**Project Report**

**Department & Institution**

Computer Science & Engineering,

Khulna University of Engineering & Technology

**(*CSE-KUET*)**

**Course**

CSE 1206 **(*Object Oriented Programming*)**

**Project**

Terminal-Based 2D Game **(*Tunneler*)**

**by**

Md. Shifat Hasan

**(2107067)**

**Introduction to OOP with C++**

In the dynamic field of computer science, Object-Oriented Programming (OOP) stands as a cornerstone methodology essential for robust, scalable, and maintainable software design.

C++, a language renowned for its efficiency and expressive power, serves at the forefront of this paradigm, contributing significantly to the creation of intricate software systems. Object-oriented programming, organizing code into reusable objects, promotes clear separation of concerns, enabling developers to design and implement complex systems comprehensibly.

The academic exploration focuses on the principles of Object-Oriented Programming with C++, covering foundational concepts such as encapsulation, inheritance, and polymorphism.

Additionally, advanced topics including templates, exception handling, and the Standard Template Library (STL) are explored, enhancing the expressive capacity of C++ for the development of scalable and maintainable software systems.

This academic journey aims to equip readers with the knowledge and skills to effectively leverage C++ for object-oriented design, fostering a nuanced understanding of the symbiotic relationship between Object-Oriented Programming and the C++ language.

**Citation:**

OpenAI. (2023). *ChatGPT* (September 25 Version) [Large language model]. https://chat.openai.com

**OOP Features in C++**

**Classes and Objects**

User-defined data types with encapsulated data and methods.

**Encapsulation**

Hides internal details and restricts data access.

**Inheritance**

Subclasses inherit properties and behaviors from base classes.

**Polymorphism**

Allows treating different classes as a common base class.

Includes compile-time and runtime polymorphism.

**Abstraction**

Simplifies complex systems through class modeling.

Abstract classes and pure virtual functions support abstraction.

**Operator Overloading**

Defines custom behaviors for operators with user-defined types.

**Function Overloading and Overriding**

Defines multiple functions with the same name but different parameter lists.

**Templates**

Supports generic programming for any data type.

**Multiple Inheritance**

Allows a class to inherit from more than one base class.

**Dynamic Memory Allocation**

***new*** and ***delete*** for efficient memory management.

**Constructors and Destructors**

Initialize and clean up object resources.

**Friend Functions**

Accesses private and protected class members.

**Static Members**

Belong to the class rather than instances.

**Namespaces**

Organizes code and prevents naming conflicts.

**Exception Handling**

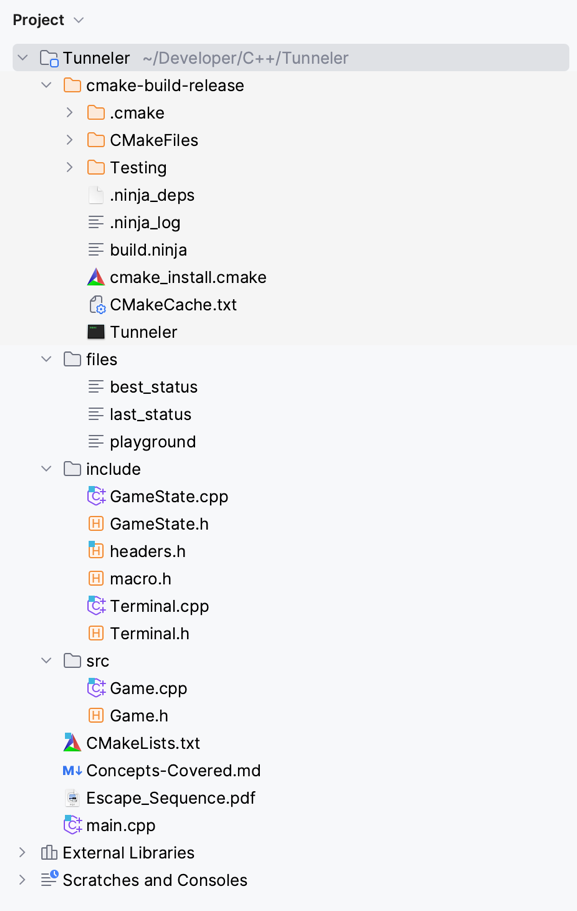
***try***, ***catch,*** and ***throw*** for robust error handling.

