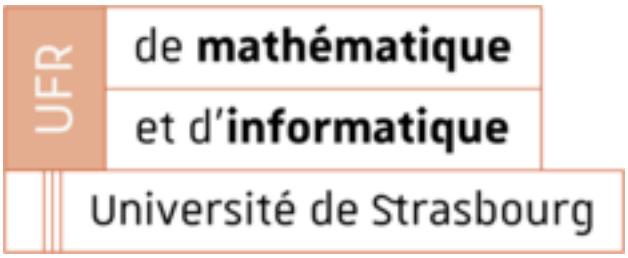
Advanced Programming

Project Report

Team:
ALLEMAND Fabien
LEBOT Samuel



Contents

List of Figures

1 Introduction

> Pac-Man, originally called Puck Man in Japan, is a 1980 maze action video game developed and released by Namco for arcades. In North America, the game was released by Midway Manufacturing as part of its licensing agreement with Namco America. The player controls Pac-Man, who must eat all the dots inside an enclosed maze while avoiding four colored ghosts. Eating large flashing dots called "Power Pellets" causes the ghosts to temporarily turn blue, allowing Pac-Man to eat them for bonus points.[6]
>
> Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code. The data is in the form of fields (often known as attributes or properties), and the code is in the form of procedures (often known as methods).[5]

The goal of this project is to recreate the Pac-Man video game from 1980 as accuratly as possible using C++. C++ is a object-oriented programming language that is very efficient and regulraly updated. In this project, many features from the latest C++ standards (C++11/14/17/20) were used. The Standar Template Library (STL)[8] and the Simple DirectMedia Layer (SDL)[7] add even more features to the basic C++ language.

GitHub: `https://github.com/FABallemand/Pac-Man`

2 Programming

2.1 Language and Libraries

The video game was programmed using the object-oriented programming language C++. As the purpose of this project was to get familiar with modern C++, many features from the latest C++ standards (C++11/14/17/20) were used: classes, heritage, polymorphism, lamnda functions...
The Standar Template Library is a C++ library used to enhance C++. Among other things, it contains very efficient and programmer-friendly containers and algorithms. These elements were massively used in order to create an efficient and readable program.
C++ does not natively provide functions to handle on screen display but the Simple DirectMedia Layer is a fairly easy to use library that provides all the objects and functions to easily handle 2D display.

2.2 Programming Agreements

In order to work efficiently as a team, some agreements regarding the repository and code organisation must be made.

The repository contains five main folders: `include`, `src`, `assets`, `build` and `doc`. The first one contains all the header files while the second one contains the corresponding source files (ie: `.cpp` files). The `assets` folders contains the sprites used in the game and the files representing levels. The two last folders are created and filled when building the project. After compiling, `build` contains the executable file and `doc` contains an *html* documentation generated with Doxygen.[2]
The `include` and `src` contains the same file tree that corresponds to the objects hierarchy. For instance, all files related to Ghost objects can be found in `game/object/moveable/ghosts`.

Throughout the entire project, the following naming scheme was used:

- Pascal Case: classes, struct, enum and name spaces (except LOG to make it stands out in the code)
- Camel Case: functions and methods
- Serpent Case: variables and attributes (with an extra "_" at the end for classes attributes)

A dedicated header file is used to store all constants by declaring them using constexpr and encapsulated name spaces following the hierarchy. For instance the size of a Ghost can be accessed in every other file using: gconst::game::object::moveable::ghost::size.

