PACMAN

Le travail peut être réalisé en binôme, il est déposé dans le répertoire suivant :

Attention, vous ne déposez que le répertoire de votre solution avec tous les fichiers. Si vous voulez apporter une modification, vous pouvez remplacer votre solution.
Ne touchez pas aux solutions de vos camarades….

L'objectif de ce tp est de vous initier au développement sous le framework XNA

Introduction

Le jeu du Pacman

Pacman consiste en un jeu vidéo créé par le concepteur Toru Iwatani pour l'entreprise japonaise Namco. Le but du jeu consiste à déplacer Pacman, un personnage en forme de camembert, à l'intérieur d'un labyrinthe, et de lui faire récolter toutes les pac-gommes qui s'y trouvent en évitant d'être touché par des fantômes.

Les différentes étapes

Description

Pacman, personnage emblématique du jeu, est un personnage en forme de rond jaune doté d'une bouche: il doit manger des pac-gommes et des bonus (sous forme de fruits, et d'une clé à 5000 points) dans un

labyrinthe hanté par quatre fantômes. Quatre pac-gommes spéciales rendent les fantômes vulnérables pendant une courte période au cours de laquelle Pacman peut les manger. Le jeuoriginal comprend 255 labyrinthes différents (le jeu était considéré comme allant à l'infini, mais l'écran se brouillait dès le 256e niveau).Avec XNA, Microsoft est le premier constructeur à ouvrir la porte au développement indépendant sur sa console Xbox 360. Les jeux produits sont distribués via le Xbox Live.

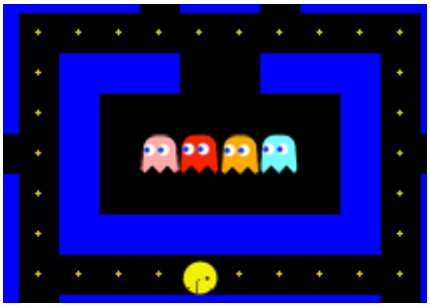
Un programme tel qu'un Pacman demande une démarche qui s'apparente à une analyse descendante. Cette dernière nous conduit à définir les étapes suivantes :

- Le plateau de jeu : labyrinthe – mur
- Ajout du Pacman et gestion de ses déplacements;
- Ajout des fantômes:
  - Intelligence artificielle pour les déplacements : algorithme de Dijkstra
  - Gestion des rencontres avec le Pacman.
- Ajout des bonus;
- Ajout des animations pour rendre le jeu un peu plus vivant.

Les caractéristiques du jeu

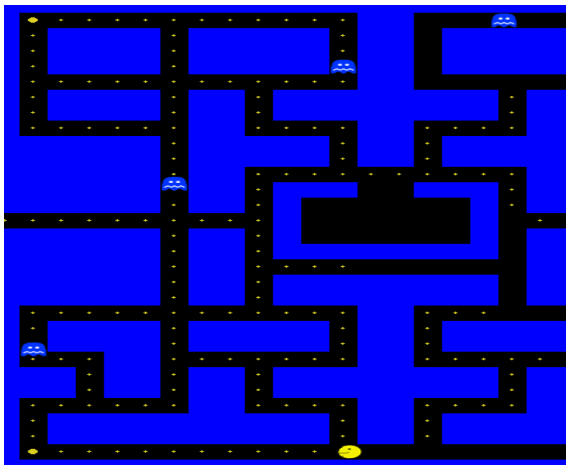
Au début du jeu, le Pacman a n vies (de préférence 3) et commence au niveau 1 avec 0 point.

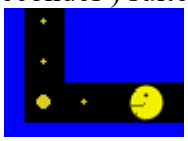
Les fantômes



Les fantômes, au nombre de 4, se déplacent aléatoirement sur le plateau. S'ils sont en mode agressif (de couleur) et si le joueur se trouve sur leur chemin, ils le traqueront. Si l'un d'entre eux le touche, le joueur perd une vie. La vitesse de déplacement des fantômes augmente de niveau en niveau afin d'accroître la difficulté des épreuves.

Vulnérabilité des fantômes



Lorsqu'ils changent de couleur, ilsdeviennent alors vulnérables pendant un laps de temps prédéfini (quelques secondes) suite à l'avalement par Pacman d'une pac –gomme spéciale (pastille de couleur située aux coins) 
Avaler un fantôme durant ce temps, crédite le joueur d'un nombre de points important (200 pts) . Attention, une fois ce délai passé, les fantômes peuvent redevenir agressifs (clignotement préalable de 1 à 2 secondes).

