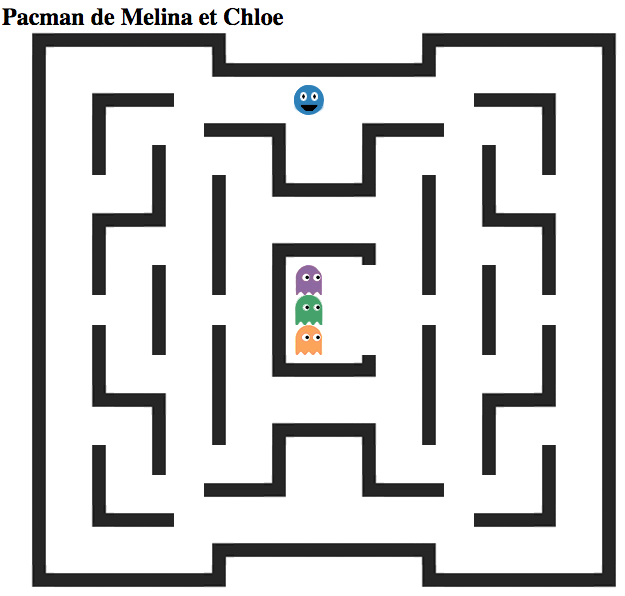
Cahier des charges

Jeu de Pacman en JavaScript



SOMMAIRE :

1. Introduction
2. Gifs
3. Affichage
4. Déplacements
5. Nourriture
6. Problèmes rencontrés
7. Sources
8. Introduction

Pour ce projet, nous avons décidé de faire un Pacman, en commençant tout d’abord sur le jeu de base, programmer le jeu classique de Pacman en JavaScript.

Nous allons apporter des modifications par la suite, afin de rendre le jeu plus personnel.

Le projet initial consiste en un Pacman avec deux personnages, Pacman en bleu et Pacgirl en rose, et plusieurs niveaux. Avant de commencer, le joueur devra définir sur quel labyrinthe il veut jouer et avec lequel des deux personnages. Celui-ci doit attraper le plus de nourriture possible, ce qui lui permet de gagner des points. Il existe deux types de nourriture, la nourriture normale et la nourriture spéciale, qui permet de rendre les fantômes vulnérables et change leur aspect. En effet le jeu comporte trois fantômes, qui font perdre le joueur lorsqu'ils attaquent Pacman. Le but du jeu est alors de gagner le plus de points possible avec la nourriture sans se faire attraper par les fantômes.

1. Gifs

Nous allons commencer par faire les graphismes, les images du labyrinthe, de pacman et des fantômes. On se sépare la tâche : Mélina fera la structure du labyrinthe et quelques fantômes, et Chloé fera le gif de pacman et les autres fantômes. Pour cela, nous nous sommes aidées de Paint et de Photoshop pour les dessiner dans un premier temps puis pour les animer, en gifs. Nous avons créé un Pacman en bleu et voulons créer Pacgirl en rose : l’utilisateur devra choisir avec lequel jouer. Le jeu est alors constitué d’un labyrinthe de 3 fantômes, de un Pacman (dont on a choisi préalablement la couleur) et de nourriture normale ou spéciale.

Afin de faire un rendu plus réaliste, moins aplati et que cela sorte de l’aspect 2 dimensions, nous avons pensé à faire 4 gifs de Pacman. Le gif par lequel est représenté Pacman varie en fonction de sa direction. Si il va à gauche, il aura le gif qui donne l’impression qu’il est tourné dans cette direction. De même pour toutes les autres directions (haut, bas, droite)

PB37 vers la gauche PB40vers le bas PB39vers la droite PB38vers le haut

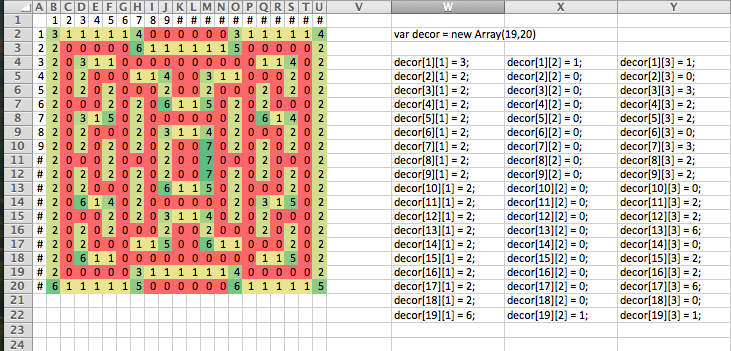
Les gifs des fantômes eux ne varient pas en fonction de la direction, car nous pensons que le focus est surtout sur Pacman, et par conséquent, l’utilisateur ne sera fixé que sur son personnage. De plus, le fait que les fantômes ne changent pas laisse un suspense sur la direction dans laquelle il va aller. Le jouer n’a donc pas d’indices sur la trajectoire choisie par le fantôme, ce qui lui laissera moins de temps pour réagir. Le jeu sera plus ludique.

