Greedy Snake

May 2024

Content

1 Overview

The Snake Game project is a desktop application developed based on C++ and the Qt framework. The game interface is mainly divided into a game display area and an operation control area. The former provides real-time dynamic display of the game for players, while the latter contains various functional buttons for game control, such as start, pause, resume, restart, load and save game progress, and exit.

In terms of game mechanics, this game uses the keyboard's arrow keys (or WASD keys) as input devices to control the snake's movement direction. The player's main task is to maneuver the snake within a closed space, increasing its length by eating appearing food while avoiding the snake's head touching itself or the boundaries. The game's challenge lies in testing the player's reaction speed and strategic planning. Before starting the game, players can manually set personalized obstacles by clicking on the mouse interface, adding to the fun and challenge. Additionally, the game features speed adjustment, allowing players to adjust the snake's movement speed according to their preferences, enhancing the game's playability and challenge. This function is implemented through a slider interface, enabling users to adjust speed in real time to cope with changes in game difficulty.

This project reproduces the basic functionality of the Snake Game while enhancing user interaction and visual experience through software development techniques, suitable for players of different age groups for entertainment and leisure. Continuously optimizing and adding new features can further enhance the gaming experience and enjoyment.

2 Language and IDE

2.1 C++

C++ is a powerful general-purpose programming language developed by Bjarne Stroustrup at Bell Labs in the 1980s. It builds upon the foundation of C language while extending the concepts of object-oriented programming, providing programmers with richer abstractions and structural tools. The design goal of C++ is to balance high performance and flexibility, suitable for system-level programming and high-performance application development, including operating systems, game engines, graphics processing, scientific computing, and database systems.

C++ possesses several key features. Firstly, object-oriented programming supports encapsulation, inheritance, and polymorphism, making code more modular and maintainable. Template programming provides powerful capabilities for generic programming, allowing developers to create generic algorithms and data structures while maintaining type safety. The Standard Template Library (STL), as part of C++, offers a set of efficient, generic data structures and algorithms, making program development more convenient. Additionally, C++ has direct hardware access capabilities and memory management features, which are particularly important in critical domains such as embedded systems and game development. The C++ standard library covers a wide range of functionalities including file I/O, multithreading, network programming, and mathematical computations, making it a comprehensive programming tool.

C++ continues to evolve to meet the demands of modern software development. New standards such as C++11, C++14, C++17, and C++20 introduce many new language features like lambda expressions, smart pointers, concurrency libraries, and modules, improving development efficiency and code maintainability. In conclusion, C++ remains one of the important tools for modern software development due to its high performance, flexibility, and versatility.

2.2 QT

2.3.1 Introduction of QT

Qt (Qt Application Framework) is a cross-platform application development framework developed by Qt Company (formerly Trolltech), providing comprehensive tools and libraries to simplify the development of desktop, embedded, and mobile applications. The Qt framework originated in the 1990s and has evolved and matured over the years to become a widely used UI design tool and application development platform globally.

The core advantage of Qt lies in its cross-platform nature. With code written once, developers can easily deploy applications to different types of devices such as Windows, Linux, macOS, etc., without the need for separate adjustments for each system. This significantly reduces development and maintenance costs while improving efficiency.

Qt also offers powerful graphical user interface (GUI) capabilities. Its Qt Widgets module allows developers to create beautiful traditional desktop application interfaces, while the Qt Quick module provides a more modern interface development approach using QML and JavaScript, suitable for mobile applications or interfaces that require rapid response.

In addition to GUI, Qt provides rich tools and modules for multimedia processing, network communication, database access, file operations, multithreading, and internationalization, among others. Its integrated development environment Qt Creator offers developers comprehensive code editing, debugging, and visual design tools, further streamlining the development process. As a highly extensible framework, Qt's modular design allows developers to tailor and expand it according to project requirements.

2.3.2 Why choose QT

Qt is a cross-platform application development framework that, compared to traditional C++, offers rich GUI components and functionality, allowing developers to easily create modern, visually appealing, and responsive user interfaces. Qt introduces the signal and slot mechanism, simplifying communication between objects. Signals are member functions of a class used to trigger specific events when they occur, but they don't contain any processing logic themselves. Signals are only used to notify that a system event has occurred, such as a button being clicked or a timer timing out. Slots are regular member functions that get called when connected to a triggered signal. Slots can be connected to any signal, regardless of whether the signal originates from a local or remote object. The connection between signals and slots is established using the QObject::connect() function, ensuring that the appropriate slot function is executed when the signal is triggered. Connections can be made between signals and regular functions, and even between different objects.