Les pac-gommes (pastilles blanches) avalées rapportent des points (10 pts) au compétiteur. Quand il n'y en a plus ni des blanches ni des rouges, celui-ci franchit le niveau et poursuit avec davantage de complexité.

Bonus du jeu (option facultative)

Il est possible de faire apparaître aléatoirement, durant un laps de temps (4 à 5 s) au centre de l'écran des fruits (cerise, orange et fraise) qui peuvent rapporter respectivement 200, 400 et 600 points.
A chaque palier de 5000 points, le joueur gagne une vie. Le nombre de niveaux est illimité, le plateau ne change pas mais les fantômes vont de plus en plus vite.

Fin

Une fois la partie perdue, votre score s'affiche et le compétiteur est renvoyé au menu du jeu.

Construction d'un labyrinthe

La construction d'un labyrinthe n'est pas une chose facile. Plusieurs solutions s'offrent à vous :

- Construction aléatoire d'un labyrinthe (difficile à mettre en œuvre)
- Dessin de labyrinthe et chargement des objets à l'écran
- Dessin d'un labyrinthe dans une matrice et affichage des objets correspondants
- …

Nous utiliserons la troisième méthode . Notre labyrinthe sera dessiné dans une matrice de bytes bdont la signification est :

- (0 pour un mur),
- 1 pour un bean (haricotà manger),
- 2 rien à afficher

Voici un exemple de matrice

```
map = new byte[VX, VY]{
```

```
            {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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 2, 2, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
            {0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0},
            {0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
            {0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
            {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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},

        };
```

Cette matrice est à insérer dans le constructeur de la classe du jeu Game1

Objets du labyrinthe

Il faut à présent définir les objets qui vont composer le labyrinthe à savoir :

- Le mur
- Les beans

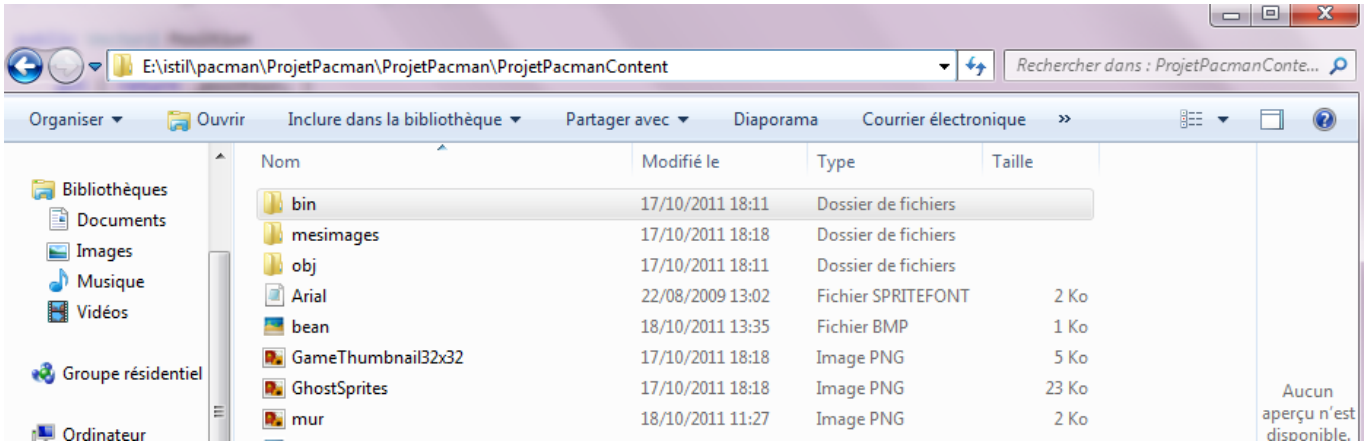
Ces objets sont de type <Texture2D> mais il faut leur donner une position pour les afficher. Nous définissons donc une classe ObjetAnime dont l'impémentation est :

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Xna.Framework.Graphics;   //   for Texture2D
using Microsoft.Xna.Framework;  //  for Vector2

namespace  ProjetPacman
{
    class ObjetAnime
    {
        private Texture2D _texture;    //  sprite texture

        public Texture2D Texture
        {
            get { return _texture; }
            set { _texture = value; }
```

