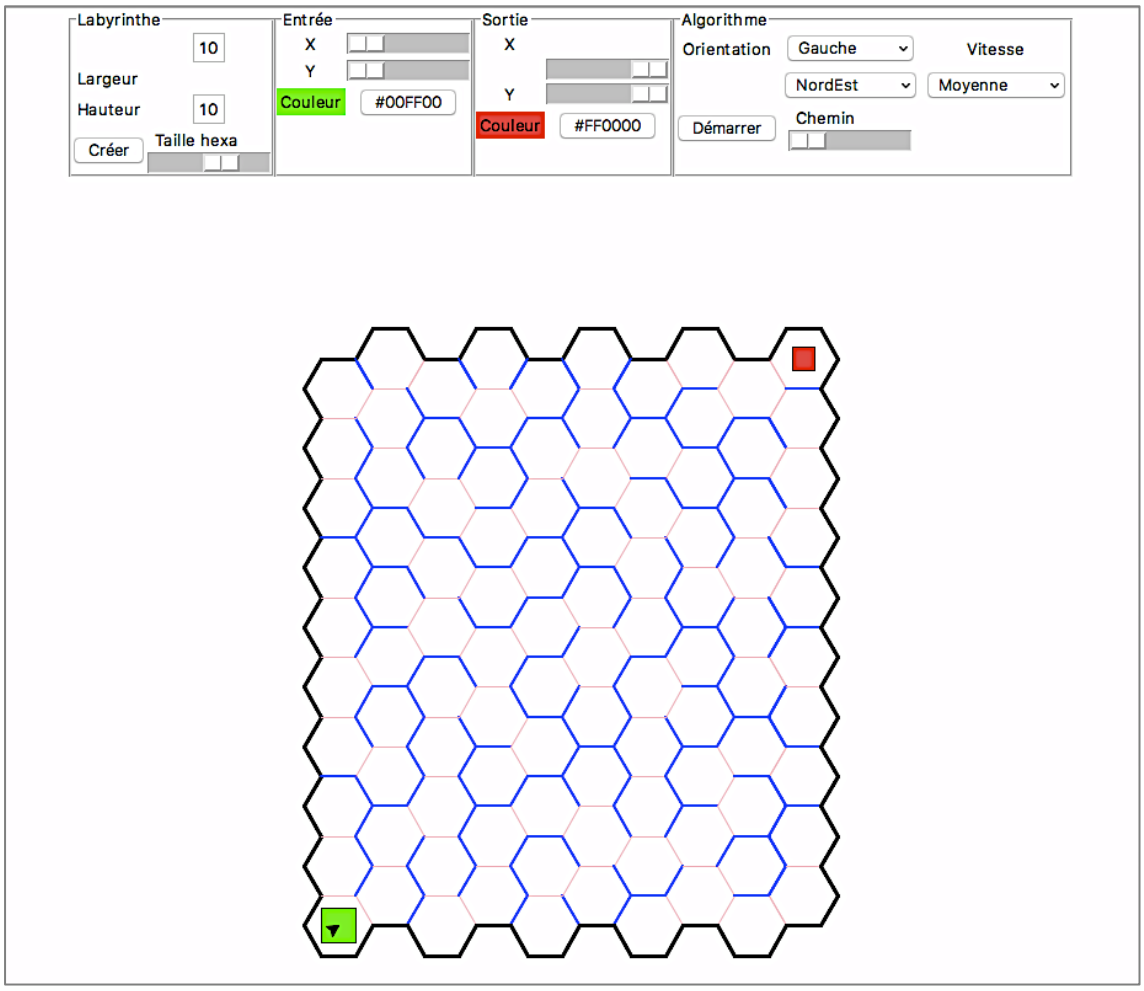
TERMINALE S ISN
Lucas Schneider
Maxence Decourtis
Maé de Montalier

PROJET ISN : LABYROBOT



I – PRESENTATION DU PROJET

Notre équipe LabyRobot s'est lancée dans la réalisation d'un projet en apparence basique mais qui en réalité demande de nombreuses connaissances. Nous sommes fiers de vous présenter notre projet final : un générateur automatique de labyrinthe parfait composé d'îlots avec un robot capable de trouver à chaque fois la sortie.

Pourquoi ce projet ?

