Arcade

Arcade is a simple core game engine adapting for simple 2D games and implementable with different graphics libraries.
It is designed to be easy to use and flexible, allowing developers to create games quickly and efficiently with two interfaces arc::IGameLibrary and arc::IGraphicLibrary.

Supported Libraries

- • Ncurses: A library for creating text-based user interfaces in a terminal.
- • SDL2: A cross-platform library for creating games and multimedia applications.
- • SFML: A simple and fast multimedia library for C++.

Game implemented

- • Snake: A classic snake game where the player controls a snake that grows longer as it eats food.
- • Nibbler: A variant of the snake game where the player controls a snake that must navigate through a maze to collect food and avoid obstacles.

Prerequisites

- • Make
- • CPP Compiler

1. Run make to build the core, games and graphicals.
2. Run ./arcade ./lib/arcade_sfml.so to start the core.
3. Select a game and a library from the menu.
4. Use the arrow keys to navigate the menu and select a game and a library.
5. You can enter your name
6. Use the Enter key to start the game.
7. Use the Esc key to quit the game.

Ideas for games and libraries

All the games and libraries are in the src/Games and src/Libraries folders. You can create your own games and libraries by following the instructions below. The name of the game or library must be in lowercase and the name of .so must be the same as the name of the game or library. For example, if you create a game called my_game, the name of the .so file must be arcade_my_game.so. The same applies for libraries.

Libraries:

- • NDK++ (arcade_ndk++.so)
- • aa-lib (arcade_aalib.so)
- • libcaca (arcade_libcaca.so)
- • Allegro5 (arcade_allegro5.so)
- • Xlib (arcade_xlib.so)
- • GTK+ (arcade_gtk+.so)
- • Irrlicht (arcade_irrlicht.so)
- • OpenGL (arcade_opengl.so)
- • Vulkan (arcade_vulkan.so)
- • Qt5 (arcade_qt5.so)

Games:

- • Minesweeper (arcade_minesweeper.so)
- • Pacman (arcade_pacman.so)
- • Qix (arcade_qix.so)
- • Centipede (arcade_centipede.so)
- • Solarfox (arcade_solarfox.so)
  How to build

Game interface - IGameLibrary

- • The IGameLibrary interface is the main entry point for the game engine. It provides methods for initializing the game, updating the game state, and rendering the game graphics.

8. 1. Create a new class that implements the IGameLibrary interface in src/Games/{GameName}/GameName.cpp and includes/Games/{GameName}/GameName.hpp in the arc::games namespace and inherit from AGameLibrary.

Example :

```
/*
* EPITECH PROJECT, 2025
* Tek 2 Arcade
* File description:
* GameName.hpp
*/

ifndef GAMENAMEHPP_

define GAMENAMEHPP_

include "Core/Abstract/AGameLibrary.hpp"

include "Core/Click.hpp"

include "Core/Event.hpp"

namespace arc::games {

class GameName : public AGameLibrary {
public:
    GameName();

    ~GameName();

    /**
     * @brief Updates the game state based on input and tick.
     *
     * @param state The click state (if used).
     * @param key The event key pressed.
     * @return True if update succeeded or game should continue,
false if it should quit.
     */
    bool Update(click state, Event key);

    /**
     * @brief Adds a renderable object to the game.
     *
     * @param name Name identifier for the object group.
     * @param component The RenderComponent to add.
```

```
     */
    void AddObject(std::string name, arc::RenderComponent component);

    /**
     * @brief Deletes a group of objects by name.
     *
     * @param name Name identifier of the group to delete.
     */
    void DeleteObject(std::string name);

    /**
     * @brief Gets the renderable objects associated with a name.
     *
     * @param name Name of the group to retrieve.
     * @return A vector of RenderComponents.
     */
    std::vector<arc::RenderComponent> GetObjects(std::string name)
const;

    /**
     * @brief Gets the current score.
     *
     * @return The score as an integer.
     */
    int GetScore() const;

    /**
     * @brief Checks if the game is over.
     *
     * @return True if the game is over, false otherwise.
     */
    bool IsGameOver() const;

    /**
     * @brief Resets the game to its initial state.
     */
    void Reset();

    /**
     * @brief Gets the name of the game.
     *
     * @return The game's name as a string.
     */
```

```
    std::string GetName() const;

    /**
     * @brief Initializes the game.
     */
    void InitGame();

    /**
     * @brief Closes and cleans up the game state.
     * @return void
 */
    void CloseGame();

    /**
     * @brief Retrieves the game map for rendering.
     *
     * @return A 2D vector of RenderComponents.
     */
    std::vector<std::vector<arc::RenderComponent>> GetMap() const
override;

    /**
     * @brief Adds or sets the current game map.
     *
     * @param map The new game map to use.
     */
    void
AddMap(std::vector<std::vector<std::shared_ptr<arc::RenderComponent>>
>) override;

private:
    std::map<std::string, std::vector<arc::RenderComponent>>
_objects; ///< Game objects by name.
    std::vector<std::vector<std::shared_ptr<arc::RenderComponent>>>
_map;                    ///< Current game map.
    int _score;
};

}

/**
 * @brief External C-style entry point for loading the game
dynamically.
```

```
 *
 * @return A void pointer to the game instance.
 */
extern "C" void *entryPoint();

endif // GAMENAMEHPP_

