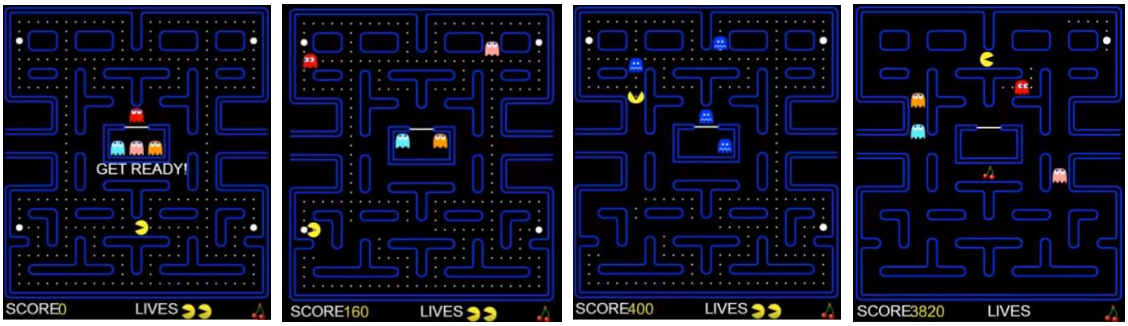
PAC-MAN

I. Il était une fois…

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 :

- Le jeu PAC-MAN original
- Finir le niveau 1 en 1min30
- La version Android sur le playStore
- La version Apple sur l'AppleStore

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 apparaître aléatoirement sous la maison des fantômes comme par exemple la cerise, la fraise, la pastèque, la pomme ou encore le raisin. Chacun bonus ramassé rapporte des points. Ces bonus n'ont pas d'intérêt particulier dans le Gameplay sinon vous forcer à vous rapprocher du centre du décors.

II. Architecture du projet

Chargez et exécutez le programme Pac-Man.py. Comme vous pouvez le remarquez, le déplacement des fantômes et de Pac-Man sont totalement aléatoires. Les Pac-gommes ne sont pas avalées et les collisions entre Pac-Man et les fantômes sont sans effet ! Dans le projet, nous utilisons une version tour par tour du jeu. A chaque itération, les fantômes ainsi que Pac-Man se déplacent d'une case. 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 !

Le code est divisé en 3 parties :

- Partie 1 : l'initialisation des données du jeu, avec notamment la mise en place du décor, le positionnement de Pac-Man et des fantômes. Vous devrez ajouter du code dans cette partie.
- Partie 2 : l'affichage, vous ne devez pas modifier cette partie.
- Partie 3 : le gestionnaire de partie. A terme, dans cette partie, vous allez gérer les différents aspects de l'IA de Pac-Man :
    - manger les Pac-gommes
    - fuir les fantômes lorsqu'ils s'approchent trop près
    - chasser les fantômes après avoir mangé une super Pac-gomme.

  Evidemment, c'est dans cette troisième partie que vous allez ajouter l'essentiel du code.

Le point d'entrée du programme est la fonction PlayOneTurn() appelée toutes les 400 ms par le système pour effectuer un tour de jeu et mettre à jour l'affichage. A chaque appel de cette fonction, on a alternativement :

- Pac-Man bouge grâce à la fonction IAPacMan()
- Les fantômes se déplacent grâce à la fonction IAGhosts()

Afin, de vous familiariser avec le code, apportez les deux modifications suivantes dans le jeu :

- Faîtes en sorte que Pac-Man mange les Pac-gommes !
- Faîtes évoluer le score de 100 points par Pac-gommes avalées, affichez ce score à l'écran.

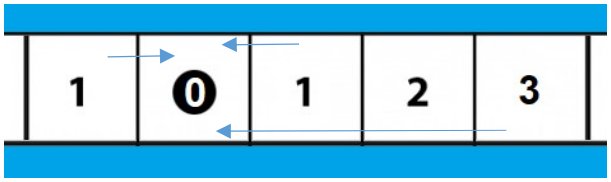
III. Donner une IA à Pac-Man

Dans cette partie, nous mettons en place l'IA de Pac-Man pour qu'il arrive à manger toutes les Pac-gommes. Nous avons pu remarquer qu'en déplaçant Pac-Man aléatoirement, il est peu probable qu'il arrive à manger toutes les Pac-gommes. Ainsi, construire une IA pour Pacman qui arrive à le guider n'est pas évident. Pour cela, nous allons introduire le concept suivant :

1. La notion de plus court chemin vers un objectif

Nous allons mettre en place un algorithme permettant à Pac-Man de prendre un chemin l'amenant vers la Pac-gomme la plus proche. Pour cela, nous cherchons pour chaque case du décor la longueur du chemin nécessaire pour aller de cette case jusqu'à la Pac-gomme la plus proche. La distance parcourue est comptée en nombre de cases traversées.