Tout d'abord, nous avons réfléchi ensemble à propos de notre futur projet. Assez rapidement, l'idée de la programmation d'un labyrinthe généré aléatoirement et automatiquement avec un robot capable de la résoudre nous est apparue comme la meilleure idée. Il nous a fallu alors peu de temps pour déterminer ce projet et ainsi, nous lancer dans sa réalisation.

Nous avons choisi ce projet pour de multiples raisons. Premièrement, étant tous les trois des étudiants en Terminale Scientifique, les algorithmes sont des suites finis d'instructions avec lesquels nous avons déjà travaillé que ce soit en mathématique ou hors cadre scolaire. Il se trouve que la programmation du projet met en jeu plusieurs algorithmes. Connaissant alors déjà un peu le fonctionnement de ces derniers, nous avons désiré en savoir plus pour pouvoir les mettre en pratique dans notre projet. La curiosité et l'excitation d'apprendre de nouvelles choses nous ont alors poussé à nous lancer dans ce travail.

Dans un second temps, nous avions auparavant lu sur internet des articles mentionnant ce type de projet. L'idée de base nous semblait intéressante mais nous avons vraiment choisi de créer ce programme car nous voulions apporter quelque chose de plus. Nous avons alors réalisé un générateur de labyrinthe qui comporte des hexagones et non des simples cases carrées. Cependant, lors des premières recherches sur le code nécessaire à la création de ce labyrinthe, nous nous sommes rendus compte que le programme était plus dur que dans un labyrinthe classique. Mais attiré par la difficulté et l'envie de se surpasser, nous nous sommes acharnés pour que notre idée puisse devenir réalité.

Dans un dernier temps, ce projet nous parut très intéressant à réaliser car il allait nous octroyer de nombreuses connaissances de programmation qui pourront être utilisées à nouveau dans notre future carrière professionnelle. La quantité de travail que nécessitait ce code ne nous a pas effrayé car nous étions très déterminés et le résultat final en valait le coup. C'était alors pour nous un vrai apport autant personnel que professionnel car nous savions que le projet n'allait pas être simple et qu'ainsi, la cohésion au sein de notre groupe ne pouvait être que renforcé. L'aspect répartition des tâches et travail de groupe nous a beaucoup plu et à travers ce travail, nous savions qu'ils seraient essentiels.

Ainsi, les enjeux du projet étaient multiples :

- Trouver l'algorithme de résolution le plus simple et le plus rapide pour le robot.
- Comprendre et utiliser de nouvelles fonctions de programmation.
- Obtenir un "jeu" facile à prendre en main par l'utilisateur au travers d'une interface clair.
- Générer un labyrinthe aléatoirement à chaque fois
- Se répartir équitablement les tâches et travailler ensemble

1. Positionnement du projet par rapport à des solutions existentielles

Notre projet étant un labyrinthe parfait hexagonal automatiquement généré avec un avec un algorithme de résolution, nous avons recherché si des projets tels que le nôtre avait déjà été réalisé auparavant. Après quelques recherches nous avons surtout trouvé des labyrinthes carrés parfaits générés automatiquement, en effet il existe de nombreux algorithmes permettant la génération de labyrinthes parfaits, et de nombreux algorithmes de résolution. Cependant nous n'avons pas trouvé nous n'avons pas trouvé de tels algorithmes pour des labyrinthes à bases hexagonales.

