

RAPPORT

PROJET INFORMATIQUE : PACOMAN

Adrien Dufraux

Alexis Durieux

David Lamidel

TABLE DES MATIERES

Introduction	2
I. Cahier des charges initial	3
A. Description	3
B. Limites	3
II. Programme Actuel	4
A. Analyse descendante	4
B. Signatures et types de données principaux	5
i. Signatures	5
ii. Types de données	8
C. Algorithmes principaux	8
i. Menu	8
i. Boucle globale de jeu	8
ii. Gestion des niveaux et des maps	9
iii. Gestion des déplacements	10
iv. Gestion des interactions	10
v. Meilleurs scores	11
vi. Analyse de la map et intelligence artificielle	11
D. Gestion audio et vidéo	17
i. SDL2	17
ii. FMOD	17
iii. Photoshop	18
E. Perspectives d'évolution	19
i. Limites du code	19
ii. Ajouts éventuels	19
III. Guide d'utilisation	20
A. Jouer	20
i. Histoire	20
ii. Entraînement	20
B. Meilleurs scores	20
C. Jeu	21
D. Fin de partie	22
IV. Retour sur le travail en groupe	23
A. Répartition et organisation du travail	23
B. Impressions personnelles sur le projet	23
i. Adrien	23
ii. Alexis	24
iii. David	24
Conclusion	25

INTRODUCTION

Dans ce rapport de projet informatique, nous allons expliquer les points importants dans la création de notre jeu le « Pacoman ». Au début du semestre, lorsque nous devions choisir un sujet, il nous a dans un premier temps été difficile de choisir un sujet. En effet nous voulions, un sujet intéressant et stimulant sans être irréalisable.

L'option du « Pacoman » s'est alors vite montrée intéressante puisque même si le pacman de base n'est pas un jeu des plus compliqué, il offre néanmoins la possibilité d'effectuer des améliorations qui rendent sa conception stimulante. Ainsi nous avons décidé de refaire un pacman mais avec quelques modifications par rapport à l'original. Sans parler du thème graphique, nous avons décidé d'implémenter des bonus (turbo et ralentissement des monstres). De plus le Pacoman nous offrait l'opportunité de travailler sur une intelligence artificielle intéressante puisque nécessaire afin de rendre le jeu jouable.

Dans un premier temps nous expliquerons donc dans ce rapport les différents problèmes rencontrés lors de la conception du jeu et les solutions que nous avons choisi (analyse descendante, signatures, algorithmes...). Puis nous exposerons un petit guide pour utilisateur non averti. Enfin nous exprimerons notre ressenti quand à ce projet informatique.

I. CAHIER DES CHARGES INITIAL

A. Description

Au début de notre projet, nous avons fixé un cahier des charges listant les différentes options de notre jeu. En voici la liste :

- PACMAN classique dans terminal (Sans bonus)
- Graphisme SDL (optionnel)
- Bonus à ramasser sur la carte (par exemple pour tuer les monstres) (optionnel)
- Monstres différents (différentes vitesses et différents comportements) (optionnel)
- Outil de création de plateau (optionnel)
- Plateau aléatoire (optionnel)
- Menu de démarrage.
- Tableau des meilleurs scores

Nous n'avons pu réaliser la plupart des fonctionnalités attendues sauf le plateau aléatoire et l'outil de création de plateau. Il va être expliqué dans la partie suivante pourquoi nous avons fait ces choix.

B. Limites

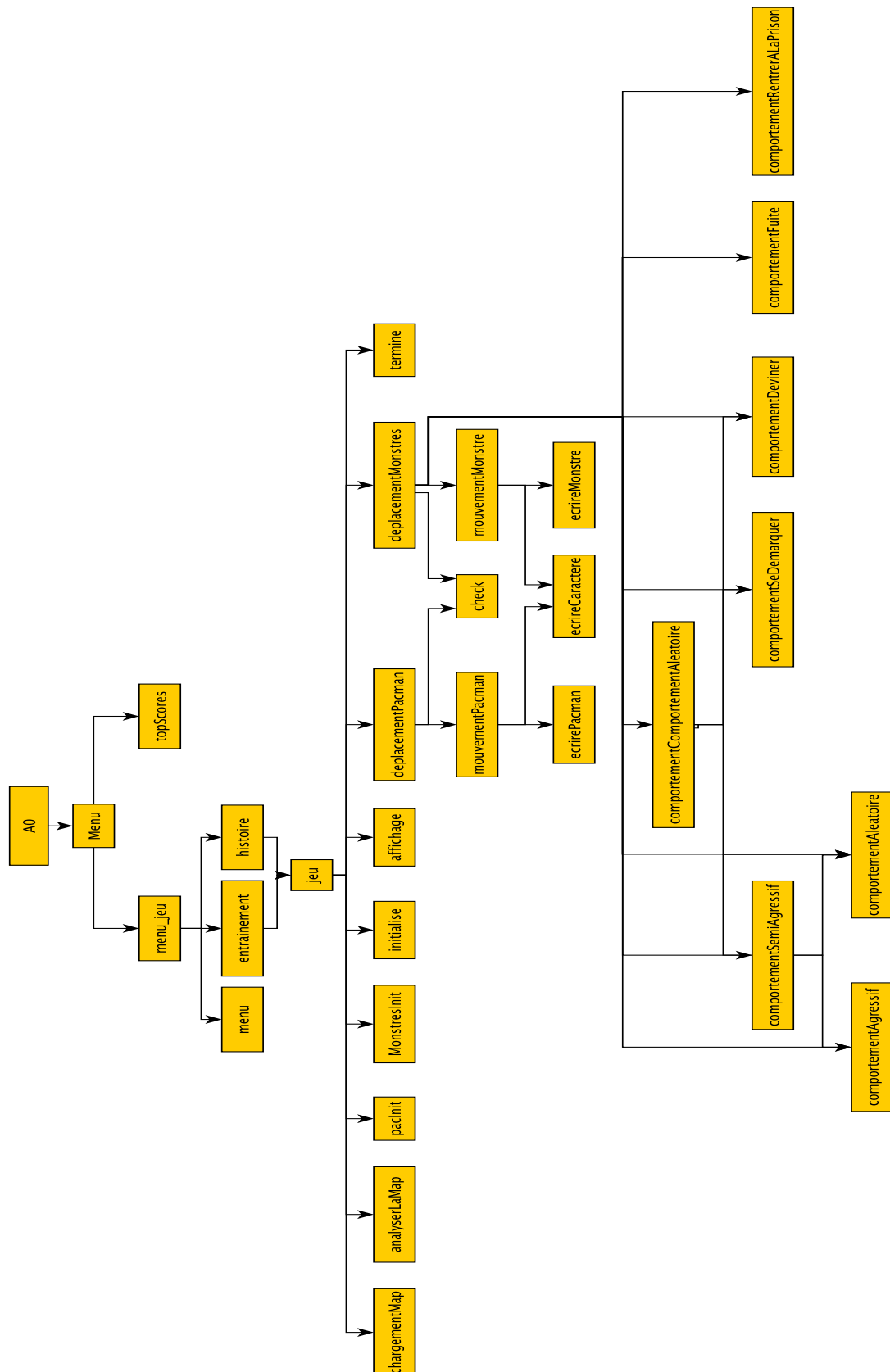
Il faut tout d'abord commencer par dire que nous avons privilégié le développement de certaines fonctionnalités plutôt que d'autres afin d'obtenir un meilleur Game Play. Par exemple, nous avons favorisé l'avancement de bonus devant l'outil de création de carte. En effet, l'outil de création de carte ne nous paraissait pas très difficile à implémenter mais cela n'apportait pas de plus valu à notre programme comparé aux différents comportements par exemple. Ensuite en ce qui concerne le générateur de plateau aléatoire, nous ne nous sommes pas rendu compte de la difficulté de cette fonctionnalité. En effet, il aurait fallu créer des cartes « jouables » c'est-à-dire intéressantes pour le joueur. Devant la difficulté d'implémentation, nous avons préféré d'autres fonctionnalités rendant le jeu plus passionnant.