```
        }
        private Vector2 _position;     //  sprite position on screen

        public Vector2 Position
        {
            get { return _position; }
            set { _position = value; }
        }
        private Vector2 _size;         //  sprite size in pixels

        public Vector2 Size
        {
            get { return _size; }
            set { _size = value; }
        }

        public ObjetAnime(Texture2D texture, Vector2 position, Vector2 size)
        {
            this._texture = texture;
            this._position = position;
            this._size = size;
        }
    }
}
```

Le mur

C'est un objet de type <texture2D> qui sera une instance de notre classe ObjetAnime. Nous partons d'un fichier **mur.png** représentant une image de 20 pixels sur 20. Nous l'incorporons à notre projet dans le répertoire



Chargement

Son chargement se fait dans la procédure

```
protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            //  changing the back buffer size changes the window size (when in windowed mode)
           graphics.PreferredBackBufferWidth = 1024;
            graphics.PreferredBackBufferHeight = 660;
           graphics.ApplyChanges();
```

```
            // on charge un objet mur
            mur = new ObjetAnime(Content.Load<Texture2D>("mur"), new Vector2(0f, 0f), new
Vector2(20f, 20f));
        }
```

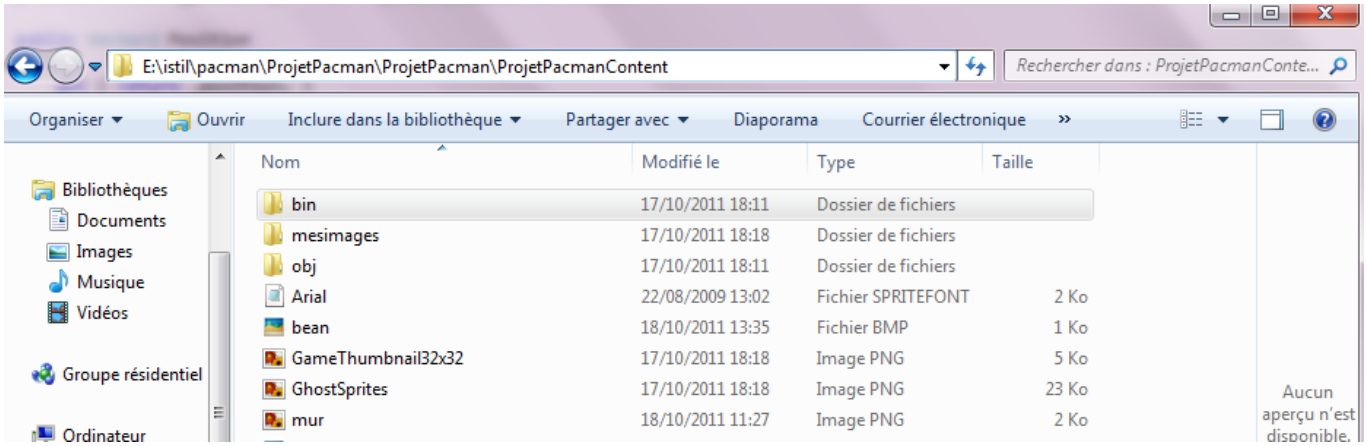
Affichage du mur

L'affichage se fait dans la procédure Draw

```
protected  override void Draw(GameTime gameTime)
{

  for (int x = 0; x < VX; x++)
            {
                for (int y = 0; y < VY; y++)
                {
                    if (map[x, y] == 0)
                    {
                        int xpos, ypos;
                        xpos = x * 20;
                        ypos = y * 20;
                        Vector2 pos = new Vector2(ypos, xpos);
                        spriteBatch.Draw(mur.Texture, pos, Color.White);
                    }
```

Le bean (haricot à manger)

Comme pour le mur, un bean est un objet de type <texture2D> qui sera une instance de notre classe ObjetAnime. Nous partons d'un fichier **bean.png** représentant une image de 13 pixels sur 12. Nous l'incorporons à notre projet dans le répertoire



Chargement

Son chargement se fait dans la procédure