2. Cahier des charges de l'équipe

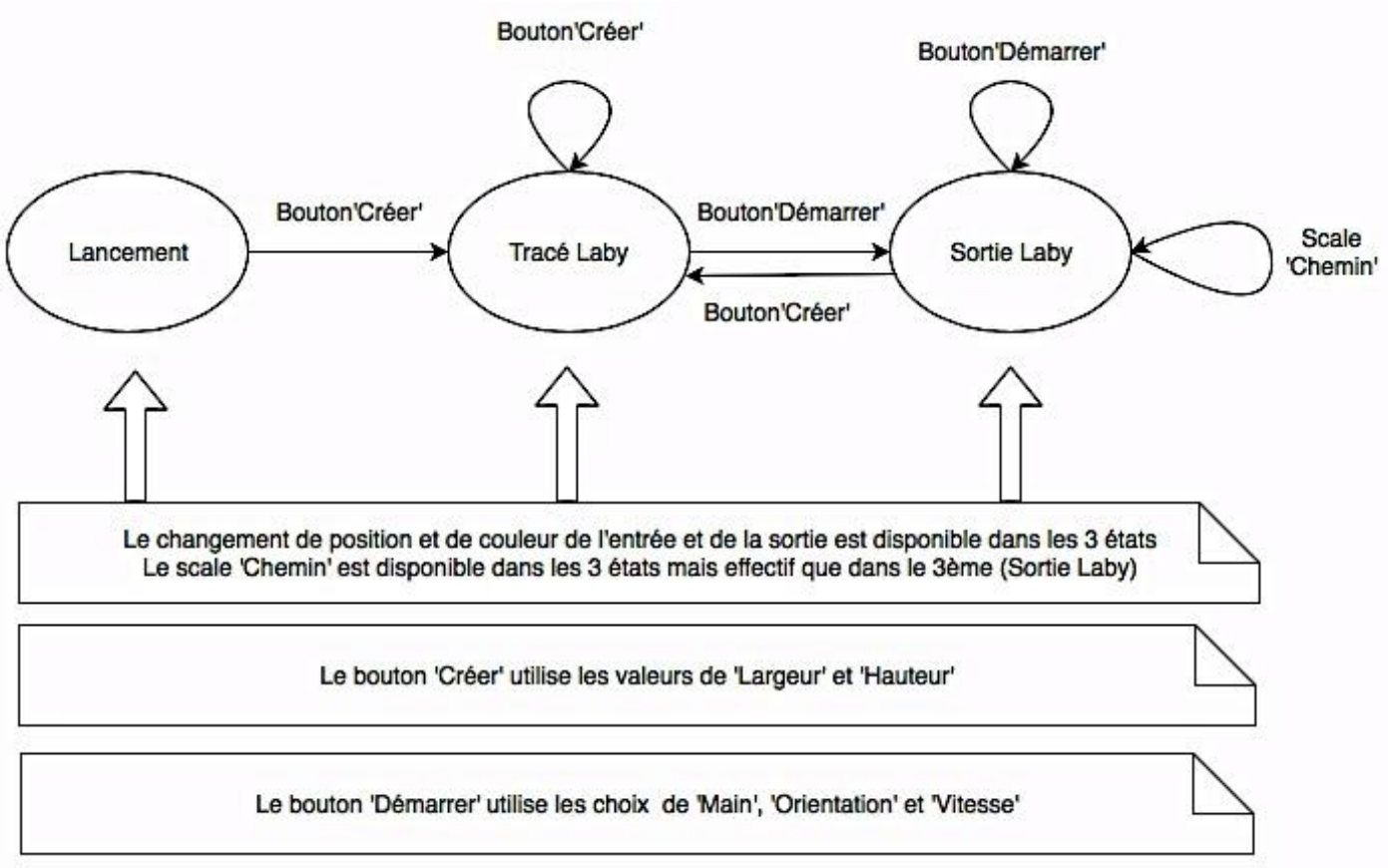
- génération automatique de labyrinthe / parfait
- cases hexagonales
- taille configurable
- définition de la position entrée et sortie
- interface utilisateur conviviale (graphique)
- pouvoir rejouer le tracé
- visualisation du déplacement (tracé pas pratique ca marche avec les numéros)
- algorithme de suivi de mur (deux options main gauche et main droite)

3. Moyens mis en œuvre (langage / matériel)

Notre programme nécessite l'utilisation de Turtle. En effet, ce module graphique permet de, comme son nom l'indique, déplacer une tortue sur un écran. Ce dernier fonctionne avec le langage de programmation Python. On a alors pu utiliser certaines de nos connaissances sur la langage pour les appliquer dans le code. De plus, nous avons eu recours à TKinter pour la création de l'interface du labyrinthe. Encore une fois, c'est le langage Python qui est utilisé car c'est un module de base intégré dans Python. Ainsi, la totalité de notre code repose sur le langage Python.

4. Structure globale du projet

Il y a 3 états dans lesquels l'utilisateur peut réaliser des actions différentes : l'état 'Lancement' obtenu au lancement du programme, l'état 'Tracé Laby' et l'état 'Sortie Laby' :



5. Tableau présentant les tâches et les avancées de notre équipe

	AVANCÉES et TÂCHES		
	LUCAS	MAXENCE	MAE
12/01/18	Choix sujet : labyrinthe parfait généré automatiquement avec un robot le qui le résout par lui même. Faire des recherches sur labyrinthes car sujet complexe et ambitieux.		
19/01/18	Idée d'un labyrinthe avec cellules en diagonales pour originalités. Trouver un programme pour l'interface graphique.		

