
# Rapport de projet

# PAC-MAN IA

# par A2CR STUDIO

Composé par :

Alexandre Bourcier
Clément Bruley
Clément Iliou
Rémi Monteil

# Table des matières


**1 Introduction** **4**

**2 Présentation du projet** **5**
2.1 Présentation de l'équipe . . . 5
2.2 Nature du projet . . . 6
2.2.1 Description du projet . . . 6
2.3 Objet d'étude . . . 6

**3 Découpage du projet** **7**

**4 Réalisé lors de la première soutenance** **8**
4.1 Graphismes . . . 8
4.1.1 GTK et Glade . . . 8
4.1.2 L'affichage de la carte et des Sprites . . . 8
4.2 Le gameplay du joueur . . . 9
4.2.1 Structure . . . 9
4.2.2 Score, mort et niveaux . . . 10
4.2.3 Déplacements . . . 10
4.3 L'IA des fantômes . . . 11
4.3.1 Le pathfinding . . . 11
4.3.2 Le mode chasseur . . . 11
4.3.3 Le mode patrouille . . . 11
4.3.4 Le mode fuite . . . 12

**5 Effectué lors de la deuxième soutenance** **13**
5.1 Finition du visuel . . . 13
5.2 Optimisation de l'IA des fantômes . . . 14
5.2.1 Le pathfinding ou A* . . . 14
5.2.2 Le mode patrouille . . . 14
5.2.3 Le mode fuite . . . 14
5.2.4 Bonus : le mode aléatoire . . . 14
5.3 L'IA de Pac-man . . . 15
5.3.1 Le réseau de Neurones . . . 15
5.3.2 Le Q-learning . . . 16

**6 Nos réalisations finales** **18**
6.1 Historique . . . 18
6.1.1 Les IA dans Pac-man . . . 18
6.1.2 Le réseau de neurones . . . 18
6.2 Le jeu en lui-même . . . 18
6.3 Création de l'IA . . . 19
6.3.1 Structure du réseau de neurones . . . 19
6.3.2 Le Deep-Q-Learning . . . 19
6.4 L'entraînement de l'IA . . . 20
6.4.1 Généralités . . . 20
6.4.2 Création de notre propre dataset . . . 21

6.4.3 La backpropagation . . . 21
6.4.4 L'algorithme de gradient . . . 21
6.4.5 Déroulé de l'entraînement . . . 21
6.5 Divers . . . 22
6.5.1 Le site web . . . 22
6.5.2 Gestion du dépôt git . . . 22
6.5.3 LaTeX . . . 22

**7 Expérience sortant de ce projet** **23**
7.1 Expérience de groupe . . . 23
7.2 Expérience personnelle . . . 23
7.2.1 Alexandre Bourcier . . . 23
7.2.2 Clément Bruley . . . 23
7.2.3 Rémi Monteil . . . 23
7.2.4 Clément Iliou . . . 24

**8 Conclusion** **25**

# 1 Introduction

Ce rapport de soutenance va présenter toutes les avancées de notre projet du S4. Ce projet a pour but de créer une intelligence artificielle et de la montrer visuellement à l'utilisateur. Le but ici n'est évidemment pas de recoder le célèbre jeu Pac-Man, mais bien de se concentrer sur comment développer une IA qui fera un gros score.

Ce rapport va donc décrire tout d'abord l'équipe et l'origine du projet avec une présentation des différents membres. Il y aura ensuite une description du projet plus approfondie. Pour continuer, nous vous proposerons une description détaillée de tout ce que nous avons pu faire depuis la dernière soutenance. Nous y ajouterons des images pour permettre de mieux visualiser notre projet. Enfin nous vous présenterons les objectifs que nous avons pour la soutenance finale.

# 2 Présentation du projet

## 2.1 Présentation de l'équipe

**Alexandre Bourcier** : Entré à l'EPITA en 2019, je me suis toujours efforcé de faire beaucoup de programmation pour avoir des acquis et une expérience solide dans cette discipline. Ainsi, avec le projet de S2, j'avais pris en charge une partie très importante de la programmation d'un jeu vidéo(multijoueur et gameplay). Lors du projet de S2, j'ai pris en charge l'interface graphique, le pré-traitement et la segmentation de l'OCR. Ainsi, toutes ces expériences m'ont permis d'avoir des acquis assez solides et d'aborder ce projet de S4 sereinement. L'idée de refaire une intelligence artificielle me plaît beaucoup, car lors de notre projet de S3, le réseau de neurones était loin d'être impeccable et je souhaite pouvoir prendre ma revanche sur ce projet.

Pendant les projets, j'apprécie toujours commencer rapidement les projets pour pouvoir vraiment étaler tout le travail sur le temps imparti et ne pas courir à la fin et rendre quelque chose de moyennement fonctionnel. Un point très important aussi est de discuter de la conception du projet, notamment le prototype des fonction et comment les faire interagir. Tout cela nous permettra, j'espère de faire un beau projet fonctionnel.

**Clément Bruley** : La plupart des élèves d'EPITA ont commencé à toucher à l'informatique depuis de nombreuses années. Pour ma part, j'ai commencé réellement lorsque je suis entré à EPITA. Lors des deux derniers projets, nous avons approfondi nos connaissances en c# lors du S2 et en C99 lors du S3. Ce projet va me permettre d'avoir de plus grandes connaissances en C qui est un langage très utile dans le domaine de l'informatique. Possiblement, nous allons pouvoir appliquer ce que nous avons appris lors du séminaire UNIX. J'ai toujours été intéressé par les IA. Grâce à ce projet, je vais pouvoir développer mes compétences dans ce domaine.

Avec le reste du groupe nous avons décider de m'attribuer le rôle de chef de projet. Il me semble que c'est important d'avoir quelqu'un qui va être leader du projet. Néanmoins, ce rôle ne va pas être très grand, car il ne s'agit pas d'un grand projet tel que pourrait le faire des entreprises.

**Clément Iliou** : Je suis étudiant en deuxième année de classe préparatoire à EPITA. J'ai effectué une terminale S spécialité SVT. Avant EPITA, je n'avais pas fait de programmation dans un cadre scolaire. Cependant, je me suis toujours intéressé à l'informatique en général. J'aime beaucoup les travaux de groupe, car l'on peut ainsi profiter des différents atouts de tous. La partie graphique du jeu m'encourage à approfondir le domaine du traitement d'image que je trouve passionnant. Ce projet est un moyen pour moi de croiser les connaissances acquises en développement de jeu vidéo durant le projet de S2 et celles sur le machine learning et les réseaux de neurones vu pendant le projet de S3. J'ai hâte après EPITA de pouvoir développer ce genre de programme dans un cadre professionnel. Ce projet devrait me permettre d'approfondir mes connaissances en C99 et en GTK.

**Rémi Monteil** : Je suis rentré à EPITA en 2019. Depuis j'ai appris plusieurs langages tels que le Caml, c# et le c. Mais surtout j'ai appris à les exploiter au maximum avec les projets. J'avais développé le gameplay du jeu de S2 et je me suis occupé du Neural-Network de l'OCR. Je suis très attiré par ce qui touche à l'IA autant dans ces utilisations exotiques que de son apprentissage si divers. La difficulté et les erreurs que j'ai commises lors de l'OCR ne sont que de l'expérience supplémentaire pour appréhender cette nouvelle IA qui me semble plus difficile par sa méthode d'apprentissage non supervisée. J'espère partager au mieux mes connaissances des projets précédents pour accompagner mes collègues dans ce projet.

## 2.2 Nature du projet

### 2.2.1 Description du projet

Le principe du projet est assez simple. Il s'agit de reproduire le jeu Pac-Man et de faire une intelligence artificielle qui essaie de faire le score le plus élevé possible. Le but de ce projet n'est pas de coder un jeu vidéo, mais de se concentrer vraiment sur de l'intelligence artificielle avec du traitement de données dans un réseau de neurones. L'implémentation visuelle du jeu nous permettra également de faire un peu de traitement d'image.

## 2.3 Objet d'étude

Ce projet peut apporter beaucoup en terme d'expérience de programmation et de conception d'IA. Déjà dans un premier temps, il faudra programmer le jeu. Cela va demander surtout d'utiliser des connaissances en GTK+ et en traitement d'image. Il faudra programmer une mini-intelligence des fantômes, les faire bouger, rendre les pacgums interactifs, activer les super pacgums, augmenter le score et évidemment afficher graphiquement tout ça dans notre fenêtre avec GTK+.
Il faudra ensuite se concentrer sur l'intelligence artificielle et là, il y a une grosse partie de réflexion sur comment nous allons faire une intelligence artificielle. Nous pensons utiliser un réseau de neurones, mais nous allons probablement devoir faire de nombreux tests de modèles d'IA, changer les paramètres d'entrée.... . .