**Standard Template Library (STL)**

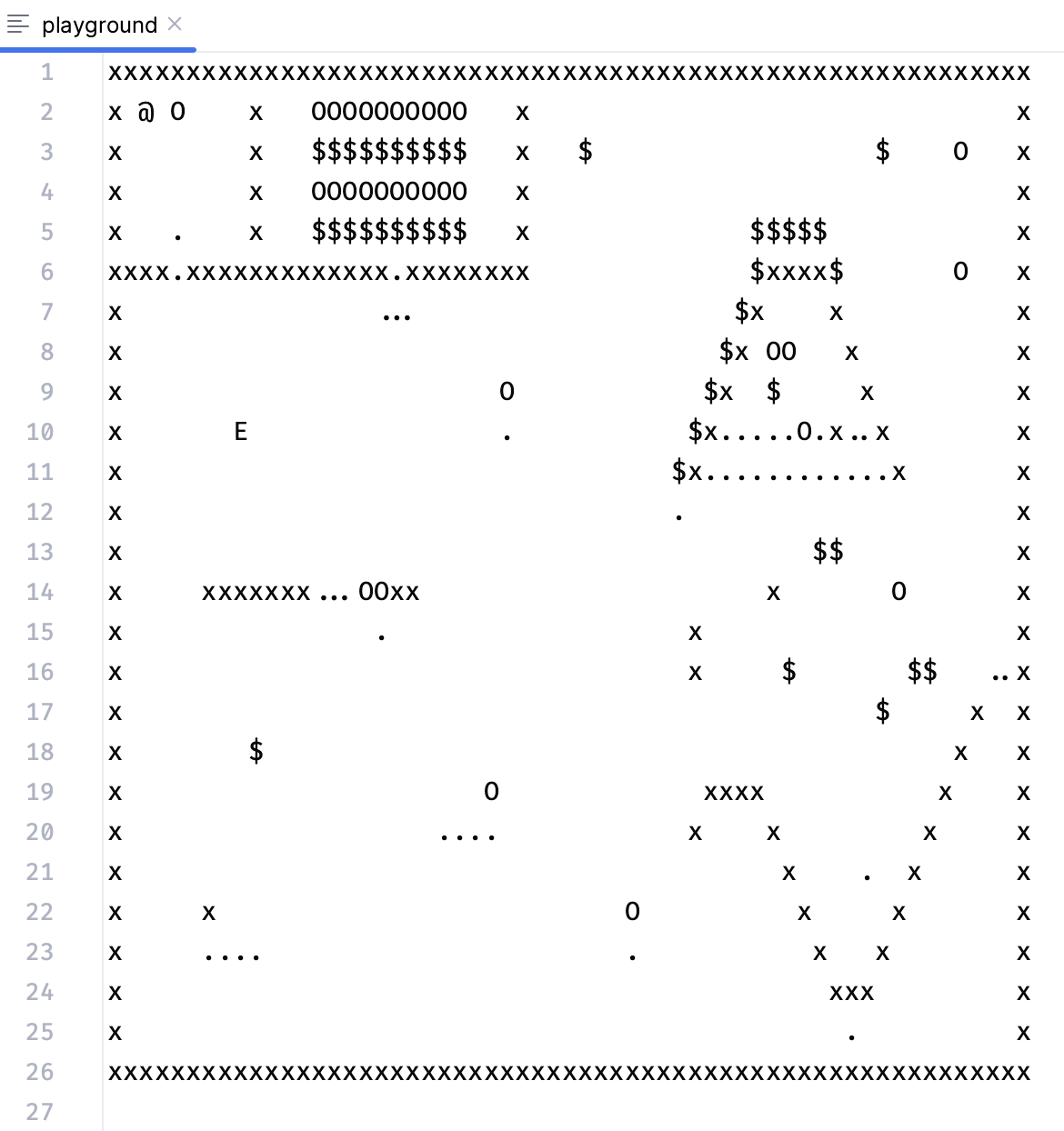
Template classes and functions for common data structures and algorithms.

