


Arcman - Personal project for C++ class

Welcome to the repository containing the game made for the second semester Object Oriented Programming class at Haute École Arc Neuchâtel . Here, you will find the source code, build instructions and a little documentation of the project.

Routes

- [Motivation and Objectives](#)
- [Compilation](#)
- [Code documentation](#)
- [UML diagram](#)



How to Use

If you want to skip the compilation phase, you can grab the release corresponding to your operating system.

If not, you can go to the [Compilation](#) section.

In the menus, use W and S to move up and down, and ENTER to select.

In the actual game, use WASD for 4 directional movement. Press P while in the game as a backdoor.



Dependencies

!! Dependencies are included directly if you use CMake to compile, and in the releases !!



CMake

- CMake



- SFML 2.6.1 or higher



Tools used for development

- Operating system: Arch Linux (& Debian, Windows for testing)
- Text editor: Neovim
- Diagrams: [clang-uml](#) and [plantuml](#)

Arcman - Motivation and Objectives

Motivation

I decided to make **ARCMAN** as my C++ Object-Oriented Programming project because it combines my passion for arcade games (especially Pac-Man) with a practical application of software engineering principles. It offers a hands-on opportunity to explore essential object-oriented programming concepts such as inheritance and polymorphism, while also enhancing my proficiency in C++.

Also, I wanted to add things that are specific to an average student studying at “HE-Arc”, like the player using the logo colors and the walls of the maze being the same color as the school wals, so the game is fun for my class mates.

Objectives

The objectives for the game are:

General Project goals

- Try to use as much of the components already provided by **C++**, like objects from the Standard Library or concepts specific to C++
- Implement the game using a Graphics Library (SFML)
- Make the game cross-platform (Windows, Linux, MacOS)
- Use *Design Patterns* where it is useful

Graphics

- Try to use the classes provided by the Graphics Engine as base classes for my own
- Draw sprites on my own
- Prioritize using maths for position and vector calculations

Classes and Objects

- Implement a class **Entity**, preferably extending the **Sprite** class provided by **SFML**, where I can manage all the general things related to the entities in the game
- Implement the **Player** as accurate as possible from the base game
- Implement **Ghosts** that chase the player (choosing direction randomly or using a search algorithm)
- Implement the physics for the maze, using entity collision or an unoriented graph
- Implement a **Logger**

Arcman - Compilation

Prerequisites

Before compiling, make sure the following tools are installed:

- **CMake** (3.16 or higher)
- **Git**
- A **C++** compatible compiler (e.g., GCC, Clang)

Windows

Install tools:

- Download and install [CMake](#)
- Install [Visual Studio](#) with C++ support or [MingGW](#)
- Install [Git](#)

Compile

- Open Command Prompt (or PowerShell)
- Clone the repository using:

```
git clone git@gitlab-etu.ing.he-arc.ch:isc/2023-24/niveau-1/1242.2-langage-c
```

- Navigate to project directory and run the installer script:

```
.\install.bat
```

Note: This script uses Visual Studio as generator for CMake. You can change the generator as you please, but I will not be documenting other generators in this tutorial.

Run:

To execute, you can either launch the executable from the File Explorer (following the path below), or launch the program from the command line:

```
.\build\bin\Release\Arcman.exe
```

Note: For technical reasons, run the executable from the root of the project, otherwise the assets are not going to load.

Linux

Install tools:

Install the necessary tools using the command:

- Debian/Ubuntu:

```
sudo apt install cmake build-essential git libsFML-dev libx11-dev libgl1-
```

- Arch Linux:

```
sudo pacman -S cmake base-devel git sfml
```

Compile:

- Open a terminal
- Clone the repository using:

```
git clone git@gitlab-etu.ing.he-arc.ch:isc/2023-24/niveau-1/1242.2-langage-cpp
```

- Navigate to project directory and run the install script:

```
./install.sh
```

Run

build/bin/Arcman

Note: For technical reasons, run the executable from the root of the project, otherwise the assets are not going to load.