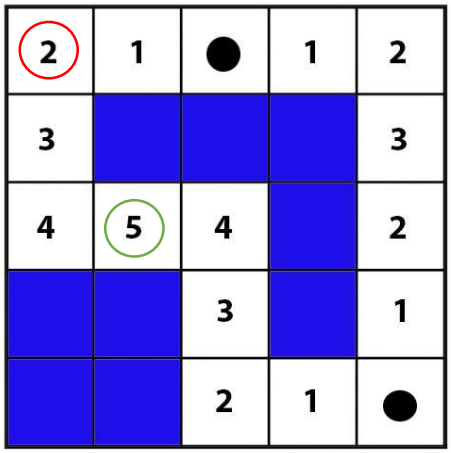
Prenons l'exemple d'un couloir entouré de murs et contenant une seule Pac-gomme. Nous calculons depuis chaque case du couloir le nombre de cases à traverser pour atteindre cette Pac-gomme :



Comment lire cet exemple ? La case où se trouve la Pac-gomme contient la valeur 0 car nous n'avons pas à nous déplacer pour l'atteindre car nous sommes déjà sur place. Les cases à gauche et à droite de la Pac-gomme contiennent la valeur 1 car il faut parcourir une seule case pour arriver à cette Pac-gomme. La valeur 3 dans la case de droite signifie que le chemin arrivant à la Pac-gomme traverse 3 cases.

2. La carte des distances associée aux Pac-gommes

Examinons une configuration plus complexe où deux Pac-gommes sont présentes dans le décor. Nous allons calculer une **carte des distances** où chaque case contient la distance minimale à une Pac-gomme du décor. Voici un exemple :

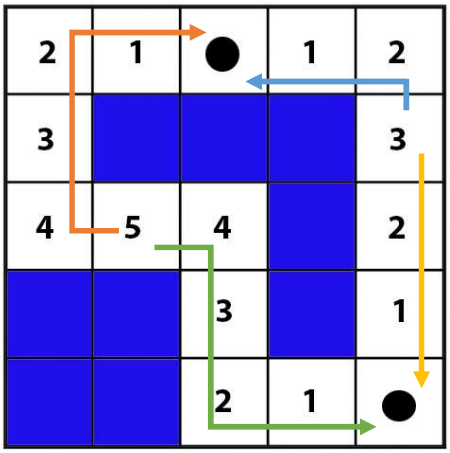


Lorsqu'on examine la case contenant le rond rouge, la distance de cette case à la Pac-gomme du haut vaut 2 cases et la distance de cette case à la Pac-gomme de droite est de 8 cases. La valeur retenue sera donc le minimum entre 2 et 8, c'est-à-dire 2. Si l'on prend la case centrale, sa distance avec la Pac-gomme de droite est de 4 cases et sa distance avec la Pac-gomme du haut est de 6 cases (pensez à prendre le chemin le plus court). La valeur associée est donc 4 = *min*(4,6). La case avec un rond vert contient le chiffre 5 car elle se situe à égale distance des deux Pac-gommes : que l'on parte à droite, ou à gauche, il faudra parcourir 5 cases pour arriver aux deux Pac-gommes.

3. Comment Pac-Man se déplace-t-il ?

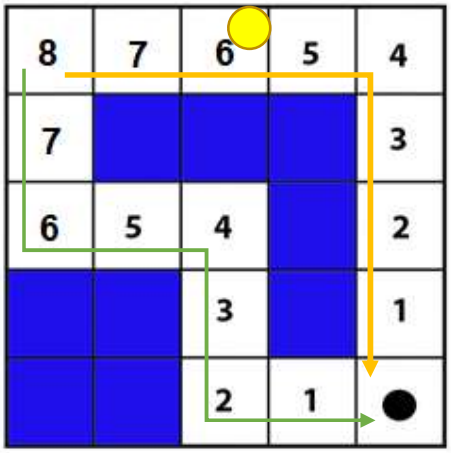
Lorsque Pac-Man se trouve sur une case, comment son IA va-t-elle savoir où aller ? En fait, l'IA se dirige vers la Pac-gomme la plus proche. Pour cela, elle examine les 4 cases avoisinantes (en haut, en

bas, à gauche et à droite) et la valeur la plus faible indique sa prochaine destination. Voici un exemple des déplacements de Pac-Man lorsqu'il suit cette règle :