```
protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            //  changing the back buffer size changes the window size (when in windowed mode)
           graphics.PreferredBackBufferWidth = 1024;
```

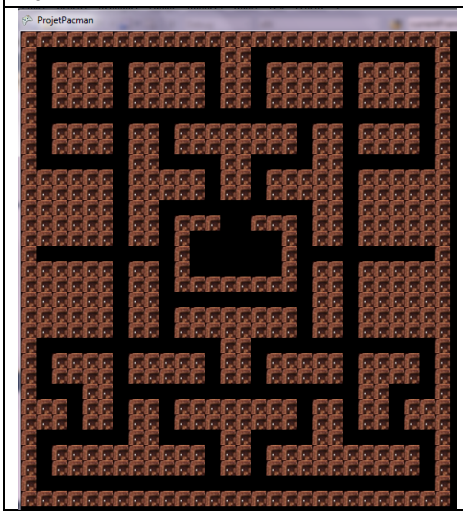
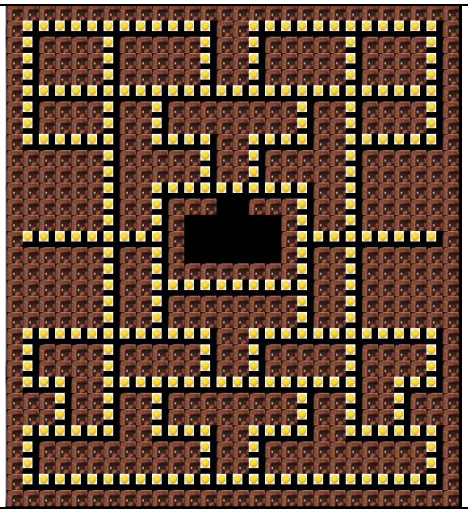
```
            graphics.PreferredBackBufferHeight = 660;
           graphics.ApplyChanges();
            // on charge un objet mur
            mur = new ObjetAnime(Content.Load<Texture2D>("mur"), new Vector2(0f, 0f), new
Vector2(20f, 20f));
            bean = new ObjetAnime(Content.Load<Texture2D>("bean"), new Vector2(0f, 0f), new
Vector2(20f, 20f));

       }
```

Affichage du bean

L'affichage se fait dans la procédure Draw

```
protected  override void Draw(GameTime gameTime)
{

  for (int x = 0; x < VX; x++)
            {
                for (int y = 0; y < VY; y++)
                {
                    if (map[x, y] == 0)
                    {
                        int xpos, ypos;
                        xpos = x * 20;
                        ypos = y * 20;
                        Vector2 pos = new Vector2(ypos, xpos);
                        spriteBatch.Draw(mur.Texture, pos, Color.White);
                    }
                 }
                    else
                        if (map[x, y] == 1)
                        {
                            int xpos, ypos;
                            xpos = x * 20;
                            ypos = y * 20;
                            Vector2 pos = new Vector2(ypos, xpos);
                            spriteBatch.Draw(bean.Texture, pos, Color.White);
                        }
                }
```

Affichage

Mur	Mur & bean
	

Affichage et animation de Pacman

Pacman représente un objet animé de type Texture2D. Sa taille est de 18 sur 16. Son image est définie dans un fichier pacman.png. Son chargement se fait comme pour le « mur » et le « bean ».

Pacman va être dirigé par les flèches de la manière suivante :