Arcman - Code Documentation

Here, I will be presenting the core mechanisms of the game.

How the Player is moving

The first problem that I encountered while developing the game was how the Player will move across the maze;

Firstly, I tried to do solid walls and collision detection between the walls and the Player, but that caused some conception problems:

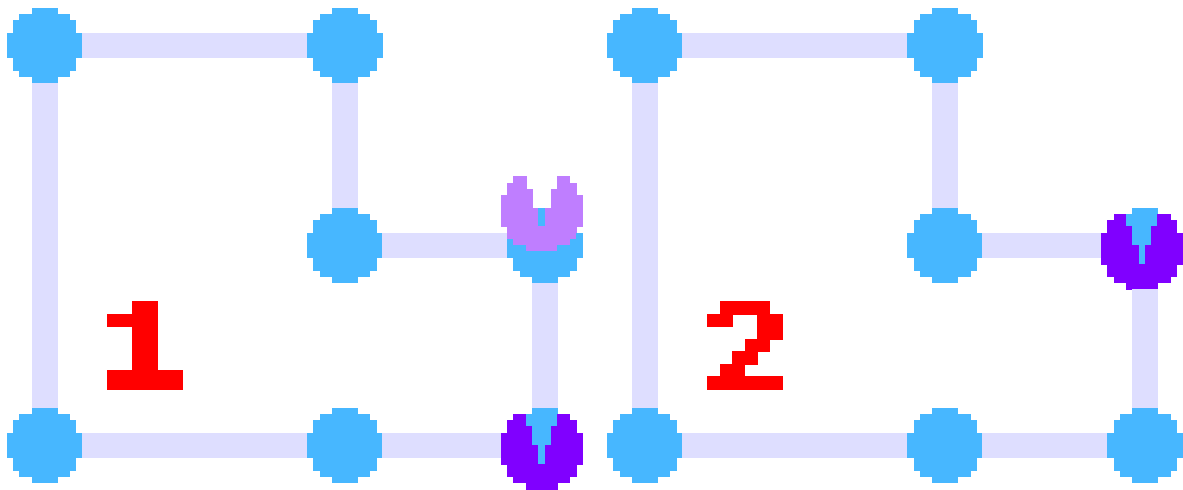
- The WASD movement needed to be extremely precise, and it was very sloppy
- There were too many collision checks to make, and the ways to reduce them were very complicated

So, to solve this problem, I chose to change the approach, and build a unoriented graph, in which the Player can move between it's nodes as following:



To make the mouvement smooth between two nodes, I implemented a **node overshoot** system, in which the Player moves in the direction of it's target node until it passes the coordinates of the node in the given direction.

When this happens, we set the Player's position to the position of the target node, and we stop the Player in place.

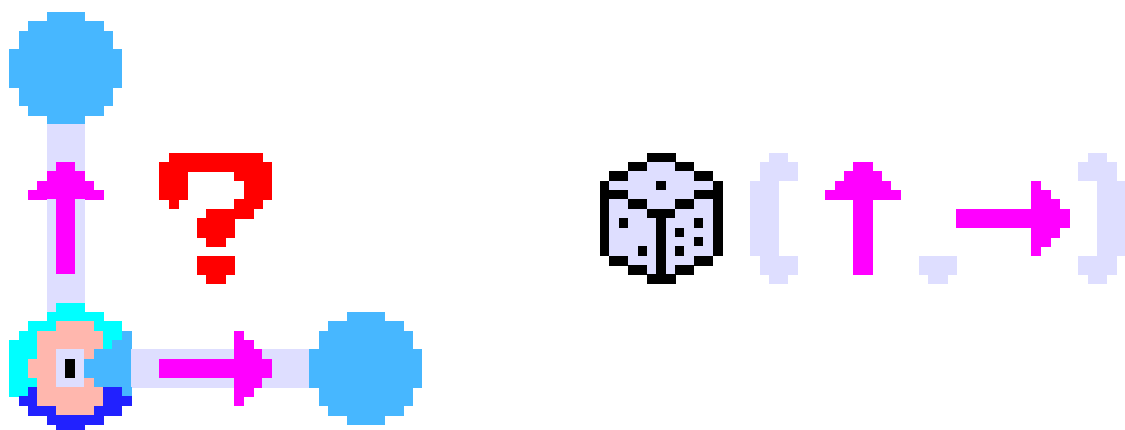


The usage of nodes has bunch of benefits, and unlocks some possibilities for future development:

- There is no need for solid tiles or walls
- Easier to implement the Ghost's movement AI
- Easier to implement a searching algorithm for the Ghosts (like Dijkstra's algorithm or A*)

How do the Ghosts work

The Ghosts, move between the nodes exactly as the Player does. The only difference is the way the Ghosts choose their next direction of movement

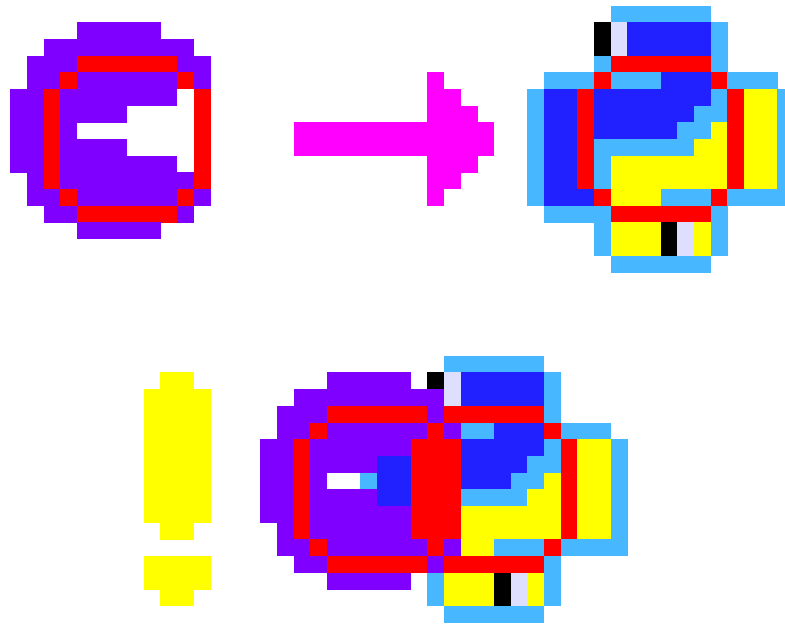


For this implementation, the Ghosts will choose a random direction from the list of available directions, and maybe in the future I will implement a custom searching algorithm for each Ghost.

Collision between Player and Ghosts

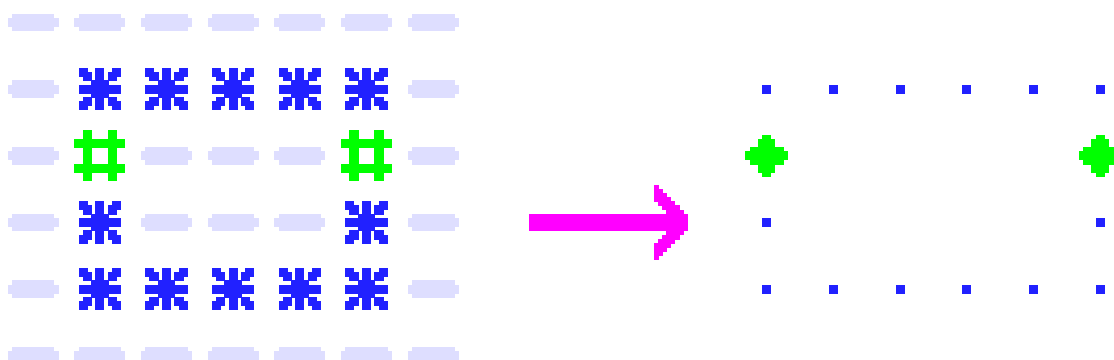
Both the Player and the Ghosts have a hitbox that is smaller than their actual size, so that we can leave a little room for error for the Player.