Cependant, devant les fonctionnalités que nous avons prévus en optionnel, nous nous sommes rendu compte qu'ils avaient une grande importance. En effet, sans le graphisme SDL le jeu aurait perdu de l'intérêt. De plus, l'implémentation des différents comportements a permis de faire varier le niveau de difficulté sans changer de carte.

Ensuite, nous avons implémenté des fonctions qui n'étaient pas prévues comme par exemple le son. En effet, nous avons utilisé la bibliothèque FMOD pour implémenter des sons à notre programme. Cela a permis d'améliorer l'interaction du programme avec le joueur. Par exemple le son a facilité l'utilisation du bonus d'invincibilité.

II. PROGRAMME ACTUEL

A. Analyse descendante



B. Signatures et types de données principaux

I. SIGNATURES

- **procedure menu() ;**

Menu d'accueil du jeu. Propose le choix entre l'accès aux différentes options de jeu ou aux meilleurs scores.

- **procedure topScores();**

Cette procédure affiche la liste des cinq meilleurs scores.

- **procedure menu_jeu();**

Cette procédure propose aux joueurs deux choix. Soit le mode de jeu entraînement où le joueur peut choisir un niveau et une carte ou le mode histoire.

- **procedure histoire();**

Cette procédure déclenche le mode histoire, c'est à dire une suite de 5 cartes au niveau de difficulté croissant dont le joueur doit venir à bout.

- **procedure entraînement();**

Cette procédure déclenche le mode entraînement, c'est à dire qu'on demande à un utilisateur de rentrer un niveau et une carte afin de s'entraîner.

- **procedure jeu(niveau : string; map : string);**

Cette procédure est lancée à chaque niveau, elle gère de A à Z le déroulement d'une partie (c'est à dire du gain ou de la perte d'une carte).

- **procedure chargementMap(var plat : plateau; nomMap : String; var nbTacos : integer);**

Cette procédure charge la carte dans une variable plat de type plateau qui est un tableau de décor à partir d'un fichier.

- **procedure analyserLaMap(plat:plateau);**

Cette procédure permet d'analyser la map afin de servir de base à notre intelligence artificielle.

- **procedure pacInit (var pac : pacoman; var N : TDateTime; var plat : plateau);**

Cette procédure initialise toutes les variables dont le Pacoman dépend.

- **procedure monstresInit(plat:plateau ; var tabM : tableauMonstre; nomNiveau : string; var k : integer);**

Cette procédure initialise les variables de base de chaque monstre.

- **procedure initialise();**

Cette procédure initialise les librairies (SDL, FMOD, SDL2_TTF) et charge les différents éléments du jeu : images, sons, fonts.

- **procedure affichage(plat : plateau);**

Cette procédure affiche la carte à partir du tableau de décort

- **procedure deplacementPacman(var plat : plateau; var p : pacoman; var N : TDateTime; var NBonus : TDateTime; var tabM:TableauMonstre; var nbTacos : integer);**

Cette procédure gère le déplacement ou non du Pacoman, en fonction du timestamp. De plus cette procédure gère également les entrées claviers : changements de direction du pacoman et utilisation des bonus.

- **procedure deplacementMonstres(plat : plateau;var p : pacoman; var tab : TableauMonstre; var N : TDateTime; Nori : TDateTime;var k : integer; var NBonus : TDateTime; var nbTacos : integer);**

Gère le déplacement des monstres en fonction de leur comportement et du timestamp

- **procedure termine(p : pacoman);**

Cette procédure ferme les différentes librairies initialisées et libère les SDL_Surface etc...

- **procedure mouvementMonstre(var tab : TableauMonstre; i : integer; plat : plateau;p:pacoman);**

Cette procédure est appelée pour effectuer le mouvement d'un monstre si le déplacement est autorisé en fonction du timestamp.

- **procedure mouvementPacman(var p : pacoman; plat : plateau);**

Même principe que mouvementMonstre mais pour le Pacoman cette fois.

- **procedure check(var tabM : tableauMonstre; var plat : plateau; var p : pacoman; var NBonus : TDateTime; var nbTacos : integer; m : musiques);**

Cette procédure gère toutes les interactions des différents éléments du jeu. Par exemple lorsque le Pacoman rencontre un tacos, un bonus ou encore un monstre, cette procédure effectue les actions nécessaire (modification de telle ou telle variable...).

- **procedure ecrireCaractere(i, j : integer; plat : plateau);**

Cette procédure affiche un élément de décor.

- **procedure ecrirePacman(p:pacoman);**

Cette procédure affiche le Pacoman. Elle prend en compte sa direction et si il avait la bouche ouverte ou fermée précédemment.

- **procedure** **ecrireMonstre**(m : monstre);

Cette procédure affiche le monstre.

- **function** **comportementAgressif**(plat:plateau;directionInitial:direction;x1,y1, x2, y2:Integer): direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement agressif. Ce comportement suit le Pacoman.

- **function** **comportementAleatoire**(plat:plateau;directionInitial:direction;x1,y1:Integer): direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement aléatoire. Ce comportement fait déplacer le monstre aléatoirement.

- **function** **comportementFuite**(plat:plateau;directionInitial:direction;x1,y1,x2,y2:Integer):direction ;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement fuite. Ce comportement fuit le Pacoman.

- **function** **comportementRentrerALaPrison**(plat:plateau;directionInitial:direction;x1,y1:Integer):direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement RentrerALaPrison. Ce comportement renvoie le monstre à la prison.

- **function** **comportementDeviner**(plat:plateau;directionInitial:direction;x1,y1:Integer):direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement Deviner. Ce comportement envoie le monstre dans la direction où le pacoman va en fonction des positions de tacos sur la map.

- **function** **comportementSeDemarquer**(plat:plateau;tabM:tableauMonstre;directionInitial:direction;x1,y1:Integer):direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement se démarquer. Ce comportement envoie le monstre dans la direction où il y a le moins de monstres.

- **function** **comportementSemiAgressif**(plat:plateau;directionInitial:direction;x1,y1,x2,y2:Integer;var compteur:Integer):direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement semi-agressif. Le comportement semiagressif alterne entre le comportement agressif et aléatoire

- **function**
comportementComportementAleatoire(plat:plateau;tabM:tableauMonstre;directionIniti
al:direction;x1,y1,x2,y2:Integer;var compt,a:Integer):direction;

Cette fonction renvoie la direction que doit prendre le monstre avec un comportement aléatoire. Ce comportement de monstre fait changer le monstre de comportement tous les 10 croisements parmi un des autres comportements.

II. TYPES DE DONNEES

Afin de simplifier l'appel de nos fonctions et procédures nous avons créé différents types de données qui nous aident à classer nos variables. Nous avons d'abord plusieurs types d'enregistrements. Les premiers sont ceux du Pacoman et des monstres, dans lesquels on trouve aussi bien leur position, leur direction mais également des variables de temps nous aidant à gérer les déplacements et les bonus. Pour les monstre nous avons créé un tableau de Monstres afin de gérer tous les monstres. Nous avons donc par exemple des structures de type « tab[i].x » pour la coordonnée en abscisse du premier monstre. Nous avons également un enregistrement « timer » qui nous sert dans notre enregistrement « Pacoman » et qui contient les différentes informations relatives à nos bonus. De plus nous avons également créé un enregistrement « musiques » stockant les différents PFSOUNDSample qui sont donc des fichiers sons de FMOD.