# 3 Découpage du projet

| Tâches à effectuer | Alexandre Bourcier | Clément Bruley | Clément Iliou | Rémi Monteil |
|---|---|---|---|---|
| Création graphique | | | ✓ | |
| Gameplay du joueur | ✓ | | ⊗ | |
| IA des fantômes | | ⊗ | | ✓ |
| IA de Pac-Man | ✓ | ⊗ | | ⊗ |
| Site web | ✓ | | | ⊗ |
| Git | ⊗ | ✓ | | |
| LaTeX | ⊗ | ✓ | | |

✓ = représentant ⊗ = *assistant*

**Les tâches énumérées précédemment ne sont pas de difficulté ni de durée équivalente. En effet, la gestion du git est très rapide à faire tandis que la création de l'IA du joueur est assez difficile.**

# 4 Réalisé lors de la première soutenance

## 4.1 Graphismes

### 4.1.1 GTK et Glade

Afin d'afficher le jeu, nous utilisons la bibliothèque GTK. Cette bibliothèque permet de créer des boutons, ainsi qu'une interface visuelle afin que le joueur puisse jouer. Pour structurer l'interface, nous utilisons Glade. Dans Glade, nous avons créé deux boutons : Start et Pause, qui permettent de démarrer ou de mettre le jeu en pause. Il y a aussi trois GtkLabel qui permettent d'afficher le score, le nombre de vies restantes au joueur et le niveau.

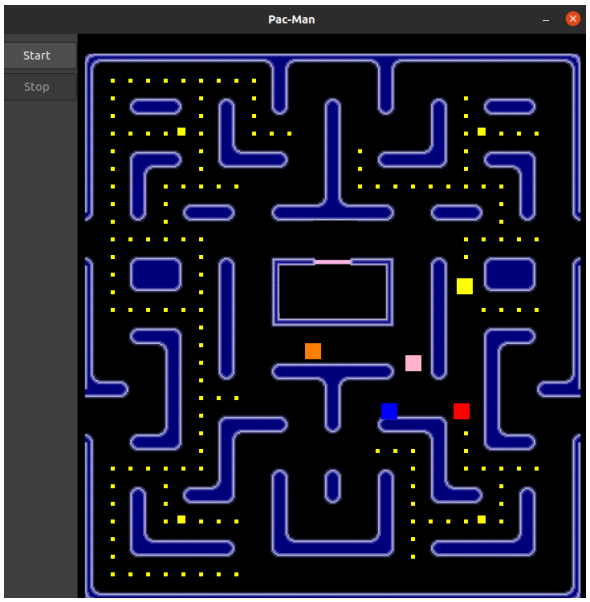



FIGURE 4.1 – Premier graphisme

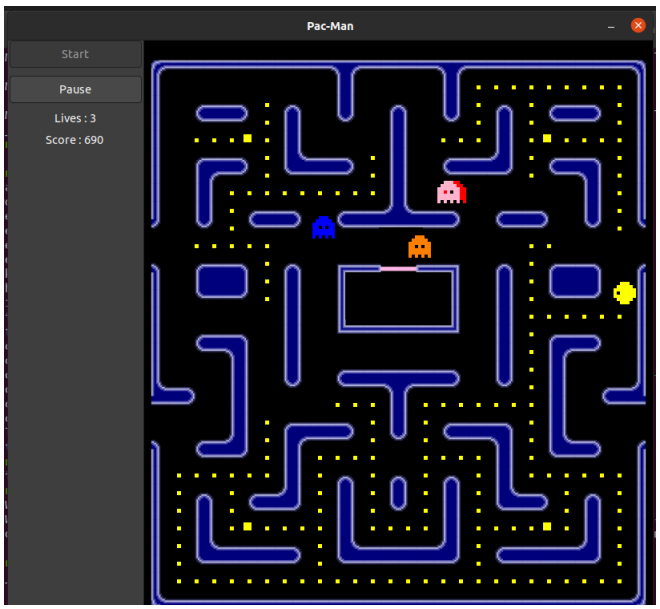



FIGURE 4.2 – Graphismes finaux

### 4.1.2 L'affichage de la carte et des Sprites

L'affichage de la carte et des Sprites sont séparés. Nous superposons grâce à un GtkOverlay une image et une Drawing Area. Nous avons choisi ce mode d'affichage, car cela nous permet d'avoir un jeu plus agréable visuellement et plus optimisé puisqu'à chaque lancement, la carte n'a pas besoin d'être reconstruite. Pour savoir si une case est un mur, un couloir vide, un couloir avec une pacgum ou une super pacgum nous avons réalisé une matrice de 28 par 31 comme dans le jeu original. Au début, de chaque partie, nous affichons toute les pacgums à partir de la matrice. Pour afficher Pac-man et les fantômes nous avons créé d'autres matrices afin de les afficher en Pixel Art. Pour Pac-man, une image sur trois celui-ci ouvre la bouche comme dans le jeu original. Pour que la direction du pixel de Pac-man change, nous changeons juste le sens du parcours de la matrice.

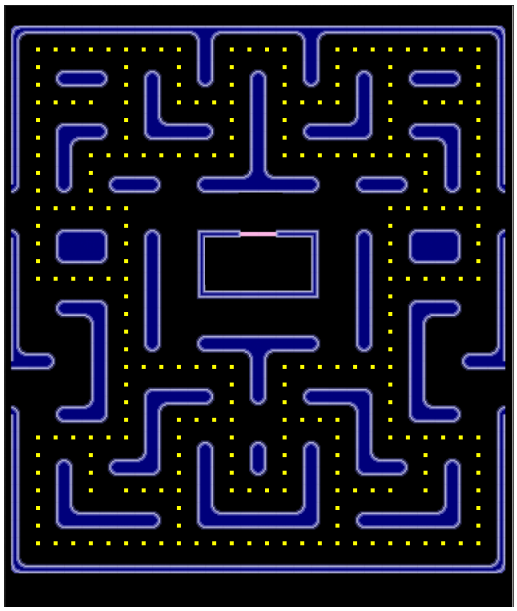


FIGURE 4.3 – Map affichée

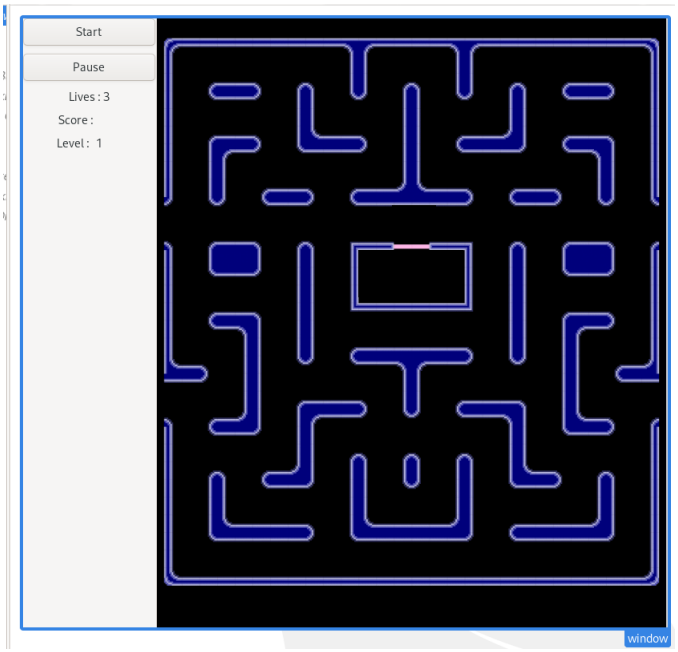



FIGURE 4.4 – Fichier glade utilisé pour faire l'interface et l'affichage de la carte

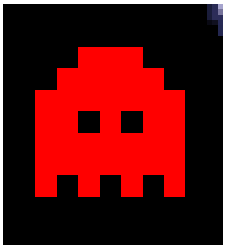


FIGURE 4.5 – Sprite de fantôme

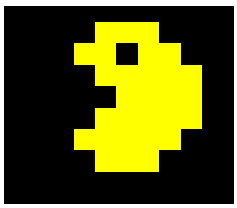


FIGURE 4.6 – Sprite de Pac-man avec la bouche ouverte

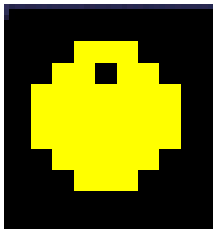


FIGURE 4.7 – Sprite de Pac-man avec la bouche fermée

## 4.2 Le gameplay du joueur

### 4.2.1 Structure

La structure du jeu comporte de nombreux paramètres. Cette structure comporte d'abord 5 joueurs : Pac-man et les 4 fantômes. Ces joueurs possèdent 3 caractéristiques : leur coordonnées (en x et en y) et leur direction. La structure du jeu possède aussi le statut pour savoir si le jeu est en pause ou en cours, le mode chasse pour savoir si Pac-man est invincible ou pas, le score, le nombre de vies et le niveau.

