Zaizafoun Sami 21600538
Mori Baptiste 21602052

Projet annuel

GitHub: https://github.com/sami-zaizafoun/projet_annuel

Génération aléatoire d'un labyrinthe PAC-man sur un dôme

M1-Informatique (DOP)
2019-2020

Sommaire

I) Introduction:

Le projet à réaliser nous demande en première partie de générer un labyrinthe rond basé sur le jeu classique PAC-man en respectant des contraintes qui créent des énigmes pour les joueurs. En deuxièmes partie, nous devons déformer l'image de labyrinthe afin de pouvoir l'afficher correctement sur un dôme. Ce projet a attiré notre attention car nous avons été curieux de découvrir comment manipuler la projection d'une image, et après avoir fait de la recherche, nous avons remarqué qu'il existe peu de données sur la génération d'un labyrinthe rond, nous avons choisis d'accepter le défi.

II) Déroulement du projet:

Contrairement à d'autres projets annuels, nous avons un concepteur de systèmes électroniques qui joue le rôle d'un client. Donc nous avons commencé par prendre rendez-vous avec lui afin de mettre en place notre cahier de charge. Durant ce rendez-vous, nous avons mis en place la priorité des taches à réaliser et nous avons discuté des améliorations possibles. Une semaine plus tard, nous avons réalisé un briefing avec notre chargé de projet et les autres élèves sous sa direction. Durant ce briefing, notre chargé de projet nous a expliqué ce qu'il attend de nous pour le rapport et la soutenance.

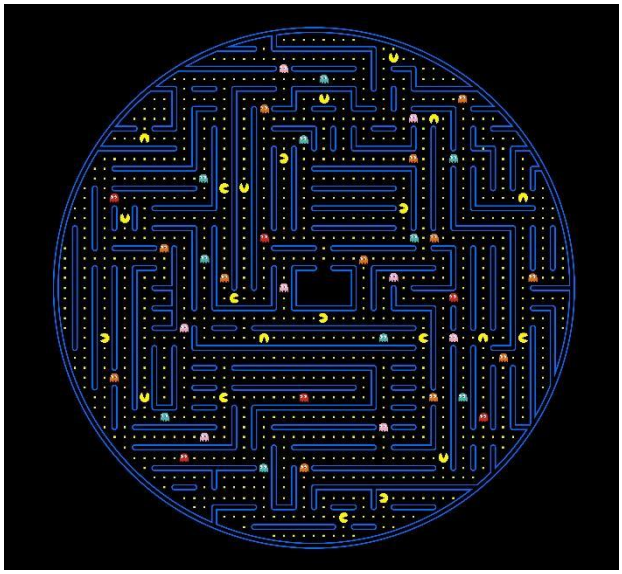
Un deuxième rendez-vous a été pris avec notre client afin de lui monter l'avancement du projet. A ce moment-là, nous avons réalisé une première version du labyrinthe qui sera affiché dans une autre partie du rapport et une déformation incorrecte pour la projection.

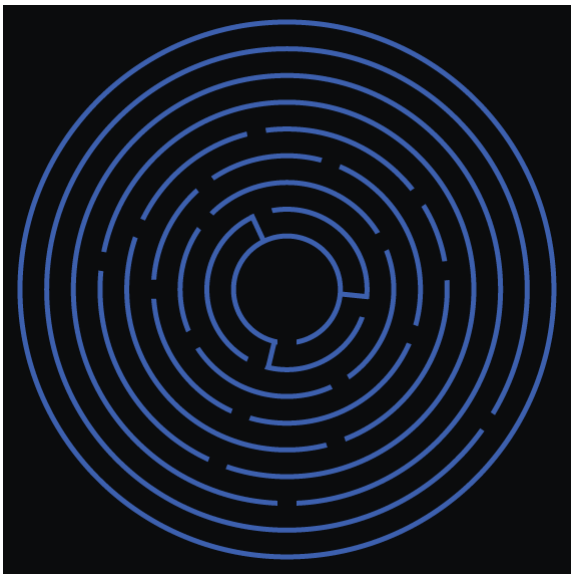
Le projet s'est mis en pause pendant la période des examens mais à la fin de cette période, nous avons repris contact et l'initiative de prendre rendez-vous afin de mettre à point notre chargé de projet et le client.

Malheureusement, à cause du confinement, nous n'avons pas pu faire la dernière démonstration de la projection et donc nous ne sommes pas sûrs de l'affichage sur le dôme.