C.Algorithmes principaux

I. MENU

Menu est la procédure qui permet de choisir les modes de jeu ou de voir les meilleurs scores. Cette procédure gère juste l'appel des différentes procédures selon le choix de l'utilisateur. Pour simplifier et clarifier le code, nous avons préféré séparer les deux modes de jeu par procédure même si la boucle de jeu est commune entre les deux.

I. BOUCLE GLOBALE DE JEU

Parlons maintenant de la boucle globale (qui effectue les déplacements du Pacoman et des monstres) présente dans les procédures jeu et histoire. En effet il nous a fallu dissocier plusieurs cas. Dans un premier temps il nous a fallu considérer deux cas pour sortir de la boucle présente dans jeu afin : soit l'on perd, soit l'on gagne. De plus si l'on perd, il faut arrêter le jeu alors que si l'on gagne il faut changer de niveau. Ainsi nous avons défini un booléen « fin » que nous avons initialisé à false dans la procédure histoire.

Ensuite dans la procédure jeu nous avons fait en sorte de sortir du repeat..until lorsque le nombre de tacos présents sur la map est égal à zéro, c'est à dire quand le joueur a gagné ou quand le nombre de boucliers du Pacoman est égal à zéro, c'est à dire quand le joueur a perdu. Ensuite lorsque nous sortons de cette boucle nous faisons un test sur la variable « p.bouclier » contenant le nombre de vies restant afin d'affecter la valeur true à « fin » si le joueur a perdu afin de sortir de la boucle présente dans histoire.

II. GESTION DES NIVEAUX ET DES MAPS

Lorsque nous avons commencé à nous interroger sur la gestion des niveaux et des maps, une problématique s'est très vite imposée à nous. Devions nous gérer indépendamment ces deux éléments ou faire correspondre à chaque map un niveau fixé. Nous avons décidé de faire un mix des deux. En effet nous avons décidé pour notre mode histoire de faire progresser le joueur sur cinq maps différentes auxquelles nous avons décidé d'associer cinq niveaux. Néanmoins nous voulions qu'il soit possible de jouer sur une carte de notre choix en choisissant notre niveau.

De ce fait, nous avons créé deux types de fichiers textes dans lesquels nous stockons respectivement les éléments de décor, les fichiers du type « map-.txt » et les informations relatives aux niveaux, « niveau-.txt ». Dans les fichiers contenant les éléments de décor, ceux-ci sont stockés sous forme de lettre, chaque lettre correspondant à un type de décor (par exemple à la lettre « m » correspond l'élément de décor « mur »). Ainsi, lorsque nous lisons le fichier texte dans la procédure chargementMap, nous faisons correspondre à chaque lettre que nous lisons un élément de décor que nous stockons dans un tableau de décors. Pour déterminer lorsque nous avons fini de lire une ligne de décor nous avons placé un « # » en fin de ligne de notre fichier texte afin de changer de ligne dans notre tableau de décor. Dans le fichier stockant les informations relatives aux niveaux on y trouve les données relatives aux 5 monstres présents sur la carte que l'on stocke ensuite dans un tableau de monstres.

La première information que l'on trouve dans ce fichier de texte est un entier correspondant au comportement du monstre. Ensuite nous avons laissé un blanc dans le fichier texte afin de délimiter les informations. Ainsi après le blanc on trouve le facteur de vitesse du monstre. Ensuite, le « # » délimite le fait que les informations suivantes lues sont celles d'un autre monstre. Grâce à ce système nous avons pu résoudre notre problème initial. En effet pour jouer sur la map deux avec le niveau cinq il nous suffit d'utiliser les fichiers textes correspondants. En effet dans la procédure histoire par exemple, nous utilisons un repeat..until dans lequel nous itérons une variable entière de un à chaque fois que l'on change d'un niveau. Ainsi lors de la première entrée dans la boucle cette variable valant un, nous allons utiliser les fichiers « map1.txt » et « niveau1.txt ». Lors du deuxième passage dans la boucle, cette variable vaudra deux. Nous lirons alors les fichiers « map2.txt » et « niveau2.txt » jusqu'à un total de cinq niveaux. Dans la procédure entraînement, en revanche nous demandons à l'utilisateur de rentrer le niveau et la carte sur laquelle il veut jouer.

III. GESTION DES DEPLACEMENTS

Dans cette partie nous allons expliquer le système que nous avons utilisé pour effectuer les différents déplacements. Le premier problème auquel nous nous sommes heurtés est le fait que nous voulions effectuer des déplacements simultanés (monstres et Pacoman en même temps) mais à des vitesses différentes. Nous avons alors assimilé la notion de vitesse de déplacement à intervalle de temps entre deux déplacements. Pour expliquer notre démarche nous allons prendre l'exemple du déplacement du Pacoman, mais le système de déplacement des monstres est identique.

Le principe est simple, à chaque déplacement du Pacoman on récupère la date à l'aide de `dateutils`, des fonctions `Now` et `MillisecondOfTheHour`. Dans un premier temps on récupère le timestamp à l'aide de la fonction `Now`. Ensuite on transforme cette valeur en millisecondes qui correspond au nombre de millisecondes écoulées depuis le début de l'heure et on l'a stocke dans une variable « `p.date` » qui contient de ce fait la date du déplacement (Attention : bug lors d'un changement d'heure car la différence entre les dates entre deux déplacements en millisecondes est inférieure à 0).

Ensuite lorsqu'on rentre une nouvelle fois dans `deplacementPacman`, on compare la date actuelle avec les mêmes fonctions que celles utilisées précédemment. Si la différence des deux dates est supérieure à une « `VITESSEDEPLACEMENT` » constante (en millisecondes) quel l'on divise par un facteur « `k` » alors le déplacement a lieu.

IV. GESTION DES INTERACTIONS

La gestion des interactions correspond à la procédure `check` contenu dans l'unité jeu. Cette procédure gère donc les différentes interactions : `pacman-carte`, `pacman-monstres`. Cette procédure actualise aussi les bonus.

La procédure `check` contient un `case of` qui actualise les variables du jeu selon la position du pacman sur la carte. Le pacman peut se trouver sur :

- Un `tacos` : on augmente alors le nombre de points.
- Un bonus invincible : le statut du pacman et des monstres sont actualisés, la vitesse des monstres est diminuée et la date de début du bonus est récupérée.
- Un bonus de vitesse : la jauge de vitesse est remplie.
- Un bonus de ralentissement : la jauge de ralentissement est remplie.
- Un `switch` : lecture du tableau de `switchs` afin de modifier les coordonnées et la direction du pacman.

Ensuite `check` s'occupe aussi l'interaction `pacman-monstres`. En effet, quand un monstre et le pacman se trouvent sur la même case, la procédure modifie les variables y compris le nombre de vie selon le statut des entités.

Cette procédure gère de même la fin du bonus d'invincibilité et la sortie de prison des monstres. De plus, elle contient une sécurité pendant le bonus d'invincibilité car il se peut qu'un monstre ait été oublié par le programme à l'activation du bonus. C'est pourquoi cette procédure actualise en permanence le statut des monstres et leur vitesse lors du bonus d'invincibilité.

V. MEILLEURS SCORES

Notre programme comporte la fonctionnalité d'un classement des meilleurs scores. Ce classement implique deux procédures : ActualisationTopScores et TopScores.

- TopScores est une lecture du fichier des meilleurs scores. Cette procédure permet aussi d'effacer le classement des scores
- ActualisationTopScores permet de vérifier à la fin d'une partie si le score du joueur entre dans le classement des meilleurs scores et si c'est le cas elle actualise le classement. La procédure met à jour le classement à l'aide de tableaux. Les tableaux sont remplis par la lecture du fichier texte « Meilleurs scores ». Un tableau est réservé pour les noms et un autre est pour les scores avec une correspondance d'indice entre les tableaux. Par exemple le troisième meilleur score est contenu dans score[3] et le nom du joueur ayant fait ce score est dans nom[3]. Cette séparation en deux tableaux est nécessaire car les types de données sont différents. On effectue alors une insertion du score dans un tableau trié. On réalise les mêmes modifications dans les deux tableaux afin de garder la correspondance des indices. A la fin de l'insertion du score dans le classement, le contenu des deux tableaux est enregistré dans le fichier texte.