<table>
<tr><td></td><td colspan="3"></td></tr>
<tr><td>26/01/18</td><td colspan="3">Il faut s'occuper de :
- L'interface graphique
- Comment générer un labyrinthe ?
- Quels outils informatiques utiliser ?
- Modélisation du robot (affichage son chemin)
- Quelle case d'entrée ? Quelle case de sortie ?
- Algorithme du robot → aléatoire ou plusieurs mode de sortie

Test pour algorithme du robot : peut-il sortir en se collant au coté gauche ou au côté droit ? → vrai à gauche et vrai à droite</td></tr>
<tr><td rowspan="2">02/02/18</td><td colspan="3">Test des labyrinthes parfait et imparfait.
Mise en commun des premières lignes de programmation et des résultats des tâches → Utilisation du logiciel Turtle pour la génération du labyrinthe, et de TK inter pour l'interface graphique.
Répartition globale des tâches :</td></tr>
<tr><td>Trouver comment générer le labyrinthe avec Turtle.</td><td>Trouver les ressources nécessaires pour la réalisation du projet.</td><td>Comprendre les techniques de TK inter pour l'interface.</td></tr>
<tr><td>09/02/18</td><td colspan="3">Cours annulé, Skype pour parler des avancements et des tâches à faire.</td></tr>
<tr><td>16/02/18</td><td>Algorithme du labyrinthe terminé par Lucas.
Et mise en forme de programme.</td><td>Avancer sur le déplacement du robot.
Savoir ce qu'il y manque dans le programme.</td><td>Début de l'interface graphique fait.</td></tr>
<tr><td rowspan="2">Pendant les vacances de février</td><td>Codage de la partie génération du labyrinthe en hexagone avec l'initialisation de Turtle</td><td colspan="2">Trouver les idées et faire des recherches pour le code de la génération du labyrinthe → idées ensemble, Lucas lui qui codait le plus + petite aide de son père, ingénieur en informatique.</td></tr>
<tr><td colspan="3">On s'est vus mardi, vendredi et samedi de la première semaine, Maé en Skype. Traçage de la grille d'hexagone centrée sur la fenêtre Turtle.
Réglages de quelques problèmes lorsque le labyrinthe contenait un nombre pair ou impair d'hexagones en largeur au niveau des murs extérieurs.
Aide avec fonction help.turtle, → beaucoup de nouvelles idées.</td></tr>
</table>

<table>
<tr><td>09/03/18</td><td>Commencer l'algorithme du robot avec l'algorithme de Pledge : sortir du labyrinthe en tournant tout le temps à gauche ou tout le temps à droite.</td><td>Faire les finitions du codage du programme du labyrinthe.</td><td>Continuer de coder l'interface graphique avec Lucas.</td></tr>
<tr><td>16/03/18</td><td colspan="3">Début programmation de l'algorithme de Pledge.
Continuer programmation génération automatique du labyrinthe.
Continuer à avancer maintenant tous ensemble sur l'interface graphique.</td></tr>
<tr><td rowspan="2">23/03/18</td><td colspan="3">Pas cours d'ISN, réunion chez Maxence.
Abandon de l'algorithme de Pledge trop compliqué, autre algorithme car un seul type de labyrinthe : le labyrinthe parfait.</td></tr>
<tr><td colspan="2">Utilisation et programmation d'un nouvel algorithme « suivre le mur », le robot va seulement suivre soit le mur de gauche ou soit le mur de droite plus simple à coder.
Nouvelle idée → slider : suivre résolution laby</td><td>Avancée au niveau de l'interface graphique (la doc de TK inter sur internet très pratique)</td></tr>
<tr><td>30-31/03/18</td><td colspan="3">Pas cours d'ISN, réunion chez Lucas pendant le week-end pour réalisation de l'interface graphique. Petite aide du père de Lucas l'organiser et comprendre les fonctionnalités de chaque fonction.</td></tr>
<tr><td rowspan="2">06/04/18</td><td colspan="3">Mise au point avec Mr Jules, → il faut avancer vite.
Idée d'un fichier audio.</td></tr>
<tr><td>Réglage d'un défaut.
Avancer codage algorithme résolution.
Interface graphique.</td><td colspan="2">Grande avancée dans l'interface graphique de la page du labyrinthe, nouvelles idées et nouvelles mises en formes.</td></tr>
<tr><td>13/04/18</td><td colspan="3">Mise au point des avancées de la semaine.
Objectif : terminer code pour rentrée pour finaliser petits détails.</td></tr>
<tr><td rowspan="2">Pendant les vacances de Pâques</td><td colspan="3">Finir l'interface et l'algorithme de programmation du robot.</td></tr>
<tr><td>Programmer l'algorithme de gauche et droite.
Test vérification de l'algorithme « suivi du mur.
Faille : robot bloqué en main gauche.</td><td colspan="2">Programmer slider pour chemin robot (aide de Lucas) + option pour l'utilisateur : dessiner parois labyrinthe avec couleur.
Programmer boutons slider et boutons main droite et main gauche.</td></tr>
</table>

