B4 - Object-Oriented Programming

B-OOP-400

Arcade

A Retro Platform

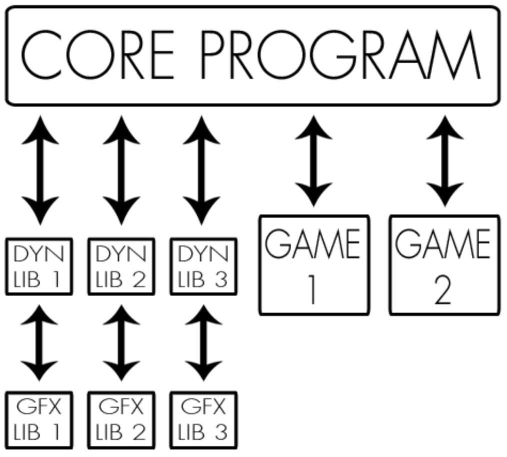
2.2

Arcade

binary name: arcade
language: C++
compilation: via Makefile (all, clean, fclean, re) or CMake 3.17

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Arcade is a **gaming platform**: a program that lets the user choose a game to play and keeps a register of player scores.
To be able to deal with the elements of your gaming plateform at run-time, your graphics libraries and your games must be implemented as **dynamic libraries**, loaded at run-time.



Each GUI available for the program must be used as a shared library that will be loaded and used dynamically by the main program.

It is **STRICTLY FORBIDDEN** to refer to a graphics library explicitly in your main program.
Only your dynamic libraries can do so.
This also applies to your games.

DYNAMIC LIBRARIES

You must use dynamic libraries at run-time.
This means that you MUST use the `dlopen`, `dlclose`, `dlsym` and `dlerror` functions to handle your dynamic libraries

`libdl` is a C library. You MUST provide an encapsulation. Also, add `-fno-gnu-unique` to your compilation flags to ensure `dlclose` unload your libraries properly.

Running `ldd` on your program must not show any dependency towards your libraries.

These dynamic libraries can be seen as plug-ins providing various capabilities to your main program.
In NO CASE must graphics libraries influence the game logic.
Game libraries must not contain any information about screen rendering or low-level events.
Your libraries (games and graphics) binaries (`*.so`) must be placed in the `./lib/` directory at the root of your repository.

You MUST NOT differentiate your libraries.
Each of your libraries MUST be handled in a generic and uniform manner.
Keep genericity!

GRAPHICS LIBRARIES

You must implement the `nCurses` (`arcade_ncurses.so`) and `SDL2` (`arcade_sdl2.so`) graphical libraries, and at least one more from the following list:

- `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`)
- `SFML` (`arcade_sfml.so`)
- `Irrlicht` (`arcade_irrlicht.so`)
- `OpenGL` (`arcade_opengl.so`)
- `Vulkan` (`arcade_vulkan.so`)
- `Qt5` (`arcade_qt5.so`)

Games libraries

You must implement at least two games from the following list:

- Nibbler (`arcade_nibbler.so`)
- Pacman (`arcade_pacman.so`)
- Qix (`arcade_qix.so`)
- Centipede (`arcade_centipede.so`)
- Solarfox (`arcade_solarfox.so`)

Descriptions and rules of these games are given at the end of this document.

Usage

The program must take as a startup argument the graphics library to use initially.
It must nevertheless be possible to change the graphics library at run-time.

```
Terminal
~/B-OOP-400> ./arcade ./lib/arcade_ncurses.so
```

When the program starts, it must display in separated boxes:

- the games libraries available in the `./lib/` directory.
- the graphics libraries available in the `./lib/` directory.
- scores.
- a field for the user to enter their name.

You MUST handle the following cases:

- if there is more or less than 1 argument, your program must print a usage message and exit properly (84).
- if the dynamic library passed as argument does not exist or is not compatible, your program must display a relevant error message and exit properly (84).

When your program is running, keys must be mapped to the following actions:

- previous graphics library.
- next graphics library.
- previous game.
- next game.
- restart the game.
- go back to the menu.
- exit.

Did you know that the `nCurses` library can handle mouse events?

Build

Makefile

Your Makefile **must** have the following rules (in addition to `all`, `re`, `clean` and `fclean`):

- `core`: it must **only** build the core of your arcade (not the games nor the graphical librairies).
- `games`: it must **only** build your games librairies.
- `graphicals`: it must **only** build your graphical librairies.

The results of running a simple `make` command in your turn in directory must generate a program, at least three graphics dynamic libraries and at least two game dynamic libraries.