III) Architecture du projet:

Comme dit précédemment, notre client nous a montré ce qu'il attendait de nous pour le projet. Nous devons appliquer le contenu de la figure 1 dans la figure 2.





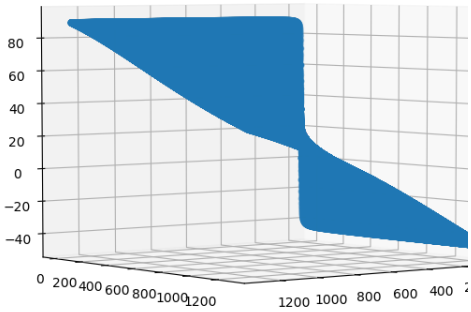
Nous avons décidé de nous diviser les taches de travail en deux. La première tâche était de faire la déformation de l'image tandis que la deuxième était la génération du labyrinthe et appliquer la déformation dessus.

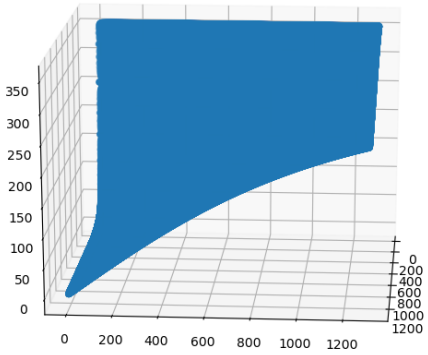
a) Effet Fisheye:

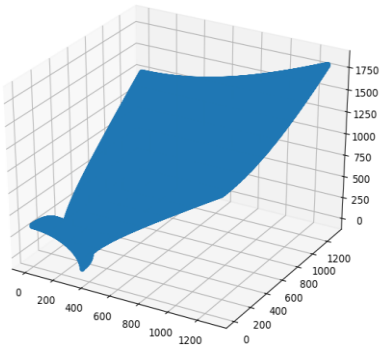
Nous avons commencé par la génération d'un dôme en 3 dimensions grâce à la coordonnée polaire pour créer une simulation du dôme en lui-même, et ainsi faire la déformation en positionnant chaque pixel de l'image sur le dôme. Nous avons par la suite

remarqué que cette façon de faire n'était pas la bonne puisque ça apporté plus de problème que de solution.

Voici quelques exemples de l'échec sur lequel nous sommes tombes en essayant cette méthode :







En effet étant donné que de cette manière nous devions sauvegarder l'image en 3 dimensions tout en l'affichant avec un vidéo projecteur cela aurait été très compliqué. Nous avons donc fait de plus amples recherches orientées sur les effets des lentilles pour arriver sur les effets de Barillets et de Fisheye.

Au regard des de la courbure du dôme et des effets optiques de la lentille, les bords de l'image à projeter doivent être plus rapprochés soit rétrécis en quelque sorte. Le centre quant à lui doit être agrandi (mettre image de dôme pour montrer l'effet optique). L'effet Fisheye s'adapte ainsi très bien à ce type de déformation.

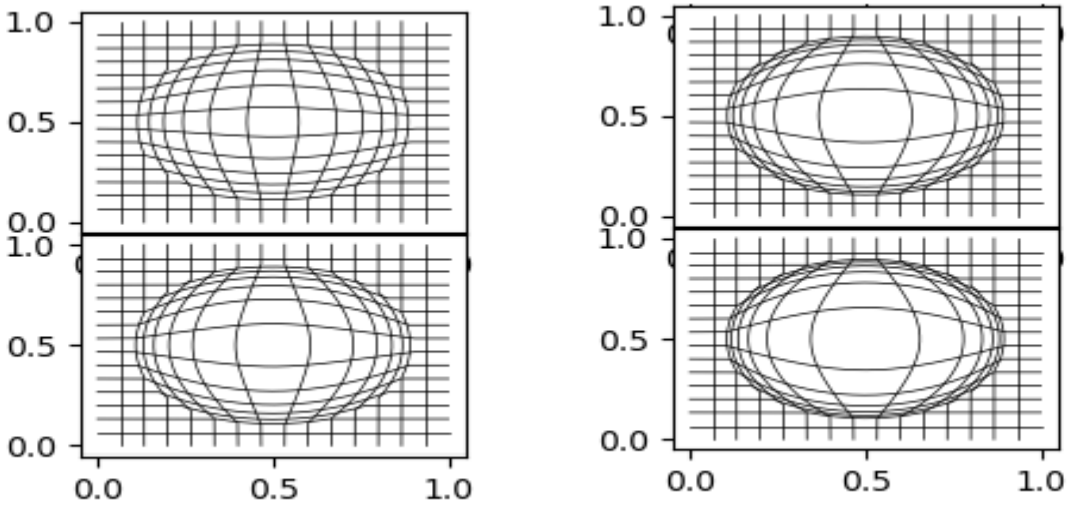
Donc pour réaliser l'effet Fisheye, nous commençons par récupérer le cercle de distorsion, c'est-à-dire tous les points qui sont dans un rayon donné, qui vont être changés de positions. Ensuite, nous calculons le ratio, qui est le pourcentage de déplacement des pixels par rapport au centre du cercle. Lorsque les pixels sont plus proches, le ratio sera moins élevé. A partir du ratio, nous pouvons calculer le taux de distorsions, c'est-à-dire que pour tous les points de ratio inférieurs à 1 nous appliquons la formule de Sarkar & Brown légèrement modifiée pour notre besoin qui donne :