Additionally, Qt provides support for cross-thread communication, making it easier for developers to handle multithreading and concurrent programming. Qt's rich library support covers graphics, networking, databases, multimedia, and more, allowing developers to easily implement various functionalities required for games, such as drawing game interfaces and playing audio. Qt provides an integrated development environment like Qt Creator, equipped with a rich set of debugging tools and design interfaces, enabling developers to efficiently carry out project development. Furthermore, Qt has comprehensive official documentation and tutorials, providing developers with learning and reference resources.

3 Program functionalities

3.1 Game Design

The game's main interface is divided into two parts: the game display area and the operation control area. The game display area shows the snake navigating through obstacles to eat the target food, while the operation control area contains game function buttons and displays the game progress.

3.1.1 Interface UI design

The game map size is set to 820x820 pixels through the GameWidget class, which serves as the main display area of the game interface. This display area not only carries the overall game visuals but also accurately showcases various elements on the game map. The game map is represented by a 2D array map_label[MAX_X][MAX_Y], where each element points to a QLabel control. These controls are displayed on the interface with a size of 20x20 pixels, with a gray background forming the blank spaces of the game area.

Building upon this foundation, elements such as the snake body, food, and boundaries are dynamically displayed on the map. The snake body is represented by gray QLabel controls, with each segment displayed independently and controlled and accessed through pointers in the array. The food is randomly generated as red QLabel controls on the blank spaces of the map, adding challenge and enjoyment to the game. The map boundaries are represented by gray QLabel controls surrounding the map's perimeter, serving both as interface decorations and limiting the snake's movement range, thus adding strategic depth to the game. It's worth noting that using a circular shape for the snake head ensures smoother transitions when the snake changes direction.

To effectively manage game elements, a 2D array map_label[MAX_X][MAX_Y] is used to represent the game map. This structural design is concise and efficient, allowing easy access and manipulation of map elements. Each array element points to a Snake structure, which includes a QLabel pointer and member variables representing the snake's position and type. This design makes the display and updating of game elements on the map convenient while also improving the game's performance and maintainability. The construction of the entire game interface and the dynamic display of elements are thus achieved, providing players with a smooth and intuitive gaming experience.

3.1.2 Obstacle design

In the Snake game, the design of obstacles is integrated into the construction of the game map. Obstacles can be randomly placed or defined by the player, becoming fixed barriers in the game. Their presence aims to increase the game's difficulty and strategic depth.

The default behavior of obstacles is to create a border around the game map, restricting the snake's movement range and preventing it from going beyond the game area. This default border behavior is implemented in the initBorder() function, which sets the type of edge cells to "border_label" and changes their QLabel style to a distinct color (usually light gray) to differentiate them from the game background.

In addition to these fixed borders, players can dynamically add obstacles before the game starts by clicking with the mouse. This dynamic addition is handled in the mousePressEvent() function, allowing players to place or remove obstacles by clicking anywhere within the game area beyond the edges. The mouse click position is calculated as the corresponding cell coordinates, then checked to see if there is already an obstacle:

- If it's already an obstacle, it's restored to a normal background cell, with the type set to "bg_label" and the style changed to the background color.
- If it's not an obstacle, the cell is set as a new obstacle, with the type set to "border_label" and the style updated to the specific style for obstacles.

During the game, obstacles significantly impact gameplay. In the moveSnake() function, each time the snake moves, it checks if the new position it's about to enter (i.e., its head) is an obstacle. If it is, the game calls the gameOver() function, indicating that the player has collided with an obstacle, resulting in the end of the game.