```
KeyboardState keyboard = Keyboard.GetState();
    if (keyboard.IsKeyDown(Keys.Right))
      {
          direction = "Right";
                //on vérifie s'il est possible de se déplacer
                // si oui on avance
```

Déplacement des fantômes

Principes de l'algorithme de Dijkstra

Publié en 1959, l'algorithme de Dijkstra est une alternative à celui de Floyd, alternative plus complexe, mais également beaucoup plus rapide. En fait, le résultat fourni par Dijkstra n'est pas tout à fait le même que celui fourni par Floyd : on se fixe un sommet source, et Dijkstra donne tous les chemins les plus courts de ce sommet source à chacun des autres sommets.

Description de l'algorithme [http://www.nimbustier.net/publications/djikstra/djikstra.html]

L'algorithme de Dijkstra est un algorithme de type glouton : à chaque nouvelle étape, on traite un nouveau sommet. Reste à définir le choix du sommet à traiter, et le traitement à lui infliger, bref l'algorithme...

Tout au long du calcul, on va donc maintenir deux ensembles :

- *C*, l'ensemble des sommets qui restent à visiter ; au départ *C*=*S*-{*source*}
- *D*, l'ensemble des sommets pour lesquels on connaît déjà leur plus petite distance à la source ; au départ, *D*={*source*}.

L'algorithme se termine bien évidemment lorsque *C* est vide.

Pour chaque sommet *s* dans *D*, on conservera dans un tableau *distances* le poids du plus court chemin jusqu'à la source, et dans un tableau *parcours* le sommet *p* qui le précède dans un plus court chemin de la source à *s*. Ainsi, pour retrouver un chemin le plus court, il suffira de remonter de prédécesseur en prédécesseur jusqu'à la source, ce qui pourra se faire grâce à un unique appel récursif (beaucoup moins coûteux que dans le cas de Floyd...).

Initialisation