**The Features I’ve Used in My Project**

* **Classes and Objects**
* **Encapsulation**
* **Inheritance**
* **Polymorphism**
* **Abstraction | Pure Virtual Function**
* **Function Overriding**
* **Constructors and Destructors**
* **Static Class-Members**
* **File Management**

**Project’s File Structure**

**Playground**

****

After Rendering

****

**macro.h**

This header file provides a collection of macros and constants for managing terminal behavior and appearance.

**Header Guard**

The ***#ifndef***, ***#define***, and ***#endif*** directives are used to create a header guard, preventing multiple inclusions of the same file.

**Enumeration Class: *Key***

Enumeration for keyboard keys with values such as ***None***, ***Up***, ***Down***, ***Right***, and ***Left***.

**Constant String: *Workspace***

Specifies the workspace path as *“/Users/\*\*\*/Developer/C++/Tunneler”*.

***TUNNELER* and *MainTerminal***

Macros for changing terminal profiles.

**FLUSH**

Defines a macro for flushing the standard output and input streams.

**Frame Dimensions and Cycle**

Specifies constants for frame rows, columns, and the frame cycle duration.

**Terminal Manipulation**

Resize the terminal window using ***WINDOW\_RESIZE***.

For clearing the screen and moving the cursor to the top-left position use ***SCREEN\_CLEAR\_ALL***.

**Cursor Control**

For hiding and showing the cursor use ***CURSOR\_HIDE*** and ***CURSOR\_SHOW***.

For setting the cursor position ***CURSOR\_POSITION***.

**Color Formatting**

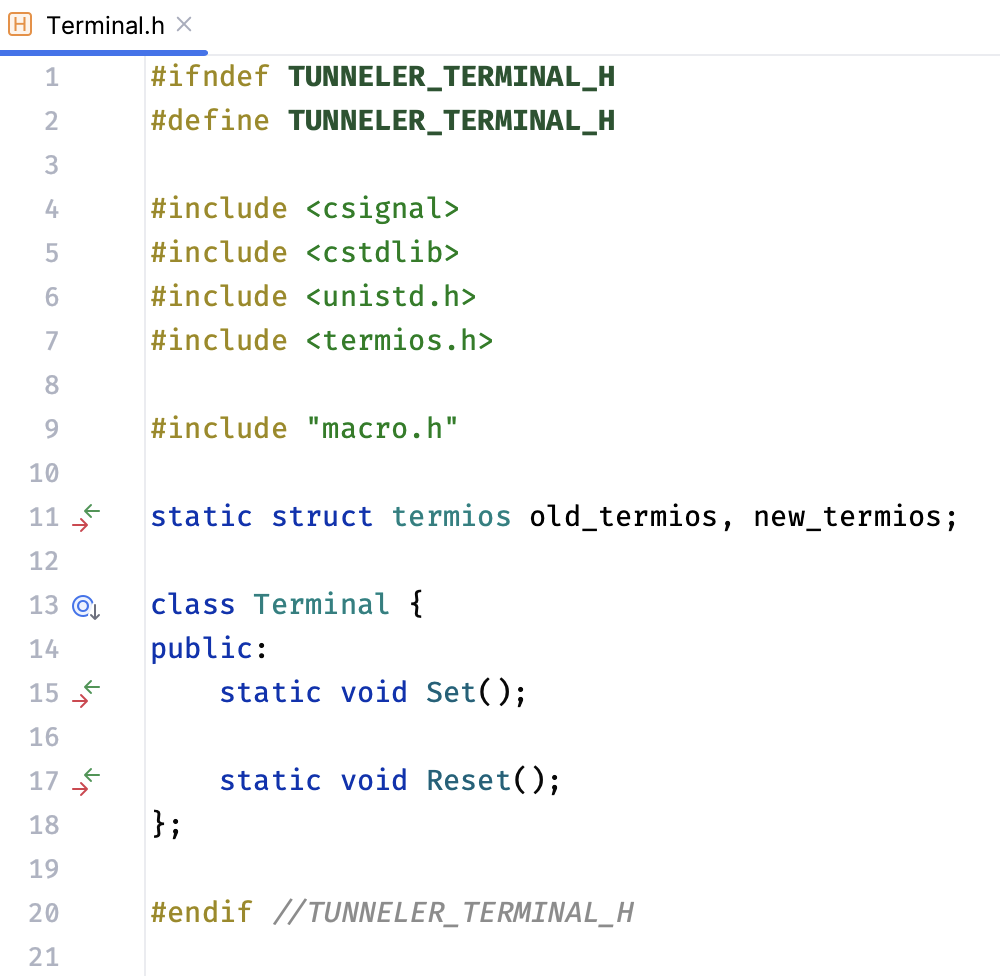
For setting foreground and background colors using RGBA values using macros ***RGBA\_FG***, ***RGBA\_BG.***

For resetting color attributes using ***RGBA\_RESET***.

