

PAC-MAN


En mai 1980, la première borne d'arcade Pac-Man était installée au Japon. Le héros en forme de rond jaune se déplace à l'intérieur d'un labyrinthe. Il doit ramasser l'ensemble des Pac-gommes pour passer au niveau suivant. Cependant, pour corser l'affaire, quatre fantômes sont à sa poursuite : Shadow en rouge, Speedy en rose, Bashful en bleu et Pokey en orange. Vous pouvez trouver étrange que l'on baptise des fantômes, mais au Japon, cela est possible ! Si un des fantômes réussit à rattraper Pac-man, c'en est fini pour lui. Mais il y a une justice en ce bas monde. Quatre supers Pac-gommes sont réparties aux quatre coins du labyrinthe. Lorsque Pac-man en avale une, c'est le monde à l'envers : il peut enfin dévorer les fantômes, qui ayant compris ce qu'il se passe, se défilent à la vitesse grand V. Bien sûr, cet effet reste de courte durée et Pac-man devra rapidement redevenir un glouton surfant entre ses ennemis.



Le jeu a connu un succès immense et a été décliné dans différentes versions sur consoles et bornes d'arcade. Il fut une époque où chaque mois sortait un clone du jeu sur ordinateur, on parlait alors du Pac-Man du mois. Et oui, c'était l'époque de la Pac-mania ! Quelques liens pour présenter le jeu :

- PAC-MAN l'original : <https://www.youtube.com/watch?v=-CbyAk3Sn9I>
- <https://www.youtube.com/watch?v=XQicy0CfQuo>
- **ANDROID** : <https://play.google.com/store/apps/details?id=com.namcobandaigames.Pac-mantournaments>
- **APPLE** : <https://apps.apple.com/fr/app/id293778748>

Au centre du décor se trouve une sorte de maison où viennent ressusciter les fantômes dévorés. Une petite porte leur permet de sortir. Mais Pac-man n'a pas accès à cette zone.

Durant la partie, des bonus peuvent être ramassés comme par exemple  la cerise, la fraise, la pastèque, la pomme ou le raisin... Ils n'ont pas d'intérêt particulier dans le Gameplay sinon vous rapporter des points en plus et vous forcer à vous rapprocher du centre du décor.

Pour simplifier le codage de l'IA, nous allons mettre en place une version tour par tour. A chaque itération, les fantômes ainsi que Pac-man vont se déplacer d'une case. C'est un prérequis pour garder une animation fluide. Effectivement, dans le jeu original, les fantômes étaient un peu plus rapides que Pac-man. Mais rassurez-vous, cela ne vous empêchera pas d'avoir des sueurs froides avec cette version simplifiée !

Etape 1 : prendre en main le programme d'exemple

- Chargez et exécutez le programme PAC-MAN.py.
- Faites en sorte que Pac-man mange les Pac-gommes !
- Faites évoluer le score en fonction.

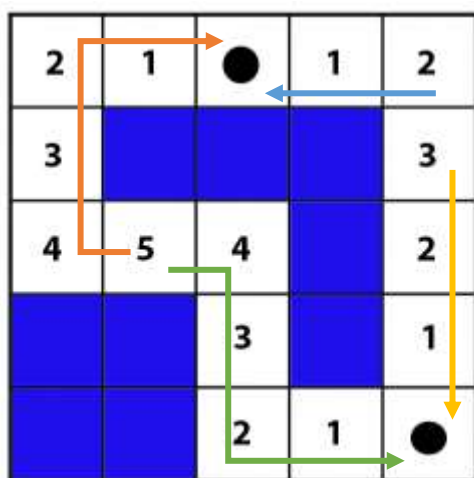
Etape 2 : donner une IA à PAC-MAN

Pour l'instant, la présence des fantômes est peu critique. L'objectif de Pac-man est centré uniquement sur avaler l'ensemble des Pac-gommes du décor. Ce n'est pas chose simple ! Pour cela, nous vous proposons de construire une carte des distances. Voici un exemple dans le schéma ci-dessous. Il y a deux Pac-gommes dans le décor. Nous ne faisons pas de différence entre les Pac-gommes standards et les super Pac-gommes à ce niveau. Les murs sont dessinés en bleu. Ce qui nous intéresse, ce sont les cases vides. Normalement, Pac-man est toujours disposé sur une case vide car il vient de dévorer la Pac-gomme de la case où il est arrivé.