Si Pac-Man se trouve en haut à droite, il va se déplacer vers la gauche pour attraper la Pac-gomme du haut (parcours bleu). S'il se trouve sur les cases tout à gauche, il va choisir d'aller vers le haut pour attraper la même Pac-gomme (parcours orange). Le raisonnement est identique pour les autres cases. Chaque flèche indique le chemin emprunté par Pac-Man pour se rendre à la Pac-gomme la plus proche. Certaines cases à égale distance de plusieurs Pac-gommes peuvent contenir plusieurs départs de chemin. Choisir l'un ou l'autre chemin est alors équivalent.

Lorsque Pac-Man avale la Pac-gomme du haut, la carte des distances se trouve alors modifiée et il faut la recalculer entièrement. Voici les nouvelles valeurs obtenues :



Maintenant, pour se diriger le plus rapidement possible vers la dernière Pac-gomme ; Pacman doit suivre le parcours en orange.

Remarque : au début de la partie, cet algorithme semble peu utile car il y a des Pac-gommes partout. Ainsi, quelle que soit la direction choisie par Pac-Man, il va avaler des Pac-gommes en grande quantité. Cependant, dès qu'il ne reste plus qu'une dizaine de Pac-gommes dans le labyrinthe, la carte des distances s'avère cruciale pour mettre en place une IA amènant efficacement Pac-Man vers les Pac-gommes restantes.

4. Critère de mise à jour

Nous abordons maintenant le problème de calcul de la carte des distances. La carte et ses valeurs seront utilisées par la fonction IAPacman(). Tout d'abord, nous créons une grille de la même taille que la grille de jeu et nous la remplissons en respectant les consignes suivantes :

- Les cases correspondantes aux murs sont initialisées avec une valeur G très grande.
- Les cases du parcours sont initialisées à une valeur M correspondant au nombre de cases présente dans le labyrinthe (sa surface totale). Cette valeur reste plus grande que les valeurs des plus courts chemins et reste inférieur à G.
- Les cases où sont présentes des Pac-gommes sont mises à la valeur 0.

Pour construire la carte des distances, nous allons mettre en place un critère permettant de déduire si une case contient une valeur pertinente par rapport à ses cases voisines ou si sa valeur doit être mise à jour. Pour s'échauffer, prenons l'exemple d'une case entourée de deux cases à gauche et à droite :

- Si les trois cases contiennent les valeurs 5 | 6 | 3 ; nous savons que la case de gauche se trouve à 5 cases d'une Pac-gomme et que la case de droite se situe à 3 cases d'une Pac-gomme. La case centrale contient la valeur 6, apparemment, la distance actuelle correspond au chemin de gauche auquel on a ajouté 1 case. Cependant, depuis la case centrale, si nous décidions de partir sur la droite, nous serions à 3+1=4 cases d'une autre Pac-gomme. Nous avons ainsi trouvé une meilleure option par la droite ; nous mettons donc à jour la valeur 6 et la remplaçons par la valeur 4. Nous obtenons ainsi comme configuration finale : 5 | 4 | 3.

Ainsi, nous avons un critère de mise à jour pour la case courante :

- Nous examinons les valeurs des cases adjacentes.
- Nous prenons la valeur minimale de ces cases et lui ajoutons 1. On obtient ainsi la longueur du meilleur chemin possible en empruntant une de ces 4 cases.
- Si la valeur calculée est meilleure que la valeur de la case courante, alors on la met à jour.

Astuce 1 : comment traiter les cases se trouvant sur le bord du décor. Il faut en effet faire attention, car en regardant les cases voisines, on va sortir du tableau. Si vous examinez le labyrinthe, vous vous apercevez que Pac-Man ne peut pas se trouver sur les bords car des murs sont présents. Ainsi, cette astuce permet d'éviter la gestion de cas particulier.

Astuce 2 : lorsque nous examinons les valeurs des cases adjacentes, comment devons-nous gérer la présence des murs ? Comme nous avons associé les murs à une valeur G très grande, 1000 par exemple, lors du calcul du minimum des 4 voisins, cette valeur ne sera jamais retournée par la fonction min, cela équivaut à ignorer les murs. Il en est de même pour les cases de valeur M.