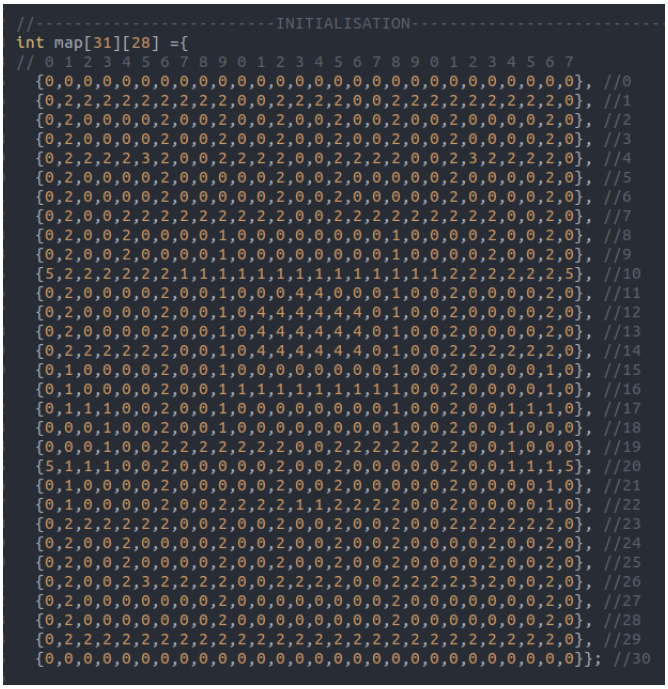



FIGURE 4.8 – Matrice utilisé pour la map

### 4.2.2 Score, mort et niveaux

Le joueur possède trois vies en début de partie, il en perd une à chaque fois qu'il se fait manger pas un fantôme. Nous vérifions que Pac-man et un des fantômes ne se trouvent pas sur la même case. Lorsque Pac-man se fait manger, celui-ci ainsi que les fantômes retournent à leur emplacement de départ. Nous utilisons alors la fonction Sleep(3) pour permettre au joueur de réfléchir à une nouvelle stratégie. Il possède 2 secondes pour réfléchir.

Lorsque le joueur mange une pacgum nous modifions la valeur de la case dans la matrice. Pour vérifier si le joueur a mangé toutes les pacgums nous utilisons un compteur du nombre de pacgum mangés. Nous réinitialisons la matrice à chaque changement de niveau. Nous ne mettons pas de limite de niveau comme dans le jeu original.

Lorsque Pac-man mange une super-pacgum (il y en a 4 par niveau) celui-ci devient bleu et il peut manger les fantômes. Les mouvements des fantômes changent et ils passent en mode fuite. L'effet de la pacgum dure environ 10 secondes, à la fin Pac-man redevient jaune.

### 4.2.3 Déplacements

Comme dit plus haut, nous avons créé une matrice de 28 par 31, et nous avons donc un premier système de coordonnées, c'est-à-dire que pour chaque entité nous avons des coordonnées dans cette matrice. Ensuite, nous avons un deuxième système de coordonnées, celles qui correspond au pixels de zone de dessin de GTK.

Ainsi, pour ne pas se perdre dans ce double système de coordonnées, nous avons décidé de séparer en 2 fichiers :

- un premier fichier GTK.c qui traite de tout ce qui est en lien avec l'interface graphique et donc le système de coordonnées relatives à la zone de dessin.
- un second fichier pac-man.c qui traite de la gestion du jeu en lui-même avec le système de coordonnées de la matrice.

Pour la direction de Pac-man, nous créons une fonction d'appel dans GTK.c, qui est déclenchée lorsqu'on appuie sur une touche. Ensuite on appelle une seconde fonction dans pac-man.c qui vérifie grâce aux coordonnées de la matrice si Pac-man ne va rentrer dans un mur en changeant de direction. On change ensuite la direction de Pac-man.

Concernant les déplacements de Pac-man, on définit la vitesse de celui-ci comme étant le nombre de pixels de déplacement par frame (le jeu est en 24 frames par seconde). Ainsi, à chaque frame, on vérifie que Pac-man ne va pas rentrer dans un mur avec les coordonnées en de la matrice et la direction

puis on modifie les coordonnées en pixel pour que Pac-man soit ensuite re-dessiner dans la zone de dessin GTK.
Pour le déplacement des fantômes, la procédure est similaire à celle de Pac-man sauf que la direction est redéfinie à tous les frames et que celle-ci tient compte des obstacles. Ainsi, il ne reste plus qu'à mettre à jours le coordonnées.

## 4.3 L'IA des fantômes

### 4.3.1 Le pathfinding

Les fantômes utilisent un algorithme de pathfinding. Au moment de la création de la fonction, nous étions et nous sommes toujours dans l'étude des graphes en cours d'algorithmique. L'algorithme privilégié pour trouver le plus court chemin était le parcours largeur. Nous avons donc adapté ce parcours de graphe à celui de l'exploration de labyrinthe dans la matrice.

Comme vu précédemment nous utilisons une matrice qui définie chaque case du plateau de jeu. Pour faire le plus simple possible nous avons attribué des numéros à chaque "état" que peut prendre la case. Par exemple les cases "0" sont des murs et "1", "2" et "3" sont des couloirs avec ou sans pacgum. Les cases "4" correspondent à la maison des fantômes et les cases 5 sont les télé-porteurs que nous ignorons pour le moment. Pour réussir l'implémentation du parcours largeur, il a fallut avoir les queues qu'on utilise en cours d'algorithmique. Nous avions trouvé la librairie "sys/queue.h". Néanmoins, suite à des difficultés de manipulation et des doutes ressentis, nous avons donc décidé de créer notre propre struct queue qui à été optimisé et corrigé au fil du projet.

Nous avons effectué des recherches sur d'autres algorithmes de pathfinding et avons trouvé le a* qui est décrit comme le plus performant. Mais après la première soutenance, nous avons conclu que ce serait une perte de temps et dangereux de le faire à 1 semaine de la soutenance d'autant plus que notre fonction de pathfinding actuel fonctionne. Mais le changement vers cet algorithme a* sera probablement envisagé et étudié pour la prochaine soutenance.

### 4.3.2 Le mode chasseur

En mode chasseur les fantômes : Blinky (le rouge), Pinky (le rose), Inky (le bleu) et Clyde (le orange) vont pourchasser Pac-man. Mais ils ont chacun leurs propre technique :

- Blinky : Il va simplement poursuivre Pac-man.
- Pinky : Il est un peu plus malin. Il va tenter de piéger Pac-man. Pour cela il se dirige vers le mur où Pac-man irait s'il continuait dans sa direction actuelle.
- Inky : Il est un peu plus compliqué. Il va considérer que Pac-man essayera de fuir en priorité Blinky (étant donné qui semble le plus agressif). Ainsi, il prend comme destination une coordonnée qui est le symétrique ou aux alentours du symétrique de la position de Blinky par rapport à Pac-man. Dans certains cas, ce symétrique se trouve hors de la map, il va alors effectuer la même chose que Blinky.
- Clyde : Il est un peu plus fantaisiste. Il va toujours fuir Pac-man. Il va vers les coordonnées étant les plus proches de lui et dans hors d'un rayon de 5 cases autour de Pac-man.

Note : les noms japonais expriment très bien leur mode chasseur respectif :

- Blinky = chasseur
- Pinky = embusqué
- Inky = inconstants
- Clyde = stupide.

### 4.3.3 Le mode patrouille

Les fantômes ne poursuivent pas constamment Pac-man. Au bout d'une certaine distance de Pac-man, ils décident d'arrêter leur chasse et de passer en mode patrouille. Ainsi le fantôme en question va se promener dans le coin le plus proche ou il se trouve. Pour le moment ce mode n'est pas activé.

