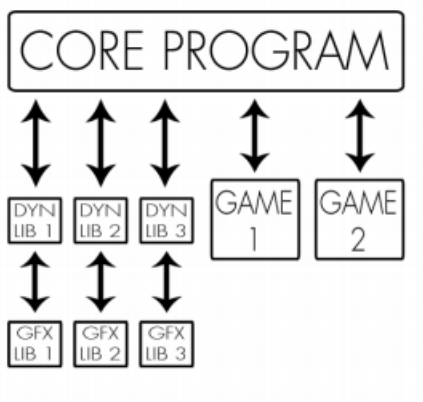
Projet Arcade

Le projet Arcade consiste à créer une plateforme de jeu : un programme qui permet à l'utilisateur de choisir un jeu auquel il peut jouer et qui tient un registre des scores des joueurs.

Pour cela, les bibliothèques graphiques et nos jeux doivent être implémentés en tant que bibliothèques dynamiques, chargées au moment de l'exécution.

Le projet :

Notre arcade comprend principalement en racine du repository un dossier *Arcade* comprenant l'intégralité de notre code, une makefile, exécutant les makefiles dans notre *./Arcade* servant à la création des librairies. Et un dossier lib, où sera stocker nos différentes librairies.



Le Dossier « ./Arcade » :

Le dossier Arcade est lui-même composer de 3 dossiers.

-Un dossier *Core*, qui fera l'appelle des librairies dynamiques.

-Un dossier *Game*, où sont implémentés les jeux.

-Un dossier *Graphics*, où sont implémentées les librairies graphiques.

Le Dossier « ./Core » :

Le Core ouvre un menu selon le graphique choisi. Grace à une fonction qui lui permet de trouver les jeux et les graphiques existent, le menu nous les montre et on peut choisir l'un des libraires possibles. Si on choisit une librairie graphique, le Core ferme le menu et le graphique qui va avec et ouvre un nouveau menu avec le graphique choisi. Si c'est un jeu qui est sélectionné, le Core fait ouvrir la librairie du jeu en question et initialise d'abord la map du jeu. Après cela il détecte chaque changement dans la map et fait la "mise a jour" de la map dans l'écran jusqu'à qu'on appuie dans le bouton quitter (Q).

Le Dossier « ./Game »:

Le dossier Game comprend deux dossiers :

- Nibbler : est un simple jeu vidéo d'arcade sorti en 1982. Son concept s'est répandu principalement grâce au jeu culte Snake.
- Pacman : est un jeu vidéo d'arcade sorti en 1980. Le but est d'explorer un labyrinthe afin de manger tous les "pacgums" qui s'y trouvent tout en évitant les fantômes.

Ainsi qu'une interface *IGame.hpp.*

L'interface va servir de classe abstraite afin que les classe Pacman (et Ghost) et Nibler, puissent en hériter. Ainsi en plus des méthodes qui seront propre au jeux Pacman et Nibbler, tous deux hériteront de l'interface IGame. Cet héritage est fondamental pour pouvoir utiliser les librairies dynamiquement et permettre à d'autre libraire quelconque d'être utiliser.

IGame :

```
class IGame
  {
     private:
        /* data */
     public:
                        virtual void                inputEvent(Arcade::Input input) = 0;
                        virtual char                 **setArray2d() = 0;
                        virtual void                reset() = 0;
                        virtual char                 **getGame(Arcade::Input in)=0;
                        virtual                    ~IGame() = default;
                        virtual void                printMap() = 0;
                        virtual void                init() = 0;
                        virtual unsigned int        getScore() = 0;
                        virtual unsigned int        getPV() = 0;
                        virtual char **             update() = 0;
   };
```

Afin de d'implémenter une libraire de jeux qui sera compatible avec notre système, il est essentiel de respecter l'utilisation des capacités décrite dans l'interface. Les principales méthodes sont :

-*inputEvent* : permet de récupérer les touches qui seront récupérer par le core depuis la libraire graphique et d'y attribuer une action.

-*setArray2d* : permet de convertir la map du jeu, en un format utilisable pour les librairies graphiques

-*reset* : permet de remettre les paramètres à leur état d'origine.

-*getGame/uptade* : sont les méthodes principales qui seront récupérer par le core afin de faire la partie communication entre map et event.

-*getScore* : permet de récupérer les scores de chaque jeu.

Le Dossier « ./Graphics » :

Le dossier Graphics comprend les éléments pour l'implémentation de deux librairies :

-Ncurses : est une bibliothèque libre fournissant une API pour le développement d'interfaces utilisateur à menu déroulant (GUI), en utilisant les caractères et couleurs d'un mode semi-graphique.

-SFML : est une interface de programmation destinée à construire des jeux vidéo ou des programmes interactifs.

Ainsi qu'une interface *IGraphics.hpp.*

L'interface va servir de classe abstraite afin que les classe Ncurse et SFML, puissent en hériter. Ainsi en plus des méthodes qui seront propre au jeux Pacman et Nibbler, tous deux hériteront de l'interface IGraphics. Cet héritage est fondamental pour pouvoir utiliser les librairies dynamiquement et permettre à d'autre libraire quelconque d'être utiliser.