2	1	●	1	2
3				3
4	5	4		2
		3		1
		2	1	●

L'idée à la fin de l'algorithme est que les valeurs dans chaque case correspondent à la distance (en cases parcourues) à la Pac-gomme la plus proche. Ainsi, lorsque l'on regarde la case du rond rouge, elle porte le chiffre 2 car si Pac-man se trouvait sur cette case, il serait exactement à deux cases de la Pac-gomme de droite. La case avec un rond-vert contient le chiffre 5. Chose amusante, elle se situe à égale distance des deux Pac-gommes. Ainsi que l'on parte à droite, ou à gauche, il faudra parcourir 5 cases à Pac-man pour arriver à destination. Lorsque Pac-man est sur une case, comment sait-il où aller ? En fait, s'il cherche à se diriger vers la Pac-gomme la plus proche, il suffit qu'il recherche sur les 4 cases avoisinantes (en haut, en bas, à gauche, à droite), la case contenant une valeur plus faible.

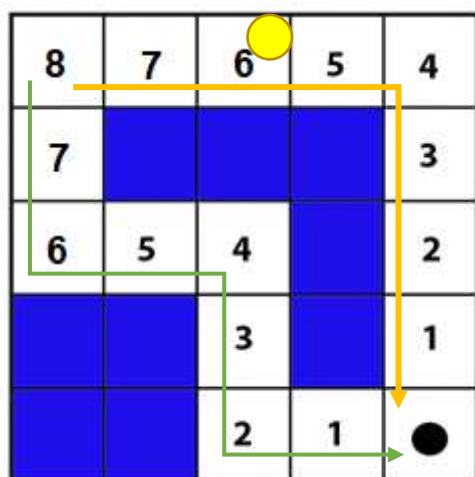
Cette case existe toujours et elle indique le chemin vers la destination. Voici en exemple le tracé des chemins suivant cette règle :



Ainsi, si Pac-man se trouve en haut à droite, il aura tendance à se déplacer vers la gauche pour attraper la Pac-gomme du haut. S'il se trouve sur la colonne de gauche, il a intérêt à aller vers le haut pour attraper la même Pac-gomme.

Mais, comment Pac-man sait-il où aller après avoir mangé la Pac-gomme en haut du décor, les cases avoisinantes lui indiquant de se diriger vers cette case. Une fois que la Pac-gomme a été dévoré, la carte des distances change. En effet, comme il n'y a plus de Pac-gomme dans ce secteur, les valeurs évoluent et

maintenant nous obtenons le tracé suivant :



Ainsi, lorsque Pac-man vient de dévorer la Pac-gomme en haut, les valeurs de la carte des distances évoluent dans la foulée. Maintenant, il doit pour se diriger le plus rapidement possible vers la nouvelle Pac-gomme suivre le parcours en orange.

Au début de la partie, cet algorithme a peu d'intérêt car il y a des Pac-gommes partout. Ainsi, quelle que soit la direction choisie par Pac-man, tout va bien se passer. Une méthode qui choisirait une direction au hasard à chaque croisement serait tout aussi efficace.

Cependant, une fois que 50%, 80% des Pac-gommes auront été mangées, les choses deviennent moins évidentes. Et lorsqu'il ne reste plus qu'une seule Pac-gomme dans le décor, si vous n'utilisez pas cette méthode ou une méthode équivalente, Pac-man risque de tourner en rond pendant un bon moment !

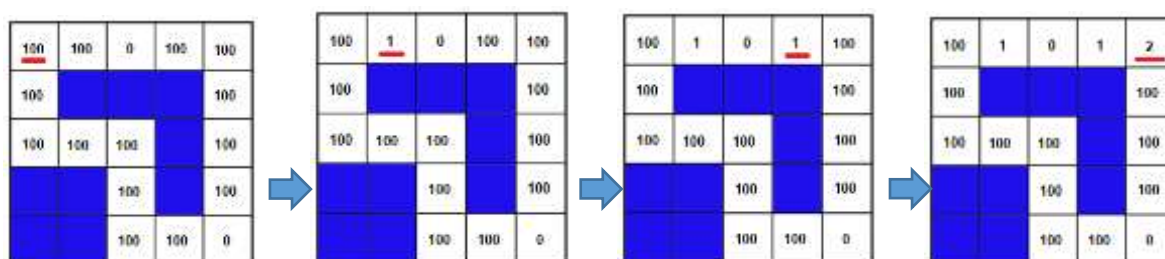
Il nous reste maintenant à présenter comment construire cette carte des distances. Nous allons en premier lieu créer une grille de la même taille que la grille de jeu. Les cases correspondantes aux murs seront initialisées avec une valeur infiniment grande. Les cases du parcours seront initialisées à une valeur correspondant au plus long parcours possible dans le décor, soit par exemple 100. Ensuite, les cases correspondantes à des Pac-gommes seront initialisées à la valeur 0.