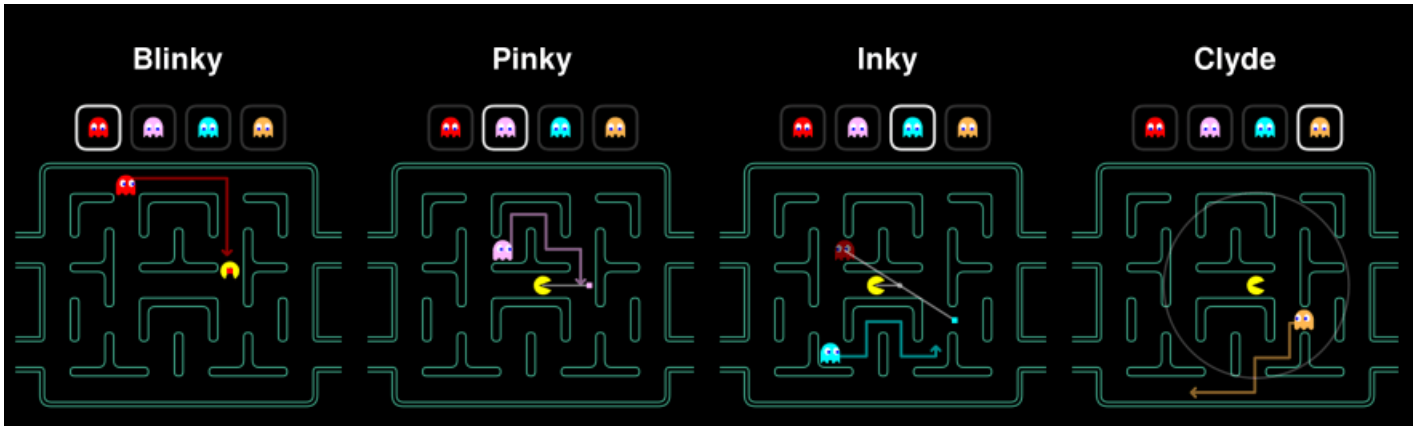



FIGURE 4.9 – Mode chasseur de chaque fantômes

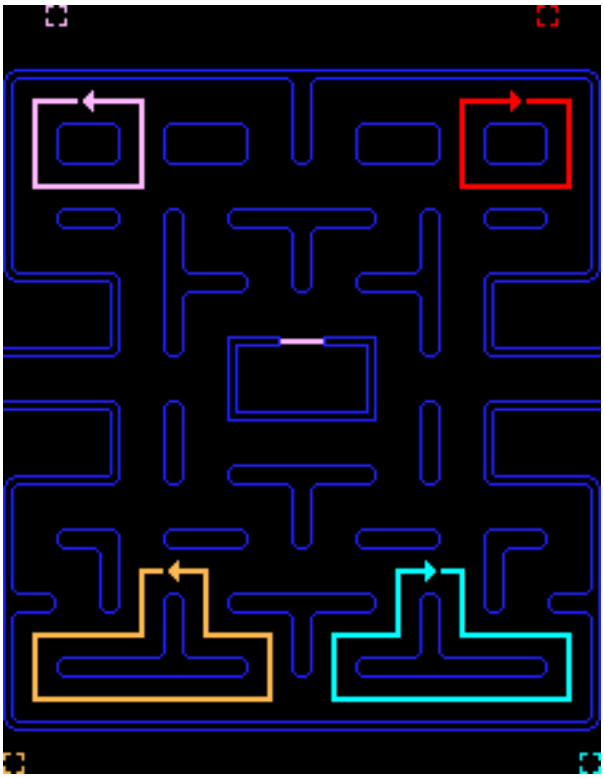


FIGURE 4.10 – Mode patrouille de chaque fantôme

### 4.3.4 Le mode fuite

Ce mode peut être déclenché uniquement par Pac-man lorsqu'il mange une super pacgum. A partir de ce moment Pac-man à 10 secondes pour manger les 4 fantômes et ainsi obtenir plus de points. A ce moment, les mouvements des fantômes sont aléatoires comme dans le jeu original.

# 5 Effectué lors de la deuxième soutenance

## 5.1 Finition du visuel

La finition du visuel est une partie importante car il rend le jeu plus agréable pour l'utilisateur. Le pixel art des fantômes a été amélioré pour les rendre plus intéressants et réalistes. Lorsque Pac-man mange une super pacgum, les fantômes et Pac-man deviennent bleus et pendant les dernières secondes de l'effet, Pac-man clignote pour indiquer au joueur que l'effet va prendre fin. De plus, lorsqu'un fantôme est mangé par Pac-man celui-ci ne peut pas se faire manger à nouveau, il faut attendre que Pac-man mange une autre super pacgum. Les fantômes ne pouvant pas être remangés gardent leur couleur d'origine.

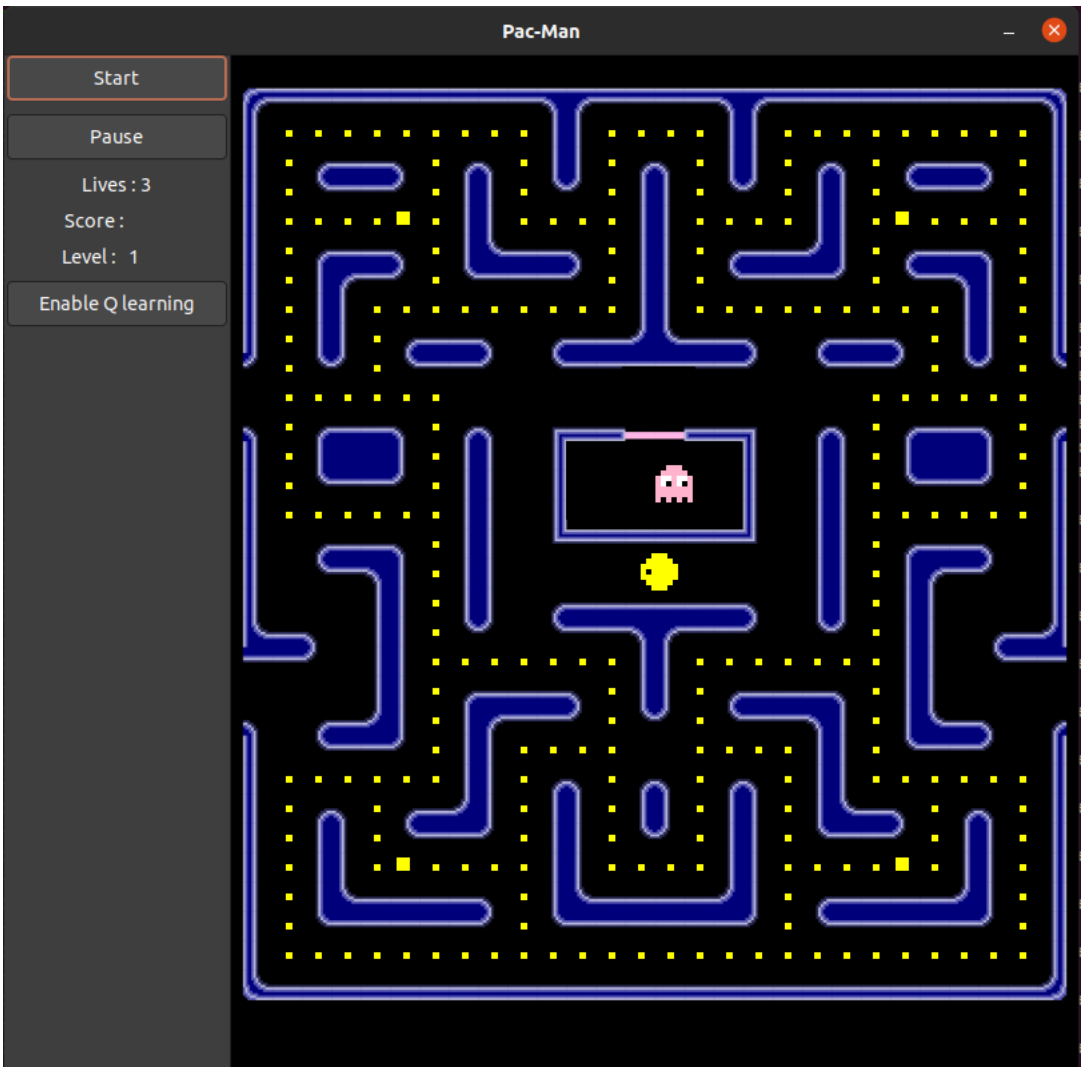



FIGURE 5.1 – Nouvelle interface

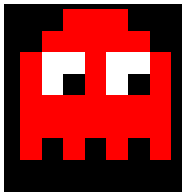


FIGURE 5.2 – Sprite de fantôme

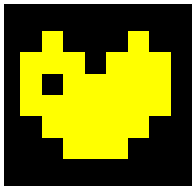


FIGURE 5.3 – Sprite de Pac-man avec la bouche ouverte

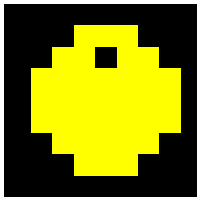


FIGURE 5.4 – Sprite de Pac-man avec la bouche fermée

## 5.2 Optimisation de l'IA des fantômes

### 5.2.1 Le pathfinding ou A*

Nous avions comme objectif d'implémenter un nouveau pathfinding en utilisant l'algorithme A*. Néanmoins près quelques recherches nous nous sommes aperçus que le temps passé pour concevoir l'algorithme était une perte de temps. En effet étant donnée que nous sommes dans un labyrinthe le parcours largeur classique est presque aussi performant que A*. Ce dernier est conçu pour être très efficace dans de grands espaces comme la recherche d'un objet dans une matrice vaste.