04/05/18	Retour des vacances → quasiment terminé juste détails. Accords sur modifs à faire pour finaliser. Répartition tâches pour rédaction dossier. Changer nom certaines fonctions car pas explicites. Ajout d'une fonction pour changer vitesse tracé chemin du robot.	

II – Partie personnelle

1. Réalisation personnelle

Mon travail au sein du groupe a débuté lorsqu'il a fallu se pencher sur les outils nécessaires à la réalisation de notre projet. En effet, il fallait que je détermine les différentes ressources que nous allions utiliser pour pouvoir mener a bien notre projet. Le premier outil a utilisé impérativement fut le logiciel Turtle. Ce dernier est essentiel pour la génération du labyrinthe. Turtle est un module graphique qui appartient au langage de programmation Python. On allait alors pouvoir appliquer les connaissances sur le langage Python que nous avions développé les six premiers mois.
Ainsi, je me suis penché sur le principe de la programmation du labyrinthe. J'ai notamment codé des parties du programme qui permettent la création des cellules hexagonales qui composent le labyrinthe.

```
def traceBasHexa(a_x, a_y, a_type) :
    # On lève le stylo
    turtle.up()
    # On va au cinquième segment (dans le sens trigo)
    turtle.goto(a_x - DemiTailleHexa / 2, a_y - DemiTailleHexa * RACINE3_2)
    # On abaisse le stylo
    turtle.down()
    # On trace le cinquième segment (dans le sens trigo) : mur 4
    turtle.pensize(EPAISSEUR_LABY[a_type])
    turtle.pencolor(COULEUR_LABY[a_type])
    turtle.goto(a_x + DemiTailleHexa / 2, a_y - DemiTailleHexa * RACINE3_2)
    return
```

Ce groupement de fonctions permet de tracer le bas d'un hexagone soit le mur 4. Ainsi, après avoir compris la programmation pour le codage de reste de l'hexagone, j'ai pu moi-même coder pour le mur du bas. Le fonctionnement de ces fonctions est relativement simple. Dans un premier temps, le programme va définir les abscisses et ordonnées du centre de l'hexagone grâce à la fonction « def traceBasHexa » suivi des coordonnées. Ici, « a_x » correspond aux coordonnées des abscisses et « a_y » aux coordonnées des ordonnées.
« a_type » permet de déterminer le type de mur de chaque côté de l'hexagone ; les murs extérieurs ne peuvent être effacés lors de la génération du labyrinthe, les murs intérieurs ou parois peuvent être supprimé à cette même étape. Ainsi, après avoir déterminer ces paramètres, le programme peut commencer à tracer le dernier mur de l'hexagone. Pour cela, il va dans un premier temps « lever son stylo », c'est-à-dire, arrêter de tracer le cinquième segment qui est mitoyen au mur du bas. C'est la fonction « turtle.up » qui est à l'origine de ce mécanisme. Pour donner suite à cela, grâce à la fonction « turtle.goto » qui permet d'aller aux coordonnées x et y données, on indique les coordonnées du cinquième segment. On se déplace à l'aide du cercle trigonométrique d'où l'apparition de coordonnées du style « RACINE3_2 ». Les coordonnées du cinquième segment définies, la fonction « turtle.down » va abaisser le stylo de sorte à commencer à tracer le dernier mur. Il faut désormais indiquer les coordonnées de l'extrémité du segment pour que le programme sache jusqu'où tracer le mur. On réutilise alors la fonction « turtle.goto » avec des coordonnées différentes, celles de l'autre extrémité

