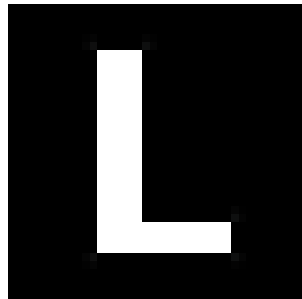


# **Projet de jeu de labyrinthe à génération procédurale**



## Table des matières :

<b>Présentation du projet</b>	p.3
<b>Idée initiale</b>	p.3
<u>Objectifs</u>	p.3
<u>Outils</u>	p.3
<b>Développement</b>	p.4
<u>Début avec Tkinter</u>	p.4
<u>Génération procédurale</u>	p.5
<u>Système de collisions et clef</u>	p.6
<u>Les pièges</u>	p.8
<u>Menu "Options"</u>	p.8
<u>Mode de jeu "Arcade"</u>	p.9
<u>Le fonctionnement des threads</u>	p.9
<u>Répartition et gestion des fichiers du jeu</u>	p.10
<u>Création de l'exécutable</u>	p.10
<b>Versions</b>	p.11
<b>Produit final</b>	p.11
<b>Conclusion</b>	p.12
<b>Sitographie/Logithèque</b>	p.13
<u>Sitographie</u>	p.13
<u>Logithèque</u>	p.13

## **Présentation du projet :**

Ce projet consiste à créer un labyrinthe à génération procédurale. Ainsi, j'allais pouvoir apprendre à utiliser une interface graphique, et commencer à découvrir la génération procédurale, en générant quelque chose de simple : un labyrinthe. Mon objectif principal était d'apprendre. Je voulais, à l'arrivée, être capable de faire de nouvelles choses et également avoir une idée concrète sur la manière de créer d'autres jeux plus ambitieux dans le futur.

## **Idee initiale :**

### Objectifs :

Initialement ce projet devait remplir trois objectifs :

- Apprendre à utiliser une interface graphique, à créer un jeu.
- Le joueur doit avoir un intérêt à jouer à ce jeu, il doit être amusant. Je ne voulais pas d'un simple jeu se terminant en quelques minutes, il doit donc y avoir une certaine rejouabilité.
- Ce jeu doit théoriquement être accessible au plus grand nombre, même si je ne le posterais jamais sur Internet.

Le labyrinthe devait donc contenir :

- Un menu depuis lequel le joueur peut choisir la difficulté de la partie. La difficulté correspondant à la taille du labyrinthe.
- À chaque nouvelle partie, un nouveau labyrinthe inédit doit être généré de manière procédurale.
- Le joueur devait commencer au milieu du labyrinthe, et la sortie devait être sur un des quatre côtés.

### Outils :

Pour réaliser ce projet, j'ai utilisé :

- Notepad++, l'éditeur de texte que j'utilise pour coder depuis mes débuts en informatique.
- Gimp, un logiciel de dessin que je connais depuis longtemps, qui m'est utile pour les graphismes.
- yEd Graph Editor, un logiciel de graphique, m'ayant permis de faire un schéma explicatif sur le fonctionnement du jeu.
- Python 3.7.4, qui est la version de python que je possède.
- Tkinter, l'interface graphique installée avec python par défaut.
- Les cours d'OpenClassrooms, leur forum, le forum de developpez.net et quelques autres sites et blogs pour me renseigner.

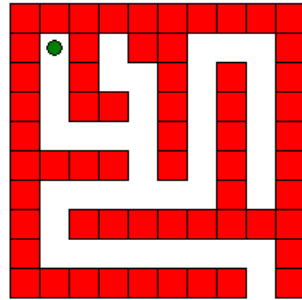
## Développement :

### Début avec Tkinter :

J'ai tout d'abord trouvé sur un forum un prototype me permettant d'avoir un labyrinthe avec Tkinter à partir d'un tableau correspondant à la map du labyrinthe. Les collisions sont gérées à partir des couleurs. Le personnage est un rond vert, les murs sont en rouges, et le fond en blanc. Si le personnage est quelque part où il y a du rouge, alors il est dans un mur, il est donc remplacé à sa position précédente.

```
laby = [  
    "+++++++",  
    "+P+  ++  +",  
    "+ +  + + +",  
    "+ ++ + + +",  
    "+      + + +",  
    "+++++ + + +",  
    "+      + + +",  
    "+ ++++++",  
    "+      +",  
    "+++++++"  
]
```