Macintosh HD:Users:chloeparker:Documents:Tle:ISN:Projet ISN:images:G1.gif Fantôme 1 Macintosh HD:Users:chloeparker:Documents:Tle:ISN:Projet ISN:images:G2.gif Fantôme 2 Macintosh HD:Users:chloeparker:Documents:Tle:ISN:Projet ISN:images:G3.gif Fantôme 3

Les graphismes du labyrinthe sont très simples, juste des traits noirs qui délimitent l’espace de jeu. Les personnages sont libres d’aller où ils veulent, tant que cela reste dans le labyrinthe et que cela ne passe pas sur ou à travers les murs. Nous expliquerons par la suite comment nous avons procédé pour coder le labyrinthe.

1. Affichage

Pour programmer le labyrinthe, nous avions pensé à utiliser canvas, où la zone de jeu sera définie par la couleur du fond. Si le fond est blanc, alors les acteurs peuvent s’y déplacer. Par conséquent, si ils arrivent près d’un mur, qui lui a une autre couleur que blanche, alors il ne pourra pas y aller. Finalement, nous avons jugé plus intéressant et accessible d’utiliser des tableaux, afin de mieux visualiser la structure du labyrinthe, et de pouvoir la modifier, sans avoir à redessiner tout un labyrinthe. De cette façon, nous utilisons que 7 images pour tout le labyrinthe, et nous pouvons changer l’aspect du labyrinthe plus facilement. Chaque image représente soit un coin, un mur horizontal, vertical, ou le vide (où pacman peut s’y déplacer). Il s’agit alors de créer des coordonnées que nous écrirons dans le code grâce à un tableau sur Excel. L’intêret d’avoir utilisé Excel est que nous avons pu avoir un visuel du labyrinthe pendant sa création, et une idée de la disposition. Ainsi, nous avons utilisé newArray pour délimiter ce tableau, et inscrire ce que représentait chaque case. Il s’agissait de nommer chaque case selon l’image que nous voulons y afficher. Ainsi deux cases qui correspondent à un coin de mur en haut et à gauche auront le même nom et une case correspondant à un espace où Pacman se déplace aura pour nom 0. L’avantage du tableau par rapport aux canvas est que nous pouvons changer librement la disposition, sans avoir à refaire le dessin du labyrinthe. Ceci nous permettra de gagner du temps, et de passer plus vite des graphiques au code. De plus, ceci permettra de créer les différents niveaux plus facilement.

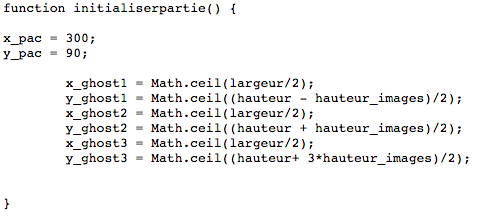


Le format d’affichage du jeu est de600px\*600px, car, d’après nos recherches statistiques, la majorité (plus de 80%) des utilisateurs ont un format d’environ 1300 \* 800 ou plus, et aucun écran n’est plus petit que 600 en hauteur, et notre labyrinthe se présentera sous forme carrée. Ainsi, tous les écrans seront compatibles. Certes, certains seront trop grand, mais ce n’est qu’une minorité qui a un écran de plus de 1600\*1200 (soit deux fois plus). Les images font donc 30px\*30px. Le tableau aura une 20 colonnes et 19 lignes (donc 600px\*570). Certes, ce n’est pas exactement un carré, mais pour des raisons d’esthétique du labyrinthe, nous étions obligés de supprimer une ligne horizontale (ce qui ne change rien concrètement au codage).