VI. ANALYSE DE LA MAP ET INTELLIGENCE ARTIFICIELLE

Concernant l'intelligence artificiel, il avait d'abord été indiqué dans le cahier des charges que l'implémentation de l'intelligence artificiel était optionnelle. Cependant il nous paraissait indispensable d'en créer une pour avoir un Game Play suffisamment évolué. Nous allons présenter ici, la manière dont a été codée cette intelligence artificielle. Aucune connaissance en intelligence artificielle n'a été utilisée ici, juste des idées simples.

La première étape a été d'analyser la map (procédure analyserLaMap()). Tous les calculs d'analyse sont effectués pendant le chargement de la map. Ensuite les différents comportements des monstres ont été codés à partir de cette analyse (unité IA). Ces comportements s'appuient sur un tableau créé par la procédure d'analyse (tabInterCroisement2).

Pour cela, on a distingué différents type de cases :

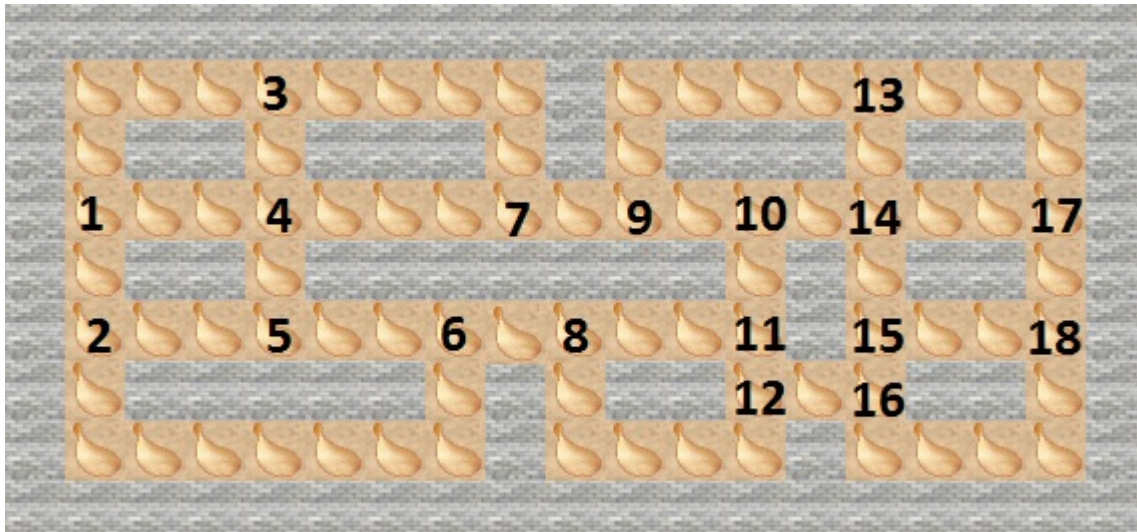
- Les murs où les monstres ne peuvent pas aller.
- Les croisements. Ce sont des cases où il y a plus de trois chemins qui partent de cette case. En effet s'il y a plus de trois chemins, c'est que le monstre peut emprunter ensuite plusieurs chemins. Sur ces cases, les monstres doivent prendre une décision. Ils ne peuvent en effet changer de direction que lorsqu'ils sont sur un croisement. La direction qu'ils doivent alors prendre est alors tout le problème. On considérera ensuite que la prison ainsi que les switches sont des croisements.
- Les autres cases : Ce n'est ni un mur, ni un croisement. Quand un monstre est sur une telle case il continue juste son chemin jusqu'à ce qu'il rencontre un croisement. Le cas particulier des impasses est traité dans le programme même si on évite d'en mettre car cela rend le jeu moins jouable.

On expliquera le fonctionnement du programme sur une map fictive, mais le programme s'adapte à n'importe quelle map.

Etape 1 : On détermine où sont les croisements.

A l'aide de la procédure `chercheCroisement()`, le programme crée un tableau `platCroisement[x,y]`, avec (x,y) les coordonnées de la case. Ce tableau est rempli de 0, sauf là où il y a un croisement. Il indique alors sur les cases avec croisement le numéro du croisement.

Ci-dessous nous avons la map affichée graphiquement avec les numéros des croisements.



Et ici le tableau `platCroisement[x,y]` correspondant à la map.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	3	0	0	0	0	0	0	0	0	0	13	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	4	0	0	0	7	0	9	0	10	0	14	0	0	17	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	0	0	5	0	0	6	0	8	0	0	11	0	15	0	0	18	0
0	0	0	0	0	0	0	0	0	0	0	0	12	0	16	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2	4
2	6
5	2
5	4
5	6
8	6
9	4
10	6
11	4
13	4
13	6
13	7
15	2
15	4
15	6
15	7
18	4
18	6

Pendant cette procédure on en profite pour créer un autre tableau `coordCroisement[numéro du croisement,1 ou 2]` (voir ci-contre) qui enregistre les coordonnées de chaque croisement. 1 étant la coordonnée en x et 2 la coordonnée en y. Grâce à cela on peut passer facilement du numéro du croisement à ses coordonnées.

A l'aide de la procédure `distanceEntreCroisement()`, on détermine pour chaque croisement quels sont les croisements qui lui sont adjacents et la distance qui les séparent. La procédure part d'un croisement et regarde si il y a une case différent d'un mur juste à côté. Si c'est le cas alors on appelle la procédure `trouveProchainCroisement (numCroisement1;x1;y1;x2;y2;distance;directionPourAllerAuCroisement)` avec `numCroisement1` le croisement d'où l'on est parti ; `x1` et `y1` les coordonnées de la case où l'on est et `x2`, `y2` les coordonnées de la case où on va et `directionPourAllerAuCroisement` la direction qu'on a emprunté au départ pour partir du croisement. La procédure regarde à chaque fois les cases autour de celle qui est analysée. Elle choisit la case qu'on n'a pas encore emprunter puis elle s'appelle elle-même pour analyser cette case (fonctionnement récursif). On s'arrête quand on tombe sur un nouveau croisement.

- `tabInterCroisement1[numCroisement1,numCroisement2,1 ou 2]`. Avec au niveau 1 la distance entre le croisement 1 et le croisement 2. Au niveau 2, la direction qu'il faut prendre pour aller du croisement 1 vers le croisement 2. Avec 1=gauche, 2= bas, 3=droite et 4=haut. Pour implémenter les switches on indiquera juste artificiellement que la distance entre deux switches qui sont reliés est égal à 1.