Astuce 3 : comment être sûr qu'une case non calculée (de valeur M) va se mettre à jour lorsque l'on aura la bonne information ? Dès qu'une des 4 cases voisines aura une valeur inférieure à M, donc une distance connue par rapport à une Pac-gomme, le critère de mise à jour va automatiquement retourner une valeur inférieure à M, entraînant la mise à jour de la valeur de cette case.

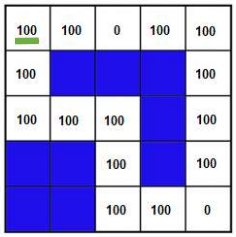
5. Balayage de la carte des distances

Un balayage de la carte des distances consiste à parcourir la totalité des cases de la grille, de gauche à droite et de haut en bas, en appliquant le critère de mise à jour sur chaque case ne correspondant pas à un mur.

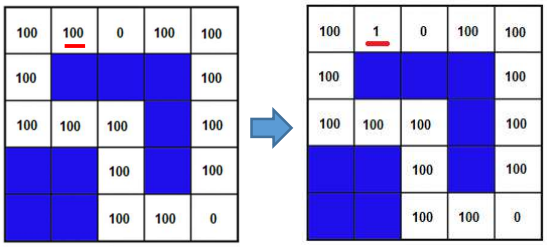
Première mise à jour

Prenons un exemple. Nous partons d'une grille venant d'être initialisée : les cases contenant une Pac-gomme sont à la valeur 0, les cases correspondant aux couloirs ont une valeur à 100 et les murs sont mis à 1000 (non indiqués sur le schéma). Nous allons effectuer une mise à jour de la carte de gauche à droite et de haut en bas :

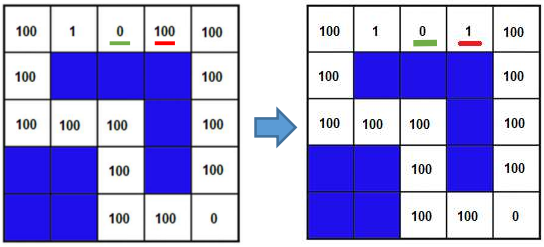
Pour commencer, nous allons successivement traiter les cases de la première ligne de la grille :



Nous partons de la case en haut à gauche. La valeur contenue dans cette case ne change pas car ses deux cases voisines contiennent la valeur 100.

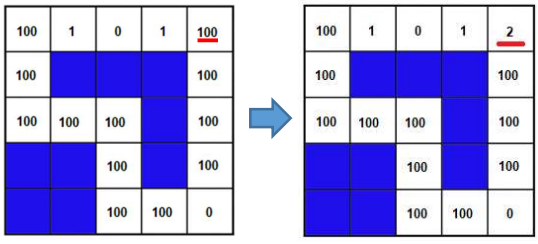


La 2ème case en partant de la gauche porte aussi la valeur 100 mais elle est entourée par les valeurs 0 et 100. D'après le critère de mise à jour, elle passe donc à la valeur 1.



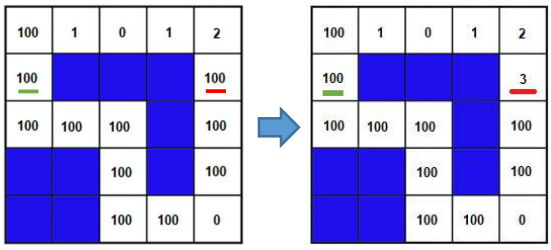
La 3ème case vaut 0, sa valeur ne va pas changer.

La 4ième case vaut 100 et elle est entourée par les valeurs 0 et 100, d'après le critère de mise à jour, sa valeur passe donc à 1.

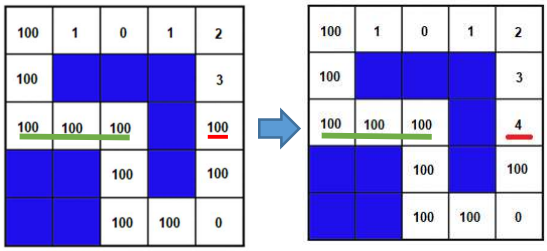


La dernière case de cette ligne porte la valeur 100, elle est entourée par les valeurs 1 et 100, sa valeur passe donc à 2.