3.1.3 The movement of the snake's body

In the game, the movement of the snake is controlled by a core timer (QTimer). The timer triggers the timeout() signal, which is connected to the snakeMoveSlots() slot function. This slot function then calls the moveSnake() function to implement the specific movement logic. In the moveSnake() function, the game step count is first updated, and the displayStepSignal(steps) signal is emitted to notify the interface to update the step count display. Next, based on the snake's current movement direction (represented by dX and dY), the new position of the snake's head is calculated. This new position is then checked for validity: if the new position is at the boundary or overlaps with another part of the snake, the game ends; if it's at the position of food, the snake grows one segment, the score increases, and the displayScoreSignal(scores) signal is triggered to update the score display. Additionally, a new food item is randomly generated on the map. If the new position is not food, the snake's head simply moves to the new position, and the position of the snake's tail is updated to the background, creating the illusion of snake movement.

Furthermore, the snake's movement direction is controlled by the player using the keyboard. The snake's movement direction is updated based on the direction keys or WASD keys pressed by the user, ensuring that the snake does not move directly in the opposite direction.

3.1.4 Create food

In the Snake game, the generation of food is a crucial feature that directly impacts the game's playability and challenge. The createFood() function encapsulates this logic and uses random number generation techniques to determine the position of the food on the map, ensuring that the food does not appear on the snake or other obstacles.

Firstly, srand((unsigned)time(0)) is used to initialize the random number generator, ensuring that the food's position is random each time the game starts. Then, rand() % MAX_X and rand() % MAX_Y are used to generate random X and Y coordinates. Here, MAX_X and MAX_Y represent the width and height of the map, indicating the maximum rows and columns available on the map.

To ensure that the food does not generate on the snake or obstacles, createFood() uses a loop to check if the randomly generated position is vacant (background label). This validation process continues to generate new positions within a while loop until a valid empty position is found. Once a suitable location is found, the function sets the type property of the corresponding Snake structure to "food_label", indicating that the position is for food. The associated QLabel control's style is set to a red background, ensuring that it is distinct from the snake body and background on the game interface.

Finally, the food is presented to the player on the interface by calling the show() method of the QLabel.

3.1.5 Background music

In the Snake game, handling background music is achieved through the Qt audio playback interface, enhancing the game's appeal and immersing players in a more attractive gaming environment.

The process involves instantiating a QMediaPlaylist object to manage the playback of audio files. This playlist can contain one or more audio files to support sequential playback or looping. The background music file (typically in .wav format) is added to the playlist, and it's embedded into the program using the Qt resource system. Finally, the playlist is configured for looped playback mode to ensure that the music automatically restarts after it ends, providing continuous background music throughout the entire game process.

3.2 Game Control

3.2.1 Control buttons

In the Snake game, players use a rich set of control buttons to perform game operations. These buttons include Start, Pause, Resume, Restart, Save, Load, Exit, and Help. Each button is bound to a specific slot function, ensuring immediate response to player actions and appropriate control of the game flow. For example, the Start button initiates the game, while the Pause button halts the game awaiting further actions. Additionally, the availability status of buttons adjusts based on the current game stage to prevent inappropriate actions and enhance the visual experience through style changes.

Noteworthy are the game's save and load functions, which are triggered instantly through the corresponding buttons and slot functions, allowing players to save and restore game states and providing convenience in interrupting and resuming gameplay.

- Game Save Function:

When the player clicks the Save button, the saveGame() function is invoked. First, the game is paused using timer.stop() to ensure that the game state remains unchanged during the save process. The function then iterates through each part of the snake, records the coordinates of each segment, and saves them in snakeInfo. Simultaneously, it traverses the entire game map, recording the positions of all borders and food items in borderInfo and foodInfo. This information, along with the current step count and score, is concatenated into a complete string. The player chooses a file path through a file dialog, and the data is written into a text file to save the current game state. Upon successful save, the player receives a popup notification indicating successful saving.

- Game Load Function

When the player clicks the Load button, the loadGame() function is called. A file dialog opens, allowing the player to select a previously saved game state file. The function reads and parses the saved data, restoring the coordinates of the snake, borders, and food items, along with the step count and score. The display status of each grid on the map is updated based on the loaded data, and the snake's movement direction is set according to the last saved state. Upon successful load, the player receives a popup notification indicating successful loading.

3.2.2 Display Window

In the game, the score and step count are displayed in real time using two separate QLCDNumber controls. These numbers are updated dynamically based on the player's performance during the game, providing feedback such as score increments and the snake's movement steps. This real-time update functionality is achieved by connecting signals emitted by the game logic to slot functions responsible for displaying the score and step count. This ensures accurate transmission of game information.