du segment. Quand à la typographie du segment, elle est modifiable grâce aux fonction « turtle.pensize » et « turtle.pencolor » qui modifie respectivement l'épaisseur et la couleur du segment. Il est possible de les changer en modifiant la variable définie à l'intérieur de la fonction. Nous voulions rendre notre labyrinthe le plus agréable possible dans son utilisation et sa compréhension. Pour cela, nous avons mis en place différentes options qui permettent de modifier des paramètres et configurations de ce dernier. Parmi ces possibilités, j'ai travaillé sur le programme qui influe sur la taille des hexagones. L'utilisateur peut alors modifier à sa guise la taille des hexagones et donc tout le labyrinthe. Le programme qui permet cala est le suivant.

```
def tailleCmd(a_val) :
    global DemiTailleHexa
    DemiTailleHexa = int(a_val)
    # Recalcul et retracé du labyrinthe s'il existe
    if InitTurtle :
        initCentrage()
        traceGrilleHexa()
    return
```

Dans un premier temps, grâce à la fonction « def tailleCmd(a_val) » le programme va définir la taille initiale de l'hexagone avec la valeur « a_val » qui détermine cette dernière. Cela est rendu possible après avec la fonction « global » qui permet d'avoir la même taille pour tous les hexagones. Ainsi, on initialise la taille des demi-hexagones avec la valeur d'un hexagone en entier défini auparavant. Cette partie du code permet à tous les hexagones d'avoir la même taille initiale lors de la génération du labyrinthe. Par suite de cela, lorsque l'utilisateur va changer la taille des hexagones, une autre partie du programme rentre en jeu. Pour cela, on instaure une condition. Cette condition est traduite par la fonction « if ». Elle est suivie de la variable « InitTurtle » qui permet de lancer la condition lorsqu'il y a changement de la taille es hexagones par l'utilisateur. Lorsque la condition est déclenchée, d'autres fonctions interviennent alors. « initCentrage() » permet d'initialiser un nouveau centre pour chaque hexagone du labyrinthe. En effet, leur taille étant modifiant, ils ne sont plus centrés de la même façon. Après avoir définit le nouveau centre respectif de chaque hexagone, la fonction « traceGrillHexa() » permet de tracer les nouveaux hexagones avec leur nouvelle taille.
Je me suis penché sur le début de la programmation du code qui permet au robot de résoudre le labyrinthe. J'ai alors travaillé sur le positionnement initial du robot avant qu'il ne débute sa résolution.

```
def demarrerCmd() :
    global Orientation
    # On retrace le labyrinthe (pour effacer les chemins turtle précédents s'il y en a)
    traceGrilleHexa()
    # Initialisation pour calcul du temps de l'algorithme
    tps = time.time()
    # Orientation de la 'turtle'
    orientation()
    # On lève le stylo
    turtle.penup()
    # On va au centre de la cellule
    (x, y) = centreHexa(Entree[0], Entree[1])
    turtle.goto(x, y)
    # Affichage de la 'turtle'
    turtle.showturtle()
    # On trace !
    turtle.pendown()
    # Couleur du chemin
    turtle.pencolor(COULEUR_LABY[CHEMIN])
    # Epaisseur du chemin
    turtle.pensize(EPAISSEUR_LABY[CHEMIN])
    # Position de départ (attention pos = Entree ne marche pas)
    pos = [Entree[0], Entree[1]]
    # Nombre de cellules visitées
    cellules = 1
    # Nombre maximum de cellules visitables
```