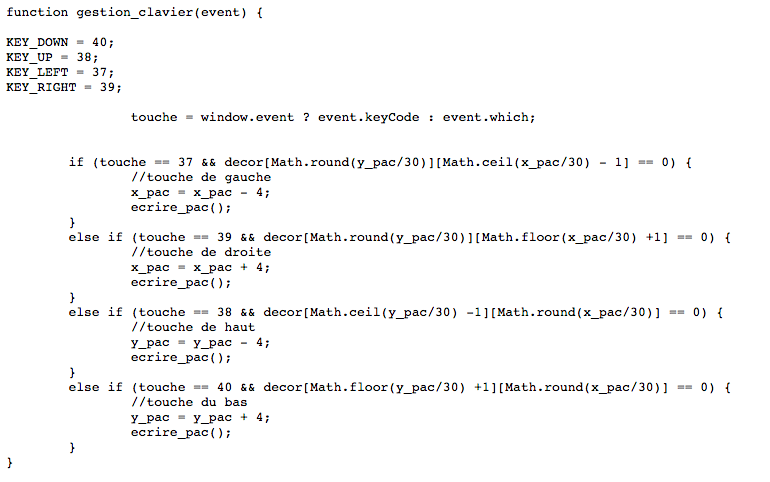
Par la suite, nous avons réussi à afficher le labyrinthe sur la page après avoir associé chaque case du tableau initial avec les images que nous avons créées. Pour ce code nous nous sommes inspirées du Jeu de Pong et du Jeu de la Vie.

1. Déplacements

Tout d’abord, il faut initialiser la partie. Pour faire cela, nous avons utilisé une fonction initialiserpartie (). Dedans, nous avons inscrit où sont situés les différentes images qui composent le jeu. Simultanément, nous « écrivons » les caractéristiques des objets, ce qui place les bonnes images aux bons endroits.



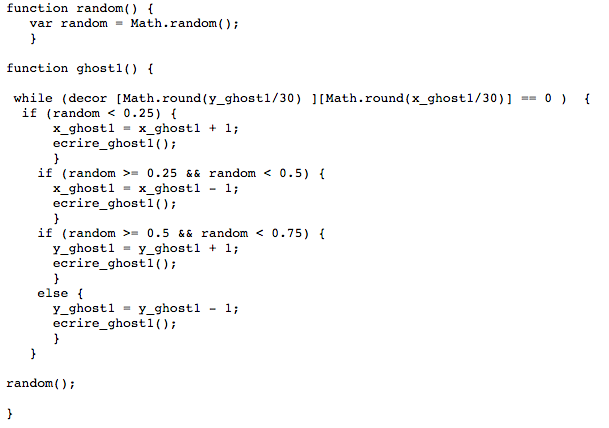
Ensuite nous avons programmé la gestion des déplacements de Pacman. Pacman se déplace en fonction de la touche appuyée par l’utilisateur. On a donc utilisé onkeydown. Nous nous sommes renseignés sur le numéro assigné aux différentes touches, puis les avons incorporé dans le programme. Si la touche appuyée est la 37 (flèche gauche), alors les coordonnées de pacman sont modifiées, et ses abscisses diminuent. Et ainsi de suite pour les autres touches. Afin d’éviter que Pacman ne se déplace dans un mur, nous avons mis une condition. Si l’utilisateur souhaite aller à gauche, mais que la case du tableau qui se situe à gauche de pacman est associée à un autre nombre que 0, alors l’action ne pourra pas être exécutée.



Une fois les coordonnées changées, il faut réécrire l’image de Pacman, et supprimer la précédente. Ainsi, on regarde si la variable qui définit Pacman existe déjà. Si elle existe, car elle a été créée dès le début du jeu, on ne fait que assimiler les coordonnées et le gif approprié. Le gif est déterminé par le numéro de la touche appuyée. Si c’est la touche 40, qui va vers le bas, alors le gif nommé PB40.gif sera celui choisi, et ainsi pour les quarte autres.