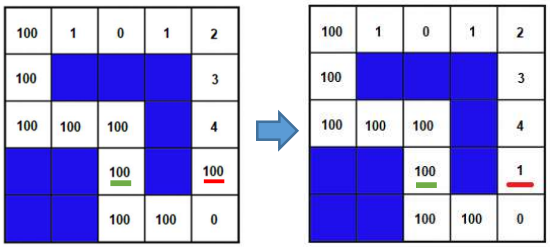
Nous traitons maintenant la deuxième ligne.



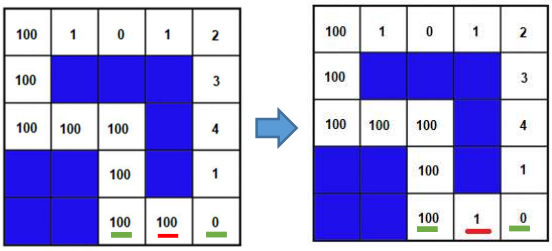
La valeur tout à gauche reste inchangée car les deux cases avoisinantes sont à la valeur 100. La case tout à droite est entourée par les valeurs 2 et 100, sa valeur passe donc à 3.



Les trois premières cases de la 3ième ligne restent inchangées car entourées de valeurs égales à 100. La case tout à droite valant 100 est modifiée car elle est entourée par les valeurs 3 et 100, elle passe donc à la valeur 4.



La première case de la 4ième ligne contient la valeur 100, elle est entourée par des cases avec la valeur à 100, sa valeur reste donc inchangée. La case tout à droite avec la valeur 100 est entourée par les valeurs 0 et 4, elle passe donc à la valeur 1.

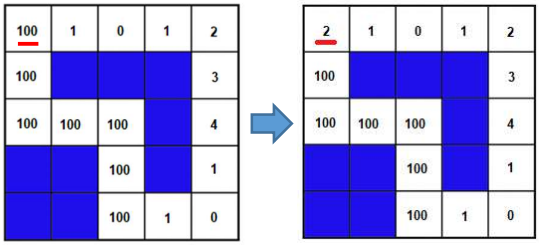


Nous traitons maintenant la dernière ligne. La première case à gauche vaut 100, étant entourée par les valeurs 100 et 100, sa valeur ne change pas. La deuxième case contient la valeur 100 mais elle est entourée par les valeurs 100 et 0, sa valeur passe donc à 1. La dernière case contient la valeur 0, elle reste donc inchangée.

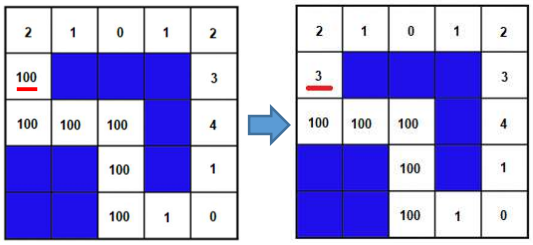
Le premier balayage de la carte des distances est terminé. Comme nous parcourons le tableau de gauche à droite et de haut en bas, nous remarquons que les modifications ont tendance à se propager plus facilement dans ces directions. Il est évident que le traitement n'est pas terminé. Nous allons donc relancer un nouveau balayage en repartant en haut à gauche du tableau.

Deuxième balayage

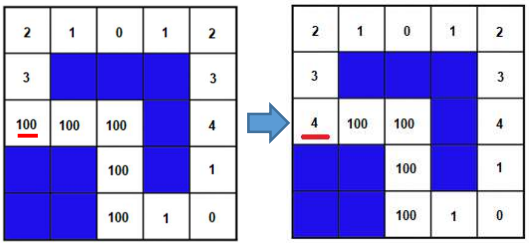
Cette fois ci, nous indiquons seulement les cases mises à jour.



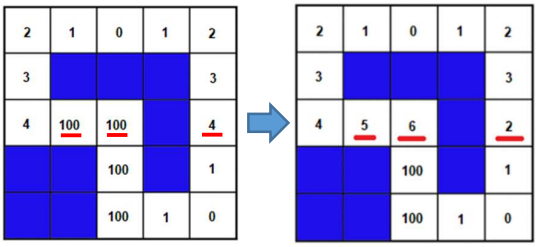
En traitant la première ligne, la case en haut à gauche contenant la valeur 100 est mise à jour pour la valeur 2. Il n'y aura pas d'autres modifications sur cette ligne.