$$P_{fisheye} = \frac{ratio}{démagnification \times (1 - ratio) + 1} \text{ avec } ratio < 1$$

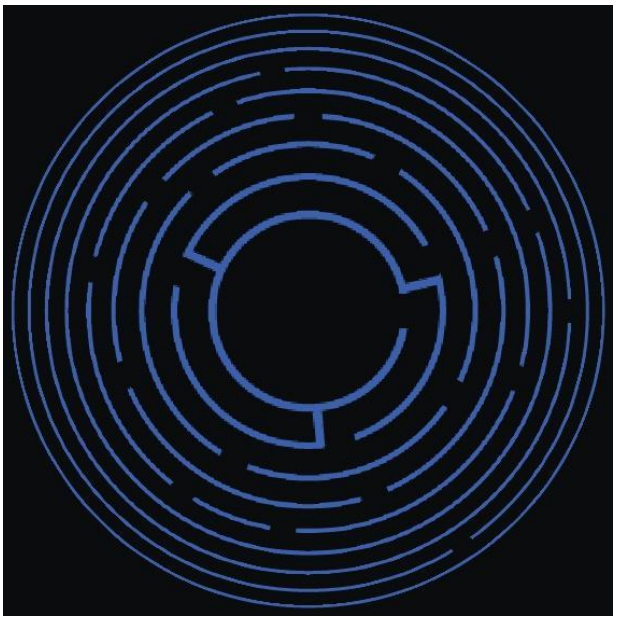
Cette formule sert à appliquer un déplacement vers l'extérieur du cercle pour créer un taux que l'on va utiliser pour recalculer les positions des pixels, plus le ratio de distorsion est élevé plus le taux est faible et inversement étant donné que les pixels des bords ont moins besoin d'être déplacer pour obtenir l'effet Fisheye.

Maintenant que l'on a le taux de distorsion qui correspond à une augmentation de la distance du pixel par rapport au centre de l'image, il est logique de passer par les coordonnées polaires en calculant thêta (l'angle du vecteur) tout en appliquant le taux sur le calcul du rayon. Cette application nous donne tout ce dont on a besoin pour revenir à des coordonnées cartésiennes et remplacer les pixels modifiés dans l'image source.

Nous pouvons par ainsi régler l'effet Fisheye grâce au paramètre de démagnification, nous pouvons voir qu'en augmentant ce paramètre, l'effet Fisheye est accentué et nous pouvons voir que les pixels s'éloignent du centre et s'approche du bord. Exemple :