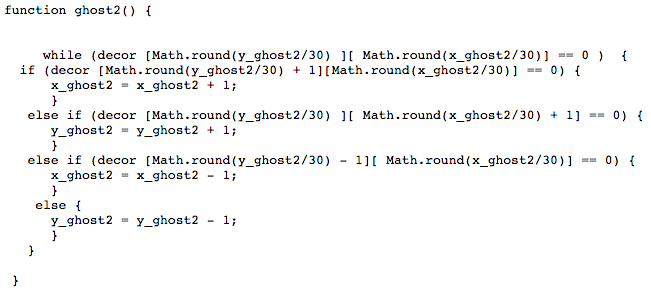


Pacman se déplace avec la commande des touches, tandis que les déplacements des fantômes sont décidés par l’ordinateur. Le premier fantôme se déplace aléatoirement. La fonction Math.random() ; va prendre un chiffre aléatoire entre 0 et 1. Si ce chiffre est compris entre 0 et 0,25, alors le fantôme se déplacera à droite, et ainsi de suite pour les 3 autres directions. La condition est la même que pour Pacman : si la case où il souhaite se diriger est associée à quelque chose d’autre que 0, alors le fantôme ne s’y déplacera pas, et on rappellera la fonction random pour tirer un autre nombre aléatoire. Ainsi, le fantôme 1 se déplacera tout droit, dans une seule direction, jusqu’à ce qu’il rencontre un mur, et il recalculera un autre chemin.



Lui aussi doit être réécrit une fois que ses coordonnées changent, de la même façon que pour le Pacman.

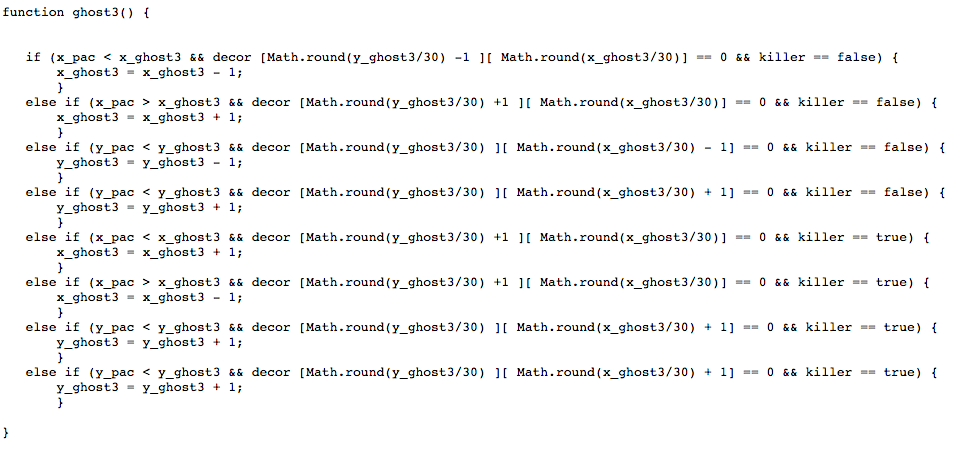
Le deuxième fantôme cherche à se déplacer dans un ordre précis de directions. Il commence par chercher si il peut aller à droite. Si il n’y a pas d’obstacles, il ira bien à droite, mais si cela lui est impossible, il cherche alors à aller en bas, puis à gauche, puis en haut.



Puis lui aussi est réécrit.



Pour le troisième fantôme, nous avons essayé de lui donner une trajectoire stratégique, et non pas indépendante de Pacman, comme les autres. En effet, ce dernier fantôme compare sa position (abscisses, ordonnées) avec ceux de Pacman. Si Pacman a une abscisse plus grande que lui, alors il va agrandir la sienne. Au contraire, si Pacman avait une abscisse plus petite, il diminuerait la sienne. De même pour les ordonnées. Cependant, il faut aussi tenir compte de ce que peut attraper le joueur durant la partie (dont nous allons expliquer le fonctionnement par la suite). Si Pacman mange de la nourriture qui lui permet de tuer les fantômes, alors le fantôme ne doit pas se rapprocher, mais plutôt s’éloigner. Ainsi, au lieu d’augmenter son abscisse si Pacman en a une plus grande, il va la diminuer. Il agira de cette façon tant que « killer » soit vrai, et dès que cette variable revient au status de « false », alors il se re-déplacera normalement.