### 5.2.2 Le mode patrouille

Nous avons ajouté le mode patrouille. Les fantômes alternent entre le mode chasseur et le mode patrouille. Au début du niveau, ils passent 7 secondes en mode patrouille puis 20 secondes en mode chasse et ainsi de suite. Pour chaque fantômes deux éléments ont été rajoutés à leur structure : une liste et un compteur. La liste correspond à une liste contenant les points que le fantôme doit parcourir durant le mode patrouille. À chaque fois qu'un point est atteint le compteur est incrémenté de 1 et donc sélectionne le point suivant dans la liste. Pinky et Blinky possèdent 4 points dans leur liste tandis que les fantômes Clyde et Inky en possèdent 5.

Pour déterminer la direction dans laquelle les fantômes se déplacent on utilise la fonction qui détermine le parcours de Blinky en y remplaçant les coordonnées de Pac-man par le point de la liste.

### 5.2.3 Le mode fuite

Le mode fuite est un mode de déplacement aléatoire. Ce mode n'est pas encore totalement achevé en effet nous utilisons la fonction de valeur aléatoire d'une des bibliothèque de base du C mais celle-ci change de valeur moins souvent que l'on lance la boucle du jeu ce qui pose certains problèmes. Certaines fois, les fantômes peuvent rester coincés dans un mur pendant quelques secondes. Nous cherchons encore une solution à ce problème.

### 5.2.4 Bonus : le mode aléatoire

Le mode aléatoire pour chaque fantôme est déjà implémenté, néanmoins nous ne comptons pas l'activer pour le moment. En effet, notre IA n'est pas du tout assez puissante pour supporter de l'aléatoire chez les 4 fantômes.

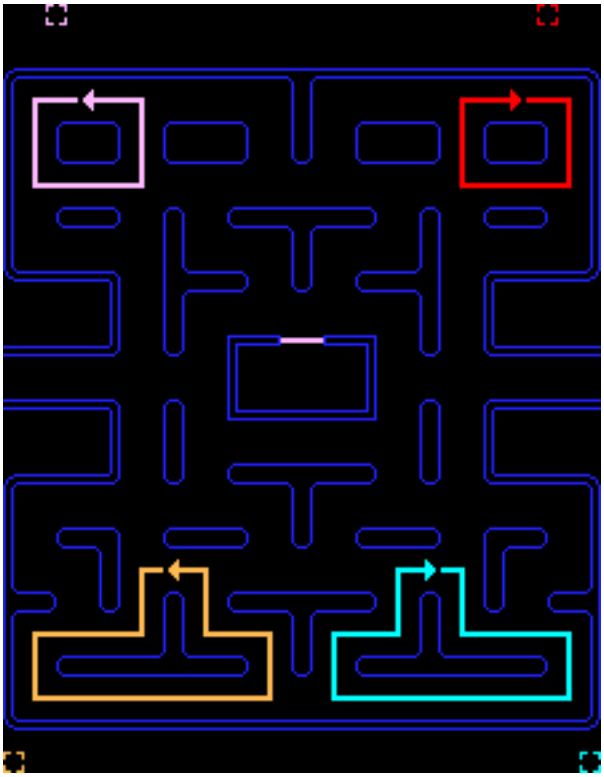


FIGURE 5.5 – Mode patrouille de chaque fantôme

## 5.3 L'IA de Pac-man

### 5.3.1 Le réseau de Neurones

#### La théorie

Pour l'IA de Pac-man, nous avons donc opté pour un réseau de neurones en ayant pour base l'expérience acquise lors du projet du S3 : l'OCR. Ainsi nous avons récupéré les codes nous permettant la création des neurones, la propagation du réseau ... Ce code sera modifié pour l'adapter au machine learning de Pac-man (modification expliquée dans la partie suivante).

Nous sommes donc encore une fois dans du deep learning. Mais le machine learning de Pac-man est diffèrent de l'OCR. Celui du projet précédent était un apprentissage supervisé. On pouvait lui indiquer à chaque exécution qu'elle était la réponse attendue et les réponses non-attendues et donc pouvoir le corriger avec de la back-propagation. Mais nous ne sommes pas dans un apprentissage non-supervisé, cela fut une erreur de ma part de dire cela en début de projet en me justifiant par mon manque de connaissance dans le sujet. On pourrait croire que cela est le bon machine learning étant donné qu'on est dans l'inconnu pour savoir qu'elle est la bonne action et les mauvaises actions. Inconnue typique à l'apprentissage non-supervisé.

Mais dans le cas des jeux tel que Pac-man cela est nuancé. On ne sait pas qu'elle est la MEILLEURE bonne action. Dans le projet actuel nous sommes donc dans un apprentissage semi-supervisé ou plus communément appelé apprentissage par renforcement : machine learning adepte des théories du jeux et des jeux. L'apprentissage par renforcement va pouvoir répondre à la question de la meilleure bonne action. Dans ce machine learning, l'IA va étudier son environnement : les murs, les fantômes, les pacgums à son état actuel dans sa partie et ainsi opter pour l'action qui va lui apporter la meilleure récompense. Il existe plusieurs algorithmes permettant le calcul de cette récompense qui se distingue par l'utilisation du réseau. Pour Pac-man, nous avons décidé d'exploiter le Q-learning (explication dans la partie suivantes) qui nous semble être le plus adéquat pour notre Pac-man.

### La pratique

En terme d'implémentation, nous avons souhaité pouvoir facilement modifier l'architecture de notre réseau de neurones, c'est-à-dire pouvoir modifier en quelques secondes le nombre de couches et de neurones par couche. Ainsi, en fonction de toutes ces données, on va constituer trois listes :

- Une liste d'entrées qui est égale au nombre total de neurones de toutes les couches.
- Une liste de poids qui est égale à la somme du nombre de neurones de la couche précédente multipliée par ceux de la couche actuelle pour chaque couche sauf pour la première.
- Une liste de biais qui est égale au nombre total de neurones sauf la première couche.

Ces trois listes suffisent alors à pouvoir mettre en place tout notre réseau. On a créé un structure "neurone" pour chaque neurone du réseau. Cette structure est alors constituée de :

- une variable "size" qui contient la taille de la liste d'entrées des neurones.
- la liste "input" qui contient la liste des entrées du neurone (toutes les sorties des couches précédentes)
- la liste "weights" qui contient tous les poids associés aux entrées.
- la liste "bias" qui contient le biais de notre neurone.

Ainsi, pour initialiser correctement le réseau, on déplace les pointeurs de nos 3 listes aux bons endroits en fonction des neurones. Pour pouvoir exécuter le réseau, on appelle une fonction d'activation pour chaque neurone qui fait la somme des poids multipliés par les entrées et son biais correspondant. On stocke ensuite le résultat dans la liste d'entrées pour la couche suivante. Après avoir exécuté les fonctions pour tous les neurones, on regarde dans la couche finale, le neurone qui a le résultat le plus élevé et l'on retourne la direction qui lui a été attribué.

Concernant les données d'entrée, nous avons 17 inputs :

- La présence des murs dans les 4 directions -> inputs à 1 s'il n'y a pas de murs dans la direction correspondant.
- Le nombre de fantômes dans les 4 directions -> inputs égal au nombre de fantômes divisé par 4 dans la direction correspondante (s'il y a un mur dans la direction, c'est 0).
- Le nombre de pacgum dans les 4 directions -> inputs égal a nombre de pacgum divisé par cases parcourues dans la direction correspondant (s'il y a un mur dans la direction, c'est 0).
- Le nombre de super-pacgum dans les 4 directions -> inputs égal a nombre de super-pacgum divisé par 4 dans la direction correspondante (s'il y a un mur dans la direction, c'est 0).

Enfin, pour finir sur le système de réseau de neurones, nous avons décidé de faire un système de génération de réseau. On a donc fait une liste de réseaux de neurones et quand le jeu est lancé, on exécute chaque réseau puis on conserve les meilleurs réseaux que l'on sauvegarde dans nos fichiers et on en fait des dérivés en modifiant les poids et biais.

## 5.3.2 Le Q-learning

Le Q-learning est une des techniques utilisée pour permettre à une IA de reinforcement learning d'apprendre rapidement. Pour cela, le q-learning va avoir besoin de connaître la récompense que nous allons donner à chaque état que va prendre le jeu. Ainsi, il va pouvoir en déduire une certaine récompense en fonction de chaque état où il se trouve.