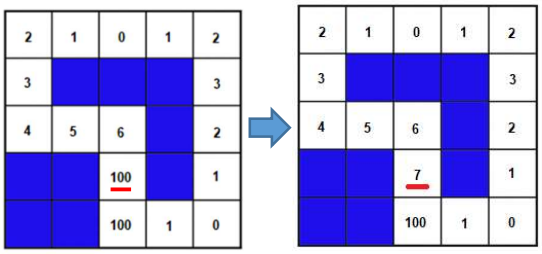
En traitant la 2ième ligne, la première case est mise à jour pour la valeur 3 car elle est à proximité d'une case valant 2. La case sur la droite n'est pas modifiée.



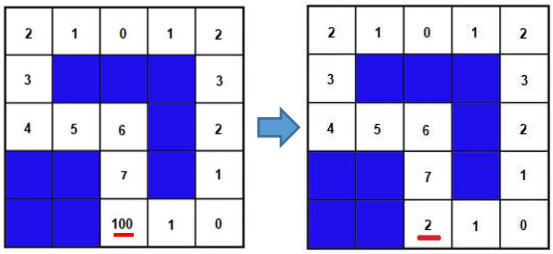
En traitant la 3ième ligne, la première case valant 100 est mise à jour pour la valeur 4, car elle est à proximité d'une case valant 3.



Les deux cases suivantes valant 100, elles passent successivement aux valeurs 5 et 6. La case tout à droite est mise à la valeur 2 car entourée par les valeurs 3 et 1.



En traitant la 4ième ligne, la première case avec la valeur 100, est mise à jour pour la valeur 7 car elle est entourée par les valeurs 6 et 100. La case tout à droite n'est pas modifiée.



Sur la dernière ligne, la valeur 100 tout à gauche est mise à jour pour la valeur 2 car elle est entourée par la valeur 7 et 1. Les autres cases restent inchangées.

La deuxième mise à jour est terminée : toutes les cases ont été traitées. Pourtant, avons-nous terminé ? Non, car dans la grille actuelle, la case portant la valeur 7 n'est pas optimale car située à côté d'une case de valeur 2.

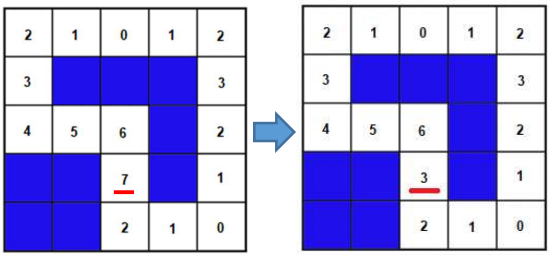
6. Critère d'arrêt

Le tableau est en évolution permanente et des cases peuvent être encore mises à jour plusieurs fois tant que des meilleurs trajets sont trouvés. Mais alors ? Comment savoir à quel moment le traitement s'arrête ? Après la fin d'un balayage, nous vous proposons le critère d'arrêt suivant :

A la fin d'un balayage de la carte des distances, si aucune case n'a vu sa valeur évoluer, alors les valeurs sont stables et nous avons terminé la construction de la carte des distances.

Astuce : pour la mise en place de ce critère, on peut initialiser un booléen à faux au départ d'un nouveau balayage. Dès que la valeur d'une case est mise à jour, ce booléen passe à vrai. Ainsi, ce booléen permet d'indiquer si une valeur a été modifiée durant le parcours.

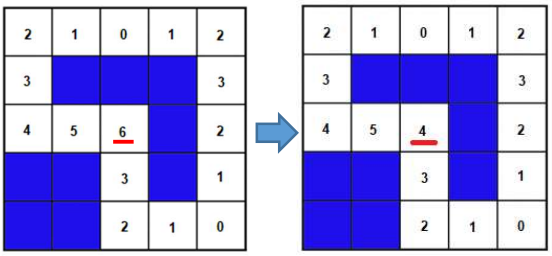
Lors du 2ème balayage, des modifications ont été effectuées, d'après ce critères, nous devons relancer un nouveau balayage du tableau. Nous enclenchons donc notre 3ième balayage de la grille.



La valeur 7 est la seule valeur mise à jour.

La case correspondante prend alors la valeur 3.

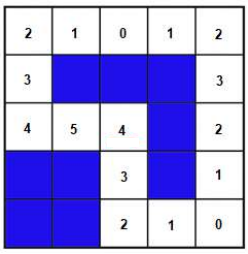
Suivant le critère d'arrêt énoncé, une modification a été effectuée durant ce balayage, nous relançons donc un nouveau balayage !



Nous enclenchons le 4ième balayage.

