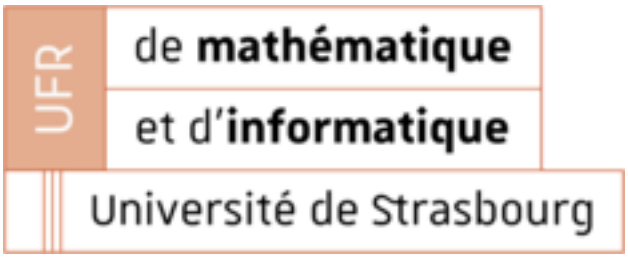
Advanced Programming

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.[5]
>
> 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).[4]

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)[7] and the Simple DirectMedia Layer (SDL)[6] were used in order to 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 such as: classes, heritage, polymorphism...
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 "$_{attheendforclassesattributes}$")

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.

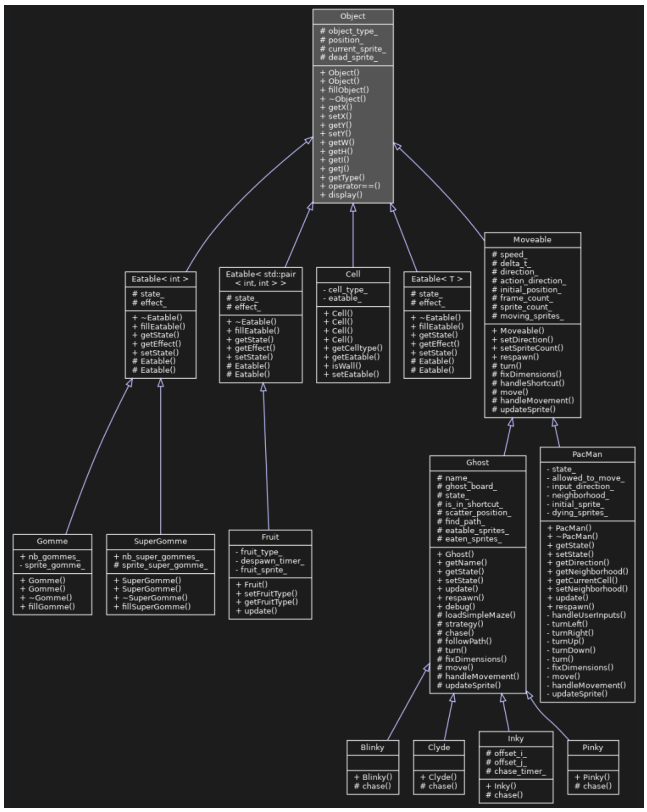


Figure 1: Game objects class hierarchy

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)

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 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.[**?**, astar]owever, std::priority$_{q}ueuedoesnotimplementthe$find$method.Usingtemplates, anotherpriorityqueueimplementingthe$find$methodhasbee$

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$_{B}litScaledfunctionisusedtodisplaysprites.Additionnaly, the$

2.5 Working in a Team

3 Conclusion

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

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

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

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