Il existe 2 phases pour le Q-learning. La première consiste à de l'exploration. Il va chercher de nouveau chemin la plupart du temps pour savoir s'il y a des chemins qui vont lui donner plus de récompenses au final ou pas. La seconde phase est l'exploitation. Cela correspond à l'utilisation des données qu'il aura accumulées au fur et à mesure de son exploration. En réalité, ces 2 phases s'effectuent en même temps lors de l'exécution du réseau de neurones. En effet, sur les premières générations, le Q-learning va privilégier l'exploration pour trouver de nouvelles récompenses plus élevées. Au contraire, il ne va faire que peu d'exploitation, car les récompenses peuvent facilement être augmentées normalement sauf si nous avons une chance incroyable dès le début. Puis au fur et à mesure que les générations passent il y aura moins d'exploration et passera majoritairement en exploitation des résultats. Il passera alors de 90% d'exploration à 10% tandis que l'exploitation passera de 10% à 90%.

Nous pouvons donc dire que la méthode du Q-learning permet à l'IA de se créer un "dataset" qui lui permet ensuite de prendre la meilleure option en fonction de son état actuel. Le nom exact pour designer le "dataset" créé par le Q-learning est Q-table (cf Figure 7.2). Comme on peut le voir sur l'image ci-dessous plus les valeurs sont élevées plus il est bon d'être dans cet état.

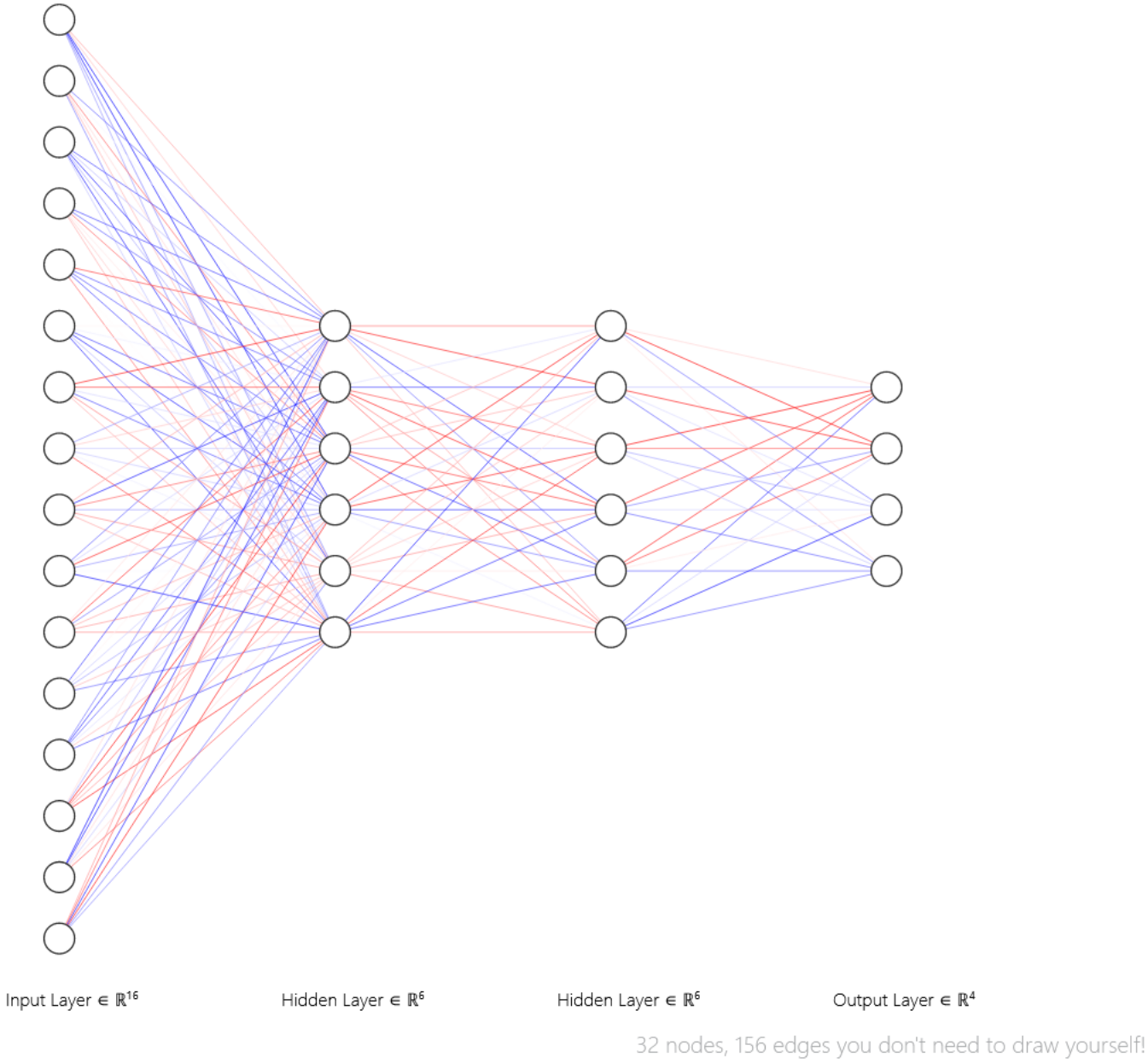



FIGURE 5.6 – Représentation graphique du réseau de neurones

Le problème que nous avons pour le moment est que nous possédons 17 inputs qui peuvent avoir chacun plusieurs états. Ainsi, si nous voulons déclarer absolument tous les états possibles que pourrait prendre le jeu, il nous faudra effectuer une imbrication de 17 'for' pour l'initialisation. Cela est énorme et n'est pas trop envisageable.

# 6 Nos réalisations finales

## 6.1 Historique

### 6.1.1 Les IA dans Pac-man

Pac-man a été un sujet de choix pour le développement d'IA. En effet ce jeu offre un challenge à l'IA tout en ayant des règles simples. Les premières IA jouant à Pac-man ont été développé au début des années 2000. Les IA développé étaient soit du côté des fantômes soit de Pac-man.Les premiers algorithme génétiques jouant à Pac-man datent de 2007, il s'agit donc un domaine relativement récent. Ces dernières années, les chercheurs se sont plutôt tournée vers Ms Pac-man. Ms Pac-man est similaire en tout point à Pac-man mais il existe 4 cartes différentes.

### 6.1.2 Le réseau de neurones

Les réseau de neurones sont un sous-ensemble de l'apprentissage automatique initié par Alan Turing en 1950. Inspirer bien évidement des réseau de neurones biologique dont les principes fondamentales ont été crées dans les années 1940, c'est en 1957 qu'apparaît le premier réseau artificiel "perceptron" capable de reconnaître des formes. Après une traverse du désert, c'est en 1982 que revient en force les réseau de neurones artificiel avec le développement de plusieurs couches cachées. Perceptron ne possède qu'une couche d'entrée et de sortie. Depuis, la technologie des réseau n'a fait que progresser en fonction des avancées informatique tel que la puissance de calcul et le développement de diverses théorie.

## 6.2 Le jeu en lui-même

Le jeu en lui-même n'a pas beaucoup changé entre la deuxième et la troisième soutenance. L'un des changement le plus important est que la boucle du jeu est maintenant appelé depuis une autre boucle dans laquelle le réseau de neurones est appelé. L'autre changement majeur est le faite que Pac-man peut passer à travers les murs. Cela peut paraître étrange voir être considéré comme une regrettions. Mais il s'agit d'une fonctionnalité essentiel nécessaire au bon apprentissage du réseau de neurone. En effet celui-ci est alerté du franchissement d'un mur par une récompense négative.

Le jeu aussi a été amélioré au niveau de l'aléatoire, en effet les fantômes en mode fuite ont des déplacements aléatoires. Ces déplacements étaient à la dernière soutenance dépendant de l'aléatoire de la bibliothèque de base du langage C. Cet aléatoire n'était pas très efficace car il changeait toute les secondes, cependant pour que le déplacement des fantômes soit fluide il fallait que le nombre change bien plus souvent. Nous avons donc implémenter notre propre fonction aléatoire. Cette fonction est bien plus efficace et a pu être utilisé dans d'autres partie du projet. Par ailleurs, le jeu en lui-même a subit d'autre changement pour permettre au réseau d'accéder plus facilement aux informations lié à Pac-man, aux fantômes et à la position des pacgums, la fonction initialisant les variables du jeu en compte maintenant 60.