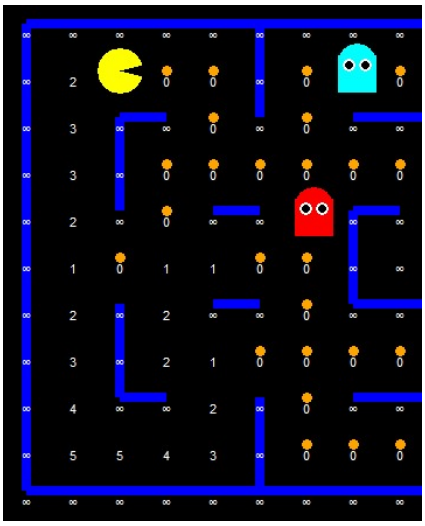
La valeur 6 est la seule valeur mise à jour pour la valeur 4.

Suivant le critère d'arrêt énoncé, une modification a été effectuée, nous relançons donc un 5ième balayage !



Cette fois, nous n'avons aucune modification. Cette itération supplémentaire, qui peut sembler inutile, sert cependant à confirmer que nous avons obtenu des valeurs optimales pour la carte des distances.

7. Debug



Une fois la carte des distances calculées, affichez son contenu en utilisant la fonction ***SetInfo1****(x,y,v)*. Vous pouvez ainsi associer à chaque case *(x,y)* une information *v* indiquant la distance actuelle ou un message en utilisant une chaîne de caractères ou encore n'effectuer aucun affichage en passant une chaîne vide.

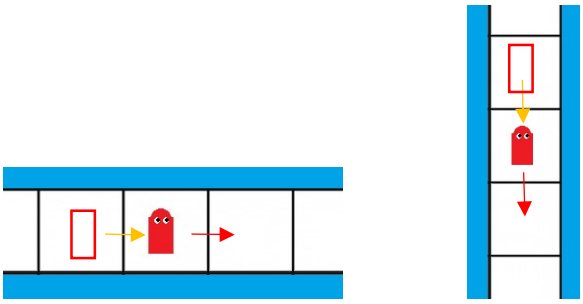
IV. Les fantômes

1. Stratégie de déplacement

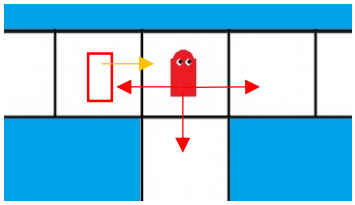
Nous allons donner une IA de déplacement aux fantômes. Cette partie du code doit être développée dans la fonction IAGhosts(). Tout d'abord, vous allez associer à chaque fantôme un paramètre supplémentaire en plus de sa position et de son nom : sa direction courante (haut, bas, gauche ou droite). Cette information est nécessaire pour construire l'IA.

Lorsqu'il parcourt le labyrinthe, un fantôme peut adopter deux comportements :

- Soit il se trouve dans un couloir et il continue d'avancer dans sa direction courante. Nous montrons deux exemples. Dans les schémas ci-dessous, l'ancienne position est indiquée par un rectangle, la flèche orange indique le déplacement et la flèche rouge le futur déplacement.



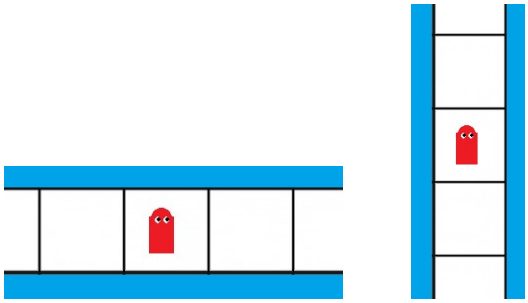
- Soit le fantôme arrive à un tournant ou à un croisement. Dans ces deux configurations, on choisit au hasard sa nouvelle direction parmi les différentes directions possibles :



2. Détection d'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



3. Gestion correcte des collisions

Pour détecter, la collision entre les fantômes et Pac-Man, vous devez faire un test :

- Après le déplacement de Pac-Man dans la fonction IAPacman().
- Après le déplacement de chaque fantôme dans la fonction IAGhosts().