Au début de l'algorithme, le chemin le plus court connu entre la source et chacun des sommets est le chemin direct, avec une arête de poids infini s'il n'y a pas de liaison entre les deux sommets. On initialise donc le tableau *distances* par les poids des arêtes reliant la source à chacun des sommets, et le tableau *parcours* par *source* pour tous les sommets.

*i*ème étape

On suppose avoir déjà traité *i* sommets, *parcours* et *distances* contiennent respectivement les poids et le prédécesseur des plus courts chemins pour chacun des sommets déjà traités.

Soit *s* le sommet de *C* réalisant le minimum de *distances*[*s*]. On supprime *s* de *C* et on l'ajoute à *D*. Reste à mettre à jour les tableaux *distances* et *parcours* pour les sommets *t* reliés directement à *s* par une arête comme suit : si *distances*[*s*] + *F*(*s*,*t*) < *distances*[*t*], alors on remplace *distances*[*t*] par *distances*[*s*] + *F*(*s*,*t*) et *parcours*[*t*] par *s*... et c'est tout !

(*n*-2)ème étape

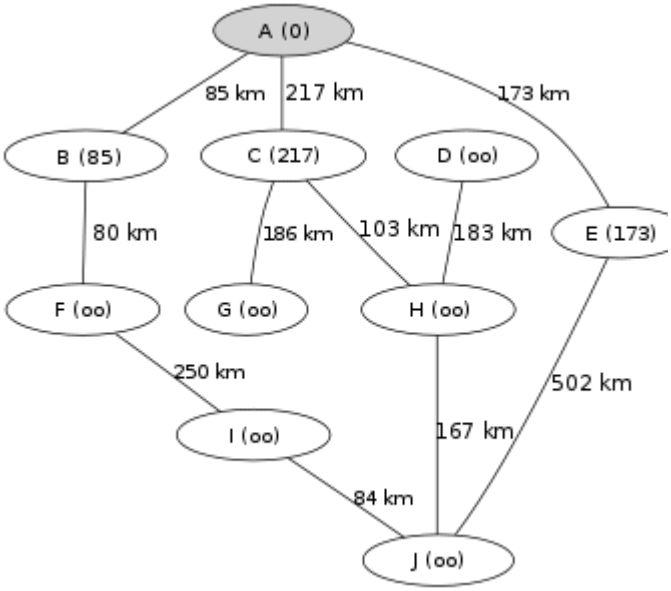
Au départ, il y a (*n*-1) sommets à visiter, mais comme on le verra ci-après, la dernière étape est inutile puisqu'elle n'apporte rien. Ainsi, dès la (*n*-2)ème étape, *distances* et *parcours* contiennent toute l'information nécessaire pour trouver des plus courts chemins de la source à chacun des autres sommets (car alors *D*=*S*):

- *distances*[*s*] est le poids du plus court chemin de la source à *s*
- *parcours*[*t*] est le prédécesseur de *s* dans un plus court chemin de la source à *s*

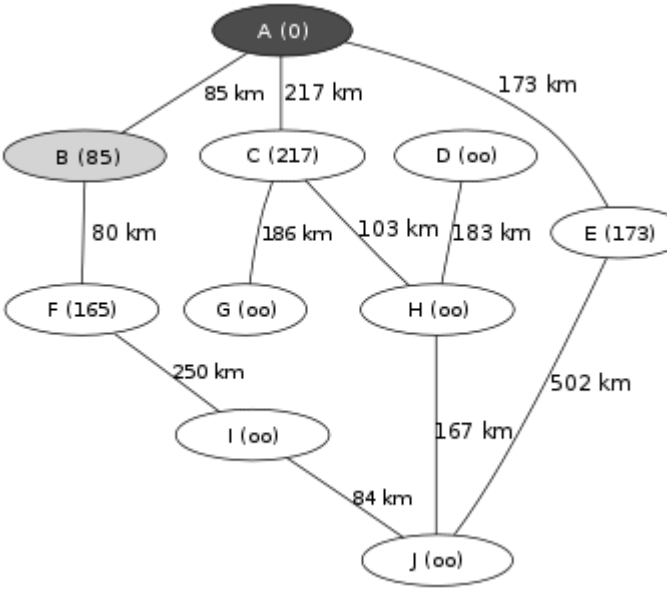
Exemple [source wikipedia : http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra]

Distance entre la ville A et la ville J

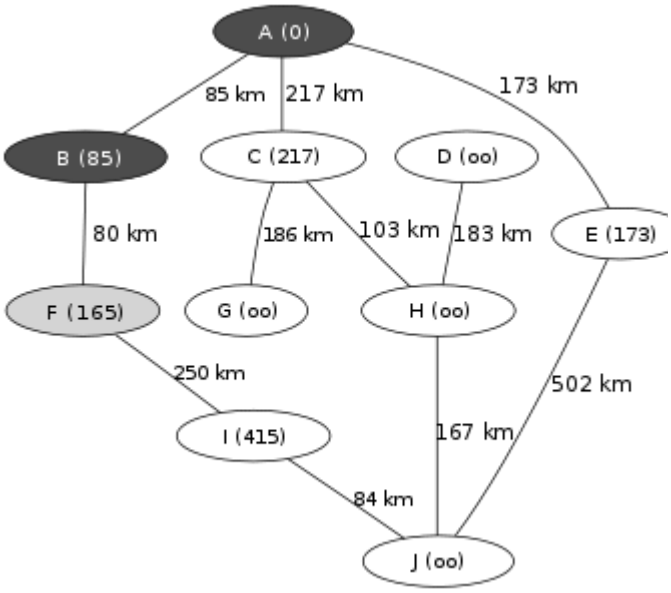
L'exemple suivant montre les étapes successives dans la résolution du chemin le plus court dans un graphe. Les nœuds symbolisent des villes identifiées par une lettre et les arêtes indiquent la distance entre ces villes. On cherche à déterminer le plus court trajet pour aller de la ville A à la ville J.