## 6.3 Création de l'IA

### 6.3.1 Structure du réseau de neurones

Lors de cette dernière soutenance, le réseau de neurone a complètement changé. En effet, les entrées et la structure sont complètement différents. Le réseau prend désormais 121 entrées qui seront expliqué dans la partie du Q-learning, 2 couches cachées de 70 neurones et 4 neurones de sorties. Nous avons toujours des neurones qui sont inter-connectés entre chaque couche et on a toujours une sommes des poids et des biais comme sur un réseau classique. Les changements se font sur les fonctions d'activation qui sont désormais différentes afin de pouvoir prendre n'importe quelle valeurs car la fonction sigmoïde ne proposait que des valeurs entre 0 et 1 ce qui été insuffisant. Nous avons donc pour les couche caché mis la fonction ELU (exponantial linear unit) avec cette formule :

$$R(z) = \begin{cases} z & \text{si } z > 0 \\ \exp z & \text{sinon} \end{cases} \tag{6.1}$$

Pour la couche de sortie, la fonction est encore plus simple puisque que nous utilisons une fonction linéaire de formule

$$R(z) = z \tag{6.2}$$

Ainsi grâce à toutes ces fonctions, nous avons la possibilité d'avoir toutes les sorties possibles ce qui va être très utile à Pac-man pour connaître la bonne direction à prendre.

Ces deux fonctions nous permettent aussi de gagner du temps de calcul car avec sigmoïde nous avions de l'exponentiel partout tandis qu'ici il n'y en a que dans les couches cachés pour des sommes de neurones négatifs.

Au niveau de l'implémentation le réseau a gardé la même structure qu'à la deuxième soutenance, une structure qui était fonctionnelle et qui n'a donc par conséquent pas besoin de modification.

Pour conclure, cette partie, le but à cette troisième soutenance était d'avoir un réseau évidement fonctionnel mais surtout assez "universel" pour que l'on puisse lui faire rentré toutes les valeurs que l'on veut avec n'importe quelle valeurs de sortie.

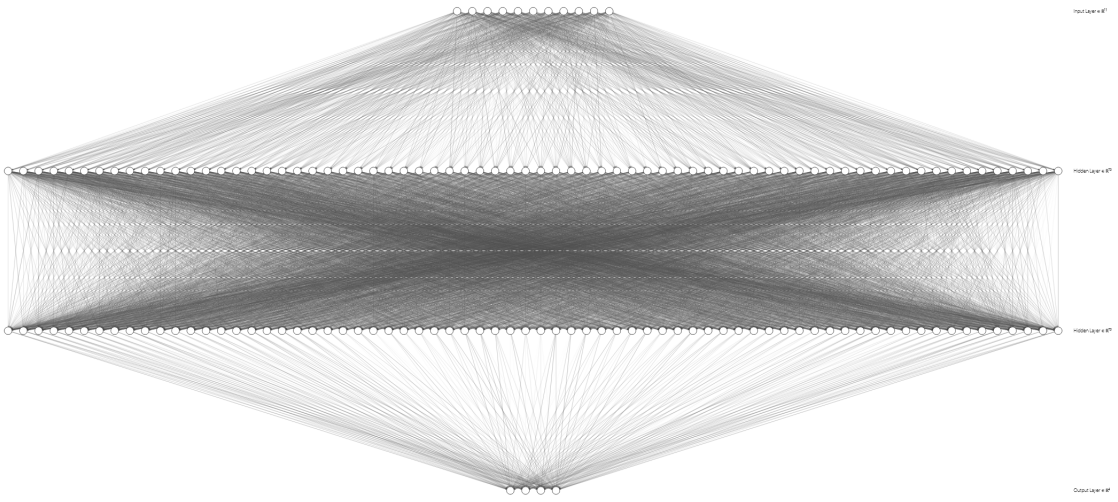


FIGURE 6.1 – Réseau final (couche du haut = 121 ici 11 pour le visuel)

### 6.3.2 Le Deep-Q-Learning

Lors de la deuxième soutenance, nous avions présenté le fonctionnement du Q-learning avec une Q-table où en fonction de l'état de Pac-man (la case où il se trouve) nous retournions quarte valeurs pour les 4 directions possibles qui correspondent à la récompense que l'on peut espérer (pacgum).

Seulement, le jeu bloquait Pac-man s'il allait dans les murs et il n'y avait pas de fantômes. Rajouter ces deux aspects, complexifie tout de suite les choses et nous arrivons a des millions de possibilité. Il est donc impossible de stocker toutes ces valeurs. C'est donc pour cela que nous allons utiliser un réseau de neurones pour faire une approximation de la Q-table.

Tout d'abord les entrées du réseau : nous avons décidé de faire une couche d'entrée de 121 neurones. Ces 121 entrées correspondent à la situation de Pac-man, c'est-à-dire que nous regardons à une distance de 5 cases de Pac-man dans toutes les directions et nous mettons cela dans la couche d'entrée. Voici un exemple pour être plus précis :

$$\begin{array}{cccccccccccc} 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & -3 \\ 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & -3 \\ 0 & -3 & -3 & -3 & -3 & 17 & -3 & -3 & -3 & 0 & -3 \\ 0 & -3 & -3 & -3 & -3 & PM & -3 & -3 & -3 & 0 & -3 \\ 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 \\ 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & -3 \\ 0 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & 0 & -3 \\ 3 & 3 & 3 & 3 & -3 & -3 & 3 & 3 & 3 & 3 & 3 \end{array} \tag{6.3}$$

Dans cette exemple-ci, les "-3" correspondent à des murs ou à la maison des fantômes, les "0" à des chemins, les "3" aux pacgums et "PM" est évidemment notre Pac-man. On reconnaît au milieu la maison des fantômes avec Pac-man qui est dedans et au dessus, on a un "17" qui correspond à la récompense que Pac-man peut espérer car les fantômes sont en mode fuite à cet instant. Il s'agit évidemment d'un état relatif au début de l'entraînement vu que Pac-man ce trouve dans la maison des fantômes, chose qu'il ne fait plus après un peu d'entraînement.

Ainsi, pour chaque frame, on envoie au réseau toutes ces entrées et il nous ressort la valeurs des récompenses que nous pouvons espérer. Voici un exemple de récompense que nous pourrions espérer :

$$\begin{array}{cc} Nord & output[0] = -3.546480 \\ Sud & output[1] = -3.135717 \\ Ouest & output[2] = 363.469684 \\ Est & output[3] = 344.693180 \end{array} \tag{6.4}$$

On voit bien ici qu'au Nord et au Sud de Pac-man il y a un mur et qu'aller à droite est ce qu'il y a de plus rentable pour lui.

## 6.4 L'entraînement de l'IA

### 6.4.1 Généralités

Nous allons devoir entraîner notre réseau afin que ces prédictions soient justes et que ainsi Pac-man fasse des hauts scores. Il a seulement, en apprentissage profond, quelques règles à respecter. Il faut tout d'abord s'assurer que les récompenses que nous mettons en entrée du réseau soient bien les bonnes. Il faut ensuite lancé le jeu et constitué une base de données avec à chaque fois les inputs, la récompense ainsi que l'action choisi. Une fois une base de données importante constituée, nous choisissons aléatoirement une de ces données pour calculer l'erreur. Il est important que ces données soient choisies aléatoirement pour éviter des corrélations et que Pac-man s'enferment dans des minimums locaux.

### 6.4.2 Création de notre propre dataset

Comme vu précédemment, pour pouvoir réaliser de bon score le réseau de neurones à besoin d'avoir des expériences sur lesquelles se baser pour ensuite pouvoir déduire la bonne solution à choisir. Pour cela nous avons réadapter la queue que nous avions implémenté pour le pathfinding. Nous avons inséré une structure composé du lidar correspondant à tout les inputs données au réseau, l'action choisi ainsi que la récompense obtenue. Pour éviter la corrélation lors du choix de l'expérience, nous avons utiliser un random entre 0 et 50. Nous avons ensuite fait tourné la queue en enlevant puis remettant les expériences tant que nous n'arrivions pas au nombre aléatoire choisi.

### 6.4.3 La backpropagation

Quand nous créons un réseau de neurones, les résultats qui sortent par rapport au entrées sont complètement anarchiques. Il faut donc modifier les poids et les biais afin d'avoir des résultats cohérent. Pour cela, nous allons utiliser l'algorithme du gradient. Seulement pour cette algorithme, nous avons besoin d'avoir une fonction de perte qui va calculer l'écart entre la valeur du réseau de neurones et la valeurs réel (la récompense). Voici la formule de cette fonction de perte :

$$L(s, a, s') = (Q(s, a) - (r + 0,99 * max(Q(s'))))^2 \tag{6.5}$$

Cette équation est connue sous le nom d'équation de Belleman. Ici la fonction $Q$ correspond au résultat du neurone correspondant à l'action $a$ avec l'entrée $s$. $s'$ correspond à l'entrée de l'état suivant (quand Pac-man a bougé) et $r$ correspond à la récompense que Pac-man a reçu en se déplaçant. On voit bien ici le coté récursif du réseau de la Q-fonction avec la valeur de l'état suivant qui est retourné et qui donc permet de comprendre où se trouvent les récompenses.