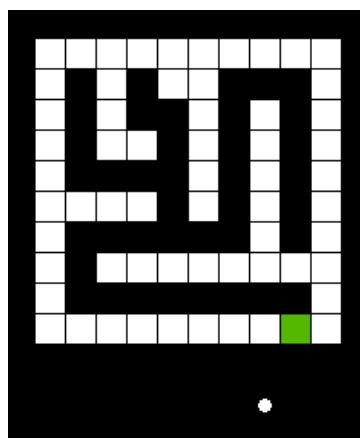
Map



Rendu à l'écran

J'ai ensuite amélioré ce prototype en changeant les couleurs et en ajoutant une sortie. Je voulais que les murs et le personnage soient blancs sur un fond noir. Cependant, les collisions étant définies par les couleurs, si le personnage est blanc, la fonction détectant les couleurs aux coordonnées du joueur détectera toujours du blanc. J'ai alors utilisé l'hexadécimal, le joueur est en #ffffffe, et les murs en #ffffff.

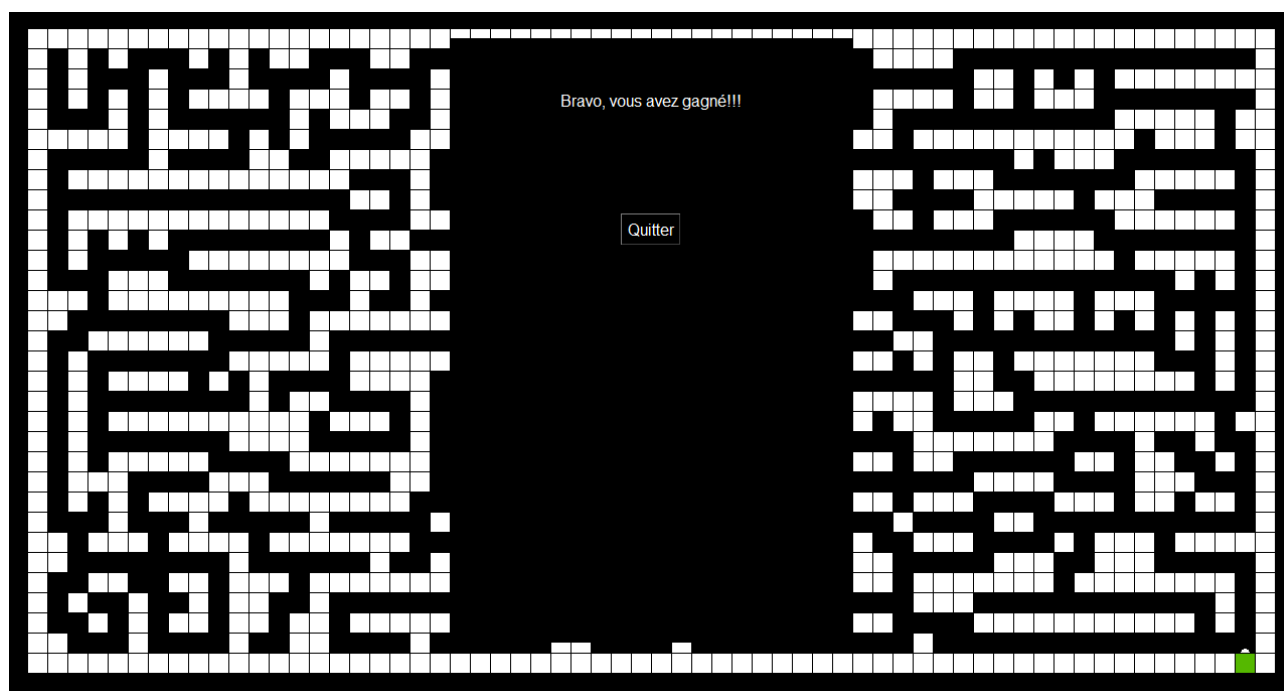
La sortie est en vert, même si elle ne sert à rien : le joueur peut la traverser, sortir du labyrinthe, et continuer à jouer, aucun message de victoire n'apparaît dans cette version.