CMake

Your CMakeLists.txt **must** build a program, at least three graphics dynamic libraries and at least two game dynamic libraries at the root of the repository.

```
~/B-OOP-400> mkdir ./build/ && cd ./build/
~/B-OOP-400> cmake .. -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release
[...]
~/B-OOP-400> cmake --build .
[...]
~/B-OOP-400> cd ..
~/B-OOP-400> ls ./arcade ./lib/
./arcade

./lib/:
arcade_ncurses.so
arcade_pacman.so
arcade_sdl2.so
arcade_sfml.so
arcade_solarfox.so
```

Documentation

More than any other kind of program, an **extensible** program like the Arcade project MUST be documented. In order to clarify the way your program and interface work, you must write a documentation. Turn-in the following document in PDF format, in a `./doc` directory:

- A document explaining how to implement new graphics libraries or game libraries compatible with your system.

As a bonus, it will be greatly appreciated if you also provide the following documents. Working on these will save you development time, as you can make sure your program is well designed before you start implementation:

- A class diagram for your program, featuring at least the relationships between classes and their public member functions.
- An explanatory manual that accompanies your diagram and descrbies how procedures are linked in your program.

Interface sharing

You must share your graphical and game library interfaces with at least one other project group. After turn-in, it will therefore be possible to run your games using their launcher and graphical libraries, and vice-versa.

You must add a README file in your repository containing, at least, the email address of the leader of the group with which you have collaborated for the interface.

GAMES

NIBBLER (`arcade_nibbler.so`)

Nibbler is a simple arcade video game released in 1982.
Its concept has spread mainly thanks to the cult game **Snake**.
Nibbler itself was inspired by another great classic: **Blockade**, itself inspired from **Tron Light Cycle**.

The simplicity and addictiveness of **Snake** made it available on almost every existing platform under various names.
As you may know, **Snake** is about moving a snake around a map.
The snake is represented by sections and must eat food in order to grow.
The game is over when the head of the snake hits an edge of the map or one of the sections.
The goal of the game is to make the snake as long as possible.

Various versions of **Snake** exist.
Some of them include obstacles, others have a core system, or bonuses, etc.

Core rules

- The game area is a finite amount of cells. The edges of the area cannot be passed through.
- The snake starts with a size of 4 cells in the middle of the area.
- The snake moves forward automatically at a constant speed. Each section of its tail follows the exact same path as the head.
- The snake can turn right or left when the corresponding key is pressed.
- The goal of the game is to feed the snake so that it can grow. The game area MUST NEVER have less than one element of food.
- A food element fills a single cell.
- When the head of the snake goes over a cell with food, the food disappears and a one-cell-long section is added at the tail of the snake. The new section appears in the first free tile next to the last cell of the tail. If there is no free cell, the game is over. If a new section is added, a new food element appears.
- When the head of the snake runs into the border of the screen or a part of its body, the game is over

Bonus ideas

- Bonus food appears for a short period of time
- The head section looks different from the other sections
- Movement speed increases throughout the game
- The game area has obstacles
- The size of the snake increases randomly when eating
- A speed boost when pressing the space bar

PACMAN (`arcade_pacman.so`)

Pacman is an arcade video game released in 1980.
The goal is to explore a maze in order to eat all the *"pacgums"* in it while avoiding ghosts.

Some *"pacgums"* let the player invert roles: **Pacman** can, for a short period of time, eat ghosts instead of being eaten.
Eaten ghosts do not disappear: their eyes head back to an unaccessible zone in the middle of the maze.
They change back to normal ghosts after a short period.

Core rules

- The game area has a specific size. Going through one side of the area makes the player appear on the opposite side. All cells that are not walls may be walked through and contain *"pacgums"*.
- In the middle of the map is a small 5-cell-wide and 4-cell-high area that contains ghosts.
- Ghosts can get out of their box 10 seconds after the game starts.
- Pacman starts the game right under the ghosts.
- Some special, larger *"pacgums"* let Pacman eat ghosts. This effect lasts 10 seconds. During this period, ghosts become blue and flee Pacman instead of hunting him. Their movement speed is slower during this time. There are only 4 *"pacgums"* of this kind on the map.
- When Pacman eats a ghost, only its eyes remain. These eyes quickly go back to the ghost box, where the ghost is healed after a short period of time.
- The player wins when Pacman eats all the *"pacgums"*. A new map is loaded after that, or the current one is reloaded and movement is accelerated.
- On screen, Pacman and ghosts must not move cell by cell, but smoothly.

Bonus ideas

- Food appears after a short period of time at Pacman's starting position. It provides a powerup or huge score bonus.
- Pacman and some ghosts can jump, like in **Pacmania**.
- The game speed increases over time.
- The game features a camera, like in **Pacmania**.

QIX (`arcade_qix.so`)

Qix is an arcade video game from 1981.
The game presents an area containing a monster: the **Qix**.
The player can move around this area and leaves a trail behind him.
When the player goes through a border of the screen, they appear on the other border and the area which was isolated from the **Qix** disappears.