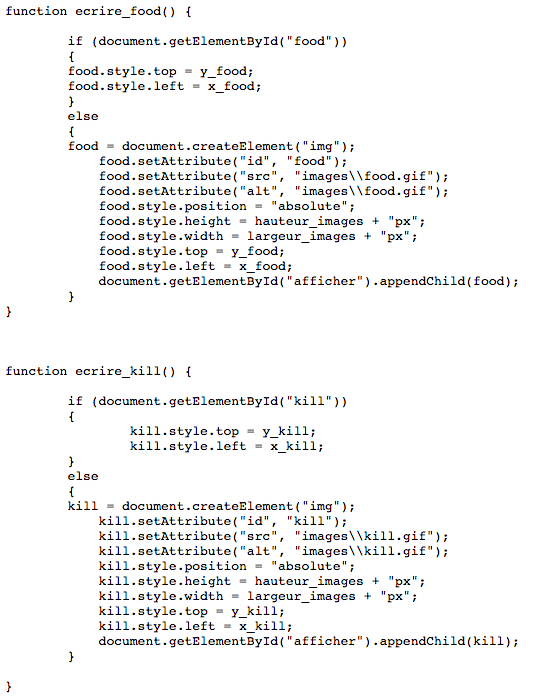
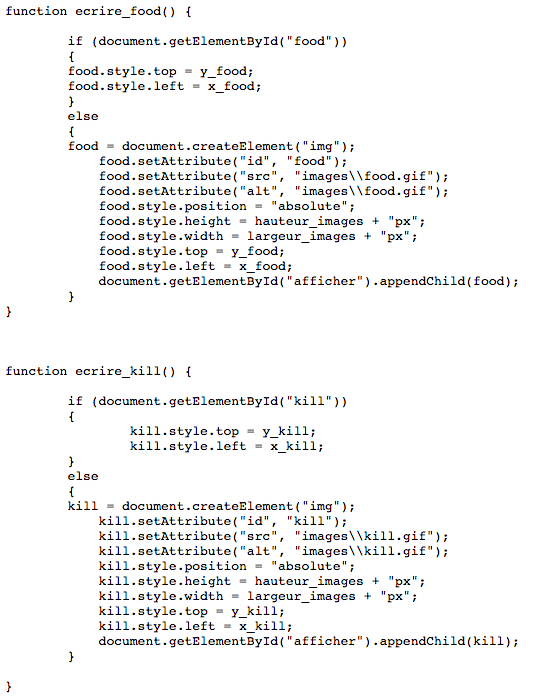
Puis de même que pour les autres fantômes.



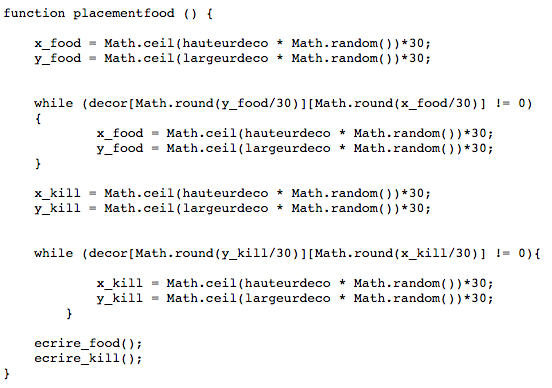
1. Nourriture

Nous étions partis sur l’idée originale de Pacman, où la nourriture jaune est disposée tout autour du labyrinthe, et le but est de toutes les manger. Nous avons décidé de personnaliser, en nous inspirant de Snake, par exemple. La nourriture apparaît donc aléatoirement dans le labyrinthe, et reste pendant 10 secondes, avant de changer encore de place.

Pour coder ceci, nous avons utilisé la même façon d’écrire les objets sur la page, en les associant avec des coordonnées et l’image correspondante.

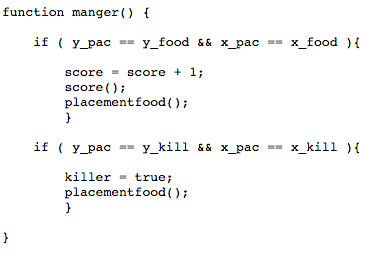


La nourriture se place aléatoirement dans le labyrinthe. Il faut donc utiliser Math.random() pour lui trouver des coordonnées. Math.random() donne un nombre aléatoire entre 0 et 1. Il faut donc multiplier ce nombre par le nombre de cases dans le labyrinthe par hauteur/largeur, puis encore multiplier par le nombre de pixels dans une image. Ainsi, ils obtiennent des coordonnées qui lui permettent de se placer dans le labyrinthe, et ceci tant que ces coordonnées correspondent à une case du tableau associée à 0.