2<sup>e</sup> version avec le joueur sorti du labyrinthe

J'ai ensuite travaillé sur une autre version, avec un labyrinthe plus grand (32 cases par 62 cases), mais toujours sans génération procédurale. Il y a désormais un menu de victoire, et le joueur ne peut plus bouger une fois la partie terminée.

De plus, lorsque le joueur appuie sur la touche "Echap", un menu pause apparaît indiquant les touches de contrôles ("Z", "Q", "S", et "D"), même si le joueur peut encore se déplacer dans le labyrinthe lorsque le menu pause est ouvert.



Labyrinthe 32x62 avec le menu de victoire

#### Génération procédurale :

La génération procédurale est une manière de générer absolument tout (donjon, labyrinthe, forêt, montagne, ou même des animaux) de manière aléatoire mais en respectant certains critères. Dans un labyrinthe, les critères principaux sont :

- Tout endroit du labyrinthe doit être accessible depuis n'importe quel endroit de ce dernier.
- Il ne doit y avoir que des couloirs, aucune grande étendue vide.

Au départ, le labyrinthe est intégralement composé de murs, et d'une case vide correspondant à l'emplacement de départ du joueur. La génération procédurale, va donc remplacer certains de ces murs par des cases vides, est ainsi les creuser. Les premiers couloirs creusés partiront tous du joueur, puis les autres partiront de ces premiers couloirs, etc. Ainsi tout endroit du labyrinthe sera alors accessible depuis n'importe quel endroit de ce dernier.

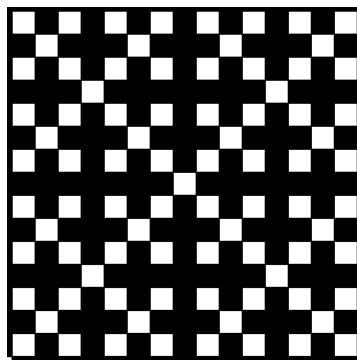
L'idée de départ était d'utiliser un algorithme de Backtracking :

- Nous partons de l'emplacement du joueur, nous sélectionnons aléatoirement une case adjacente et si nous pouvons la creuser, nous la creusons.
- Nous recommençons ensuite à partir de cette nouvelle case creusée, si nous ne pouvons pas creuser cette case, alors nous repartons depuis la dernière case où nous avons changé de direction. De plus si la case est déjà creusée, alors nous reprenons depuis cette case déjà creusée pour chercher un nouveau chemin à creuser.

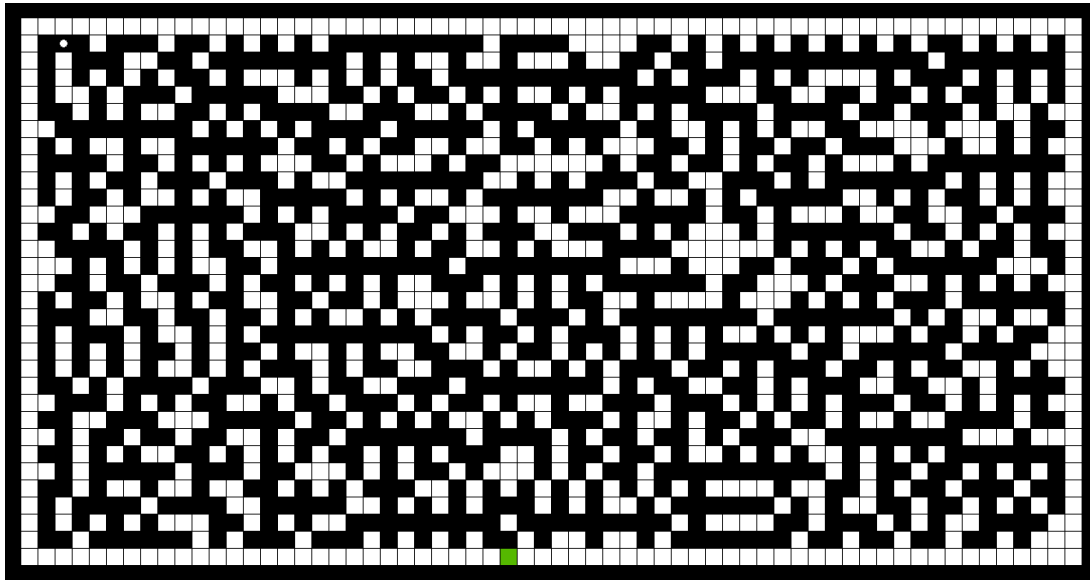
Cependant, après plusieurs versions et une semaine d'acharnement, la génération était peu satisfaisante : de grandes étendues vides apparaissaient un peu partout dans le labyrinthe, et la génération était extrêmement lente, plus de 15 min pour un labyrinthe d'environ 15 cases par 25.

