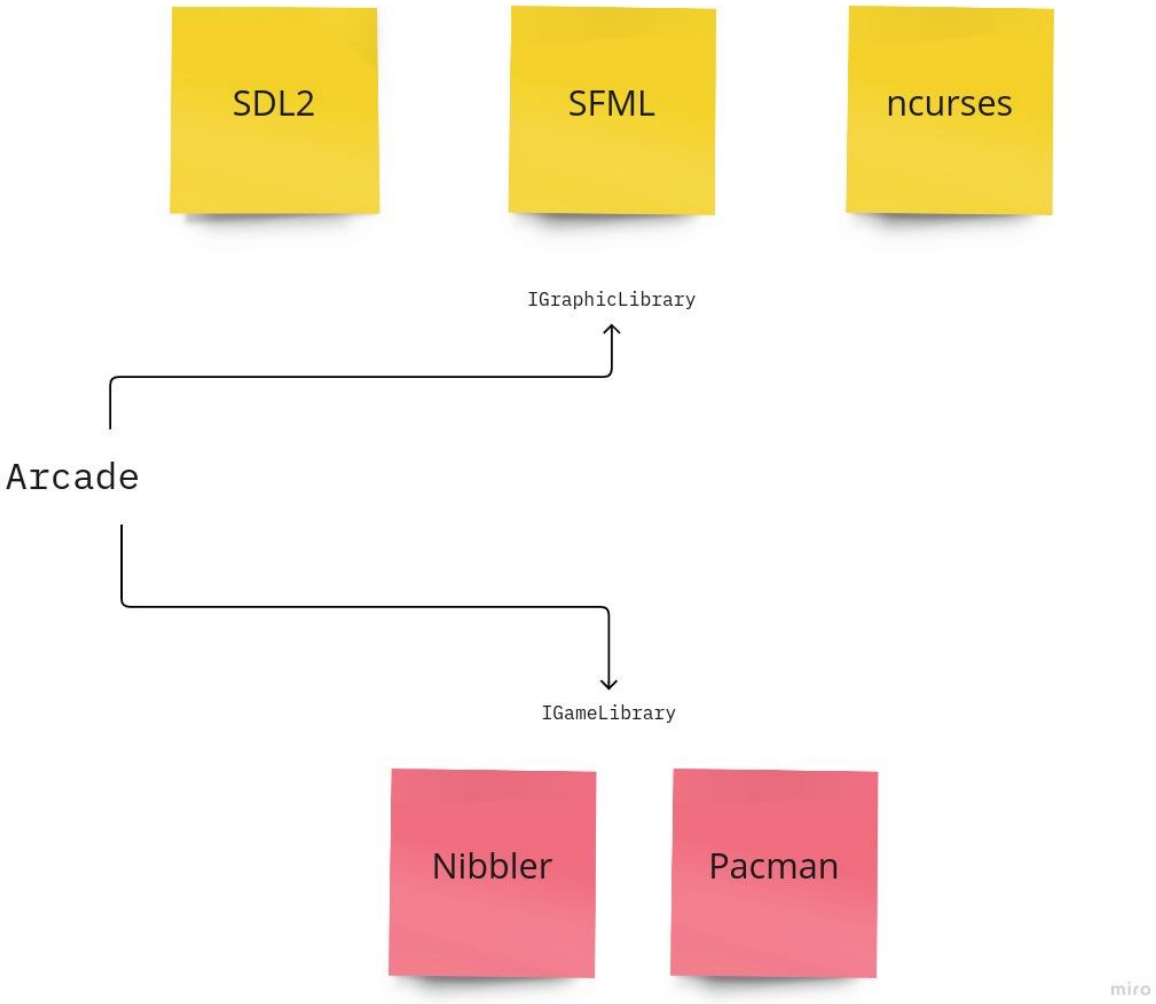
Arcade Documentation



IGraphicLibrary:

```
virtual void createWindow() = 0;
virtual std::string getName() const = 0;
virtual void closeWindow() = 0;
virtual void loadObjects(std::vector<object> gameObjects) = 0;
virtual KeyEvent loop() = 0;
```

CreateWindow() -> create a window with the dynamic lib you want.