IGraphics :

```
class IGraphic
  {
    private:
      /* data */
    public:
      virtual                   ~IGraphic() = default;
      virtual std::string       createWindow(std::vector<std::string> files, std::vector<std::string> filesGame) = 0;
      virtual Arcade::Input     getInput() = 0;
      virtual std::string       GameName(std::string files) = 0;
      virtual std::string       getNameLib() = 0;
      virtual void              initMap() = 0;
      virtual void              initGame(char **map) = 0;
      virtual void              display() = 0;
      virtual int               isGame() = 0;
      virtual void              close() = 0;
      virtual void              playMusic() = 0;
      virtual void              closeMusic() = 0;
      virtual int               Close() = 0;
      virtual int               Time(clock_t current_time) = 0;
      virtual void              showScore(unsigned int score) = 0;
  };
```

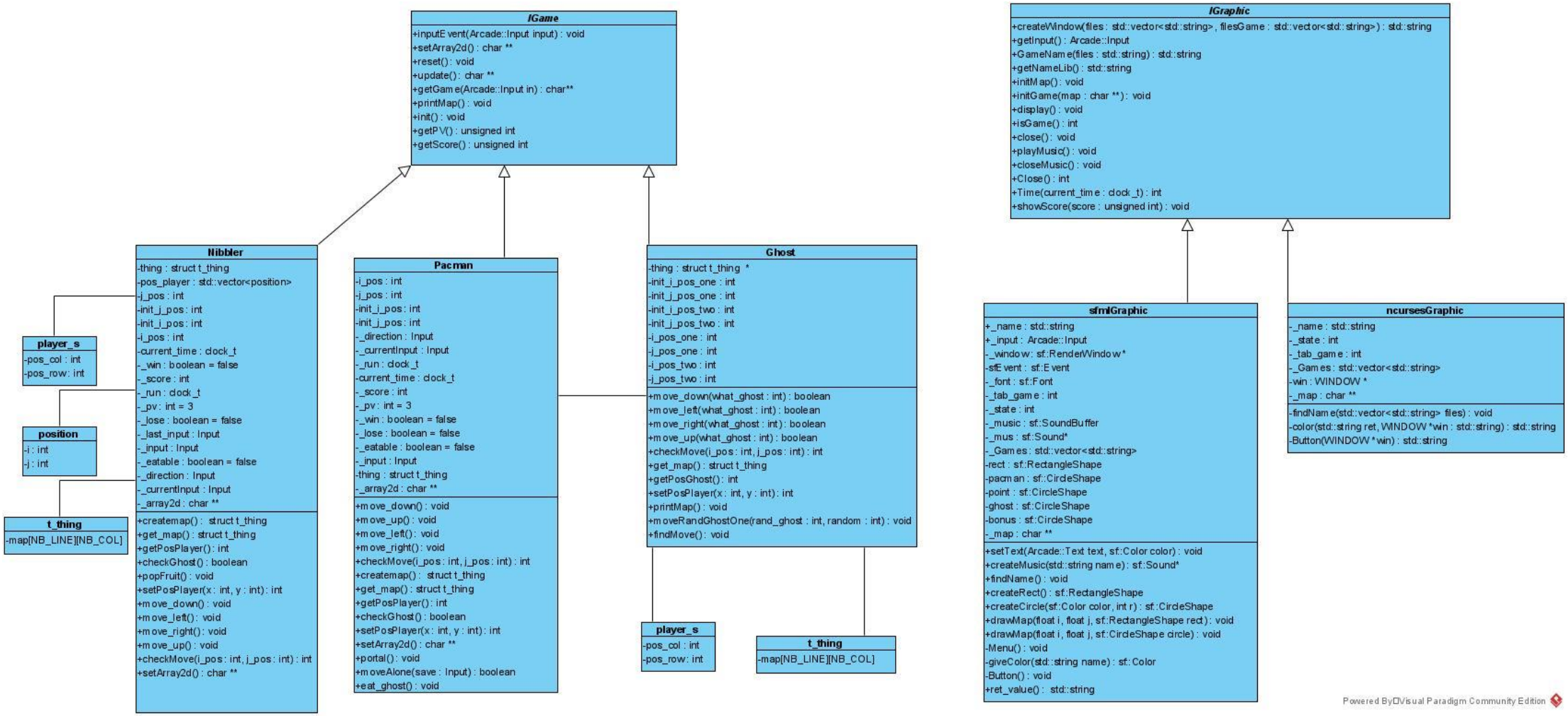
Afin de d'implémenter une libraire de graphiques qui sera compatible avec notre syteme, il est essentiel de respecter l'utilisation des capacités décrite dans l'interface. Les principales méthodes sont :

-createWindow : rassemble les méthodes qui seront utiliser pour l'utilisation de la librairie dans une seule méthodes.