J'ai alors tenté d'utiliser un autre algorithme, l'algorithme de Growing Tree : le principe est que le labyrinthe sera intégralement lu plusieurs fois, et des couloirs seront petits à petits creusés, comme pousseraient les branches d'un arbre.

Le tableau est lu par une boucle "for" tant que des cases sont creusées. À chaque nouvelle case non creusée sélectionnée, nous vérifions s'il y a une seule et unique case déjà creusée adjacente, si oui, alors il y a une chance sur 2 que cette case soit creusée. Si les cases étaient systématiquement creusées, nous nous retrouverions avec des fractales.



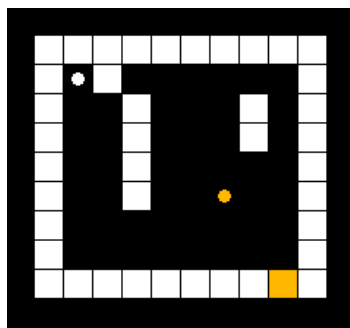
Exemple de fractales



Un labyrinthe généré de manière procédurale avec l'algorithme de Growing Tree

#### Système de collisions et clef :

J'ai ensuite rajouté une clef qu'il est nécessaire d'aller chercher pour déverrouiller la sortie. Cette clef est symbolisée par le point jaune.



Prototype avec la clef

Cependant, cette clef est peu intuitive : il n'est pas évident pour le joueur que ce point jaune soit la clef. J'ai alors voulu remplacer ce point par un gif animé. Pour cela il fallait refaire tout le système de collisions qui fonctionnait encore avec les couleurs. J'ai alors modifié la fonction "lecture", qui crée le labyrinthe à l'écran, pour me permettre de stocker les coordonnées des murs dans la liste "walls", ainsi que les coordonnées de la sortie et de la clef dans les variables "E" et "K".

La fonction "collisions" capture ensuite les coordonnées du joueur à chaque déplacement et les compare à celles stockées dans "walls", "E", et "K" pour savoir si le personnage est dans un mur, à la clef, ou à la sortie.

```

dico = {}
walls = []
for nb, lines in enumerate(map):
    dico[nb] = {}
    for nnb, lettre in enumerate(lines):

        if lettre == 1:
            line = Canva.create_rectangle(X+0, Y+0, X+20, Y+20, fill = '#ffffff')
            dico[nb][nnb] = line
            X = float(X)
            Y = float(Y)
            co = [X+5, Y+5, X+15, Y+15]
            walls.append(co)

        elif lettre == 9:
            line = Canva.create_rectangle(X+0, Y+0, X+20, Y+20, fill = '#ffffff')
            dico[nb][nnb] = line
            X = float(X)
            Y = float(Y)
            co = [X+5, Y+5, X+15, Y+15]
            walls.append(co)

        elif lettre == 'E':
            fin = Canva.create_rectangle(X+0, Y+0, X+20, Y+20, fill = '#ffb700')
            dico['End'] = line
            a, b = X, Y
            X = float(X)
            Y = float(Y)
            E = [X+5, Y+5, X+15, Y+15]

        elif lettre == 'P':
            ov_player = Canva.create_oval(X+5, Y+5, X+15, Y+15, fill = '#ffffffe')
            dico['Player'] = ov_player
            P = [X+5, Y+5, X+15, Y+15]

        elif lettre == 'K':
            Key = Canva.create_oval(X+5, Y+5, X+15, Y+15, fill = '#ffb701')
            dico['Key'] = Key
            X = float(X)
            Y = float(Y)
            K = [X+5, Y+5, X+15, Y+15]

    X += Z
X = 20
Y += Z

```

Code de la fonction "lecture"