The hitboxes are represented as a circle with a certain radius from the texture's center, and the two Entities collide if the distance between the two circles is lesser than their radius.



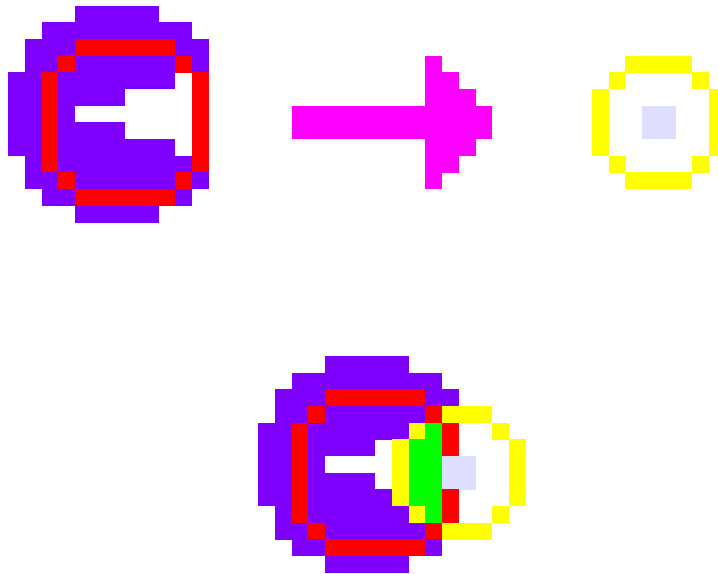
How the Pellets are generated

The Pellets are generated using a matrix of characters. If the character at a certain grid position is a '*', we will generate a Small Pellet, if it is a '#', we will generate a Big Pellet, and if it is any other character, we will generate an empty Entity.



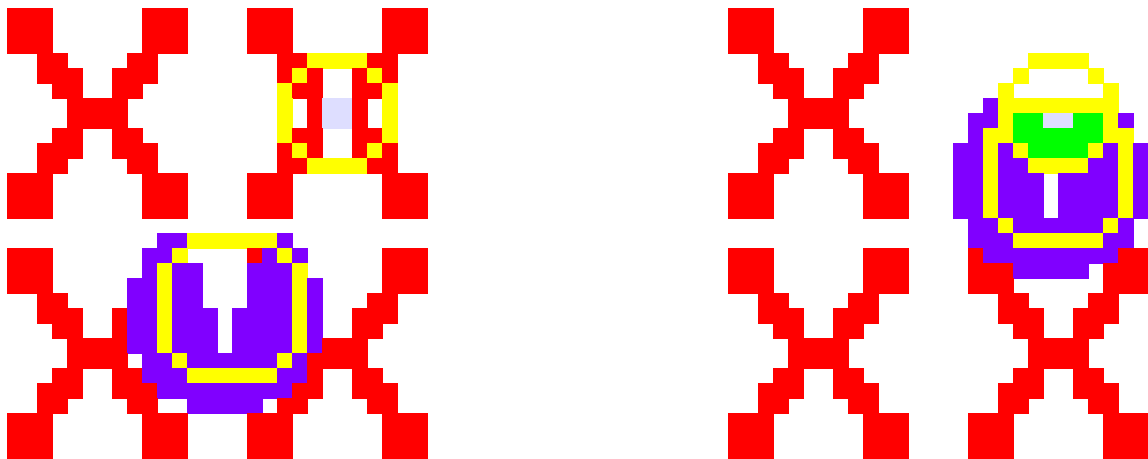
Collision between Player and Pellets

The collision between the Player and the Pellets is done in the same way as with the Ghosts.



The problem that arises here is that the number of pellets on the map is very big, and if we check each collision every frame, we will do bunch of redundant checks.

To ameliorate that, we will only check if the Player is colliding with Entities that are in it's proximity, like so:



By only checking the 4 squared surrounding the Player, we reduce the number of collision checks to just 4.