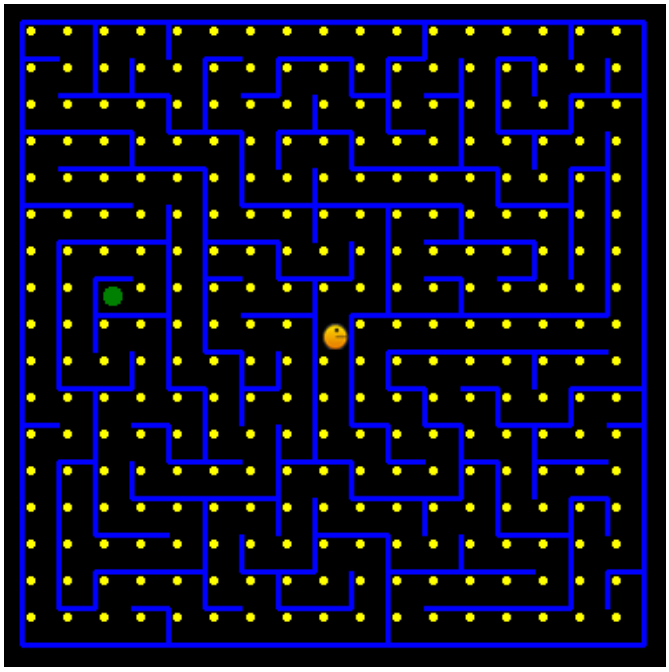


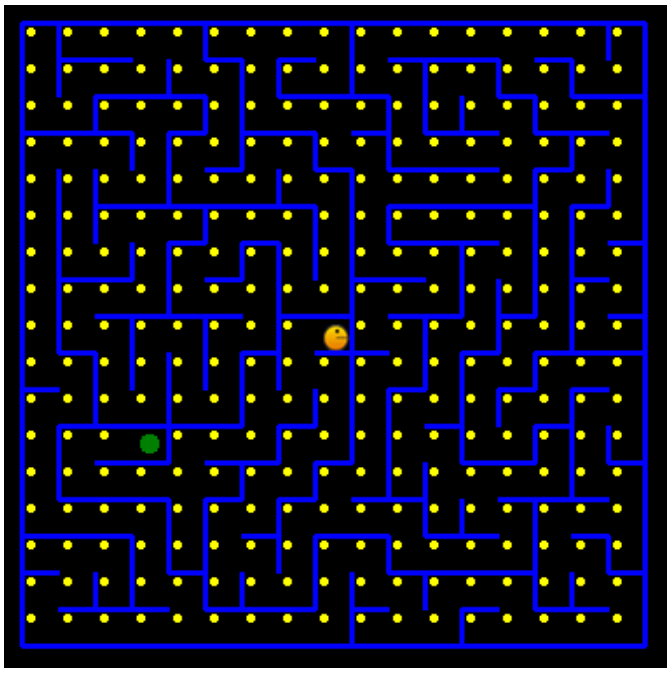
Donc pour conclure l'effet Fisheye, voici une image d'un labyrinthe et le résultat avec le filtre appliqué avec une démagnification de 1 :



b) Génération du labyrinthe:

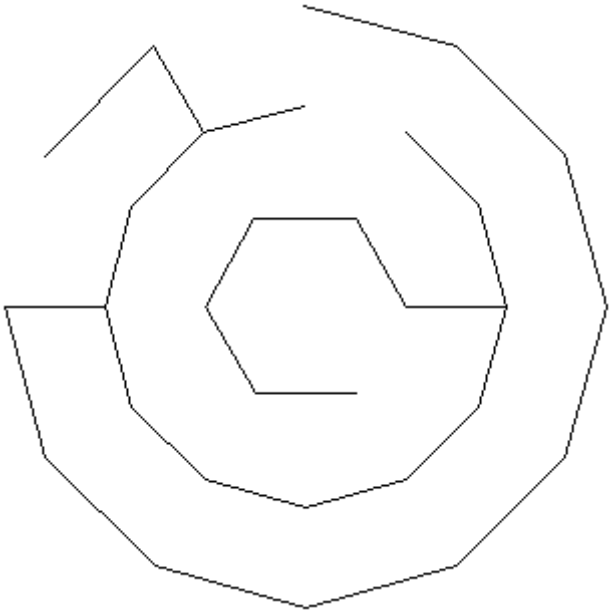
La première idée pour la génération du labyrinthe était de créer un logiciel de labyrinthe aléatoire carré et ensuite d'appliquer l'effet de Barillets sur le résultat. Pour rendre le labyrinthe aléatoire, nous avons utilisé un algorithme qui crée un chemin avec une solution. Nous avons ensuite dessiné des murs aléatoires autour de ce chemin avec un algorithme de backtracking pour ne pas repasser par les mêmes chemins. Voici deux exemples, de générations aléatoires :





En essayant d'appliquer l'effet de Barillets sur ce labyrinthe, nous avons remarqué que nous avions une trop grosse déformation sur le carré du milieu, cela aurait alors agrandi la déformation au milieu mais pas les côtés, ce qui donne un résultat pas très propre et trop compliqué à régler.