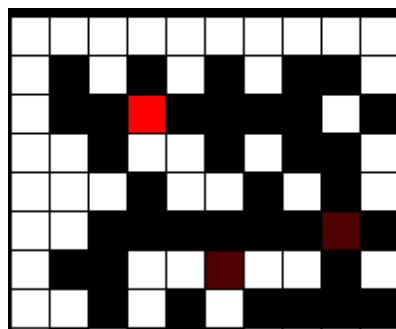


Désormais, les collisions ne dépendent plus de la couleurs des objets. Cependant, il fallait définir si la sortie était ouverte ou non. Pour cela il fallait créer certaines variables propres à la fonction "collisions". C'est pourquoi au début de chaque partie la fonction "collisions" est initialisée. Elle est appelée, et si le joueur se trouve à la position de départ, alors les variables nécessaires sont créées. La variable indiquant la position initiale du joueur est ensuite remplacée par une liste vide, pour éviter que si le joueur repasse par l'emplacement de départ, la sortie ne se referme. En effet, un nouveau passage par la case de départ entraînerait une réinitialisation de la fonction "collisions". Ces variables, bien que définies dans la fonction "collisions", doivent être globales, puisque la fonction est appelée plusieurs fois.

### Les pièges :

J'ai alors voulu rajouter des ennemis avec un déplacement aléatoire, mais leurs déplacements étaient justement trop imprévisibles.

J'ai décidé de remplacer les ennemis par des pièges qui s'activent, pendant 2 secondes, en rouge puis se désactivent en marron. Juste avant de s'activer, le temps de 0,5 seconde, le piège est en jaune, pour prévenir le joueur. Le nombre de pièges est défini par la taille du labyrinthe, et leur activation est gérée par un thread. Les coordonnées des pièges actifs sont transmises à la fonction "collisions" qui est alors appelée par le thread pour déterminer si le joueur a perdu ou non.

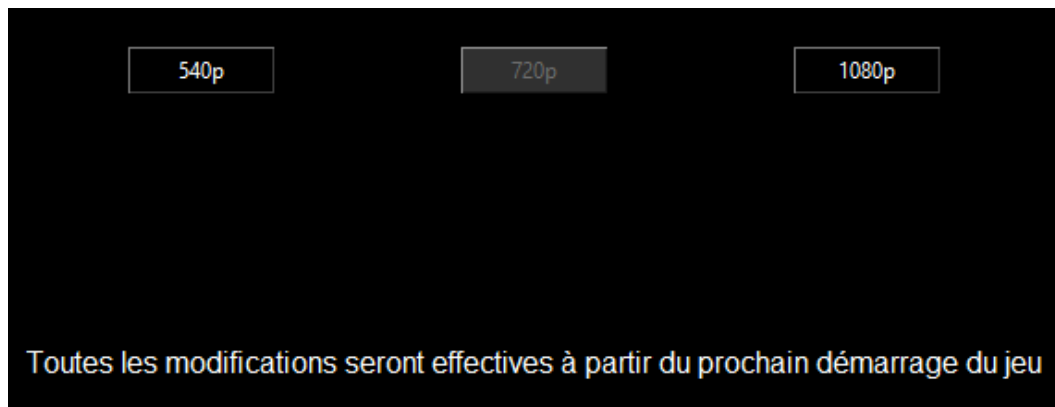


Exemples de pièges

### Menu "Options" :

Le menu "Options" permet de changer les contrôles du personnage et la résolution du jeu.

- Je me suis rendu compte en faisant tester mon jeu que certaines personnes n'aimaient pas ces contrôles, et j'avais déjà remarqué que sur certains écrans la résolution ne serait pas adaptée. Le joueur peut donc utiliser n'importe quelle touches du clavier, sauf la touche "Echap", qui annule la modification, ou une touche déjà utilisée pour un déplacement. Il est impossible d'avancer et de reculer avec la même touche.
- Le joueur a aussi le choix entre 3 résolutions : le 540p, le 720p, et le 1080p. Le jeu a été conçu en 720p.



Choix de la résolution dans le menu "Options"

Pour le choix de la résolution, je n'ai pas utilisé de RadioButtons, je n'arrivais pas à leur donner une esthétique qui me convenait. J'ai alors créé trois boutons qui fonctionnent sur le même principe : le bouton actif est désactivé et a une autre couleur.

