Pac-Man design report

Cedric De Vijt

s0225346

Introduction

In this project, a C++ implementation of the classic game Pac-Man is presented, focusing on the game logic and visual representation. This report provides an overview of the design choices made in the project, highlighting key aspects of the code structure and functionality.

Design Choices

Object-Oriented Design

The project adopts a robust object-oriented design, encapsulating entities and their behaviours into classes. This promotes code modularity, readability, and maintainability. Each entity has a corresponding class, adhering to the principles of encapsulation and separation of concerns.

Model-View-Controller

The application is developed using the Model-View-Controller pattern, this is reflected in the directory structure. The model is built as a static library, reducing the dependency on SFML to the implementation of the view and controller. This means it can be easily swapped out for a different visual implementation.

Singleton Pattern

The Stopwatch and Random classes implement the Singleton pattern, ensuring a single instance throughout the program's execution. This pattern enhances control and coherence in managing time and randomness within the game.

Resource Management

The FontFactory, SpriteFactory and SoundEffects classes demonstrate a robust approach to resource management. Fonts and sound effects are loaded and managed through singleton instances, ensuring efficient resource utilisation and providing a centralised point for resource retrieval. Notably, these classes are instantiated using a Singleton template, adding flexibility to the design.

Observer Pattern

Employing the Observer pattern for event handling, the Observer class and its implementations allow entities to respond to game events independently of the core game logic. This decoupling enhances flexibility, enabling visual representation and sound effects to evolve independently.

Exception Handling

The code incorporates exception handling, throwing runtime errors when critical resources (such as fonts or sound files) fail to load. This ensures that potential issues are caught early in development or deployment, contributing to the overall robustness of the code.

Visualisation with SFML

The SpriteFactory, EntityView and its sub-classes effectively utilize SFML for rendering sprites and visualizing entities. The SpriteFactory class centralises the creation of various game sprites, while the EntityView classes handle the visual representation of different entities. This design choice promotes maintainability and ease of extension for future sprite additions or modifications.

Convincing Points for Higher Grades

Feature complete
The implementation is feature-complete, meeting all specified requirements and delivering a fully functional Pac-Man game.

Code Readability and Organisation
The code stands out for its well-organized structure, featuring clear and concise class definitions, methods, and variable names. Adequate comments are provided where necessary, enhancing code readability and making it accessible for future development or collaboration.

Modularity and Extensibility
The modular structure of the code facilitates ease of extension. Each entity is encapsulated within its class, and the use of design patterns like Observer and Singleton enhances the code's adaptability to future changes or additions.

Robust Resource Management
The use of singleton classes for font and sound effect management demonstrates a robust approach to resource handling. This design choice ensures efficient resource utilisation and centralised control over critical game assets.

Extra Features

Sound
Both background music and foreground sound effects have been added to make the game more immersive. The foreground sounds are implemented as observers, maintaining separation between the model and sound effects without requiring additional events.

Extra Levels
Four distinct mazes corresponding to different game levels introduce variety and complexity, enhancing the overall gameplay experience.

Extra state
The inclusion of an intermission state after each level completion provides a clear separation between game levels, adding depth to the gaming progression.

Doxygen
All the documentation follows the Doxygen style comments, so a full set of HTML documentation is available.

Conclusion
In conclusion, the Pac-Man project showcases a well-crafted, object-oriented implementation following the Model-View-Controller design pattern. With a strong emphasis on modularity, extensibility, and resource management, the project adheres to best practices, readability, and thoughtful design choices. The inclusion of extra features and meticulous attention to detail make a compelling case for a positive evaluation.