3.2.3 Speed adjustment

The speed adjustment is implemented through a QSpinBox control, allowing players to set the snake's movement speed before starting the game or restarting it. The adjustment of speed is achieved by modifying the time interval of the game loop, directly affecting the snake's responsiveness and the difficulty level of the game. Changes in the speed adjuster's value are handled by connecting signals to corresponding slot functions, enabling dynamic speed adjustments during the game. This design not only enhances the game's playability but also adds strategic depth, as players need to choose the appropriate speed based on their reaction time and the progress of the game.

4 Demonstrations

4.1 Main Interface

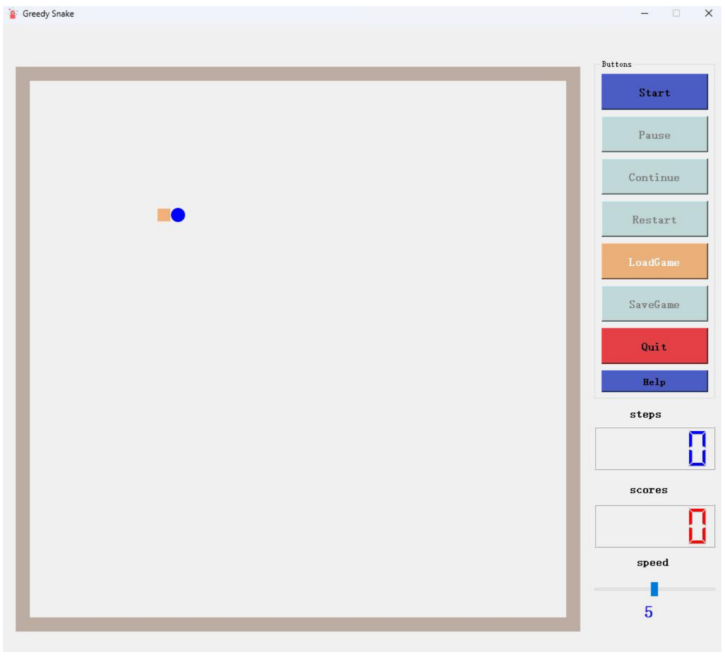
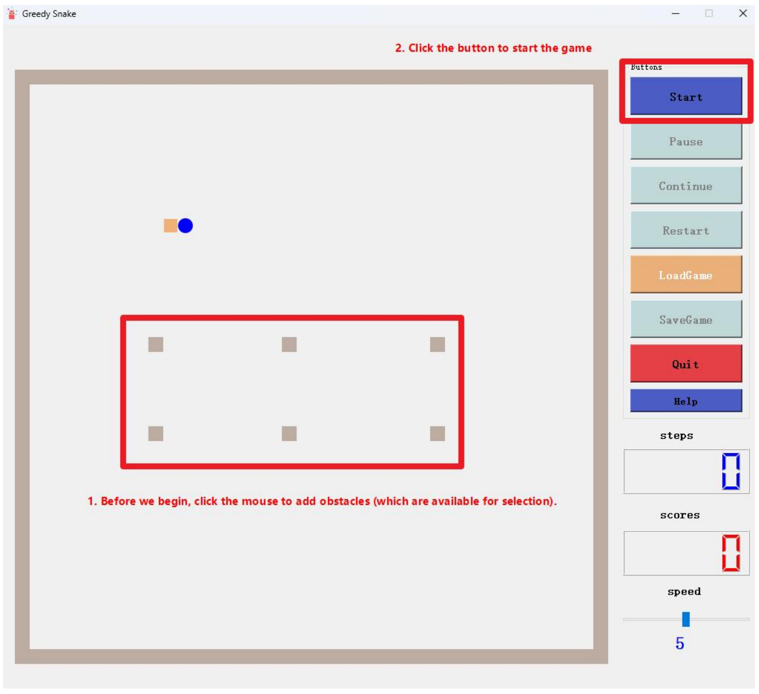


