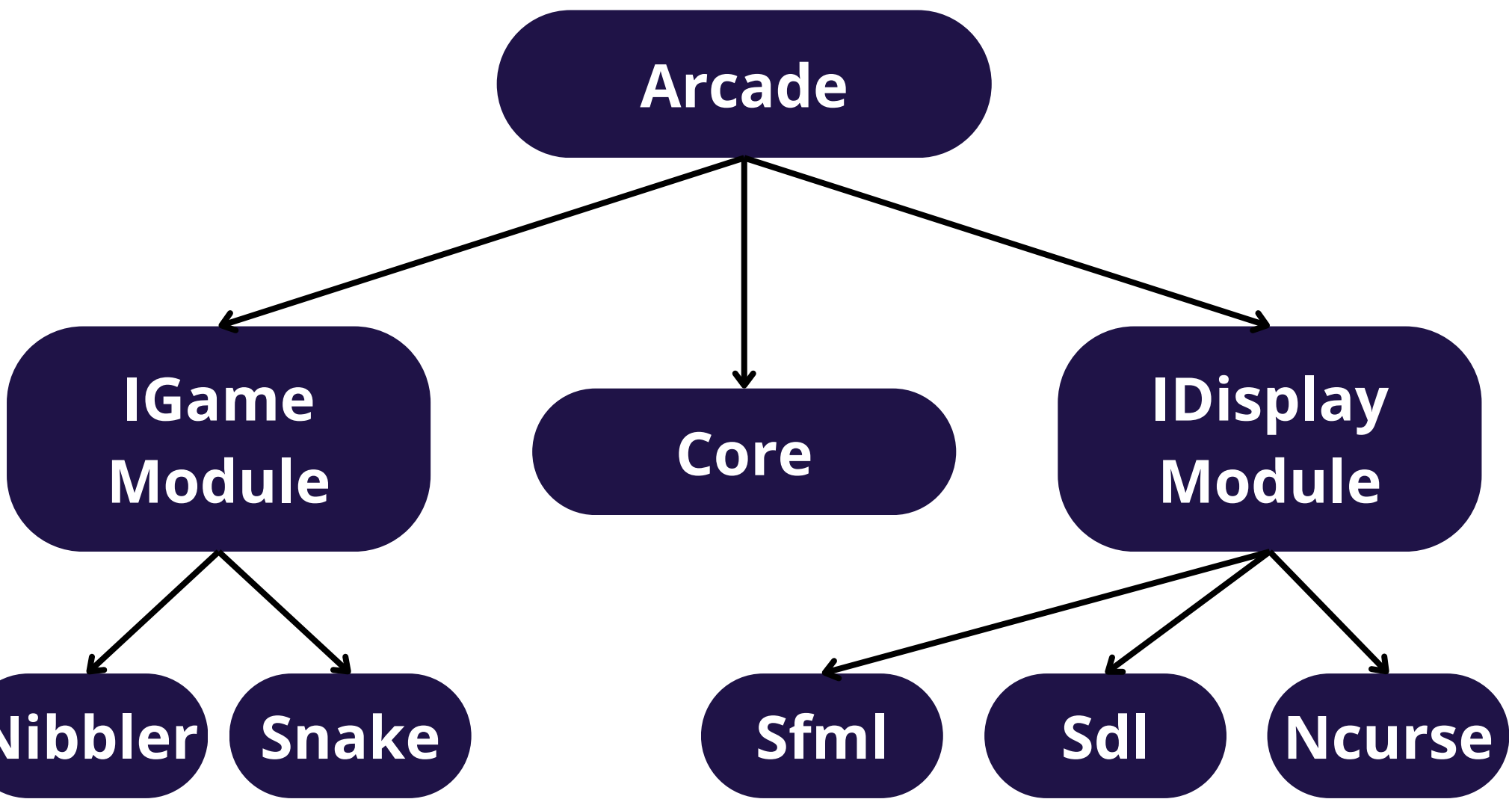
Documentation Arcade



Core

```
Core();
 void run(std::string lib);
 void displayMenu(void (Core::*drawFunc)
(std::vector<Arcade::text_object_t>&,
std::vector<Arcade::rectangle_t>&, Arcade::color_t&));
 void gameloop();
 ~Core();

void loadLib(std::string lib);
 void loadGame(std::string game);
 void scanDirectory(const std::string& path);
 bool isValidLibrary(const std::string& path, const
std::string& entryPoint);

 void
drawSFMLMenu(std::vector<Arcade::text_object_t>&
text, std::vector<Arcade::rectangle_t>& rectangles,
Arcade::color_t& color);
 void
drawSDL2Menu(std::vector<Arcade::text_object_t>&
text, std::vector<Arcade::rectangle_t>& rectangles,
Arcade::color_t& color);

void
drawNcursesMenu(std::vector<Arcade::text_object_t> &
text, std::vector<Arcade::rectangle_t>& rectangles,
Arcade::color_t& color);
```

IGameModule

```
IGameModule() : gameOver(false), dir(Direction::UP),
x(width/2), y(height/2 + 4), score(0), speed(100) {};

virtual ~IGameModule() = default;
virtual void Input(Arcade::Input input) = 0;
virtual void Draw(Arcade::IDisplayModule &display) = 0;
virtual void Logic() = 0;
virtual void RestartGame() = 0;
virtual void saveHighScore() = 0;
virtual bool isBestScore() = 0;
virtual int getHighScore() = 0;
```

