Dimanche 7 Mai 2023

Université de Strasbourg

Projet Programmation Avancée
Réalisation du jeu PAC-MAN (Namco 1980)

Contributeurs :
Jardot Charles
Derousseaux Nathanaël

Introduction :

Pour mener à bien le projet PAC-MAN, nous avons décidé de nous organiser de la manière suivante : Nathanaël Derousseaux (moi-même), organise la structure du projet, l'affichage et les collisions au niveau des différentes instances. Charles Jardot s'est occupé de la gestion des fantômes et a également participé à la structure du projet.

En ce qui concerne les attendus du projet, la jouabilité, la gestion des fantômes ainsi que les différentes animations (fin de niveau, déplacements, ...) ont bien été implémentées. La gestion du score n'a pas été quant à elle implémentée.

Ce rapport s'articule en deux temps. D'abord, nous présenterons les différents choix de développement du jeu puis, nous pourrons conclure avec la structure du projet, la gestion de l'affichage et des collisions.

I / Choix d'implémentation

Pour gérer les différents objets sur affichés sur la surface SDL, nous avons décidé de créer un objet Window, qui a comme attribut un container. Ce container est une liste d'objets *éléments* (ou classes hérités). On parcourt la liste à chaque itération du jeu, et on appelle pour chaque instance d'*élément* la fonction *update*.
Lors de l'appel de cette fonction, l'élément réagira comme il doit : pacman bougera selon les inputs claviers, changera son sprite, etc…, les fantômes exécuteront leur algorithme de chasse et le terrain gérera les collisions avec les pac-gommes.

Un tour de jeu dure 16ms.
Nous avons décidé de donner à chaque objet *movable* (classe héritée d'élément), la possibilité de se déplacer de X pixels par tour. Pacman peut bouger de 4 pixels par tour, et les fantômes de 3 pixels en mode CHASE.

II / Tâches personnelles

a / Structure du projet

Le projet est structuré comme suit:

- **Classe Window:** Classe qui gère la fenêtre, elle contiendra la liste de tous les éléments à afficher et gére les événements SDL. C'est la classe racine du projet.
- **Classe Game:** Classe qui gère le jeu, elle s'occupera des règles du jeu, et managera les autres classes. C'est elle qui dit à la classe field quand régénérer les pac gommes (partie finie) ou qui ordonne aux fantômes de se remettre à leur point de départ.

Ici on voit que la boucle principale de game appelle element->update() toutes les 16ms.

```
void Game::main_loop() {
   bool running = true;
   while (running) {

    // Gestion des évènements
    running = Window::get_instance()->handle_events();
    if (!running)
      break;

    // Fait "réagir" chaque élément
    for (Element * element : Window::get_instance()->get_elements())
      element->update();

    // On met à jour la fenêtre
    Window::get_instance()->update();

    // On attend 16ms
      SDL_Delay(16);
   }
}
```

- **Classe Element :** Classe d'un élément. Contient la position de l'élément et quelques méthodes virtuelles (react, set_sprite, etc…)

```
// Met à jour l'élément (react + animate)
void Element::update() {
  // On ne réagit pas si le jeu est en pause
  if (Game::get_instance()->get_state() != GAME_PAUSE)
    react();
  animate();
}
```

La fonction update appelle react, qui est redéfinie pour chaque élément et qui permettra à l'élément de réagir à son environnement (prendre les entrées clavier pour pacman ou changer de direction pour les fantômes). Certains éléments ne réagissent pas, comme field. Animate sera décrite plus loin.

- **Classe Field :** Classe fille de élément, symbolise le terrain. Elle contient les sprites du terrain. Elle s'occupe aussi d'initialiser et de gérer les pac gommes sur le terrain.