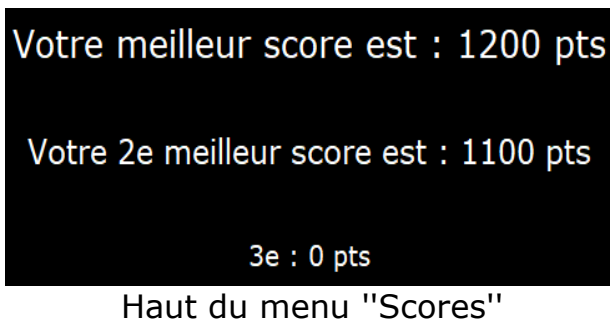
### Mode de jeu "Arcade" :

Le mode de jeu "Arcade" permet de remplir un des objectifs initiaux, celui de la rejouabilité. Dans ce mode "Arcade", le joueur a 5 minutes (soit 300 secondes) pour terminer le plus de labyrinthes possibles. À chaque fois qu'il termine un labyrinthe, son score s'incrémente de 100 points. La partie se termine soit quand le temps est écoulé, soit quand le joueur meurt dans un piège. Le score est alors enregistré dans un fichier.

297,00 s score : 0 pts

Timer et score lors d'une partie

Le joueur peut consulter ses scores dans un menu dédié.



Votre meilleur score est : 1200 pts

Votre 2e meilleur score est : 1100 pts

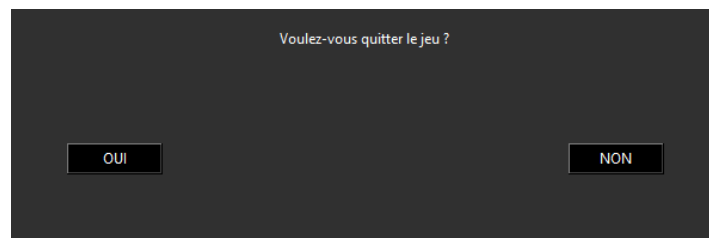
3e : 0 pts

Haut du menu "Scores"

Dans ce mode de jeu, le timer, le score, l'appel de nouveaux labyrinthes... sont gérés par un thread.

#### Le fonctionnement des threads :

Tout au long du développement du jeu, les différents threads provoquaient des erreurs lors de la fermeture. J'ai réussi à tous les fermer correctement sans avoir d'erreur, mais pour cela il est nécessaire d'avoir un message de confirmation lors de la fermeture du jeu. La partie en cours sera en revanche systématiquement détruite.