Nous avons donc choisi une autre approche qui est de générer un labyrinthe rond. Pour ceci, nous commençons par initialiser chaque cercle concentrique, du plus petit au plus grand en mettant un numéro à chaque croisement de murs. Après cette initialisation, nous initialisons l'ensemble des murs possibles pour créer un graphe non orienté et à chaque arrête de ce graphe (à chaque mur), nous ajoutons un poids que nous affectons de façon aléatoire pour générer un labyrinthe lui aussi aléatoire. L'initialisation faite, nous obtenons un graphe non orienté possédant des poids ce qui nous permet de créer des chemins en sélectionnant les murs que l'on souhaite. C'est-à-dire, nous pouvons par exemple utiliser un algorithme, tel que Kruskal, pour générer des chemins dans ce labyrinthe. Donc, nous obtenons à partir de là, un graphe contenant des murs et des passages ce qui nous permet ensuite de dessiner les arrêtes et par la suite nous permettant de dessiner les murs de manière circulaire. Le résultat est le suivant :



Lors de la génération du labyrinthe des contraintes doivent être imposées en fonction des besoins, c'est pourquoi on peut passer en paramètre une classe permettant de générer

le labyrinthe à partir du graph initialisé. Dans notre cas le labyrinthe nécessite plusieurs contraintes un peu spéciales telles que la génération du chemin qui conduira vers la clé pour résoudre l'énigme et la génération des faux chemins et des fausses clés pour tromper les joueurs. Certaines contraintes ont déjà été réalisées dans la version carrée du labyrinthe mais il faut les adapter à la version circulaire. Cette adaptation risque d'être compliquée car la version circulaire du labyrinthe est construite à partir des arrêtes (murs) alors que la version carrée est construite à partir des cellules.

L'une des solutions pourrait être, à partir du graphique : générer une liste de listes de case que l'on ferait correspondre aux listes des murs pour que quand nous souhaitions passer d'une cellule A à une cellule B, cela revient à dire que nous cassons le mur entre ces deux cellules. Cela aurait pour effet d'enlever ce mur de la liste des murs restant.

Sur le labyrinthe carré, les contraintes qui sont respectées sont :

- La génération du chemin correcte : Un chemin partant d'un point A (le milieu) à un point placé aléatoirement dans le labyrinthe, sur ce chemin nous générons 9 pacmans et une clé, ces pacmans représentent la solution.

- La génération des faux chemins : des chemins avec des pacmans qui dirigent les joueurs vers une fausse clé (nombre de pacmans < 9)

IV) Difficultés rencontré:

En travaillant sur ce projet, nous avons rencontré quelques difficultés. Une des premières difficultés rencontrées était de calculer les coordonnés polaires car il a fallu comprendre, appliquer et adapter les formules pour répondre à nos besoins ce qui n'était pas évident au début car nous n'avions jamais travaillé sur ce genre de

déformation. Ensuite, réaliser l'effet Fisheye a posé un problème car il n'y a quasiment que des documentations sur comment résoudre cet effet et ne pas sur comment le créer. Nous avons plongé profondément dans la recherche pour trouver ce qu'il nous fallait. Un des problèmes que nous avons eu au cours de la programmation de déformation de l'image est que nous ne sommes pas capables de tester la projection car nous n'avions pas accès permanant au matériel de projection. Finalement, comme nous avions commencé par générer un algorithme carré, nous avons eu un peu de difficulté de voir comment générer un labyrinthe rond en utilisant les algorithmes de génération d'un labyrinthe classique.

Conclusion:

En conclusion, l'effet Fisheye est fonctionnel, nous pouvons le calibrer grâce à des paramètres lancer en ligne de commande (à regarder dans le fichier « *read_me.txt* »). La génération de labyrinthe constitue une base à approfondir et nécessite quelques modifications pour ajouter les contraintes, les adapter de la génération carré a la génération circulaire.

Annexes:

https://en.wikipedia.org/wiki/Spherical_coordinate_system

- Génération d'un labyrinthe
  https://www.codeproject.com/Articles/416453/Circular-maze
  https://en.wikipedia.org/wiki/Maze_generation_algorithm
  https://fr.wikipedia.org/wiki/Algorithme_de_Kruskal#Pseudo-code

- Fisheye:
  https://observablehq.com/@benmaier/a-visually-more-appealing-fisheye-function
  ftp://ftp.cs.brown.edu/pub/techreports/93/cs93-40.pdf