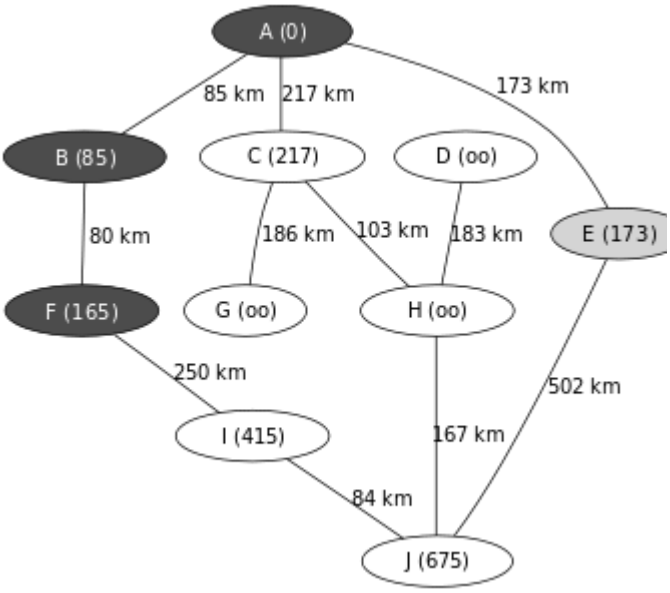
Étape 1 : à partir de la ville A, 3 villes sont accessibles, B, C, et E qui se voient donc affectées des poids respectifs de 85, 217, 173, tandis que les autres villes sont affectées d'une distance infinie.



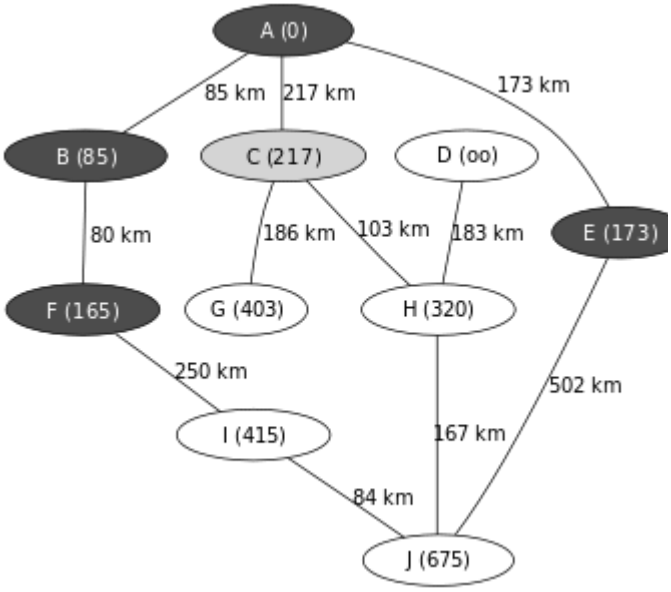
Étape 2 : la distance la plus courte est celle menant à la ville B. Le passage par la ville B ouvre la voie à la ville F (85+80 = 165).



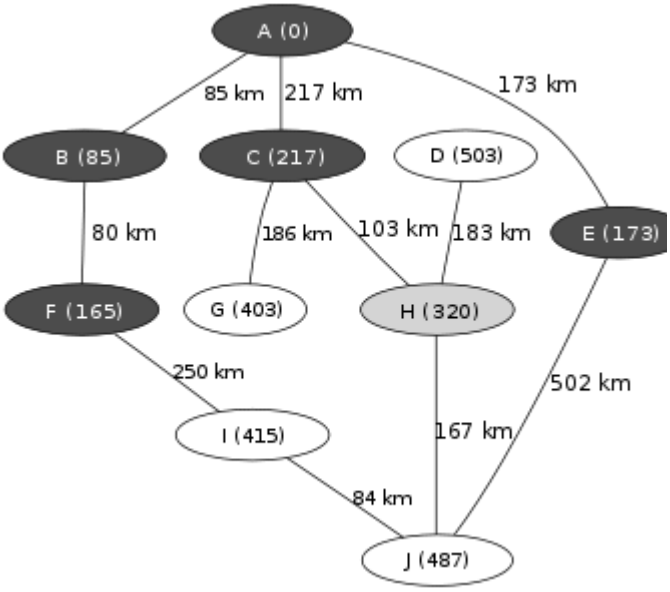
Étape 3 : La distance la plus courte suivante est celle menant à la ville F. Le passage par la ville F ouvre une voie vers la ville I (415).



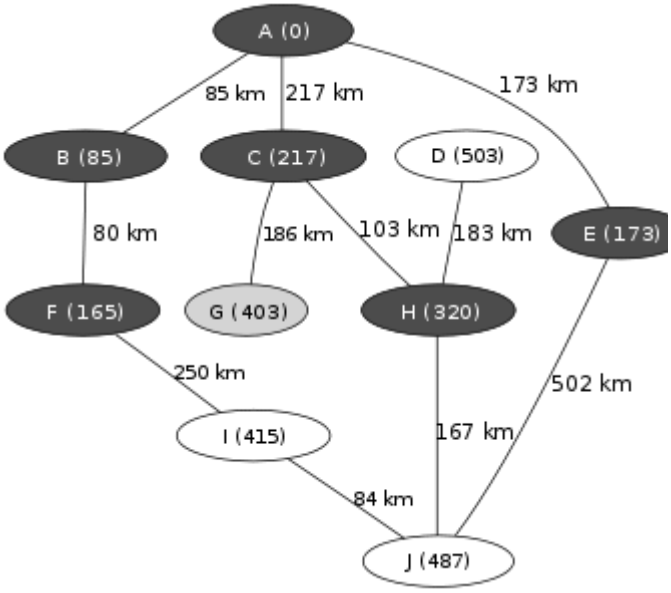
Étape 4 : La distance la plus courte suivante est alors celle menant à la ville E. Le passage par la ville E ouvre une voie vers la ville J (675).



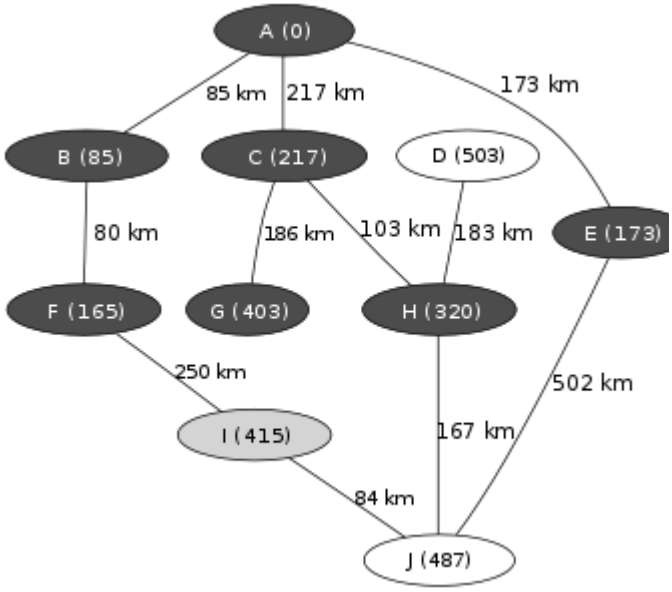
Étape 5 : la distance la plus courte suivante mène alors à la ville C. Le passage par la ville C ouvre une voie vers la ville G (403) et la ville H (320).



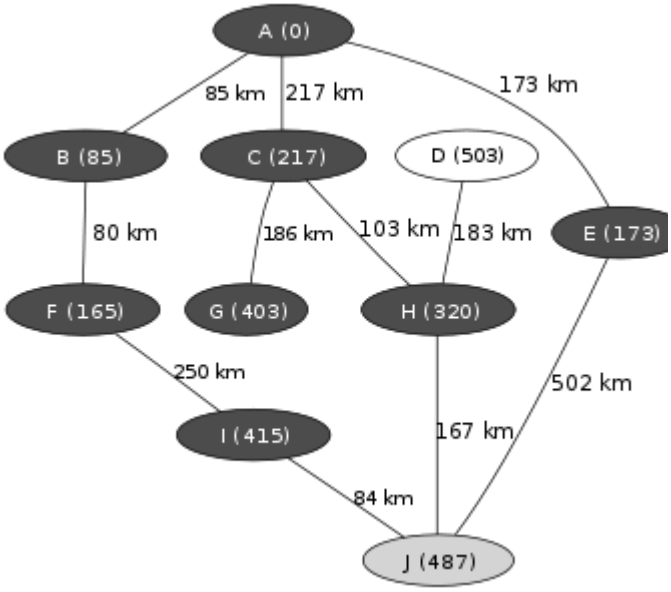
Étape 6: la distance la plus courte suivante mène à ville H(320). Le passage par la ville H ouvre une voie vers la ville D et un raccourci vers la ville J (487< 675).



Étape 7 : la distance la plus courte suivante mène à la ville G et ne change aucune autre distance.



Étape 8 : la distance la plus courte suivante mène à la ville I. Le passage par la ville I ouvre un chemin vers la ville J qui n'est pas intéressant (415+ 84 > 487).



Étape 9 : la distance la plus courte suivante mène à la ville J (487).

On connait ainsi le chemin le plus court menant de A à J, il passe par C et H et mesure 487 km.

Application de l'algorithme à notre situation de jeu

On connaît :

- les coordonnées du fantôme
- les coordonnées du pacman

On veut connaître :

- la direction que doit prendre le fantôme pour atteindre le plus vite possible le pacman

Principe de l'algo

// on définit les classes suivantes