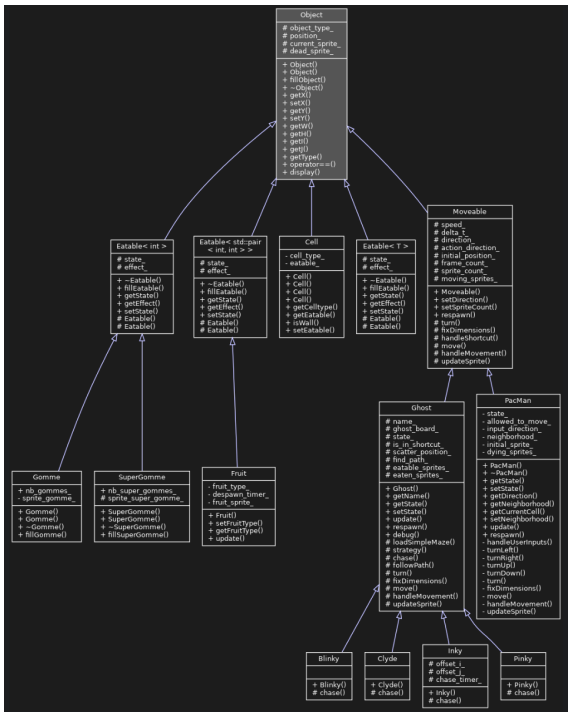


Figure 1: Game objects class hierarchy

All files were formated using the C/C++ VSCode extension from Microsoft and code documentation was generated by the Doxygen Documentation Generator VSCode extension.[3][1] In an effort to keep the code readable, explicit functions like constructors, destructors, *getters* and *setters* do not have documentation. Additionnaly, all header files starts with a the preprocessing directives to include the file only once, followed by general imports then imports related to the project. In the same maner, source files start by including their associated header file.

2.3 C++ Features

The entire program relies on classes. From the Application to the small Gommes every element has been programmed as a class. For instance, the class LOG implements a logger. Using heritage, it is easy to add features to a class whitout starting from scratch. Following the afore mentionned file hierarchy, the class Ghost is a sub-class of Moveable which itself derives from Object. (Figure 1)
Some classes are virtual meaning they contain virtual function(s) that must be implemented by the sub-class. For instance Moveable is a virtual class as it does not represent a real game object and contains among other move which is a virtual function. The definition of this method can be found either in PacMan or in Ghost, the two classes that inherits from Moveable.

Ghost objects contains a PathFinder attribute to find the best path leading to the target position. This attribute is a *functor*, that is to say an object used as a function by overloading the *()* operator.

Game object that can be eaten by Pac-Man have an effect_ attribute that is used to apply an effect to the game (namely, update the score). This object is a *lambda function*.

An particular attention has been paid to public, protected and private sections in order to restrict access to object attributes and methods and improving code reliability. In the same maner, const marker is used to avoid programming errors.

2.4 Standard Template Library Features

During the entire game, a single object of the class Fruit is created and updated, that is to say: its fruit_type_ attributes is set to the correct fruit. In order to keep the lambda function, the Eatable class was templated:

Gommes are Eatable<int> (the lambda function takes a single integer argument that corresponds to the current score and returns the new score) and Fruit are Eatable<std::pair<int,int>> (the lambda function takes a pair of interger as argument where the first value is the score of the game and the second value is a array index used to add the value corresponding to a fruit to the score).

All containers are std::array or std::vector. For instance in the class Game the board_ attributes used to represent the game board is a two dimentional array made by creating an array of arrays (see name space Board).

std::pair are used to represent positions on the board for all methods and attributes related to the movement of the Ghost objects and in the effect_ lambda function of Fruit as seen before.

All text objects are of type std::string.

Using this containers allows to use operators (==, std::min...) and pre-emplemented algorithms such as find, fill and tolower.

The PathFinder functor required priority queues to implement the Astar algorithm.[4] However, std::priority_queue does not implement the find method. Using templates, another priority queue implementing the find method has been created from std::priority_queue. (See PriorityQueue)

2.5 Simple DirectMedia Layer Features

The SDL was used in order to create a window to display the game inside. Special functions used to initialise and quit the SDL, load and unload image assets are declared in `SDL_utils.h` and `SDL_utils.cpp` files.

The SDL_BlitScaled function is used to display sprites. Additionnaly, the SDL_FillRect function was used to draw object bouding boxes during developement.

A simple timer relying on the SDL_GetTicks64 has been implemented as a class (Timer).

2.6 Game Reproduction

The goal of this project is to recreate the original Pac-Man game from 1980. Eventhough the game looks really simple, this is not an easy task as every object has multiple states, actions or effects.

Starting with the main character, the main difficulty was to properly handle the movement as it can move continuously on a discrete environnement. In other words, there is always one dimension that is fixed: if PacMan is moving to the left, its y coordinate is fixed to a precise value. Moreover, PacMan can cut corners to enhance playability. To do that we created an object called Cell representing a cell in the maze thus discretizing the board. Cell objects are used to keep track of the cells around PacMan so it is easy to look for walls.

Ghost movement is also difficult to implement as it is not relying on user input but rather on an artifical intelligence. According to its state a Ghost move to a specified target position by following the shortest path. In order to do that, the Astar algorithm has been implemented in the PathFinder class. However, there are constrains that can forbid the Ghost to follow the best path (ie: a Ghost cannot make a 180 degree turn). So if it cannot go in the best computed direction, the Ghost simply follow the walls.
In the original game, every single ghost has its own strategy to chase Pac-Man. Each of the Ghost sub-classes redefines the chase method: Blinky (red) directly chase PacMan, Inky (blue) has a target randomly choosen around PacMan and Clyde (orange) and Pinky (pink) try to go four cells in front or behind PacMan.