Comme la nourriture se replace toutes les 10 secondes, il a fallu utiliser setInterval(placementfood, 10000). Ceci permet de refaire fonctionner cette fonction au bout de 10 secondes (10000milisecondes). Toutes les 10 secondes, placementfood() va recalculer des coordonnées pour la nourriture, qui vont donc apparaître quelque part d’autre dans le labyrinthe.

Le joueur a donc dix secondes pour essayer d’attraper la nourriture. La nourriture est considérée « attrapée » si les coordonnées abscisses/ordonnées du Pacman sont égales à celles de la nourriture. Ceci augmentera le score du joueur (inscrit sur le côté du labyrinthe).



Pour la nourriture spéciale, si ses coordonnées et celles du Pacman sont en concordance, alors cela activera le status de « killer ». Si ce status est activé, alors le fantôme 3 changera de trajectoire, et cela permettra au Pacman de tuer les fantômes au lieu que cela ne soit l’inverse.

Le but du jeu sera donc d’attraper le maximum de nourriture, sans se faire attraper par un fantôme. Si on se fait attraper, un pop-up « Game Over » apparaitra, ce qui signifiera la fin de la partie. Le joueur aura la possibilité de manger les fantômes, afin de les faire disparaître temporairement pour 10 secondes. La pastille verte représente cette nourriture spéciale. Elle n’apporte pas de points au joueur. Le fonctionnement d’ « attrapage » est le même que pour la nourriture normale, seules les conséquences qui en suivent diffèrent.

La rencontre entre Pacman, nourriture et fantômes est basée sur la concordance des coordonnées. Les conséquences qui en sont issues dépendes des circonstances des rencontres (si Pacman rencontre un fantôme, il faut voir si il a mangé de la nourriture spéciale préalablement etc)

1. Problèmes rencontrés

Au cours de la réalisation du jeu, nous avons rencontré des problèmes d’affichage, notamment pour le labyrinthe mais que nous avons réussi à résoudre. Par exemple, le fait d’oublier un point virgule ou une accolade quelquepart dans le code, pour lesquels il fallait faire appel aux outils développeurs de la page web.

Nous avons aussi rencontré des difficultés pour que l’image de Pacman soit bien mise en mouvement quand les coordonnées changent. Pacman était immobile car nous même si les coordonnées changeaient, elles n’étaient pas attribuées à l’image. Il fallut donc ajouter la fonction ecrire\_pac() à chaque fois que les coordonnées étaient modifiées, afin qu’elles lui soit attribuées. Il fallut aussi ajouter cela aux autres éléments qui restaient eux aussi immobiles (fantômes).

De plus, la nourriture allait parfois sur des parties du tableau codées par 0, donc sans mur, mais qui ne faisaient pas pour autant parti du domaine jouable dans le labyrinthe. Elle tombait parfois sur la bande en haut ou en bas du labyrinthe, qui est inaccessible pour Pacman. Il a donc fallut créer une autre image blanche, similaire à celle codée par 0, mais qui était codée par un autre chiffre. Ainsi, la nourriture n’y allait plus, car cet endroit du tableau est désormais codé par un nombre autre que 0.

Nous avons abandonné l’idée de Pacgirl et des différents niveaux, faute de temps. Nous savons comment nous aurions pu les instaurer, en créant un menu qui permette de choisir les options, et qu’un labyrinthe/personnage différent s’affiche en fonction du choix. Cependant, nous avons préféré nous concentrer sur un jeu, certes plus simple, mais plus fini.

Avec plus de connaissances en Javascript et en intelligence artificielle, nous aurions voulu créer des fantômes plus performants, capables de traquer Pacman et ses déplacements plausibles (comme le fait le jeu original), au lieu de juste se repérer en fonction des coordonnées instantanées ou aléatoirement.

1. Sources :

<http://www.apprendrelesmaths.com/>

<http://www.masswerk.at/JavaPac/legacy/JS-PacManPlus.html>

<http://sciences-du-numerique.fr/projet-javascript-pour-la-specialite-isn/code-source-du-jeu-de-snake/66>

<http://ericsowell.com/blog/2013/10/1/introduction-to-html5-canvas-with-pacman-action>

<http://www.w3schools.com/js/>

* ce qu'on veut faire, décrire projet
* en quoi ça consistait
* comment on a conçu le programme, comment on a structuré
* puis détails, répartition des rôles
* difficultés rencontrées, si ça n'a pas abouti où on en est, si ça aboutit ce qu'on aurait aimé ajouter
* lignes du code en annexe
* 5-10 pages