```
class Sommet
    {
        public const int INFINI = 1000000;
        public int Potentiel;
        public bool Marque;
        public Coord Suivant;

        public Sommet()
        {
            Potentiel = INFINI;
            Marque = false;
            Pred = null;
        }
    }

class Coord
    {
        public int X, Y;

        public Coord(int x, int y)
        {
            X = x;
            Y = y;
        }

        public Coord(Vector2 pos)
        {
            X = (int)pos.X / 16;
            Y = (int)pos.Y / 16;
        }

        // on surcharge l'opérateur == pour l'égalité entre les coordonnées
        public static Boolean operator==(Coord c1, Coord c2)
        {
            return ((c1.X == c2.X) && (c1.Y == c2.Y));
        }
        // on surcharge l'opérateur == pour la différence entre les  coordonnées
        public static Boolean operator!=(Coord c1, Coord c2)
        {
            return ((c1.X != c2.X) || (c1.Y != c2.Y));
        }
    }
```

// Phase d'initialisation de tous les sommets de notre matrice

```
pour chaque case de la matrice
  si non mur
    alors
      mesSommets[i,j] <- new Sommet
 finsi
finpour
```

// le sommet d'arrivée devient le sommet courant
// On met son potentiel à 0
// boucle
  Tant que (couarnt non égal sommet_depart)
    // On crée un sommet Z égal au sommet_courant

```
 // On le marque
// On recherche une direction
   Exemple
   Si HAUT est possible
    alors
      Si sommet non nul
         Alors
          On crée un sommet s égal à sommet_courant.X, sommet_courant.Y-1
            Si (s.getPotentiel ( )> z.getPtentiel ( )+1)
               Alors
                  s.pred  = sommet_courant
             finsi
       finsi
      finsi
   minimum = Sommet.INFINI
   boucle sur la largeur
      boucle sur la hauteur
        si non mesSommet[i,j].Marque && sommet.getPotentiel()  <  minimum
         alors
             minimum = mesSommet[i,j].getPotentiel ()
              sommet_courant = new Coord(i,j]
        finsi
     fin pour
finpour
```

On peut affecter la direction du fantôme en fonction des coordonnées de la case précédent la case courante trouvée.

	Travail à faire
2.1	Réaliser un jeu Pacman - Vos fantômes seront placés au centre de l'écran - Vos pacs-gommes sont laissés à votre initiative Remarque : vous pouvez modifier : - Le mur – le bean - Le labyrinthe Option - Gestion des bonus - Algorithme de Dijkstra pour la gestion des déplacements des fantômes.

Simulation de jeu :


Pacman
Score : 1070
Vies : 2
Etat : 0