[illegible][illegible]

13

- infoCases[x,y,1 et 2 et 3]. Ce tableau donne des informations sur les cases qui ne sont pas des croisements. Avec au niveau 1 un croisement qui est relié à la case de coordonnée (x,y). Au niveau 2 l'autre croisement. Au niveau 3 la distance entre la case et le premier croisement.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	3	3	0	7	7	7	7	0	13	13	13	13	0	17	17	17	0	0
0	3	0	0	4	0	0	0	7	0	13	0	0	0	14	0	0	17	0	0
0	0	4	4	0	7	7	7	0	9	0	10	0	14	0	17	17	0	0	0
0	2	0	0	5	0	0	0	0	0	0	0	11	0	15	0	0	18	0	0
0	0	5	5	0	6	6	0	8	0	11	11	0	0	0	18	18	0	0	0
0	6	0	0	0	0	0	6	0	12	0	0	0	16	0	0	0	18	0	0
0	6	6	6	6	6	6	6	0	12	12	12	12	0	18	18	18	18	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EXEMPLE 3 INFORMATIONS CASES NIVEAU 1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	3	3	3	3	0	9	9	9	9	0	13	13	13	0	0
0	1	0	0	3	0	0	0	3	0	9	0	0	0	13	0	0	13	0	0
0	0	1	1	0	4	4	4	0	7	0	9	0	10	0	14	14	0	0	0
0	1	0	0	4	0	0	0	0	0	0	0	10	0	14	0	0	17	0	0
0	0	2	2	0	5	5	0	6	0	8	8	0	0	0	15	15	0	0	0
0	2	0	0	0	0	0	2	0	8	0	0	0	12	0	0	0	16	0	0
0	2	2	2	2	2	2	2	0	8	8	8	8	0	16	16	16	16	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EXEMPLE 4 INFORMATIONS CASES NIVEAU 2

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	2	1	0	5	4	3	2	0	4	3	2	1	0	4	3	2	0	0
0	4	0	0	1	0	0	0	0	1	0	5	0	0	1	0	0	1	0	0
0	0	2	1	0	3	2	1	0	1	0	1	0	1	0	1	0	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
0	0	2	1	0	2	1	0	1	0	2	1	0	0	0	2	1	0	0	0
0	9	0	0	0	0	0	0	1	0	5	0	0	1	0	0	0	1	0	0
0	8	7	6	5	4	3	2	0	4	3	2	1	0	5	4	3	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EXEMPLE 5 INFORMATIONS CASES NIVEAU 3

Étape 3 : Associer les deux croisements à une case

La procédure associerlesDeuxCroisementsAUneCase() prend toutes les cases et associe les deux croisements qui lui sont reliés mais cette fois ci par ordre de distance par rapport à la case. On obtient alors un nouveau tableau croisementLePlusProche[x,y,1 ou 2] avec (x,y) les coordonnées des cases. Au niveau 1 on a le croisement le plus proche et au niveau 2 on a l'autre croisement.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	3	3	0	3	3	7	7	0	9	13	13	13	0	13	13	17	0	0
0	1	0	0	4	0	0	0	7	0	9	0	0	0	14	0	0	17	0	0
0	0	1	4	0	4	7	7	0	9	0	10	0	14	0	14	17	0	0	0
0	2	0	0	5	0	0	0	0	0	0	0	11	0	15	0	0	18	0	0
0	0	2	5	0	5	6	0	8	0	8	11	0	0	0	15	18	0	0	0
0	2	0	0	0	0	0	6	0	8	0	0	0	16	0	0	0	18	0	0
0	2	2	2	6	6	6	6	0	8	12	12	12	0	16	16	18	18	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EXEMPLE 6 CROISEMENT LE PLUS PROCHE NIVEAU1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	1	1	0	7	7	3	3	0	13	9	9	9	0	17	17	13
0	3	0	0	3	0	0	0	3	0	13	0	0	0	13	0	0	13
0	0	4	1	0	7	4	4	0	7	0	9	0	10	0	17	14	0
0	1	0	0	4	0	0	0	0	0	0	0	10	0	14	0	0	17
0	0	5	2	0	6	5	0	6	0	11	8	0	0	0	18	15	0
0	6	0	0	0	0	0	2	0	12	0	0	0	12	0	0	0	16
0	6	6	6	2	2	2	2	0	12	8	8	8	0	18	18	16	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EXEMPLE 7 CROISEMENT LE PLUS PROCHE NIVEAU 2

Etape 4 : Chemin le plus court.

On arrive enfin à l'étape la plus importante. On détermine quel est le chemin le plus court pour aller d'un croisement à un autre. Pour cela on utilise l'algorithme de Dijkstra. Grâce aux tableaux qu'on a créés précédemment, on a construit un graphe avec des nœuds qui sont les croisements et des arrêtes qui sont les distances entre ces croisements. L'algorithme est utilisable seulement pour des graphes à poids positifs, ce qui est le cas ici. On ne détaillera pas le fonctionnement de l'algorithme mais nous avons étudiés de la documentation¹ pour le comprendre.

On arrive à obtenir alors le chemin le plus court entre deux croisements. Par exemple entre les croisements 3 et 12 l'algorithme nous renvoi le chemin 3-4-5-6-8-11-12. On utilise alors le `tabInterCroisement1` pour connaître la direction qu'il faut prendre pour aller du croisement 1 au croisement 4 (puisque'ils sont adjacents). Cette direction obtenue est en fait aussi celle pour aller du croisement 3 au croisement 12. On utilise ainsi l'algorithme dans un double for pour savoir comment aller de n'importe quel croisement vers n'importe quel autre. Un monstre pourra alors cibler n'importe quel croisement. On en profite pour avoir la distance entre les différents croisements. On enregistre ces résultats dans un tableau : `tabInterCroisement2[croisement1,croisement2,1 ou 2]`. Au premier niveau on a la distance entre le croisement1 et le croisement2. Au second niveau on a la direction pour aller du croisement1 vers le croisement2.

0	2	5	3	5	8	7	10	9	11	13	14	15	13	15	16	16	18
2	0	7	5	3	6	9	8	11	13	11	12	17	15	15	14	18	18
5	7	0	2	4	7	6	9	8	10	12	13	14	12	14	15	15	17
3	5	2	0	2	5	4	7	6	8	10	11	12	10	12	13	13	15
5	3	4	2	0	3	6	5	8	10	8	9	14	12	12	11	15	15
8	6	7	5	3	0	9	2	9	7	5	6	11	9	9	8	12	12
7	9	6	4	6	9	0	9	2	4	6	7	8	6	8	9	9	11
10	8	9	7	5	2	9	0	7	5	3	4	9	7	7	6	10	10
9	11	8	6	8	9	2	7	0	2	4	5	6	4	6	7	7	9
11	13	10	8	10	7	4	5	2	0	2	3	4	2	4	5	5	7
13	11	12	10	8	5	6	3	4	2	0	1	6	4	4	3	7	7
14	12	13	11	9	6	7	4	5	3	1	0	7	5	3	2	8	6
15	17	14	12	14	11	8	9	6	4	6	7	0	2	4	5	5	7
13	15	12	10	12	9	6	7	4	2	4	5	2	0	2	3	3	5
15	15	14	12	12	9	8	7	6	4	4	3	4	2	0	1	5	3
16	14	15	13	11	8	9	6	7	5	3	2	5	3	1	0	6	4
16	18	15	13	15	12	9	10	7	5	7	8	5	3	5	6	0	2
18	18	17	15	15	12	11	10	9	7	7	6	7	5	3	4	2	0

EXEMPLE 8 TABINTERCROISEMENT2

¹ <http://openclassrooms.com/courses/le-pathfinding-avec-dijkstra>

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

EXEMPLE 9 TABINTERCROISEMENT NIVEAU 2

Avec ces tableaux, on va pouvoir maintenant définir les différents comportements des monstres. Grâce au système mis en place les monstres peuvent emprunter les switches de façon intelligente. Si le chemin le plus court pour aller vers la cible emprunte un switch alors le monstre l'utilise.

Une fonction est appelée dans tous les comportements différents : `directionPourContinuerChemin(plat; directionInitial; x, y)`. Elle est en effet utilisée dès que le monstre n'est pas sur un croisement car les monstres n'ont pas le droit de faire demi-tour s'ils ne sont pas sur un croisement. Si le monstre est sur un croisement il appelle alors une fonction en fonction de son comportement. Cette fonction renvoie alors directement quelle direction prendre.