Figure 4.1 Game main interface

4.2 Game Execution



Greedy Snake
2. Click the button to start the game
Buttons
Start
Pause
Continue
Restart
LoadGame
SaveGame
Quit
Help
1. Before we begin, click the mouse to add obstacles (which are available for selection).
steps
0
scores
0
speed
5

Figure 4.2 Start game and manually adding obstacles

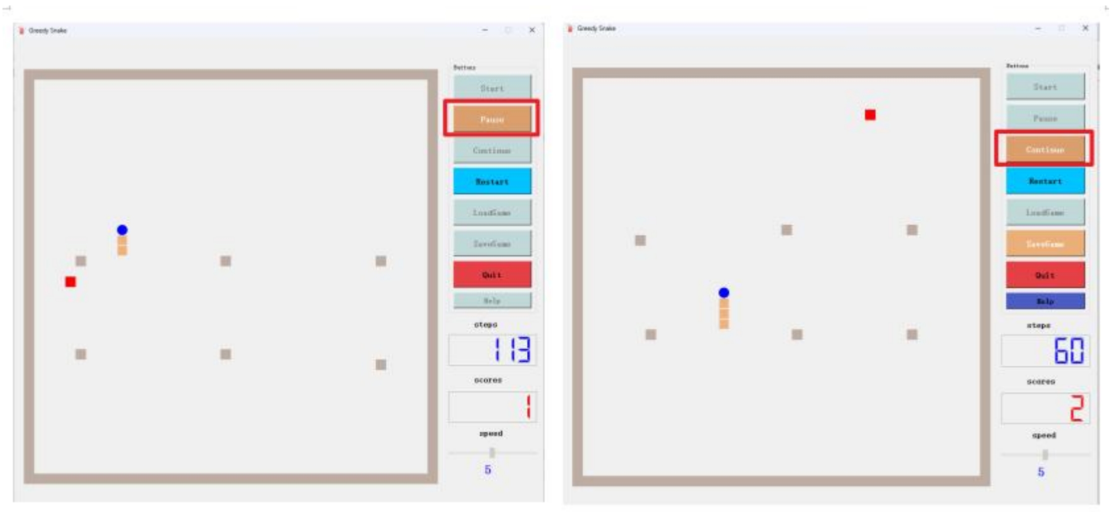
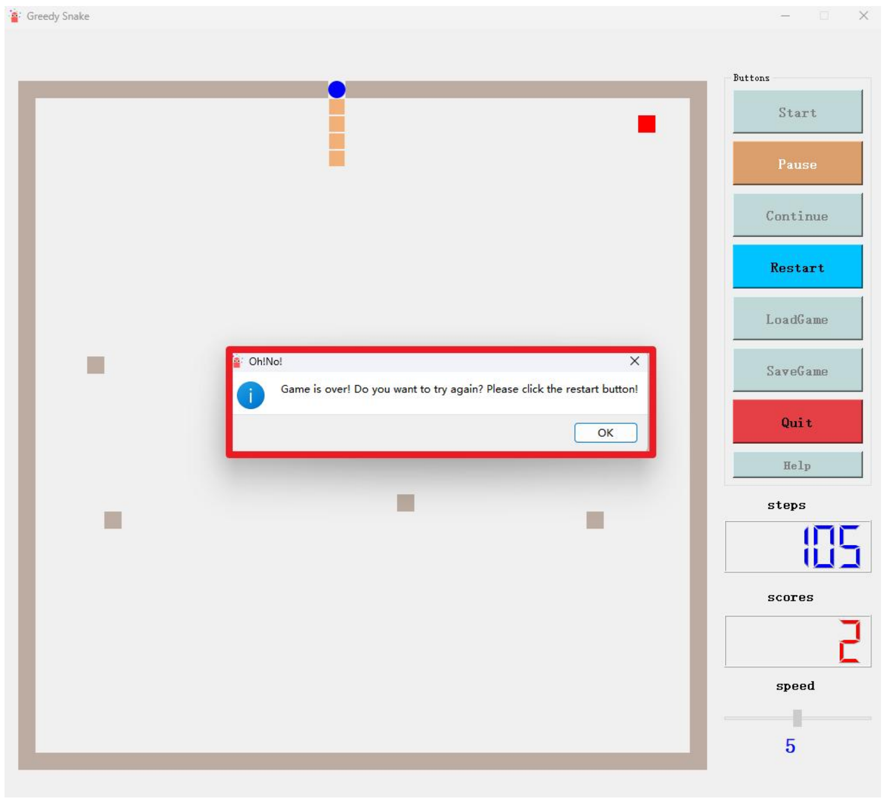


Figure 4.3 Pause and continue games



Oh!No!