IDisplayModule

```
virtual void LoadBackground(const std::string&
imagePath) = 0;
 virtual void Window(size_t width, size_t height) = 0;
 virtual void Draw(std::vector<std::pair<int, int>> _wall,
std::vector<std::pair<int, int>> _apple,
std::vector<std::pair<int, int>> _snake,
std::vector<std::pair<int, int>> _empty, int score, int
best_score) = 0;
 virtual void Clear(bool menu) = 0;
 virtual void DrawRectangles(const
std::vector<rectangle_t>& rectangles, color_t color) = 0;
 virtual void Update() = 0;
 virtual void Destroy() = 0;
 virtual void
DrawText(std::vector<Arcade::text_object_t> text, int
size) = 0;
virtual Inputs Input() const = 0;
virtual void Sleep() = 0;
virtual ~IDisplayModule() = default;
```

Nibbler

```
nibbler();
 void getLogic() {
 return (Logic());
 }
 void Draw(Arcade::IDisplayModule &display) override;
 void Input(Arcade::Input input) override;
 void Logic() override;
 void RestartGame() override;
 void saveHighScore() override;
 bool isBestScore() override;
 int getHighScore() override;
 void addWalls(int startX, int endX, int startY, int endY,  int
incrementX, int incrementY);
 void generateFruit();
```

Snake

```
snake();
 void getLogic() {
 return (Logic());
 }
void Logic() override;
void Draw(Arcade::IDisplayModule &display) override;
void Input(Arcade::Input input) override;
void RestartGame() override;
void saveHighScore() override;
bool isBestScore() override;
int getHighScore() override;
```

Sfml

```
void Window(size_t width, size_t height) override;
 void Destroy() override;
 void Clear(bool menu) override;
 void Draw(std::vector<std::pair<int, int>> _wall,
std::vector<std::pair<int, int>> _apple,
std::vector<std::pair<int, int>> _snake,
std::vector<std::pair<int, int>> _empty, int score, int
best_score) override;
 void Update() override;
 void DrawText(std::vector<Arcade::text_object_t> text, int
size) override;
 void Sleep() override;
 Arcade::Inputs Input() const override;
 void DrawRectangles(const
std::vector<Arcade::rectangle_t>& rectangles,
Arcade::color_t color) override;
 void LoadBackground(const std::string& imagePath)
override;
```

Sdl

```
sdl();

 void Init(const char* title, size_t width, size_t height,
Uint32 flags);
 void Window(size_t width, size_t height) override;
 void Destroy() override;
 void Clear(bool menu) override;
 void Draw(std::vector<std::pair<int, int>> _wall,
std::vector<std::pair<int, int>> _apple,
std::vector<std::pair<int, int>> _snake,
std::vector<std::pair<int, int>> _empty, int score, int
best_score) override;
 void Update() override;
 void DrawText(std::vector<Arcade::text_object_t> text,
int size) override;
 void Sleep() override;
 Arcade::Inputs Input() const override;
 void DrawRectangles(const
std::vector<Arcade::rectangle_t>& rectangles,
Arcade::color_t color) override;
 void LoadBackground(const std::string& imagePath)
override;

 ~sdl();
```

Ncurse

```
void Screen();
 void Window(size_t width, size_t height);
 void SetWindowProperties();
 void InitializeWindow(size_t width, size_t height);
 void Destroy() override;
 void Clear(bool menu) override;
 void Draw(std::vector<std::pair<int, int>> _wall,
std::vector<std::pair<int, int>> _apple,
std::vector<std::pair<int, int>> _snake,
std::vector<std::pair<int, int>> _empty, int score, int
best_score) override;
 int ConvertColorToPair(Arcade::color_t color);
 void Update() override;
 void DrawText(std::vector<Arcade::text_object_t> text,
int size) override;
 void Sleep() override;
 Arcade::Inputs Input() const override;
 void DrawRectangles(const
std::vector<Arcade::rectangle_t>& rectangles,
Arcade::color_t color) override;
 void LoadBackground(const std::string& imagePath)
override;

 ~ncurse();
```

C_encapsulation

```
C_encapsulation() = default;
 ~C_encapsulation() = default;
 void *dlopenC(const std::string& args, int flag);
 void *dlsymC(void *handle, const std::string& args);
 int dlcloseC(void *handle);
 std::string dlerrorC();
```

Error

```
Error(std::ostream &os, const std::string &str) throw();
 virtual ~Error(void) throw();
```

Définition

scanDirectory : récupère les .so présents dans un dossier indiqué en paramètre.

isValidLibrary : regarde si la librairie donnée en paramètre correspond au attente (est bien une librairie qui finit par un .so)

Logic : c’est dans cette fonction que la logique du jeu est implémentée (collision, ajout des pommes…)

isBestScore : si le score obtenu par le joueur est supérieur au meilleur score obtenu sur le jeu en question alors la fonction retourne true.

addWalls : fonction qui permet de placer des mures sur la map du jeu.

DrawRectangles : fonction qui permet d’afficher un rectangle autour d’un texte afin que le joueur puisse voir quel jeu est sélectionné.

```
C_encapsulation() = default;
 ~C_encapsulation() = default;
 void *dlopenC(const std::string& args, int flag);
 void *dlsymC(void *handle, const std::string& args);
 int dlcloseC(void *handle);
```

Ensemble de fonction qui permet de créer ou d’ouvrir les fichiers en .so .

Error : fonction permettant d’arrêter le programme et de retourner un message d’erreurs si une erreur est survenue.