**Citation:**

ANSI escape code. (2023, September 28). In *Wikipedia*. https://en.wikipedia.org/wiki/ANSI\_escape\_code

**Terminal.h**



**Header Guard**

The ***#ifndef***, ***#define***, and ***#endif*** directives are used to create a header guard, preventing multiple inclusions of the same file.

**Header File Inclusions**

Includes necessary header files, such as <csignal>, <cstdlib>, <unistd.h> and <termios.h>.

Also, includes a custom header file named ***macro.h*** (containing macro definitions).

**Static Variables**

Declares two static instances of the ***termios*** structure: ***old\_termios*** and ***new\_termios***. These will be used to store the terminal settings.

**Class: Terminal**

Declares a class named ***Terminal*** with public methods only.

**Static Methods**

**Set():** This static method is responsible for setting the terminal configurations. It modifies the terminal settings for specific functionality.

**Reset():** This static method is intended to reset the terminal configurations to their original state, using the stored values of ***old\_termios***.

**Terminal.cpp**



**Terminal::Reset()**

**Color Reset and Cursor Position:**

Resets the terminal color attributes using the ***RGBA\_RESET*** macro.

Moves the cursor to the top-left position using ***CURSOR\_POSITION(1, 1)***.

**Restore Terminal Settings:**

Restores the original terminal settings saved in ***old\_termios*** using ***tcsetattr(STDIN\_FILENO, TCSANOW, &old\_termios)***.

**Terminal::Set()**

**Save Original Terminal Settings:**

Saves the original terminal settings in ***old\_termios*** using ***tcgetattr(STDIN\_FILENO, &old\_termios)***.

**Modify Terminal Settings:**

Creates a copy of the original settings in ***new\_termios***.

Modifies ***new\_termios*** for ***non-canonical mode*** and ***no echo*** (disabling line buffering and local echo).

Configures ***VMIN*** and ***VTIME*** to set the terminal to ***non-blocking mode***.

Applies the modified settings using ***tcsetattr(STDIN\_FILENO, TCSANOW, &new\_termios)***.

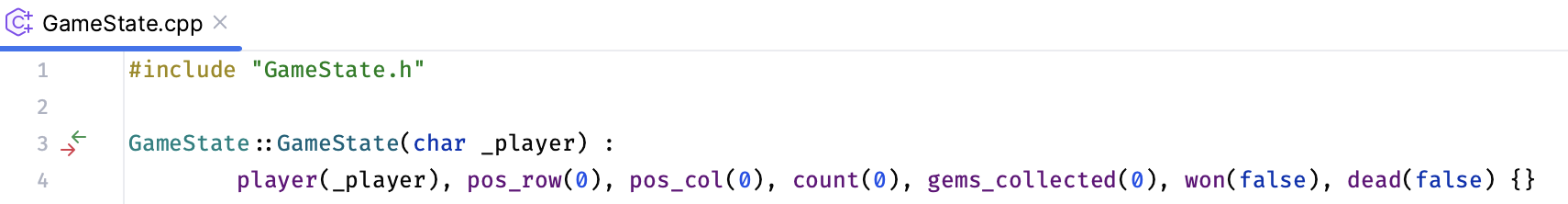
**GameState.h**



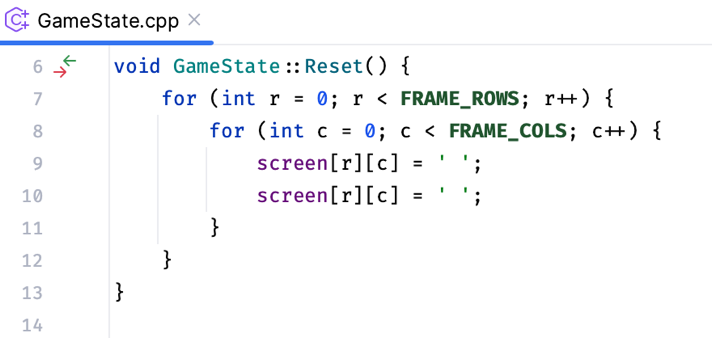
These declarations are quite self-explanatory.