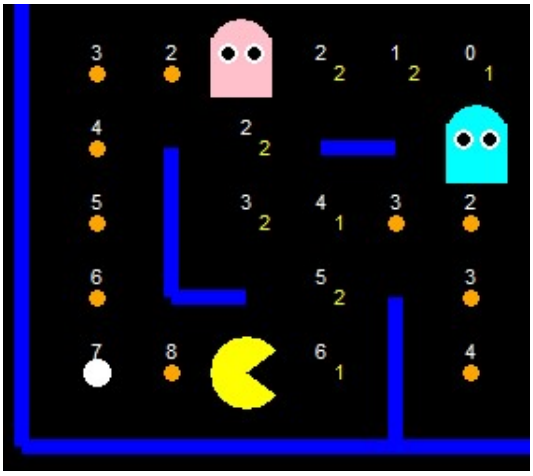
Dans le cas contraire, si par exemple vous ne testez que la collision après le déplacement des fantômes, des croisements entre Pac-Man et les fantômes vont se produire. Voici un exemple ci-dessous :

- Sur la première image, Pac-Man va vers la droite. A ce niveau, le test de collision a été oublié, même si Pac-Man se retrouve sur la même case qu'un fantôme, aucune collision n'est détectée.
- A la deuxième image, c'est au tour du fantôme de bouger, il se déplace vers la gauche. Le fantôme s'est déplacé sur une case vide, la détection de collision ne détecte aucun problème.

Lorsque qu'une collision se produit, vous devez stopper la partie : Pac-Man ainsi que les fantômes ne bougent plus, cependant, l'écran de jeu doit rester affiché.

V. Eviter les fantômes

1. Construction d'une carte des distances des fantômes

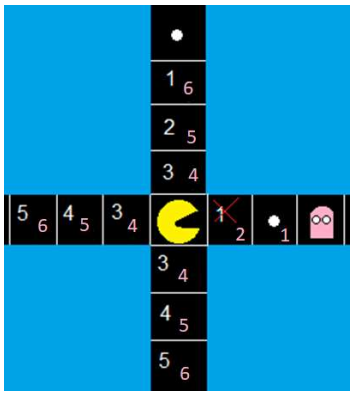


Comme pour les Pac-gommes, nous allons appliquer la même logique pour calculer une carte des distances correspondant à la proximité des fantômes.

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 ?

Nous vous demandons de modifier l'affichage afin de fournir pour chaque case les valeurs de la carte des distances aux fantômes. Pour cela, vous devez utiliser la fonction *SetInfo2(x,y,v)* au niveau de la fonction IAPacman(). Vous pouvez ajouter des tests pour ne pas afficher les valeurs extrêmes.

2. IA de déplacement de Pac-Man



Pour construire l'IA de déplacement de Pac-Man, nous vous proposons la politique suivante :

- Si Pac-Man se trouve sur une case à une distance > 3 des fantômes, alors il passe en mode « recherche des Pac-gommes ». Il choisit ainsi parmi les cases avoisinantes la plus proche d'une Pac-gomme.
- Si la case où se trouve Pac-Man se trouve à 3 cases ou moins d'un fantôme, alors il passe en mode « fuite ». Dans ce cas, nous allons uniquement considérer la carte de distances des fantômes et choisir une case nous permettant de nous éloigner d'eux si possible.

Remarque : pour construire le comportement de fuite, certains pourraient sélectionner parmi les cases adjacentes à Pac-Man une case ayant une distance aux fantômes ≥ 3. Cette logique semble sensée. Cependant, cette approche n'est pas correcte. En effet, si un fantôme se trouve sur une case adjacente à Pac-Man, les cases autours de Pac-Man ont une distance à ce fantôme égale à 2. Dans cette configuration, Pac-Man ne pourrait sélectionner aucune case pour fuir alors que des options existent :

VI. Les super Pac-gommes

Nous considérons maintenant que 4 super Pac-gommes se trouvent dans les coins du labyrinthe. 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 adjacente ayant la plus faible distance à un fantôme. Lorsque que Pac-Man arrive sur la case d'un fantôme, il le dévore. Augmentez alors le score de 2 000 points et téléportez le fantôme au centre du décor. On ne gère pas d'attente particulière, le fantôme mangé peut se déplacer immédiatement. Pour montrer que Pac-Man est en mode « chasse », faites en sorte qu'il prenne une couleur différente ou qu'il clignote.

Attention, lorsqu'ils sont mangés, les fantômes renaissent au centre du décors. Ainsi le calcul de la carte des distances des fantômes doit uniquement prendre en compte les fantômes en dehors de leur base de départ. Dans le cas contraire, Pac-Man peut se fixer devant l'entrée de leur base alors qu'il n'est pas sensé détecter la présence des fantômes à cet endroit.