The player wins when the remaining area containing the **Qix** represents less than 25% of the original area.
The player loses if the Qix crosses their trail.

The player's trail burns from its origin towards the player itself if the player stops. If the fire catches up to them, they lose.

Other monsters appear in the area "drawn" by the player. If one of these monsters touches the player, they lose.
Core rules

- The game area has a specific size. A cell can contain three different values that indicate its type: the cell can be walkable, non-walkable, or a border. A border is a specical walkable area.
- Non-walkable cells are space that was taken out by player action. Borders are cells that are directly in touch with a walkable cell and at least one non-walkable cell.
- The walkable area contains a monster: the Qix, which is several cells long and moves randomly.
- If the Qix touches the player or their trail, the player loses and goes back to the border they came from
- When the player walks on a walkable cell, they leave a trail behind them. This trail ignites if they stop. The player cannot cross the trail.
- When the player touches a border, the walkable area splits: the part containing the Qix remains and the other becomes non-walkable.
- Borders contain special monsters called Sparks. The player loses the game when they touch a Spark. Sparks can also move along the trail or old borders, even if they are in a non-walkable area, if it helps them hunt the player. A Spark cannot turn back.
- The player wins when the Qix is sealed inside less than 25% of the starting area.

Bonus ideas

- Powerups appear in the walkable area. The player may touch the powerup directly or with their trail. The effects of the powerups are up to you.
- Maps may contain obstacles or scenery.

CENTIPEDE (`arcade_centipede.so`)

Centipede is an arcade video game released in 1981.
The game features an area with a lot of empty space and some boxes.
The player is at the bottom of the scrceen and can move in all directions with some limitations: they can only move up and down by a little.
However, they can move freely left and right.
The player can also shoot projectiles towards the top of the screen.

Regularly, **Centipedes** appear at the top of the screen.
They move from left to right or right to left and go down when they encounter a box or the border of the screen.
If one of them touches the player, they lose.
A **Centipede** has a head, a tail and a body between the two.

When a shot from the player hits a **Centipede**, the part that was hit turns into a box and the body is split in half, becoming two smaller **Centipedes**. The **Centipede** spawned from the tail part encounters the newly formed box and heads back the way it came from, after having moved down.

Boxes can be destroyed by several shots from the player.

There can only be one shot one the screen at any given time.
Core rules

- The game area is split in two: a walkable area and a non-walkable area. The walkable area is at the bottom of the screen. It fills the entire width of the screen but only 20% of its height.
- Centipedes come from the top of the screen. They move from side to side and encountering an obstacle or screen border makes them move down a line. A centipede is a snake composed of several parts.
- The player can shoot. When a shot hits a centipede, it is split in half. The part that was hit turns into an obstacle, the head part keeps moving on and the tail part collides with the obstacle, moving down a line and turning back.
- If a centipede touches the player, they lose the game.
- If a centipede touches the bottom of the screen, the player loses score.
- The player wins if they survive 20 centipedes. The map is then reset and the game starts over.
- A Centipede map contains randomly generated obstacles.
- An obstacle can be destroyed by shooting it 5 times.
- There can only be one shot on the screen at any given time.

Bonus ideas

- A new centipede can appear while one is already on screen.
- Several types of centipede, with varying length, speed and movement patterns.
- The player controls two characters instead of one.

Solar Fox (`arcade_solarfox.so`)

Solar Fox is an arcade video game from 1981.
The game takes place in space and the player is in command of a space ship.
The play area is a grid, containing some powerups that can be picked up by shooting them.

Enemy guns appear on the border of the game area and shoot in a straight line. The player continuously moves forward and must dodge enemy shots.
They can also shoot to pick up powerups and intercept enemy shots.

Solar Fox is a dodge and collect game.
Many clones added functionalities and speed, making it a classic arcade game.
Core rules

- The game area is split in two: a central walkable area, with a margin of 2 or 3 cells between its limit and the screen border. The rest of the game area is non walkable and contains opponents.
- The player can move around the walkable area in every direction, but cannot turn back directly.
- The player cannot stop and always moves forward. A key lets them move faster.
- The walkable area is filled with powerups that the player must shoot to win.
- Opponents appear on the borders of the walkable area and shoot.
- Shots from the opponents can be destroyed by the player's shots.
- The player's shots only have a range of two cells. Their speed is three or four times faster than that of the player.
- The player loses the game if their spaceship is hit by a shot, special bad powerups or the walkable area's borders.
- The spaceship, lasers and opponents must not move cell per cell, but smoothly.

Bonus ideas

- Opponents with different types of shots.
- Moving opponents.
- Obstacles in the game area.