Par la suite, nous allons parcourir l'ensemble des cases. Si une case correspond à un mur, nous l'ignorons. Si une case porte une valeur positive, alors nous prenons les valeurs des 4 cases avoisinantes. Nous calculons la valeur minimale de ces valeurs et nous mettons à jour la valeur de la case courante avec cette valeur minimale + 1 **uniquement si cela diminue la valeur de la case courante**. La logique est la suivante :

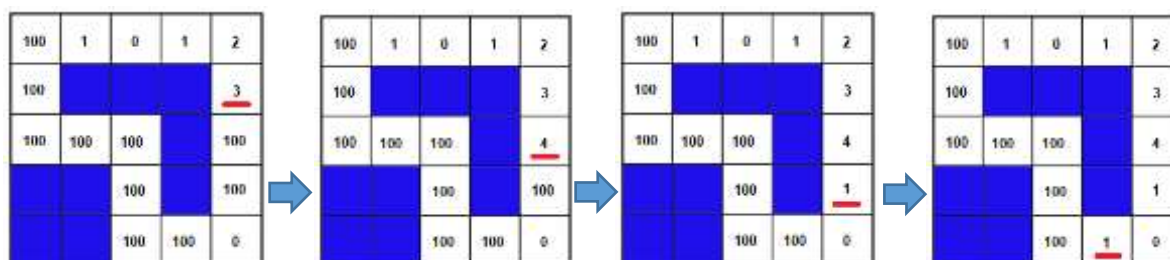
- Si je suis sur une case de valeur 0 entourée par deux cases de valeurs 100, la valeur de la case courante ne passe pas à 101. Idem pour une case de valeur 100 entourée de plusieurs cases à la valeur 100, sa valeur ne passe pas à 101 car cela augmenterait sa valeur actuelle.
- Si la case à ma gauche est à 3 cases d'une Pac-gomme, que la case à ma droite est à 6 cases d'une autre Pac-gomme et que les cases en haut et en bas sont des murs, alors j'en conclus que la pac gomme la plus-proche de ma case se situe à $3 + 1$ cases.

Pour traiter la grille, nous allons parcourir les cases de gauche à droite puis de haut en bas.

La valeur contenue dans le coin en haut à gauche ne change pas car les valeurs des cases voisines sont à 100. La deuxième case en partant de la gauche porte la valeur 100 et elle est entourée par les valeurs 0 et 100, elle passe donc à la valeur 1. La troisième case vaut 0, sa valeur ne peut changer. La 4^{ème} case vaut 100 et elle est entourée par les valeurs 0 et 100, sa valeur passe donc à 1. La dernière case de cette ligne porte la valeur 100, elle est entourée par les valeurs 1 et 100, sa valeur passe donc à 2.



Nous continuons à la deuxième ligne. La valeur 100 reste inchangée car les deux cases avoisinantes sont à la valeur 100. La case tout à droite est entourée par les valeurs 2 et 100, sa valeur passe à 3. Comme nous parcourons le tableau de gauche à droite et de haut en bas, les modifications ont tendance à se reporter plus facilement dans ces directions. La prochaine case modifiée est celle sous la valeur 3 qui passe alors à la valeur 4. La case se trouvant sous la valeur 4, se trouve maintenant entourée par les valeurs 0 et 4. Ainsi, d'après nos règles de transformation, sa valeur passe à 1. La dernière valeur modifiée est celle de la case à gauche de la valeur 0. Elle est entourée par les valeurs 0 et 100, elle passe donc à la valeur 1. Il s'agit de la dernière mise à jour effectuée lors de ce premier parcours de grille.



Il est évident que ce calcul n'est pas terminé. Nous allons donc relancer un nouveau parcours.

Cette fois la case en haut à gauche est mise à jour pour la valeur 2. Il n'y aura pas d'autres modifications sur cette ligne. La case en dessous passe ainsi à la valeur 3. La valeur 3 en fin de ligne reste inchangée. La case en dessous du 3 dans la colonne de gauche passe à la valeur 4 et les cases immédiatement à sa droite se modifient pour les valeurs 5 et 6 :

2	1	0	1	2
100				3
100	100	100		4
		100		1
		100	1	0

→

2	1	0	1	2
3				3
100	100	100		4
		100		1
		100	1	0

→

2	1	0	1	2
3				3
4	100	100		4
		100		1
		100	1	0

→

2	1	0	1	2
3				3
4	5	6		4
		100		1
		100	1	0

La valeur 4 toute à droite de cette ligne est entourée par un 3 et par un 1. Sa valeur est donc mise à jour pour la valeur 2. A la ligne suivante, la valeur 100 est mise à jour pour la valeur 7. La valeur 1 en fin de ligne ne change pas. A la dernière ligne, la valeur 100 entourée par les valeurs 7 et 1 est alors mise à jour pour la valeur 2.