Voici les différents comportements implémentés dans le jeu :

- **Agressif** : Le monstre cible systématiquement le croisement le plus proche du Pacoman. Si le monstre est déjà dessus, il prend l'autre croisement. Ce comportement est difficile à jouer car le monstre colle systématiquement le Pacoman. Il faut se servir des bonus pour espérer gagner.
- **Aléatoire** : A chaque croisement le monstre choisit une direction aléatoirement, sauf celle d'où il vient. C'est le comportement le plus facile à jouer.
- **Semi agressif** : Le monstre passe successivement du comportement Agressif à Aléatoire tous les 10 croisements qu'il traverse. Il est assez fourbe puisqu'on ne sait pas quand il va attaquer.
- **Deviner** : Le monstre essaie de deviner où le joueur va aller en fonction des tacos qu'il reste sur la map. A chaque case qui n'est pas un mur, le programme calcul le nombre de tacos qu'il y a dans un carré autour de cette case. Il cible alors l'endroit où il y a le plus de tacos. Ce monstre n'est dangereux qu'en fin de partie puisque qu'il va rester aux endroits où il reste des tacos. Il faudra souvent se garder un bonus pour pouvoir finir le niveau.
- **SeDémarquer** : Le monstre va cibler une case où la densité de monstre y est la plus faible. On va en effet calculer la densité de monstres sur chaque case à l'aide d'une autre fonction. Le monstre n'est pas vraiment dangereux puisqu'il ne cible pas du tout le Pacoman.
- **Fuite** : Ce comportement est utilisé quand le Pacoman a pris un bonus invincible. Les monstres vont essayer de fuir le Pacoman pour ne pas se faire manger. Le programme regarde autour du monstre et cible le croisement qui se trouve le plus loin du Pacoman.

- ComportementAleatoire : Le monstre alterne entre les comportements agressifs, aléatoire, deviner, se démarquer et fuite tous les 5 croisements qu'il traverse. Le monstre n'attaque pas souvent le Pacoman donc il n'est pas très dangereux mais il faut quand même le surveiller.
- Rentrer à la prison : Quand un monstre touche le Pacoman, celui-ci retourne tout le temps à la prison. Le Pacoman ne perd pas de vie s'il est invincible. Il y retourne simplement en ciblant le croisement qui correspond à la prison.

Avec tous ces comportements on peut adapter la difficulté des niveaux pour essayer rendre le jeu facile au début et dure à la fin.

D. Gestion audio et vidéo

I. SDL2

Pour la gestion des graphismes de notre programme, nous avons décidé d'utiliser la bibliothèque SDL2. La principale difficulté que nous avons rencontrée pour son utilisation est le manque d'informations sur celle-ci en pascal et l'installation. Néanmoins après avoir compris le principe, il nous a été relativement facile d'effectuer la transition d'une interface terminal vers une interface graphique. Malgré cela nous sommes conscients que notre code concernant la partie graphique possède des lacunes en terme d'optimisation. En effet, le code utilisé pour arriver à nos fins est redondant. Il aurait été plus « propre » de créer une image dans sa totalité à chaque tour de boucle et de l'afficher ensuite au lieu de faire comme nous le faisons actuellement, c'est à dire de modifier des petits bouts d'interface successifs comme une toile de peinture sur laquelle on peindrait petit à petit.

II. FMOD

Pour la gestion du son, nous avons dans un premier temps pensé à utiliser SDL_mixer, néanmoins à cause d'un soucis d'installation nous nous sommes rabattus sur la bibliothèque FMOD. Grâce à celle-ci nous avons pu charger simplement des sons au format « wave » et les jouer. Une des difficultés que nous avons rencontrée dans la mise en place du son, est la gestion des différents canaux de son. En effet, il y a parfois la possibilité que plusieurs interactions s'effectuent en même temps comme par exemple un entrée clavier pour activer le bonus qui active un son et le Pacoman qui rencontre le monstre. Il nous a donc fallu gérer ce problème à l'aide de différents canaux de sons.

III. PHOTOSHOP

Dans notre programme, nous avons voulu une charte graphique, un thème graphique différent du Pacman classique. Notre idée s'est basée sur un Pacman mexicain appelé « Pacoman » qui mange des tacos sur la carte. Pour réaliser, ce graphisme nous avons utilisé un logiciel de retouche de photo : « Photoshop ». En effet, le but était de personnaliser notre jeu de Pacman en pacoman.

Nous avons récupéré sur internet les images du Pacman classique et nous avons ajouté à l'aide de calque des objets pour créer le thème (sombbrero pour le Pacoman, maracas pour les montres). De plus, nous avons changé les bonus (piments, tête de mort, bouteille), transformé les points en tacos et le fond noir en une couleur sable. Je vais maintenant expliquer les différentes couches graphiques du jeu.

² Tout d'abord, il a fallu créer les images de fond c'est-à-dire le sable, les murs et les switchs. Ces images sont la première couche graphique du jeu. En ce qui concerne les murs, nous avons pensé à des cactus pour entrer dans le thème mexicain mais on s'est vite rendu compte que ça chargeait le visuel du jeu. De plus, il fallait une couleur qui tranche avec les autres couleurs du jeu. C'est pourquoi nous avons mis ce type de mur classique gris qui crée une continuité le long du mur et qui marque une différence avec les couleurs vives et chaudes du jeu.

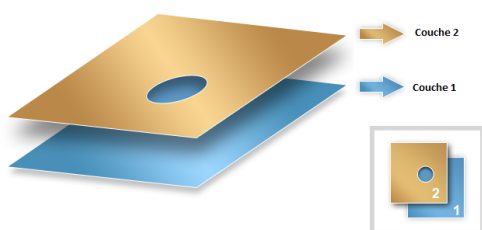


IMAGE 1 COUCHES GRAPHIQUES

Ensuite, on peut discerner en une deuxième couche graphique qui contient toutes les autres images fixes du jeu : tacos, bonus et prison. Il était nécessaire que ces images soient en un format comme le png pour laisser le fond apparent. Cela crée donc une sorte d'objet posé sur le sable comme pour les piments par exemple. Contrairement à la première couche graphique qui reste tout le long du jeu, la deuxième couche varie pendant le jeu. L'affiche de cette deuxième couche dépend donc des informations lues dans la matrice.

Enfin la dernière couche graphique comprend les monstres et le Pacoman. Cette couche est différente des autres car elle est actualisée à chaque déplacement et elle se pose sur les deux couches précédentes. Pour le Pacoman, nous avons créé 8 images (4 directions * bouche ouverte/fermée) pour fabriquer une image réaliste pour chaque déplacement. Par contre nous n'avons pas jugé utile de faire pareil pour les montres. Comme il a été dit précédemment, nous avons personnalisé l'image avec des calques. Ici le Pacoman est un bon exemple :

Calque 1	Calque 2	Ensemble

Nous pouvons donc dire que nous avons accordé une importance non négligeable à l'interface graphique afin d'augmenter le plaisir de jouer.

² <http://openclassrooms.com/courses/debuter-sur-adobe-photoshop/la-fenetre-de-calques>

E. Perspectives d'évolution

I. LIMITES DU CODE

La chasse aux bugs fut longue et difficile puisqu'à chaque ajout de contenu on avait toujours des bugs qui apparaissaient. Beaucoup furent corrigés : on pourra citer par exemple le bug du monstre qui se promène en dehors de la map et qui envahit le terminal. Dans beaucoup de cas la source du bug fut corrigée mais parfois il a été nécessaire de le corriger moins proprement. Les comportements sont par exemple codés de façon à ne jamais faire aller le monstre vers un mur. Cependant dans de rares cas, le monstre prenait quand même la décision d'aller vers un mur. Pour corriger cela, on dit simplement de prendre aléatoirement une autre direction qui fonctionne.

Pendant le développement un bug assez rare persistait. Celui-ci n'a malheureusement jamais été corrigé. Le terminal renvoie l'erreur : EAccessViolation. Une case mémoire est ouverte alors que celle-ci n'est pas définie. Le jeu s'arrête lorsque le bug se produit.