Le programme est lancé par la fonction « def demarrerCmd() ». Dans un premier temps, il faut que le labyrinthe soit réinitialisé s'il il y'a eu déjà un chemin Turtle de fait. C'est la fonction « traceGrilleHexa() » qui est à l'origine de la génération d'un labyrinthe vierge de chemin Turtle. Dès que le robot commence à résoudre le labyrinthe, un timer se lance pour voir le temps que mets le robot, en fonction de la vitesse que l'on a choisi, pour atteindre la sortie du labyrinthe. Le temps est initialisé et lancé grâce à « tps = time.time() » qui début le chronomètre. Il faut que la Turtle est une orientation de base, qu'elle pointe vers une direction précise qui sera toujours la même à chaque lancement de l'algorithme de résolution. La fonction basique « orientation() » joue ce rôle. Afin de pouvoir tracer le chemin du robot et qu'il soit visible, il faut que l'algorithme comporte des fonctions qui permettent de faire apparaître le chemin que prends le robot. Au début, à l'instant 0, il n'y a pas de Turtle au centre de la cellule. Il faut alors une fonction qui permet au programme de se rendre au point de départ sans tracer de chemin. Pour cela, on utilise la fonction « turtle.penup() » qui lève le stylo. Ainsi, par la suite, grâce à la fonction qui va déterminer la position où va se rendre le stylo, l'algorithme ne va pas faire apparaitre le chemin du stylo à cet instant-là. La fonction qui permet au traceur de se rendre au point de départ du robot, soit le centre de l'hexagone, s'appuie sur la fonction « turtle.goto » qui se traduit par « va à + les coordonnées ». Ainsi, les coordonnées du centre de l'hexagone sont données et la fonction « turtle.goto » va permettre, grâce aux coordonnées (x ; y), au traceur de se rendre au centre de l'hexagone. Jusqu'à cet instant, la Turtle n'était pas visible n'y aucunes des actions précédentes. Il faut désormais faire apparaitre la Turtle ainsi que le stylo qui va tracer le chemin du robot. Les fonctions « turtle.showturtle() » et « turtle.pendown() » vont respectivement afficher la Turtle et le tracé du stylo. Des fonctions comme « turtle.pencolor() » ou « turtle.pensize » permettent de modifier le style calligraphique du traceur.

Je me suis principalement penché sur des parties du code qui nécessitaient beaucoup de fonction « Turtle ». Cependant, j'ai aussi travaillé sur la partie interface de notre projet et plus particulièrement sur les fonctions qui font apparaitre des boutons de modifications. En effet, dans l'interface de notre projet, plusieurs boutons permettent de changer les paramètres initiaux du générateur.

2. Intégration et validation

Afin de vérifier si l'algorithme que nous allions utiliser pour la résolution du labyrinthe par le robot, j'ai réalisé des tests manuscrits. En effet, le principe de cet algorithme étant de toujours suivre le mur de droite (ou de gauche), j'ai alors dans un premier temps, créer un labyrinthe classique (sans alvéoles) pour comprendre et voir le principe de fonctionnement. Le labyrinthe a été dessiné de façon qu'il y ait un unique chemin qui mène à la sortie (labyrinthe parfait). À la suite de cela, j'ai testé l'algorithme qui consiste à suivre le mur de droite. J'ai alors constaté que ce dernier m'avait bien mené à la sortie. J'ai alors réalisé l'expérience une deuxième fois sur le même labyrinthe sauf que cette fois-ci, je m'attelais à suivre uniquement les murs de gauche. J'ai alors obtenu le même résultat ; je suis parvenu jusqu'à la sortie. J'ai alors partagé mes résultats avec mes deux camarades. Nous avons alors validé cet algorithme en tant que futur algorithme de résolution de notre projet. Cependant, notre labyrinthe n'est pas classique car à la place des cases carrées classiques qui composent un labyrinthe classique, le notre est composé d'alvéoles hexagonales. Je me suis alors penché sur la question pour que l'on puisse adapter cet algorithme à notre labyrinthe. Ainsi, pour notre type de case, le programme de résolution, j'ai déduit qu'il fallait qu'il teste chaque option dans un certain sens, soit le sens trigonométrique soit l'inverse. Cela revient au principe de l'algorithme « suivi du mur de gauche ou droite » car le robot vérifie s'il y a un mur interne pour pouvoir avancer dans un sens donné. Pour cela, il fallut diviser les cellules en deux pour pouvoir mettre en place ce système de sens. À la suite de nos avancées personnelles, nous sommes parvenus a créé cet algorithme qui permet au robot de tester les options dans un sens. S'il ne trouve pas de sortie alors qu'il a testé toutes les possibilités d'un côté, nous avons décidé qu'il devrait alors recommencer cette vérification mais sur l'autre côté. Ainsi, l'algorithme permet au robot d'avancer qu'importe l'emplacement de la sortie. Mes recherches personnelles et le travail de mes camarades combinés ont conduit à l'aboutissement de cet algorithme.
Je me suis occupé d'une partie de l'interface présente au-dessus du labyrinthe. En effet, je me suis attelé à comprendre comment on pouvait faire apparaitre des boutons qui allaient avoir un impact sur l'apparence du labyrinthe. Ces boutons permettent de changer la taille des hexagones qui composent le labyrinthe, la couleur du carré entrée et sortie et encore d'autres. J'ai aussi travaillé en collectif avec Maé pour les boutons qui changent la taille des hexagones. C'était très intéressant car on a réussi à avancer chacun de notre côté pour après mettre en commun et échanger sur ce qu'on avait trouvé. Ainsi, on a pu avancer beaucoup plus vite tout en développant notre esprit d'équipe. De plus, lors des difficultés que nous avons rencontrées, le fait de travailler à deux a permis de comprendre plus facilement les erreurs que nous avions commises.