```
```

The entry point is the function entryPoint that returns a pointer to the game instance. This function is used by the dynamic library loader to load the game.
2. Implement the IGameLibrary interface in your class as you want.
3. Create Makefile in src/Games/{GameName}/Makefile to compile the game as this :
```

```
NAME = arcadegamename.so

SRC = GameName.cpp

OBJ = $(SRC:.cpp=.o)

CXX = g++
CXXFLAGS = -Wall -Wextra -Werror -fPIC -I ../../../includes
LDFLAGS = -shared

all: $(NAME)

$(NAME): $(OBJ)
    $(CXX) $(CXXFLAGS) -o $(NAME) $(OBJ) $(LDFLAGS)
    mkdir -p ../../../lib
    cp $(NAME) ../../../lib/$(NAME)
%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $< -o $@
    @ echo -e "\033[1;32m$(NAME) successfully created!\033[0m"

clean:
    rm -f $(OBJ)
```

```
fclean: clean
    rm -f $(NAME)
    rm -f ../../../lib/$(NAME)

re: fclean all

.PHONY: all clean fclean re
```

4. Add the game to main makefile in `Makefile` :

```
all: core games graphicals

core:
    $(MAKE) -C src
games:
    # ...
    # Append the game to the games list
    $(MAKE) -C src/Games/GameName
graphicals:
clean:
    $(MAKE) -C src clean
    # ...
    # Append the game to the games list
    $(MAKE) -C src/Games/GameName clean
fclean: clean
    $(MAKE) -C src fclean
    # ...
    # Append the game to the games list
    $(MAKE) -C src/Games/GameName fclean
re: fclean all
```

``

4. You can now build the game with make and run it with./arcade` and select your game in the menu!

Graphic interface - IGraphicLibrary

- • The IGraphicLibrary interface is the main entry point for the graphic engine. It provides methods for initializing the graphic engine, updating the graphic engine, and rendering the graphic engine.

9. 1. Create a new class that implements the IGraphicLibrary interface in src/Libraries/{LibraryName}/LibraryName.cpp and includes/Libraries/{LibraryName}/LibraryName.hpp in the arc::libraries namespace and inherit from AGraphicLibrary.

Example :

```
/*
* EPITECH PROJECT, 2025
* Tek 2 Arcade
* File description:
* LibraryName.hpp
*/

ifndef LIBRARYNAMEHPP_

define LIBRARYNAMEHPP_

include "Core/Abstract/AGraphicLibrary.hpp"

namespace arc::libraries {

class LibraryName : public AGraphicLibrary {
public:
    LibraryName();

    ~LibraryName();

    void Initialize() override;
```

```
    void Open() override;
    void Close() override;
    bool IsOpen() override;
    void Display() override;

    void DrawComponent(std::shared_ptr<arc::RenderComponent>)
override;
    void DrawComponents(const
std::vector<std::shared_ptr<arc::RenderComponent>> components)
override;
    void DrawMap(const
std::vector<std::vector<arc::RenderComponent>>& map) override;
    void DrawMenu(const arc::MenuComponent& menu) override;
    void DrawText(std::shared_ptr<arc::TextComponent>text) override;
    void DrawScore(const int& score,
std::shared_ptr<arc::TextComponent>text) override;

    std::pair<int, int> GetMousePos() const override;
    arc::click GetMouseState() const override;
    Event HandleEvent() override;
    char GetKeyPressed() override;
    std::string GetName() const override;

    void LoadResources(std::string filepath, type type) override;
    void UnloadResources(std::string filepath, type type) override;
    void LaunchMusic(std::string filepath) override;
    void StopMusic(std::string filepath) override;

private:
    std::string _name = "LibraryName";
};

}

/**
 * @brief External C-style entry point for loading the library
dynamically.
 *
 */
extern "C" void *entryPoint();

endif // LIBRARYNAMEHPP_
```

10. 1. Implement the IGraphicLibrary interface in your class as you want.
11. 2. Create Makefile in src/Libraries/{LibraryName}/Makefile to compile the library as this :
    ```

```
#

EPITECH PROJECT, 2025

Tek 2 Arcade

File description:

Makefile

#

NAME = arcadelibraryname.so

SRC = LibraryName.cpp

OBJ = $(SRC:.cpp=.o)

CXX = g++
CXXFLAGS = -Wall -Wextra -Werror -fPIC -I ../../../includes
LDFLAGS = -shared

all: $(NAME)

$(NAME): $(OBJ)
    $(CXX) $(CXXFLAGS) -o $(NAME) $(OBJ) $(LDFLAGS)
    mkdir -p ../../../lib
    cp $(NAME) ../../../lib/$(NAME)

clean:
    rm -f $(OBJ)
```
```

```
fclean: clean
    rm -f $(NAME)
    rm -f ../../../lib/$(NAME)

re: fclean all

.PHONY: all clean fclean re
```

4. Add the library to main makefile in `Makefile` :

```
all: core games graphicals

core:
    $(MAKE) -C src
graphicals:
    # ...
    # Append the library to the libraries list
    $(MAKE) -C src/Libraries/LibraryName
clean:
    $(MAKE) -C src clean
    # ...
    # Append the library to the libraries list
    $(MAKE) -C src/Libraries/LibraryName clean
fclean: clean
    $(MAKE) -C src fclean
    # ...
    # Append the library to the libraries list
    $(MAKE) -C src/Libraries/LibraryName fclean
```

``

4. You can now build the library with make and run it with./arcade` and select your library in the menu!

How to use

12. 1. Run make to build the core, games and graphicals.
13. 2. Run ./arcade to start the game.

Documentation

You can find the documentation in the html folder or from brower here. To learn more about classes and methods used for example TextComponent, RenderComponent, MenuComponent, etc.

Authors

- • Martin Delebecque
- • Mathis Jusy
- • Julien Parsing