Si on change de fenêtre en cours de partie, le jeu s'arrête aussi de fonctionner et le programme ne répond plus. Ce bug n'est pas vraiment gênant, il vient sûrement de la façon dont on a utilisé la SDL.

Enfin, des erreurs s'affichent dans le terminal et nous ne comprenons pas pourquoi. Elles empêchent de bien voir le menu lorsqu'on a fini une partie.

II. AJOUTS EVENTUELS

Avec le jeu que l'on a créé il est assez facile d'ajouter du nouveau contenu :

On pourrait par exemple ajouter des bonus qui sont sur le sol. Quand on traverserait ce type de bonus, le bonus ne disparaîtrait pas et l'entité qui passe dessus accélère pendant un court laps de temps. Les monstres peuvent alors aussi accélérer. Lors du calcul du chemin le plus court, ces bonus seraient pris en compte pour privilégier les chemins où il y a des bonus.

On peut utiliser des mines qu'on pose sur le terrain. Lorsqu'une entité passe dessus, celle-ci explose : Le Pacoman perd une vie s'il est dans le champ d'action de la bombe, les monstres meurent aussi et les murs autour de la bombe sont détruits. Il faudrait alors recommencer la procédure d'analyse pour que les monstres puissent passer dans les chemins qui ont été créés.

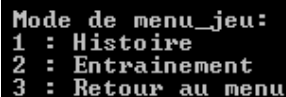
L'ajout d'un éditeur de map est aussi assez simple. A l'heure actuelle on a créé les maps dans un fichier texte. Il faudrait alors proposer une interface graphique pour que cela soit plus ludique.

Les possibilités sont infinies, il faut juste avoir un peu d'imagination.

III. GUIDE D'UTILISATION

A. Jouer

Le sous-menu jouer vous propose différents modes de jeu qui sont les suivants : histoire et entraînement.

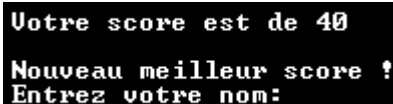


```
Mode de menu_jeu:
1 : Histoire
2 : Entraînement
3 : Retour au menu
```

IMAGE 2 MENU DU JEU

I. HISTOIRE

Le mode Histoire est le mode carrière du programme c'est-à-dire qu'il vous propose un jeu continu. En effet, le mode histoire est un enchainement de 5 cartes avec un niveau différent pour chaque carte. Pour passer au niveau suivant, il est nécessaire de passer le niveau précédent. A chaque début de niveau, le nombre de vies est initialisé à 3 et les bonus du niveau précédent sont remis à zéro. Ce mode de jeu est le mode principal du jeu. Il y a qu'uniquement dans ce mode de jeu que le score est pris en compte. En effet à la fin d'une partie, si vous avez fait un score qui entre dans les 5 meilleurs scores, le programme vous demande alors d'entrer votre nom.



```
Votre score est de 40
Nouveau meilleur score !
Entrez votre nom:
```

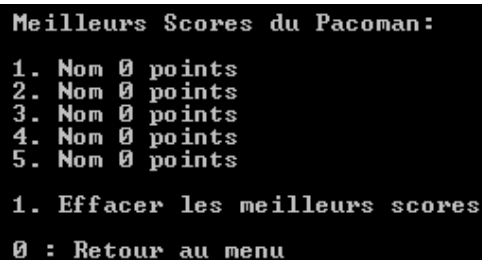
IMAGE 3 FIN DE PARTIE : MODE HISTOIRE (EN CAS DE NOUVEAU MEILLEUR SCORE)

II. ENTRAÎNEMENT

Le mode entraînement est comme son nom l'indique un entraînement pour le mode histoire. L'utilité de ce niveau est de s'améliorer sur les niveaux et cartes du mode histoire afin d'améliorer son score. En effet le mode histoire étant linéaire, il n'est pas possible de s'entraîner sur les derniers niveaux sans faire les niveaux précédents. C'est pourquoi le score du mode entraînement n'est pas comparé aux meilleurs.

B. Meilleurs scores

Le menu meilleurs scores permet de visualiser les meilleurs scores et de réinitialiser le tableau avec la procédure « Effacer les meilleurs scores ».



```
Meilleurs Scores du Pacoman:
1. Nom 0 points
2. Nom 0 points
3. Nom 0 points
4. Nom 0 points
5. Nom 0 points

1. Effacer les meilleurs scores
0 : Retour au menu
```

IMAGE 4 TABLEAU DES MEILLEURS SCORES

C. Jeu

Les deux modes de jeu vous emmènent sur la fenêtre suivante :

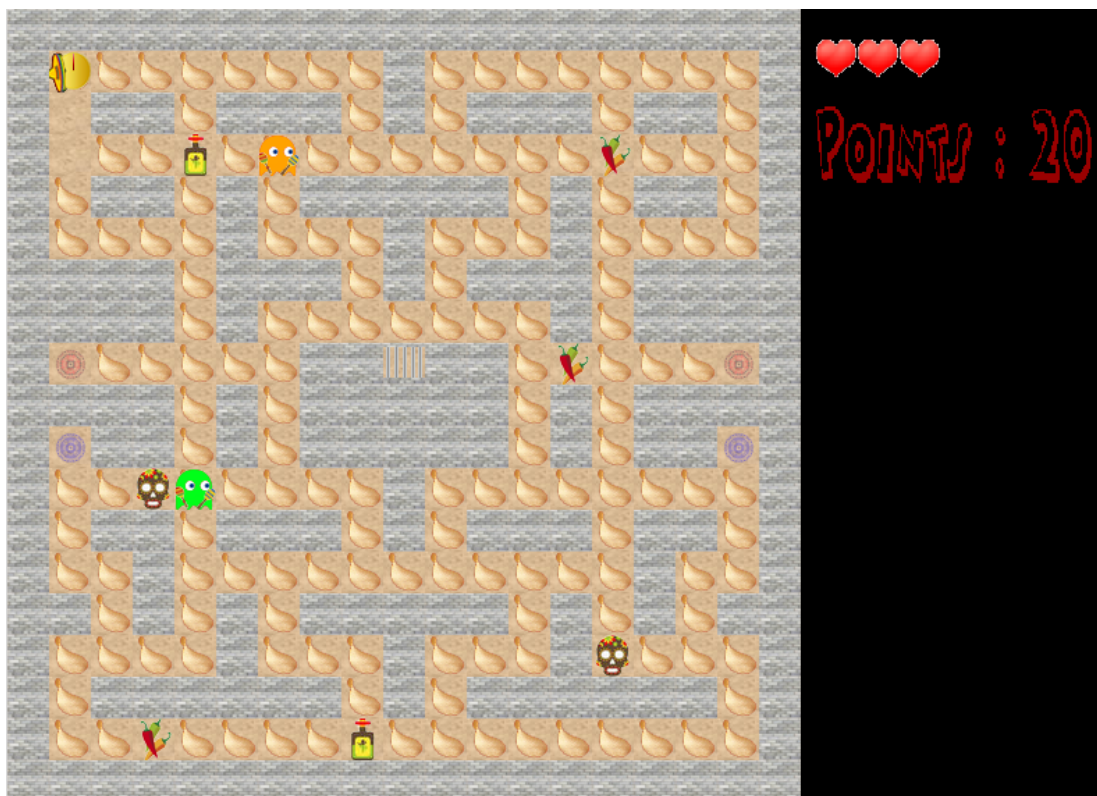









IMAGE 5 INTERFACE DE JEU (MODE HISTOIRE ET MODE ENTRAÎNEMENT)

Maintenant, il va être listé les différentes parties graphiques du jeu :

	Nombre de points
	Nombre de vies
	Bonus de vitesse
	Bonus d'invincibilité
	Bonus de ralentissement
	Monstre comportement agressif
	Monstre comportement aléatoire
	Monstre comportement démarque
	Monstre comportement devine
	Monstre comportement fuite
	Monstre comportement semi-agressif

	Monstre comportement comportement aléatoire
	Pacoman
	Prison
	Switchs
	Tacos
	Jauge du bonus ralentissement
	Jauge du bonus vitesse

Pendant le jeu, vous utilisez les flèches (haut, bas, gauche droite) pour déplacer le Pacoman. Pour utiliser le bonus de vitesse, il faut appuyer sur la touche A et pour le bonus de ralentissement sur la touche Z.