Every type of eatable object from the original game are implemented. Gommes and SuperGommes classes correspond to the dots Pac-Man can eat all over the maze and Fruit represents the fruit that appears at given score values.

The overlay uses PacString to display information. This simple class convert a string into an Object that can easily be displayed on the game window.

Level changements as well as the end of the game are properly recreated and bug-free.

3 Project Management

This project requires many hours of work to obtain a game that is both good looking and playable. In order to work efficiently, we set up some rules and define project organisation.

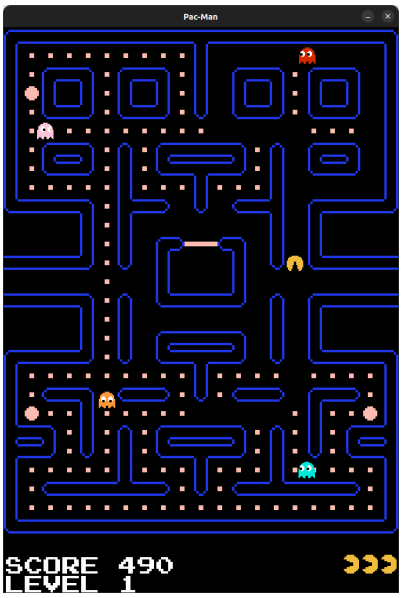


Figure 2: Game screenshot

We decided to use a GitHub repository in order to share files and keep track of versions. GitHub also propose project management features like *issues* that are really usefull when it comes to assign tasks to a member of the team.

Most of the time we worked together on the same computer or via a VSCode *Live Share* as it allows us to think, discuss and program with less errors as one of us is typing while the other is supervising. However, we sometimes distributed the tasks in order to work in parallel and speed up the developement. In this case, we made sure to read the modifications from the other as soon as possible using the commit history on GitHub.

From time to time, we organised ten-minutes-long meetings on a white board in order to plan and start thinking about future tasks.

We usually used Discord in order to communicate via text messages to share ideas or information outside of meetings or via calls during the Live Share sessions.

4 Conclusion

The resulting program is a good looking and very playable Pac-Man video game. The main elements from the iconic video game are present: Pac-Man, ghosts, dots, energizers, fruits... and work as one would expect. Retro gamers will definetly notice it is not exactly the original game from Samco but this project should be considered as a proof of concept rather than a real game as values (like speeds, timers, scores...) and ghost strategies has not been fine tuned to deliver Pac-Man's well known challenging experience. Nor do they change according to levels. However, all the elements required to recreate the full experience are implemented.

Learning advanced techniques of C++ programming while creating a complex program is not an easy task. Basic C++ features like classes and heritage are massively used and we tried to take advantage of many modern C++ features (iterators, functor, lambda function...). Of course, more experienced programmers will find flaws in the program. To name but one, multiple heritage should have been used in order to split object competencies. For example, ghosts are both moveable and eatable objects. But because we worked in a timely constrained environnement, we had to focus on developping a working program rather than creating a video game engine from scratch.

Working on this project was very challenging but also quite enjoyable as we learned advanced C++ programming in an interesting and funny way.

References

[1] doxdocgen. https://github.com/cschlosser/doxdocgen. Accessed: 2023-05-07.

[2] Doxygen. https://www.doxygen.nl/. Accessed: 2023-05-07.

[3] Microsoft. vscode-cpptools. https://github.com/microsoft/vscode-cpptools. Accessed: 2023-05-07.

[4] Wikipedia. Algorithme a*. https://fr.wikipedia.org/wiki/Algorithme_A*. Accessed: 2023-05-07.

[5] Wikipedia. Object-oriented programming. https://en.wikipedia.org/wiki/Object-oriented_programming. Accessed: 2023-05-07.

[6] Wikipedia. Pac-man. https://en.wikipedia.org/wiki/Pac-Man. Accessed: 2023-05-07.

[7] Wikipedia. Simple directmedia layer. https://www.libsdl.org/. Accessed: 2023-05-07.

[8] Wikipedia. Standard template library. https://en.wikipedia.org/wiki/Standard_Template_Library. Accessed: 2023-05-07.