- **Classe Dot :** Classe fille de élément, symbolise une pac gomme. Elle contient un sprite, et une méthode statique contenant la position de toutes les pacgommes.
- **Classe Gomme:** Classe fille de élément, symbolise une super pac gomme. Elle contient un sprite et une méthode statique contenant la position de toutes les super pac gommes.
- **Classe Moveable :** Classe fille de élément, symbolise tous les éléments déplaçables. Elle surcharge la fonction *update* pour ajouter une sous fonction : move (désactivée si l'attribut state de *game* est en pause). Elle possède aussi la direction de l'élément, la prochaine intersection à atteindre, etc… Une méthode statique permet de mesurer l'écart entre deux éléments.
    - **Classe Fantom :** Classe fille de moveable, représente un fantôme. Outre les surcharges habituelles de éléments (sprites, animations, etc…), elle contient la fonction de déplacement des fantômes. Plusieurs méthodes statiques permettent de gérer le minuteur SDL qui switch entre les différentes phases des fantômes (SCATTER puis CHASE).
        - **Classe Blinky :** Classe fille de Fantom, représente le fantome rouge. Il suit pacman.
        - **Classe Pinky :** Classe fille de Fantom, représente le fantome rose. Il essaye de se mettre 4 cases devant pacman.
        - **Classe Clyde :** Classe fille de Fantom, représente le fantôme orange. Il fuit Pacman s' il prêt de plus de 8 cases de lui, et sinon il suit pacman comme Blinky.
        - **Classe Inky :** Classe fille de Fantom, représente le fantôme bleu. Il tire un vecteur entre lui, et 2 cases devant pacman, puis multiplie ce vecteur par 2. Il tente de rejoindre la destination de ce vecteur.
    - **Classe Pacman:** Classe fille de moveable. Représente pacman. Il possède des fonctions pour réagir aux entrées utilisateur, pour détecter les collisions avec les pacgommes et les fantômes.

b / Gestion des sprites

Comme dit plus haut, chaque instance de la classe élément possède une liste de SDL_Rect. Cette liste est statique, et donc commune à toute la classe. Mais cette liste est surchargée pour chaque classe fille de élément.

Chaque SDL_Rect représente la position et la taille du sprite sur la liste des sprites. Ensuite, dans la fonction principale de game, game pourra faire *element.get_sprite* pour afficher le bon sprite à l'écran.
Chaque élément possède une fonction *animate*, qui s'exécute à chaque tour de jeu. Chaque élément possède aussi un attribut animation permettant d'enregistrer sa phase dans son animation.

Cette phase peut aussi être modifiée en fonction de l'état du jeu.
À l'issue d'une animation, on peut aussi effectuer certaines actions, comme redémarrer la partie. Par exemple dans l'animation de field :

```
// Change le sprite du terrain
void Field::animate() {
   // Si on est en train de jouer, le sprite est 1.
   if (Game::get_instance()->get_state() != GAME_WIN )
      set_current_sprite(1);

   // Sinon, on a gagné, le sprite clignote
   else if (Game::get_instance()->get_state() == GAME_WIN) {

      // On change le sprite toutes les 10 frames
      int phase = (_animation / 10)%2;

      set_current_sprite(phase + 1);

      // Si on a fini l'animation, on recommence le jeu
      if (_animation == 100) {
         _animation = 0;
         Game::get_instance()->restart(true);
         return;
      }
      _animation++;
   }
}
```

c / Les colisions

Les collisions sont gérées par Game. Cette classe possède une méthode qui vérifie que 2 éléments n'ont pas les sprites qui se touchent, en prenant en compte leur position et la taille des sprites. Cela à demandé un découpage consciencieux et laborieux des sprites.

```cpp
// Vérifie si l'élément est en collision avec un autre élément
Element * Game::check_collision(Element * element) {
  // On regarde si il y a un élément à la même position
  for (Element * other : Window::get_instance()->get_elements()) {
    //Si l'élément n'est pas lui même, et qu'il n'est pas un champ
    if (other == element || dynamic_cast<Field*>(other) != nullptr)
      continue;
    if (
      element->get_pos()->x + element->get_pos()->w >= other->get_pos()->x &&
      element->get_pos()->x <= other->get_pos()->x + other->get_pos()->w &&
      element->get_pos()->y + element->get_pos()->h >= other->get_pos()->y &&
      element->get_pos()->y <= other->get_pos()->y + other->get_pos()->h
    )
      return other;
  }
  return nullptr;
}
```

III / Bibliographie

[1] Chad Birch, Understanding Pac-Man Ghost Behavior 2010, https://gameinternals.com/understanding-pac-man-ghost-behavior

[2] Wiki de la librairie SDL2 , https://wiki.libsdl.org/SDL2/FrontPage