3. Bilan et perspectives

Le projet en lui-même m'a apporté beaucoup que ce soit au niveau des connaissances mais aussi au niveau de ma personnalité. En effet, au début du projet, je ne connaissais rien de toutes les fonctions que j'allais devoir utiliser et mettre en pratique. Il a fallu alors que je découvre ces fonctions et que j'apprenne à les utiliser pour pouvoir les insérer dans mes programmes. Cependant, la programmation et les fonctions, du fait de leur nombre important et leur complexité, ont fait apparaitre des difficultés. J'ai alors découvert une forte détermination en moi qui m'a poussé à travailler plus pour pouvoir comprendre totalement les codes de programmation. J'ai alors appréhendé les choses sous un autre aspect ce qui m'a permis de contourner les obstacles. De plus, grâce à ce projet j'ai découvert de nombreuses choses sur l'informatique en générale, la programmation et comment on peut l'utiliser pour faire des programmes entiers. J'ai été fasciné de voir le résultat final de notre projet car je ne m'imaginais pas qu'un simple jeu comme ça nécessitait une programmation aussi importante. J'ai pris conscience de l'importance de maîtriser des langages

informatiques car dans notre société qui est de plus en plus digital, ils ont une désormais une place fondamentale.
Le travail collectif que nous avons réalisé tout au long du projet fut très enrichissant pour chacun de nous. Nous avions déjà l'habitude de travailler en groupe depuis le collège car La Source met l'accent sur ce type de travail. Cependant, nous n'avions jamais travaillé tous les trois ensemble. Nous nous sommes alors découvert sous un nouvel aspect. J'ai personnellement beaucoup apprécié le travail que nous avons mené, il y avait une vraie cohésion au sein du groupe ce qui m'a motivé à m'investir davantage pour pouvoir tirer le groupe vers le haut. Je n'avais jamais travaillé en groupe sur ce genre de projet car l'informatique est une science peu enseignée dans les écoles. Il a alors fallu appréhender le travail sous un angle différent car le projet demandait une rigueur et un investissement qui diffèrent des autres matières enseignées au lycée. Ce groupe m'a alors appris de nouvelles façons de faire et m'a permis de pouvoir prendre des initiatives que je ne n'aurais jamais prise dans d'autres matières. J'ai pu alors me développer personnellement et prendre en confiance sur moi-même.
Malgré le résultat satisfaisant que nous avons obtenu, plusieurs modifications seraient envisagables afin d'améliorer quelques aspects de ce dernier. En effet, pour améliorer ce dernier, je proposerai d'ajouter une fonction qui permet de pouvoir arrêter la résolution du robot à n'importe quel temps. Cela permettrait de pouvoir stopper l'algorithme de résolution du robot et de pouvoir observer ces mouvements antérieurs et anticiper les suivants. En plus de cette fonction, j'ajouterai une autre fonction qui permet lorsque le robot est arrêté de pouvoir, à l'aide d'un simple clic, ajouté un mur intérieur partout dans le labyrinthe. Grâce à cet ajout, on pourrait observer la capacité de résolution du robot et voir comment ce dernier réagit dans un labyrinthe qui n'ait plus parfait.
De plus, je pense que pour rendre notre projet encore plus intéressant, nous pourrions l'améliorer en proposant deux modes de générations de labyrinthe : un mode génération d'un labyrinthe parfait (comme celui actuel) et un autre mode génération d'un labyrinthe imparfait. Cela impliquerait de coder un autre algorithme, l'algorithme de Pledge. Ainsi, on pourrait observer les capacités de résolutions du robot dans deux types de labyrinthe différents et deux algorithmes différents. Un labyrinthe imparfait ne possédant pas un unique chemin qui permet de sortir, l'algorithme de Pledge permettrait au robot de résoudre quand même ce dernier et ainsi de sortir.