-getInput : qui permet d'attribuer une valeur aux touches selon la libraire graphique qui sera ensuite recuperable par le core puis par le game.

-initMap : permet d'attribuer les éléments de la map envoyer par le jeu à des élément graphiques de la lib.

-intiGame : initialise les différents éléments qui serons t appeler lors de l'utilisation de la lib.


IGame
+inputEvent(Arcade::Input input) : void
+setArray2d() : char **
+reset() : void
+update() : char **
+getGame(Arcade::Input in) : char**
+printMap() : void
+init() : void
+getPV() : unsigned int
+getScore() : unsigned int
IGraphic
+createWindow(files : std::vector<std::string>, filesGame : std::vector<std::string>) : std::string
+getInput() : Arcade::Input
+GameName(files : std::string) : std::string
+getNameLib() : std::string
+initMap() : void
+initGame(map : char **) : void
+display() : void
+isGame() : int
+close() : void
+playMusic() : void
+closeMusic() : void
+Close() : int
+Time(current_time : clock_t) : int
+showScore(score : unsigned int) : void
Nibbler
-thing : struct t_thing
-pos_player : std::vector<position>
-i_pos : int
-init_j_pos : int
-init_j_pos : int
-i_pos : int
-current_time : clock_t
-_win : boolean = false
-_score : int
-_run : clock_t
-_pv : int = 3
-_lose : boolean = false
-_last_input : Input
-_input : Input
-_eatable : boolean = false
-_direction : Input
-_currentInput : Input
-_array2d : char **
+createmap() : struct t_thing
+get_map() : struct t_thing
+getPosPlayer() : int
+checkGhost() : boolean
+popFruit() : void
+setPosPlayer(x : int, y : int) : int
+move_down() : void
+move_left() : void
+move_right() : void
+move_up() : void
+checkMove(i_pos : int, j_pos : int) : int
+setArray2d() : char **
Pacman
-i_pos : int
-j_pos : int
-init_i_pos : int
-init_j_pos : int
-_direction : Input
-_currentInput : Input
-_run : clock_t
-current_time : clock_t
-_score : int
-_pv : int = 3
-_win : boolean = false
-_lose : boolean = false
-_eatable : boolean = false
-_input : Input
-thing : struct t_thing
-_array2d : char **
+move_down() : void
+move_up() : void
+move_left() : void
+move_right() : void
+checkMove(i_pos : int, j_pos : int) : int
+createmap() : struct t_thing
+get_map() : struct t_thing
+getPosPlayer() : int
+checkGhost() : boolean
+setPosPlayer(x : int, y : int) : int
+setArray2d() : char **
+portal() : void
+moveAlone(save : Input) : boolean
+eat_ghost() : void
Ghost
-thing : struct t_thing *
-init_i_pos_one : int
-init_j_pos_one : int
-init_i_pos_two : int
-init_j_pos_two : int
-i_pos_one : int
-j_pos_one : int
-i_pos_two : int
-j_pos_two : int
+move_down(what_ghost : int) : boolean
+move_left(what_ghost : int) : boolean
+move_right(what_ghost : int) : boolean
+move_up(what_ghost : int) : boolean
+checkMove(i_pos : int, j_pos : int) : int
+get_map() : struct t_thing
+getPosGhost() : int
+setPosPlayer(x : int, y : int) : int
+printMap() : void
+moveRandGhostOne(rand_ghost : int, random : int) : void
+findMove() : void
player_s
-pos_col : int
-pos_row: int
position
-i : int
-j : int
t_thing
-map[NB_LINE][NB_COL]
player_s
-pos_col : int
-pos_row: int
t_thing
-map[NB_LINE][NB_COL]
sfmlGraphic
+_name : std::string
+_input : Arcade::Input
-_window: sf::RenderWindow*
-sfEvent : sf::Event
-_font : sf::Font
-_tab_game : int
-_state : int
-_music : sf::SoundBuffer
-_mus : sf::Sound*
-_Games : std::vector<std::string>
-rect : sf::RectangleShape
-pacman : sf::CircleShape
-point : sf::CircleShape
-ghost : sf::CircleShape
-bonus : sf::CircleShape
-_map : char **
+setText(Arcade::Text text, sf::Color color) : void
+createMusic(std::string name) : sf::Sound*
+findName() : void
+createRect() : sf::RectangleShape
+createCircle(sf::Color color, int r) : sf::CircleShape
+drawMap(float i, float j, sf::RectangleShape rect) : void
+drawMap(float i, float j, sf::CircleShape circle) : void
-Menu() : void
-giveColor(std::string name) : sf::Color
-Button() : void
+ret_value() : std::string
ncursesGraphic
-_name : std::string
-_state : int
-_tab_game : int
-_Games : std::vector<std::string>
-win : WINDOW *
-_map : char **
-findName(std::vector<std::string> files) : void
-color(std::string ret, WINDOW *win : std::string) : std::string
-Button(WINDOW *win) : std::string
Powered By Visual Paradigm Community Edition