### 6.4.4 L'algorithme de gradient

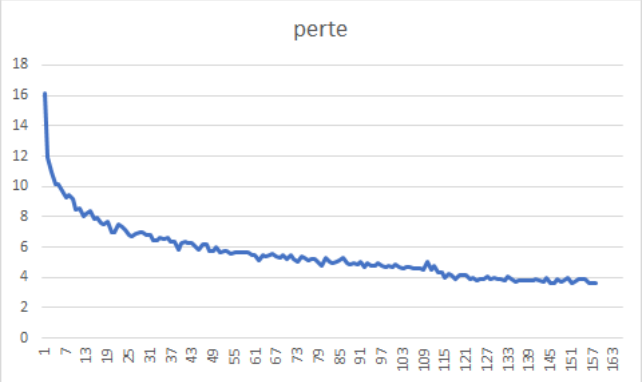
L'algorithme du gradient a été la partie la plus complexe à réaliser car elle fait appelle à des notions mathématiques un peu avancé mais surtout elle demande de gérer une grosse quantité de données. Nous ne rentrerons pas dans les détails techniques mais il s'agit de trouver le gradient de chaque poids de chaque neurone de chaque couche à partir de la fonction de perte pour voir à quel point ce poids a faussé la valeur de sortie et ainsi le corriger. Cela se fait donc avec le fonction de perte et nous effectuons de multiplications de dérivé afin d'arriver au poids souhaité.

### 6.4.5 Déroulé de l'entraînement

Nous avons passé près de deux semaines à entraîner le réseau de neurones et cette partie a été complexe car nous n'avions aucune idée de ce qu'était une bonne valeur de sortie. C'est pour cela que nous avons tâtonne pour trouver une bonne configuration. Nous avons eu plusieurs paramètre à modifier :

- le nombre de neurones pour ces deux couches : plus ce nombre est élevé plus le réseau sera précis mais lent à entraîner.
- le learning-rate : cette valeur permet de décider à quel point on va modifier les poids lors de l'entraînement : si elle est trop élevée les poids se modifient trop et tout le réseau finit par sortir des valeurs délirantes. Dans notre cas, nous avons eu des nan et -nan ce qui correspond à l'infini. Si elle est trop faible, le réseau met trop de temps à s'entraîner.
- Epsilon : cette valeur est un pourcentage indiquant si nous laissons le réseau décider de son destin ou si nous le laissons choisir l'action aléatoirement dans la base de données. En effet, l'aléatoire permet au réseau de découvrir de nouvelles stratégies mais Pac-man peut aussi finir dans les murs et donc faire perdre des entraînements intéressants.

Ainsi en entraînant le réseau nous avons eu au tout début une diminution franche de la perte mais au bout d'un certaine temps (100 000 entraînements), elle s'est stabilisé autour de 3 ou 4.




Au niveau de gameplay, on a observé qu'à ce stade l'IA a compris qu'elle ne doit pas traversé les murs mais elle est loin de faire de gros score. Nous nous interrogeons donc sur si il ne fallait pas plusieurs semaines pour entraîner cette IA

## 6.5 Divers

### 6.5.1 Le site web

Le site web est complété de tous le reste des informations manquants tels que le dernier chapitre sur l'évolution de notre projets, nos derniers rapports de soutenances et autres informations.

### 6.5.2 Gestion du dépôt git

Pour cette dernière soutenance, nous sommes tous resté sur la même branche car nous travaillions sur le même sujet a savoir l'IA. Nous avons par conséquent su apprendre à gérer les conflits git mais aussi à voir la puissance de ce gestionnaire de code. Finalement durant ce projet nous n'avons pas eu de soucis majeurs qui nous a ralenti.

### 6.5.3 LaTeX

Nous avons pris l'habitude d'écrire du texte sur LaTeXet nous avons plus vraiment de difficultés a faire nos rapport dessus. Pour cette dernière soutenance nous avons juste appris à écrire des équations pour les formules de fonctions d'activations notamment.

# 7 Expérience sortant de ce projet

## 7.1 Expérience de groupe

Ce projet a été plus complexe que les précédents à réaliser en groupe. En effet, le second confinement a empêché de nous voir physiquement et donc il y a forcement plus de difficultés à comprendre le code. Nous avons eu aussi des difficultés sur l'architecture du code et surtout sur la compréhension du code. Une fonction de 1200 lignes, même si elle est fonctionnelle est extrêmement complexe à comprendre. Il a donc fallu à plusieurs reprise repasser sur le même code afin de le fragmenter pour finir avec du code lisible par tous le monde. Concernant l'avancement individuel sur le code, chacun avait sa partie et donc la maîtrisait bien mais quand il s'agissait de mettre le travail en commun, cela a été plus complexe. Nous devons aussi faire la remarque que concernant l'IA, il faut avoir de bonnes connaissances mathématiques dessus et même de bonnes connaissances sur ce qu'est du Q-learning. Si l'on ne se plonge pas dessus durement, c'est impossible de coder quelque chose.

## 7.2 Expérience personnelle

### 7.2.1 Alexandre Bourcier

Ce projet a été le plus compliqué à réaliser pour moi. C'est un mix entre le projet de S2 où il s'agissait de réaliser une jeu vidéo et le projet de S3. Nous avons pu travailler sur des papiers de recherche assez récent car le concept de Deep-Q-learning n'a été créé qu'en 2015, de quoi donner encore plus de motivation. Cependant coder une IA à partir de rien a été particulièrement complexe et nous avons vraiment eu beaucoup de difficultés à réussir à faire quelque chose de fonctionnel et le résultat nous semble vraiment dérisoire par rapport au travail fourni.

### 7.2.2 Clément Bruley

Ce projet à été pour moi le plus enrichissant de tous bien que complexe à réaliser. J'ai pu acquérir de nombreuses connaissances dans le domaine de l'intelligence artificielle. Je n'y connaissais presque rien à part le concept des réseaux de neurones. J'ai pu découvrir le reinforcement learning utilisé avec le q-learning, une notion très récente et d'une très grande puissance d'autant plus lorsqu'il est liée avec un réseau de neurones. Je n'étais pas très à l'aise avec le C avant de commencer ce projet et suis maintenant beaucoup plus rapide.

### 7.2.3 Rémi Monteil

Ce projet a été très passionnant. Entre le plaisir de recréer un jeu et de réutiliser un réseau, je fut comblé. Ce plaisir fut plus grand avec la découverte d'un nouveau machine learning : le semi-supervized ou plutôt le reinforcement learning. Bien plus complexe, ces concepts malgré une difficulté de compréhension fut néanmoins plaisante. Ce machine learning est nettement plus compliqué à implémente que celui de l'OCR. Au delà de ça, découvrir le jeu Pacman et certaine spécificité fut intéressant.

### 7.2.4 Clément Iliou

Ce projet a été extrêmement intéressant. En effet ce projet m'a permis d'apprendre a créé un programme sans consignes mais toujours avec des contraintes. Je me suis chargé de la partie graphique et du jeu en lui même. Le développement d'un interface graphique et logique de A à Z fut une expérience très enrichissante. Le jeu a du s'adapter au besoin du réseau de neurones que ce soit au niveau logique que visuelle. J'ai aussi pu apprendre à me servir de la bibliothèque GTK, bibliothèque très polyvalente et pratique ; mais après mur réflexion je pense qu'il existe de meilleur méthodes pour développer un jeu en C.

# 8 Conclusion

Pour conclure se projet, nous pouvons dire que nous avons tous appris de nombreuses choses dans le domaine de l'intelligence artificielle. Les réseaux de neurones ainsi que les différents types d'apprentissages sont tout aussi fascinant que compliqué à mettre en oeuvre lorsque nous ne connaissons pas encore le concept. Lors de la première soutenance nous avions réussi à finir pratiquement toute l'implémentation du jeu Pac-man. Il nous a fallu comprendre les mécanismes du jeu pour ensuite les implémenter. Heureusement nous ne sommes pas les premiers à créer une intelligence artificielle, il y a donc une documentation qui explique les fonctionnements du jeu. Néanmoins, nous avons eu beaucoup plus de difficulté à savoir comment implémenter l'intelligence artificielle. Il en existe de nombreux types. Nous avons pris beaucoup de temps à comprendre commet fonctionne le reinforcement learning que nous avons utilisé. Après avoir finalement réussi à implémenter l'intelligence artificielle, l'entraînement nous a donné du fils à retordre. Trouver les paramètres permettant de faire de bons scores est très compliqué. Nous n'avons malheureusement pas trouvé des paramètres qui ne nous font passer que le premier niveau au maximum.