2	1	0	1	2
3				3
4	5	6		2
		100		1
		100	1	0

→

2	1	0	1	2
3				3
4	5	6		2
		7		1
		100	1	0

→

2	1	0	1	2
3				3
4	5	6		2
		7		1
		2	1	0

La deuxième passe est terminée. Depuis le début du traitement, toutes les cases ont été mises à jour. Pourtant, avons-nous terminé ? Rappelez-vous de la case sur la colonne de droite à la valeur 4, qui finalement a été modifiée une deuxième fois pour prendre la valeur 2 à cause d'une valeur inférieure arrivant par le bas. Ainsi, le tableau est en évolution permanente et des cases peuvent être mises à jour plusieurs fois tant que des meilleurs trajets se forment. Dans la grille actuelle, par exemple, la case portant la valeur 7 n'est pas optimale car située à côté d'une case de valeur 2. Cette valeur 2 va se propager pour créer un 3 et puis un 4. Mais alors, comment savoir à quel moment le traitement s'arrête ?

Critère d'arrêt : lors du traitement complet d'une grille, si aucune case n'est mise à jour, alors les valeurs ont fini de converger, elles correspondent aux valeurs de la carte des distances.

2	1	0	1	2
3				3
4	5	6		2
		3		1
		2	1	0

Nous enclenchons donc notre 3^{ème} parcours de grille.

La valeur 7 est la seule valeur mise à jour. La case correspondante prend alors la valeur 3.

Suivant le critère d'arrêt énoncé, une modification a été effectuée, nous relançons donc le processus.

2	1	0	1	2
3				3
4	5	4		2
		3		1
		2	1	0

Nous enclenchons donc notre 4^{ème} parcours de grille.

La valeur 6 est la seule valeur mise à jour pour la valeur 4.

Suivant le critère d'arrêt énoncé, une modification a été effectuée, nous relançons le processus. Cette fois, nous n'avons aucune modification. Cette itération supplémentaire, qui peut sembler inutile, sert à nous confirmer que nous avons convergé vers la solution. Les valeurs trouvées correspondent à la carte des distances.

A des fins de vérification et de debug, modifiez la fonction d'affichage pour que dans chaque case soit affichée la distance à la Pac-gomme la plus proche. Voici un exemple d'affichage :

Pour cela, la carte des distances peut être mise en variable globale.



Astuce : Devons-nous gérer les murs à part ? Oui, car techniquement les murs n'interviennent pas dans le calcul. Cependant, cela génère des tests et des lignes de codes supplémentaires. Lorsqu'on écrit un algorithme, il faut faire en sorte que la boucle de traitement puisse aussi traiter/absorber les cas particuliers. Dans le cas contraire, on peut avoir une explosion de cas particuliers devant être traité à part et cela rajoute du code long et compliqué. Notre traitement consiste à examiner les quatre cases voisines et à prendre la valeur minimale. Comment faire en sorte que les murs soient traités comme une case standard ? Par exemple, si l'on choisit une valeur très grande pour les murs, comme 999 par exemple, sachant que les distances sont initialisées à 100, cela sous-entend qu'on ne retiendra jamais un mur comme une direction possible car la fonction « min » va filtrer naturellement les murs. Le seul cas particulier qui pourrait exister serait celui où PacMan se retrouve entouré par 4 murs, mais si l'on regarde le labyrinthe, cela n'arrive pas !

D'une manière générale, il faut éviter de créer du code avec des imbrications complexes de if et de elseif. Le code généré devient difficile à maintenir et détecter une erreur semble impossible.

Etape 3 : fantômes

Nous allons donner une IA de déplacement assez basique aux fantômes. Chaque fantôme va avoir une direction courante : haut, bas, gauche, droite. Dans un couloir, chaque fantôme avance dans sa direction courante. Cependant, dès lors qu'il arrive à un tournant ou à un croisement, les différentes directions possibles sont analysées et une direction est tirée au hasard.

Pour déterminer si Pac-man se trouve sur une case correspondant à un tournant ou à un croisement, on peut par exemple vérifier si nous sommes dans un couloir. Un couloir correspond à une des deux configurations suivantes :

- Un mur en haut et en bas et une case de déplacement à gauche et à droite.
- Un mur à gauche et à droite et une case de déplacement en haut et en bas

Détectez la collision entre un fantôme et Pac-man. Même si les IA des fantômes sont faibles, ils sont assez nombreux pour lui barrer la route et l'obliger à réagir. Ce test doit être fait à la fin du tour, après que Pac-man et chacun des fantômes se soient déplacés.