Game is over! Do you want to try again? Please click the restart button!

OK

Figure 4.4 Game failed

4.3 Game Restart

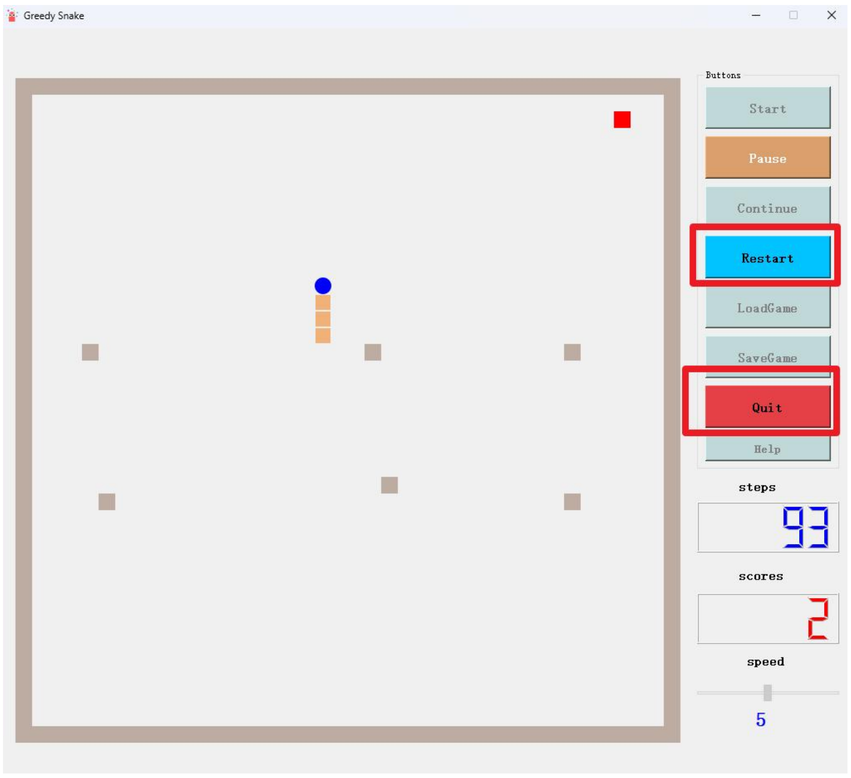


Figure 4.5 Restart or exit the game

4.4 Game Save and Load

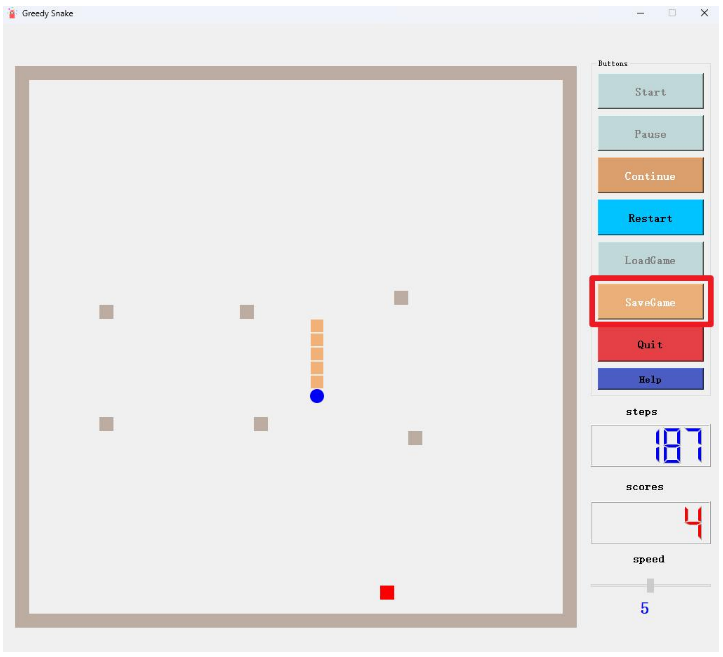