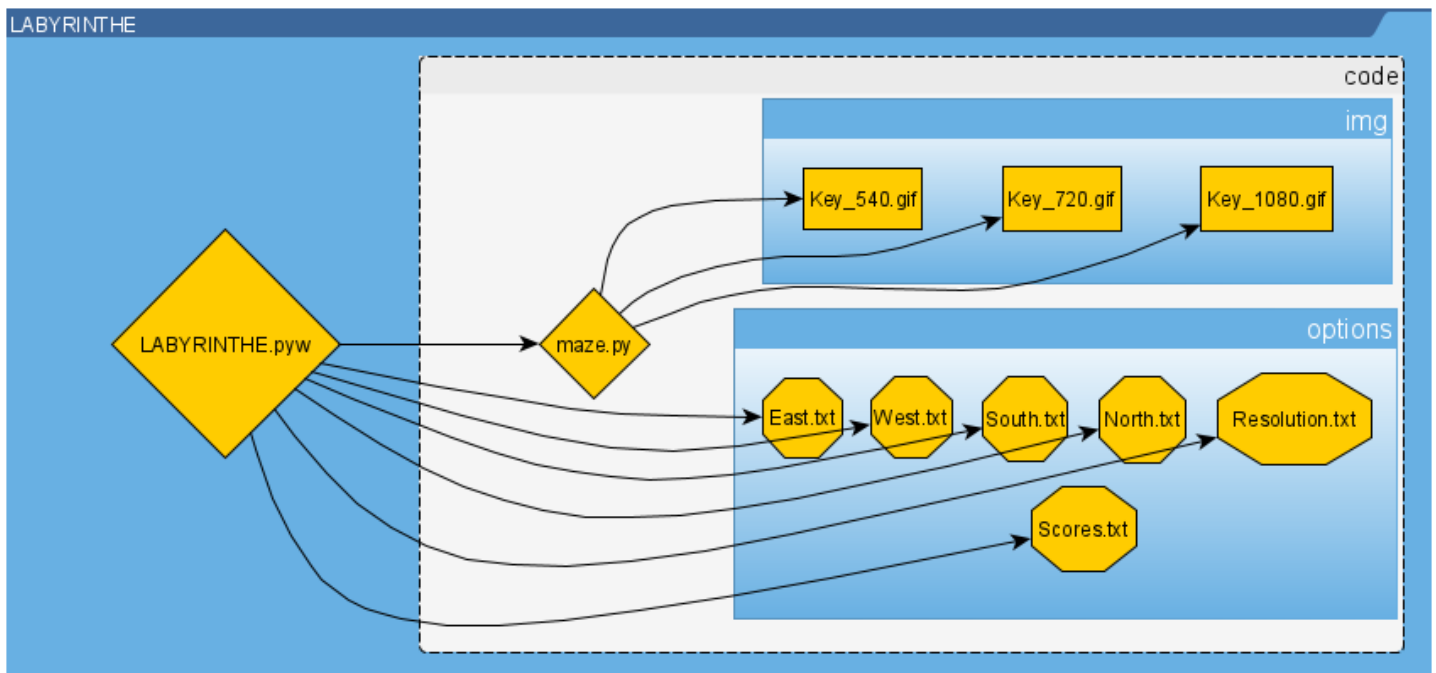


Message de confirmation de fermeture du jeu

Les threads peuvent aussi communiquer entre eux grâce à la fonction "collisions" qui contient plusieurs variables que les threads lisent pour savoir ce qu'ils doivent faire. Par exemple, un thread appelle la fonction "collisions" pour modifier une de ses variables, ensuite lorsqu'un autre thread lira cette variable, il pourra réagir en conséquence.

## Répartition et gestion des fichiers du jeu :

Les fichiers du jeu sont répartis de la manière suivante :



maze.py est un module permettant de générer les labyrinthes, de les créer sur le canva, ainsi que de gérer les collisions et les déplacements. maze.py choisit également l'un des trois gif animés en fonction de la résolution. Le module maze.py importe le gif, l'animation est effectuée par un thread dans LABYRINTHE.pyw.

Les documents East.txt, West.txt, North.txt et South.txt correspondent aux différentes touches de déplacements.

Resolution.txt correspond à la résolution du jeu.

Scores.txt est la liste des scores enregistrés en mode de jeu "Arcade".

## Création de l'exécutable :

Après cette version, dans l'objectif que le jeu soit accessible au plus grand nombre, j'ai transformé le jeu en un exécutable Windows contenant l'interpréteur python et les modules nécessaires. Pour créer cet exécutable, j'ai utilisé le module python cx\_Freeze.

lib	01/05/2020 13:26	Dossier de fichiers	
Labyrinthe	01/05/2020 13:26	Application	20 Ko
python37.dll	08/07/2019 20:29	Extension de l'application	3 522 Ko

Exécutable Windows du jeu

## **Versions :**

- Prototype : prise en main de Tkinter, création d'un premier labyrinthe.
- 1.0 : création d'un labyrinthe de 32 cases par 62 à la main ainsi que d'une sortie fonctionnelle, affichant un message de victoire.
- 1.1 : intégration de la génération procédurale et de la clef.
- 1.2 : la gestion des collisions ne se fait plus en fonction des couleurs, mais des coordonnées enregistrées pour chaque objet. Intégration du gif animé pour la clef. Ajout d'un menu principal et d'un sous-menu pour choisir la difficulté de la partie et la présence ou non d'une clef dans cette partie.
- 1.3 : ajout du menu "Options", et donc de la possibilité de modifier les contrôles du personnage et la résolution du jeu.
- 1.4 : ajout des pièges.
- 1.5 : ajout du mode de jeu "Arcade".
- 1.6 : dernière version, correction des derniers bugs, et dernière modification pour améliorer le jeu, comme le menu pause, qui permettait de chercher son chemin sans perdre de temps dans le mode "Arcade" par exemple.

## **Produit final :**

Le labyrinthe final possède donc :

- Un mode de jeu standard dans lequel le joueur peut choisir la difficulté et s'il veut jouer avec une clef ou non.
- Un mode de jeu "Arcade" avec un score. Les scores sont enregistrés et l'objectif est de battre son propre record ou celui d'un ami s'il a également le jeu.
- Un menu "Options" permettant de changer les contrôles du personnage et la résolution du jeu.