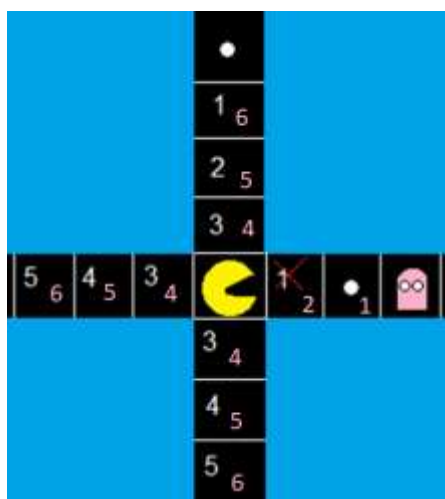
Etape 4 : éviter les fantômes



Comme pour les Pac-gommes, nous allons appliquer la même logique pour calculer une carte des distances correspondant à la proximité des fantômes.

Pour cela, dans un nouveau tableau, nous initialisons notre calcul en mettant à zéro les cases où se trouvent les fantômes et nous construisons itérativement la carte des distances. Quelle est la signification exacte des valeurs contenues dans cette carte ?

Comme précédemment, nous vous demandons de modifier l'affichage afin de fournir pour chaque case les valeurs des deux cartes de distance comme l'exemple ci-contre.



Nous vous proposons la stratégie suivante. Si Pac-man se trouve sur une case à une distance des fantômes > 3 , alors il passe en mode « chasse aux Pac-gommes ». Il choisit alors parmi les cases avoisinantes la plus proche des Pac-gommes.

Si la case où se trouve Pac-man est à une distance inférieure ou égale à 3 des fantômes, alors il passe en stratégie « évitement / fuite ». Dans ce cas, nous allons uniquement considérer les distances aux fantômes et choisir une case qui nous éloigne le plus d'eux.

On pourrait considérer les cases adjacentes à Pac-man ayant une distance aux fantômes > 3 et choisir parmi ces dernières la plus proche d'une Pac-gomme. Cette logique semble juste.

Cependant, cette approche n'est pas correcte. En effet, si un seul fantôme se trouve sur une case adjacente à Pac-man, toutes les cases adjacentes à Pac-man ont une proximité à ce fantôme égale à 2 cases. Pac-man n'aurait donc aucune case pour se déplacer alors qu'il lui reste des options de fuite.

Astuce : vous devez coder deux choses d'une part la logique qui décide dans quel mode se trouve PacMan (normal, fuite, chasse...) et d'autre part les actions à effectuer (les déplacements) en fonction du mode de PacMan. Il est sage de séparer le traitement de la logique du traitement des actions dans le code. Ainsi, on doit trouver quelques lignes qui décident dans quel mode se trouve PacMan et ensuite on trouve les différentes actions à effectuer en fonction du mode de PacMan. Mélanger les lignes de code de la logique et les lignes de code gérant les actions est possible, mais le code associé devient long et difficile à lire. Rajoutez un nouveau mode ou une nouvelle action dans un tel code devient vite un casse-tête.

Etape 5 : Les super Pac-gommes

Positionnez 4 super Pac-gommes dans les coins du labyrinthe. Faites en sorte que leur taille soit plus grosse que les Pac-gommes standards, modifiez l'affichage en conséquence.

Lorsque Pac-man mange une super Pac-gomme, il passe alors en mode « chasse aux fantômes » pendant les 16 affichages suivants. Dans ce mode, Pac-man choisit comme déplacement une case

avoisinante ayant la plus faible distance aux fantômes. Lorsque que Pac-man arrive sur la case d'un fantôme, augmentez le score en conséquence et téléportez le fantôme au centre du décor. On ne gère pas d'attente particulière, le fantôme mangé peut se déplacer immédiatement.

PROJET

Le document : <http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior> (mis en copie dans le répertoire d'exercices) décrit les IA des fantômes utilisées dans le jeu original. Il serait intéressant et c'est à votre portée de mettre en place les 4 IA des différents fantômes. En effet, chacun d'entre eux avait une personnalité propre. Le rouge était plus combatif et avait tendance à foncer vers Pac-man, un autre essayait d'arriver à revers, un autre était plutôt craintif....

Une fois les personnalités des fantômes établis, nous vous conseillons d'améliorer l'IA existante pour faire en sorte d'optimiser votre score.