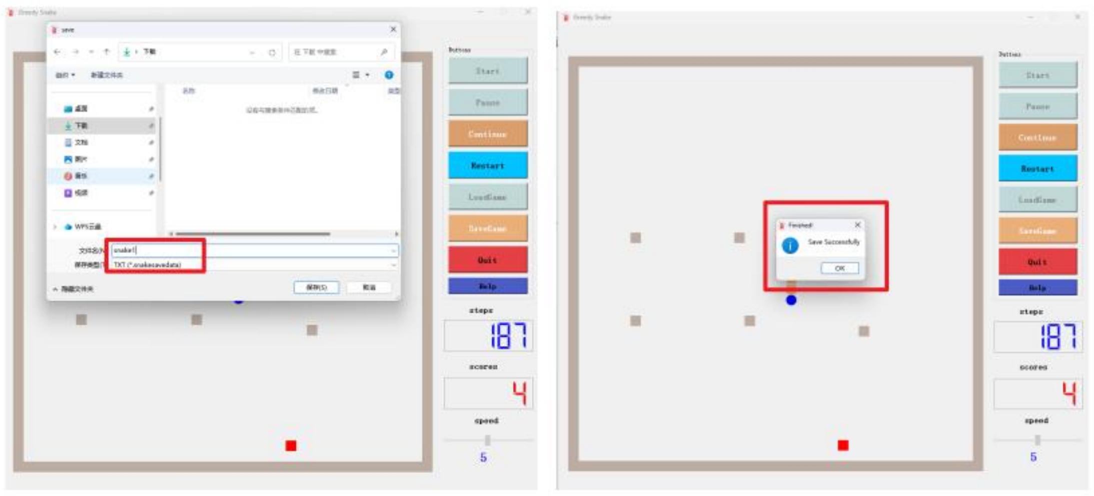


Figure 4.6 Click the save game button



Finished

Save Successfully

OK

Figure 4.7 Save current game progress

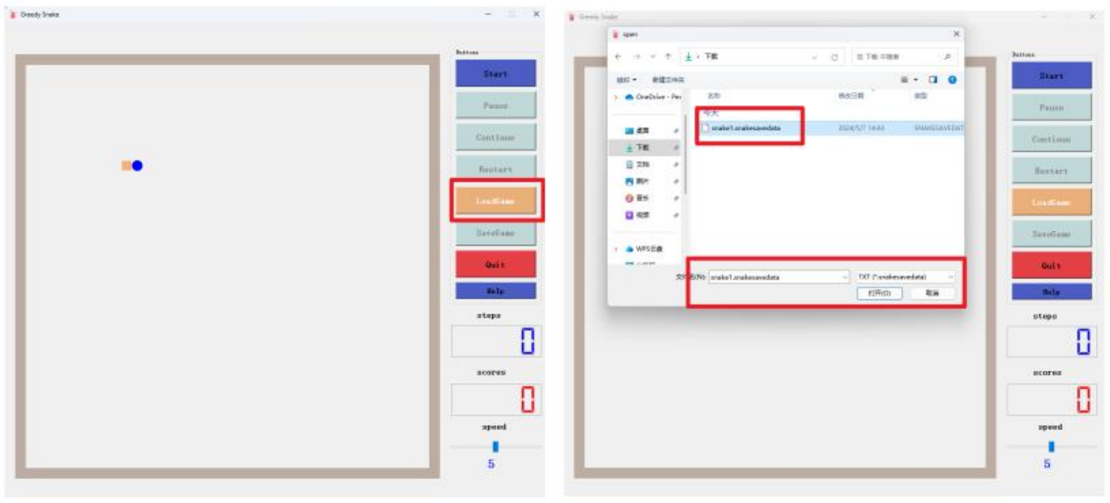
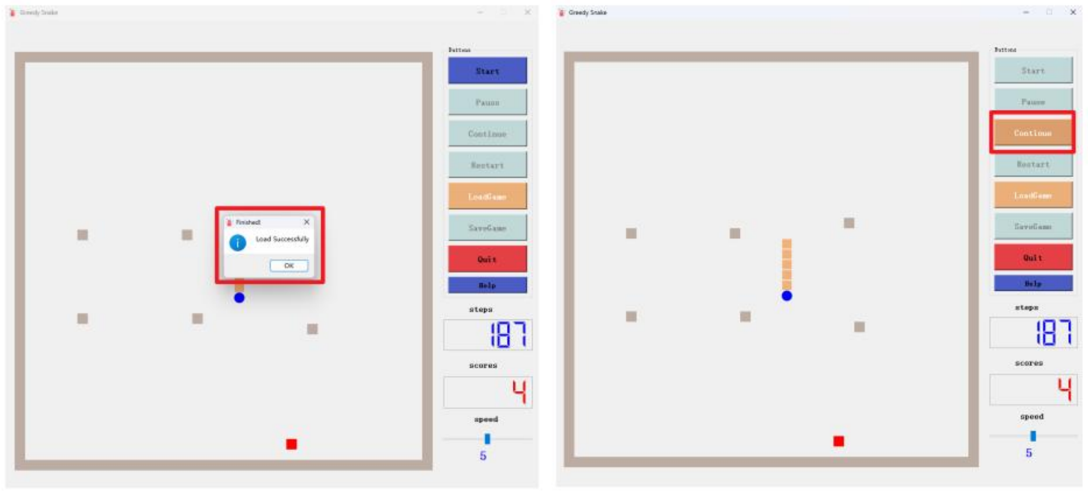