Tous les problèmes ont été résolus et aucun bug n'a été trouvé durant les tests. Le labyrinthe est donc plus complet que la version prévue initialement.

Ce labyrinthe comporte cependant quelques défauts :

- L'utilisation des threads a généré des erreurs m'ayant pris beaucoup de temps à résoudre. Même si ces problèmes concernent principalement la fermeture du programme, je les aurai tout de même eus avec un programme qui n'utilise pas d'interface graphique.
- Les changements d'options ne sont effectifs qu'après un redémarrage du jeu.
- La clef est floue dans certaines résolutions.

- L'enregistrement des scores du mode "Arcade" : tous les scores sont enregistrés dans un fichier, il n'y en a jamais de supprimé, bien que seuls les 10 meilleurs scores soient affichés dans le menu du jeu, cette gestion des scores n'est donc pas optimisée.

Cette expérience m'a appris énormément de choses : utiliser Tkinter et les threads par exemple.

Ce projet a donc rempli tous les objectifs initiaux :

- M'apprendre de nouvelles choses : entre autre, utiliser une interface graphique.
- Avoir une rejouabilité presque infinie, même si le jeu d'une manière générale est lassant.
- Être accessible au plus grand nombre de personnes, grâce à l'exécutable Windows.

Même si ce jeu comporte des défauts, ils ne sont pas gênants au point d'entraver l'expérience de jeu du joueur, ce qui est l'essentiel.

Je pourrai ensuite améliorer ce projet :

- En utilisant des avatars à la place du point blanc pour le joueur.
- En prenant le temps de faire des ennemis qui cherchent un chemin pour aller sur le joueur, et non des ennemis qui ont des déplacements imprévisibles.
- Ce que je rajouterais en premier, c'est une bande son. Je n'ai pas eu le temps de m'intéresser au son dans python pour ajouter des musiques ou bruitages ce qui aurait permis d'avoir une ambiance propre au jeu. Ce pendant, je ne considère pas l'absence de bande son comme un véritable défaut : le jeu a un design simpliste, et n'a pas réellement besoin d'une bande son.

## **Conclusion :**

En conclusion : ce projet m'a permis de créer un jeu, d'apprendre, et le résultat final est réussi. Le jeu ne comporte pas de bugs et il ne manque pas de fonctionnalités importantes au jeu. Je suis donc satisfait de ce que j'ai réussi à produire.

Je n'ai pas prévu améliorer ce jeu de labyrinthe. Je pense plutôt m'orienter vers de nouveaux projets :

- Apprendre de nouvelles choses en utilisant une autre interface graphique, comme pygame par exemple.
- En créant une bande son pour un jeu.
- En créant une interface graphique à des jeux que j'ai déjà créés il y a quelques temps, comme le jeu de mastermind.

Ce jeu m'a donc permis d'avoir une première expérience sur un projet de cette envergure, et c'est justement ce dont j'avais besoin pour progresser.

## **Sitographie/Logithèque :**

### Sitographie :

Cours d'OpenClassrooms :

<https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en->

[python](#)

Forum d'OpenClassrooms :

<https://openclassrooms.com/forum/>

Forum de developpez.net :

<https://www.developpez.net/forums/>

Documentation python :

<https://docs.python.org/fr/3/>

Différents sites ou blogs :

[http://fsincere.free.fr/isn/python/cours\\_python\\_tkinter.php](http://fsincere.free.fr/isn/python/cours_python_tkinter.php)

<https://perso.esiee.fr/~delemarn/Labyrinth/labyrinthes.html>

<https://www.softfluent.fr/blog/generation-et-resolution-de-labyrinthe/>

[https://python.jpvweb.com/mesrecettespython/doku.php?id=cx\\_freeze](https://python.jpvweb.com/mesrecettespython/doku.php?id=cx_freeze)

### Logithèque :

Notepad++ :

<https://notepad-plus-plus.org/downloads/>

Gimp :

<https://www.gimp.org/downloads/>

yEd Graph Editor :

<https://www.yworks.com/products/yed>

Python :

<https://www.python.org/>

cx\_Freeze :

Ce module s'installe en utilisant la console PowerShell directement dans le dossier de l'interpréteur python avec la commande :  
python -m pip install cx\_Freeze