GetName() -> simple c++ getter to have the lib name you want.

CloseWindow() -> close a window (this function can be called by the destructor for example).

LoadObjects() -> load objects from the struct object.

Loop() -> main loop for the lib you want.

IGameLibrary:

```
virtual ~IGameLibrary() = default;
        virtual void setGameObjects() = 0;
        virtual std::vector<object> getGameObjects() = 0;
        virtual void updateGameObjects() = 0;
        virtual void setKeyEvent(KeyEvent event) = 0;
        virtual int getLastScore() = 0;
        virtual bool hasGameEnded() = 0;
```

SetGameObjects() -> simple c++ setter for the game objects.

GetGameObject() -> simple c++ getter for the game objects.

UpdateGameObject() -> update the game objects.

SetKeyEvent() -> setter for the input keys.

GetLastScore() -> getter to stock the last score.

HasGameEnded() -> check if the game is ended.

Pacman:

```
void setGameObjects() override;
        void updateGameObjects() override;
        void initWall(int posX, int posY, int index);
        void initPacman(int posX, int posY);
        void initPhantoms(int posX, int posY, char chr);
        void initPacgums(int posX, int posY);
        void initBigPacgums(int posX, int posY);
        void initTeleporters(int posX, int posY);
        void readMap();
        void handlePacmanMovement();
        void handleTeleportation();
        void setDirection();
        bool checkMovement(object &entity, float speed);
        bool updateByTime();
        bool checkColision(object &entity, float speed);
        static bool isInt(float val);
        void setPacmanRotation();
        void handlePacgumColision();
        void handlePhantomsMovement();
        void setRandomDirection(object &entity);
        void handlePhantomColision();
        void gameEnd();
        void resetGame();
        void resetPhantom(int posX, int posY);
```

```
void resetPacman(int posX, int posY);
void resetPacgums();
void handleInvicibility();
void handlePacmanEatPhantom(object &i);
void handleDeadPhantom(object &i);
void updateScore();
bool handleBeginOrEnd();
void handleWin();
```

Nibbler:

```
void initNibbler(int posX, int posY);
void readMap();
void initWall(int posX, int posY, int index);
void initApples(int posX, int posY);
void updateScore();
bool handleBeginOrEnd();
void setGameObjects();
void updateGameObjects();
void handleNibblerMovement();
void setDirection();
bool checkMovement(object &entity, float speed);
bool updateByTime();
bool checkColision(object &entity, float speed);
bool isInt(float val);
void setNibblerRotation();
void handleAppleColision();
void gameEnd();
void handleWin();
void resetNibbler(int posX, int posY);
void resetGame();
void resetApples(int posX, int posY);
```

Struct object:

```
Type type;
std::shared_ptr<State> state { new State(State::ALIVE) };
std::string texturePath;
std::shared_ptr<std::string> text { new std::string("") };
char chr;
std::shared_ptr<float> posX { new float(0) };
std::shared_ptr<float> posY { new float(0) };
std::shared_ptr<int> animX { new int(0) };
std::shared_ptr<int> animY { new int(0) };
std::shared_ptr<int> animW { new int(0) };
std::shared_ptr<int> animH { new int(0) };
std::shared_ptr<int> spriteW { new int(0) };
std::shared_ptr<int> spriteH { new int(0) };
int sizeW;
int sizeH;
std::shared_ptr<double> rotation { new double(0) };
std::shared_ptr<int> mirrored { new int(0) };
bool isAnimated;
int currentFrame = 0;
int maxFrame = 0;
std::shared_ptr<Direction> direction { new Direction(Direction::RIGHT) };
```

```
std::shared_ptr<Direction> bufferedDirection { new Direction(Direction::RIGHT) };
std::shared_ptr<int> alpha { new int(255) };
```

Class Exception

This class allows us to throw an Exception in all the code with a std::string message.

Class DynamicLibrary

This class allows us to create the IGraphicLibrary and the IGameLibrary dynamically thanks to the function create() with dlopen() and dlsym().

Open() -> use dlopen() to open the lib you want.

Create_game() -> function to create a game lib.

Create() -> function to create a graphic lib.