Figure 4.8 Read historical game progress



Finished

Load Successfully

OK

Figure 4.9 Continuing the game

5 Originality and external Tools

5.1 Originality

The project aims to develop this Snake game using QT instead of a console terminal type, building upon the existing open-source Snake game's data structure design (such as game map design, snake structure design, etc.) and game logic (game initialization logic, random food generation logic, etc.). The game will feature a well-designed graphical user interface with different color schemes. In terms of functionality, the game will allow interaction with the game environment by placing or removing obstacles before starting. It will include a speed adjustment slider that dynamically affects the snake's movement speed. The game will have comprehensive control buttons, including Start, Pause, Resume, Restart, Save, Load, and Exit. Background music will be incorporated to enhance the gaming atmosphere. The game will also feature save and load functions, allowing players to save their progress at any time and resume later without losing progress. Additionally, there will be a dedicated Help section accessible via a button, providing players with game rules and instructions.

5.2 External Tools

The development mainly utilizes Qt's built-in library, which provides a rich set of functionalities for building graphical user interfaces and handling multimedia elements. This greatly simplifies the process of developing complex user interfaces. Additionally, the project uses Git as the version control system. Git supports distributed operations, making it convenient for collaborative work and managing version history. The project code is hosted on Gitee, which not only facilitates version tracking and collaborative development but also provides tools for code review and issue tracking, enhancing the visibility and openness of the project.

6 Program highlights

The gains from this project development are as follows:

① Enhanced project architecture planning skills: This includes understanding the overall project logic and designing every detail, such as game logic, UI interface, and resource management, making the project more flexible and maintainable. This practical experience not only deepened the understanding of system architecture design but also cultivated the ability to think globally and finely control during the development process.

② Strengthened ability to solve code bugs: Through debugging and identifying the source of bugs, learned to systematically analyze errors, quickly locate and fix defects in the code. Faced with code, cultivated habits of writing clearer and more maintainable code, and learned to use debugging tools to accurately trace the process of bug occurrence.

③ Enhanced communication and collaboration skills: Software development is a collaborative process where effective communication is crucial. In this project, Git and Gitee were used as collaboration platforms, which not only helped manage code versions but also provided a platform for communication and collaboration. Team members could effectively exchange ideas, review code, and provide improvement suggestions, greatly enhancing the overall quality of the project and the efficiency of collaboration among team members.

7 References

[1] https://blog.csdn.net/w1u1kan/article/details/124073931

[2] https://juejin.cn/post/7022846211786801159

[3] https://www.kancloud.cn/kancloud/qt-study-road-2/99470

[4] https://www.jb51.net/article/212485.htm

[5] https://github.com/Maserhe/MyeatSnack

[6] https://github.com/Wytrix/Gluttonous-Snake-Game

[7] https://github.com/muzhi621/Snakes

[8] https://github.com/liplay/QT-Snake