D. Fin de partie

A la fin d'une partie, un menu vous propose différent choix :

```
1. Nouvelle partie
2. Retour au menu
3. Quitter le jeu
```

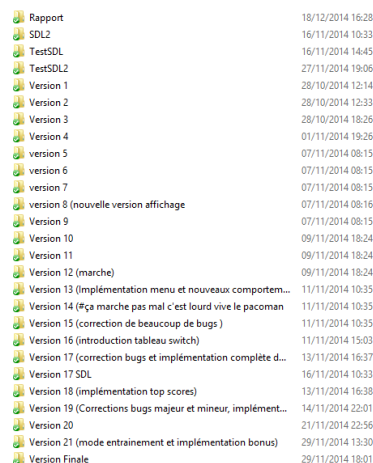
IMAGE 6 ECRAN DE FIN DE PARTIE

Vous êtes maintenant prêt à jouer à Pacoman !

IV. RETOUR SUR LE TRAVAIL EN GROUPE

A. Répartition et organisation du travail

En ce qui concerne l'organisation du travail en groupe, nous avons mis deux systèmes en place. Tout d'abord, pour faciliter la transmission des fichiers, nous avons créé une Dropbox c'est-à-dire un fichier commun en ligne. Ensuite, nous avons pris la décision de faire des versions pour deux raisons : voir l'évolution du projet et avoir une trame à la fin mais aussi avoir des sauvegardes qui compilent et s'exécutent. Cela nous permettait donc d'avoir une sécurité et de ne pas perdre le travail déjà effectué. Dès qu'il y a avait une modification du programme, cela impliquait la création d'une nouvelle version et de sa mise en ligne (une sorte de publication) sur la Dropbox. La limite de ce système est que si deux personnes travaillaient en même temps sur une version alors les modifications effectuées n'étaient pas prises en compte dans la mise en ligne des nouvelles versions. C'est pourquoi nous avons complété ce système par un groupe Facebook. Cette page nous permettait d'informer efficacement et dans la totalité le groupe sur les nouvelles versions ainsi que sur les travaux en cours.



Rapport	18/12/2014 16:28
SDL2	16/11/2014 10:33
TestSDL	16/11/2014 14:45
TestSDL2	27/11/2014 19:06
Version 1	28/10/2014 12:14
Version 2	28/10/2014 12:33
Version 3	28/10/2014 18:26
Version 4	01/11/2014 19:26
version 5	07/11/2014 08:15
version 6	07/11/2014 08:15
version 7	07/11/2014 08:15
version 8 (nouvelle version affichage	07/11/2014 08:16
Version 9	07/11/2014 08:15
Version 10	09/11/2014 18:24
Version 11	09/11/2014 18:24
Version 12 (marche)	09/11/2014 18:24
Version 13 (implémentation menu et nouveaux comportements...	11/11/2014 10:35
Version 14 (#ça marche pas mal c'est lourd vive le pacoman	11/11/2014 10:35
Version 15 (correction de beaucoup de bugs)	11/11/2014 10:35
Version 16 (introduction tableau switch)	13/11/2014 15:03
Version 17 (correction bugs et implémentation complète d...	13/11/2014 16:37
Version 17 SDL	16/11/2014 10:33
Version 18 (implémentation top scores)	13/11/2014 16:38
Version 19 (Corrections bugs majeur et mineur, implément...	14/11/2014 22:01
Version 20	21/11/2014 22:56
Version 21 (mode entrainement et implémentation bonus)	29/11/2014 13:30
Version Finale	29/11/2014 18:01

IMAGE 7 DOSSIER DROPBOX

B. Impressions personnelles sur le projet

I. ADRIEN

En ce qui me concerne, j'ai beaucoup aimé travaillé sur un projet qu'on construit de A à Z, et surtout en groupe ! En effet si chacun d'entre nous avait dû le programmer seul, le jeu n'aurait jamais été aussi élaboré. Chacun y a apporté son savoir-faire. J'ai particulièrement aimé me pencher sur l'intelligence artificielle des monstres : Réfléchir au problème, trouver petit à petit une solution, coder ses idées et voir prendre forme petit à petit le projet. C'est très valorisant d'avoir réussi à faire ce qu'on voulait, et même d'avoir fait plus que ce qu'on pensait être capable de faire. Le rendu est assez satisfaisant, ce qui nous a montré qu'on est capable de faire des choses complexes à l'aide d'outils simple.

L'élaboration de ce projet a été aussi l'occasion d'assimiler complètement toutes les notions qu'on a apprises l'an dernier. Même si je ne veux pas faire ASI, ce projet m'a donné envie de commencer un nouveau projet de ce type.

II. ALEXIS

Ce projet fut très enrichissant car il m'a permis de travailler sur un projet de programmation relativement abouti. En effet, voulant poursuivre mes études dans l'informatique j'ai ainsi pu découvrir ce que pouvait représenter un autre aspect de la programmation, celle de la création d'un petit jeu vidéo.

En effet par l'intermédiaire de ce projet nous avons pu réfléchir sur des notions différentes de celle de la programmation classique d'un programme comme le gameplay par exemple. De plus j'ai également découvert la difficulté que pouvait représenter la programmation en groupe, comme par exemple la nécessité de se mettre d'accord sur des noms de variables et le « versioning ».

Enfin, ce projet m'a permis d'enrichir mes connaissances en terme de programmation avec la découverte des bibliothèques SDL2 et FMOD.

III. DAVID

Ce projet m'a vraiment intéressé d'un côté pour la conception d'un jeu et d'un autre côté par l'aspect projet. En effet, ce qui est plaisant dans la conception d'un jeu c'est que l'on peut tester son programme après avoir implémenté une nouvelle fonctionnalité et donc d'avoir un ressenti direct. Tout au long de l'élaboration du programme, j'ai aimé la liberté d'implémentation des fonctions (même si un cahier des charges était fixé), nous avons créé un jeu de telle sorte à ce qu'il nous plaise mais aussi pour que d'autres personnes s'amuse dessus.

De plus, ce projet m'a permis d'améliorer mes compétences en informatique et d'augmenter mes connaissances. En effet avant le projet, je ne connaissais pas la SDL, ni comment se passait l'interface homme machine au niveau du clavier.

Et pour finir, en ce qui concerne l'aspect projet, ce qui m'a plus est le fait de réfléchir ensemble autour du programme ce qui le faisait avancer à bonne vitesse. Cela était alors entraînant de voir l'évolution du projet.

CONCLUSION

Bien que satisfaisant selon notre point de vue, nous sommes néanmoins conscient que celui-ci possède quelques lacunes. En effet, notre programme fonctionne, l'affichage est correct, nous avons réalisé une intelligence artificielle correcte mais il est vrai que nous aurions certainement pu mieux coder certaines parties de notre programme.

Prenons par exemple l'affichage via SDL. Nous avons réussi à afficher ce que nous voulions à l'écran. Néanmoins la manière dont nous l'avons codé est loin d'être optimale car nous découvrons la librairie, et ainsi dès qu'une solution marchait nous l'avons appliqué sans forcément aller plus loin. Cette anecdote montre bien que même si nous avons découvert de nombreux aspects de la programmation durant ce projet informatique comme l'utilisation de librairies tels que SDL ou FMOD, il faut beaucoup de temps pour apprendre à utiliser pleinement ces librairies et un langage de programmation plus généralement.