jeu et ainsi avoir en temps réel les changement effectués.

- **Classe Game** : Pour un maximum de flexibilité, on a opté pour la création d'une classe qui va gérer tout les paramètre modifiable du jeu et les stocker dans son constructeur, ainsi tout les paramètres comme la taille, la difficulté, les couleurs seront facile d'accès.

6 Conclusion

6.1 Récapitulatif

Les principaux buts posés lors de la réalisation de ce projet ont été atteints. Le labyrinthe a été généré automatiquement en se référant à l'un des algorithmes associés qui est l'exploration *exhaustive*, le moteur physique qui consiste au déplacement de la balle étant soumise à la gravité sur le labyrinthe afin d'atteindre la sortie a été mise en place. Et pour finir la maniabilité et la rotation du labyrinthe et de la balle marche parfaitement et harmonieusement avec le moteur physique.

6.2 Problèmes rencontrés

Lors de la réalisation de ce projet, on a rencontré des problèmes qui nous ont bloqués pendant un certain moment et il a fallu intensivement nous creuser la tête pour trouver des solutions adéquates à chacun d'entre eux. Pour en citer quelques-uns, on a :

- La détection de la collision entre la balle et les murs du labyrinthe est sans doute le casse tête le plus désagréable, après avoir recherché, suivie, et appliquer les algorithmes nécessaires a son bon fonctionnement des problèmes mineurs surviennent toujours tels que la balle qui traverse les murs une fois sur deux, ou quand elle se fige subitement sur un mur, on as fini par choisir de simuler les collisions et la gravité avec le module *Pymunk* qui est très facile à comprendre et a employer.
- La génération aléatoire du labyrinthe nous a aussi posé quelques problèmes, car l'algorithme sur le quel on s'est basé pour la programmer utilise des cellules et notre programme ne contient pas de définition pour ces cellules, alors on as du ruser en projetant chaque cellule (4 points) sur un seule point, ainsi, pour avoir accès au trois autres points de la cellule on les exprime a l'aide des coordonnée de ce point.
- l'affichage du labyrinthe et de la balle a aussi représenté un challenge pour nous, car ça devait se faire uniquement avec la méthode *Pygame.draw* de *Pygame* et non pas avec des images qui nous aurait grandement facilité la tâche.

6.3 Propositions d'améliorations

Parmi les objectifs qu'on s'est fixé, il y a eu certains qui n'ont malheureusement pas pu être réalisé dû à notre manque de connaissances et de temps, on voulait implémenter un compte à rebours qui dès qu'il arrive à 0 la partie est perdu, afin d'ajouter un peu de challenge au jeu. on avait aussi suggéré d'ajouter des téléporteurs dans le labyrinthe et des objets qu'on peut collecter et qui confèrent au joueur certains bonus comme la capacité de flotter pendant un certain temps.
En tout cas, pour un premier projet de cette envergure, on est content et fière que notre groupe aie atteints ses objectifs majeurs et que le jeu soit bien codé avec le *minimum* de bugs possibles.