This header file declares a ***GameState*** class, which is a base class for managing the state of a game. It inherits publicly from the ***Terminal*** class and includes several data members and member functions for game state management. Some functions are virtual, indicating that they need to be implemented by derived classes.

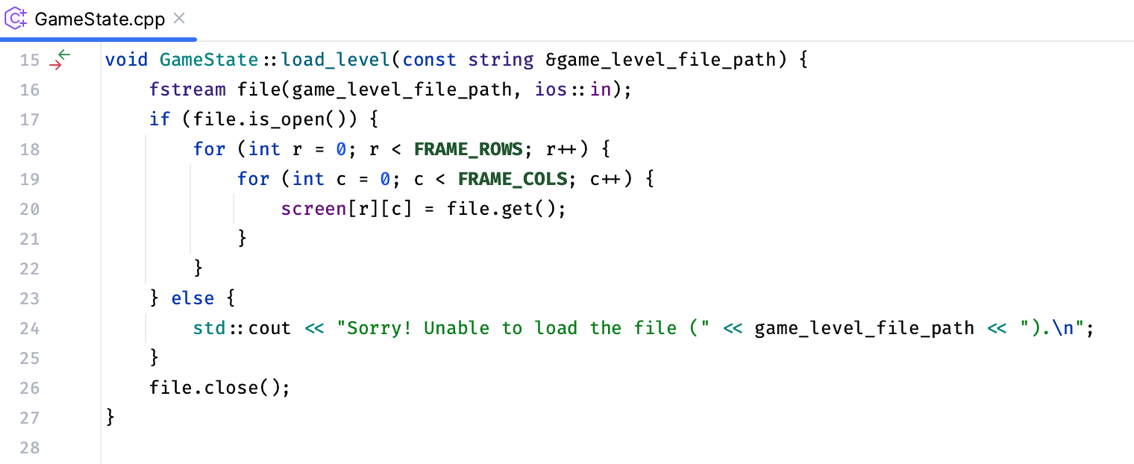
**GameState.cpp**



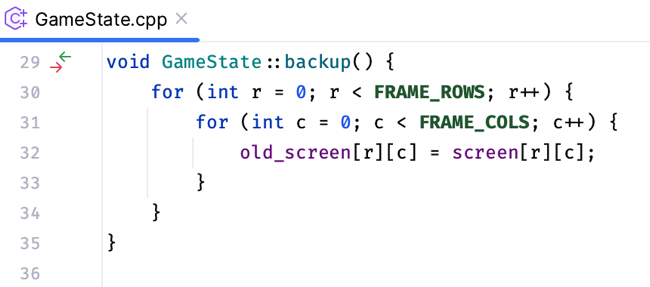
It’s the constructor to define the player.



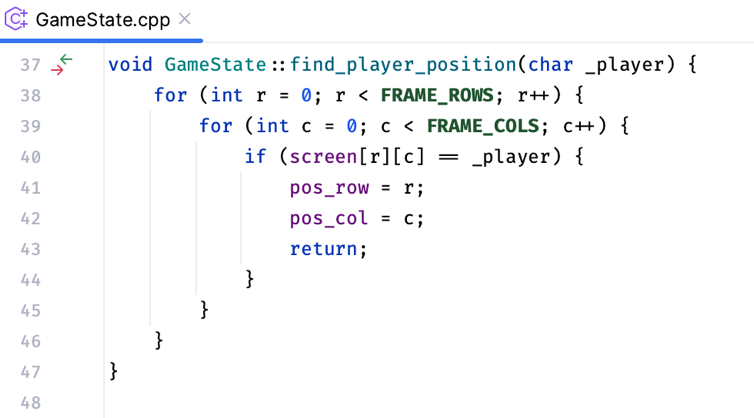
It can reset the screen variables to space [‘ ‘] character.



it’s to load the level or playground in the terminal.



It’s to backup the latest screen to the previous screen.

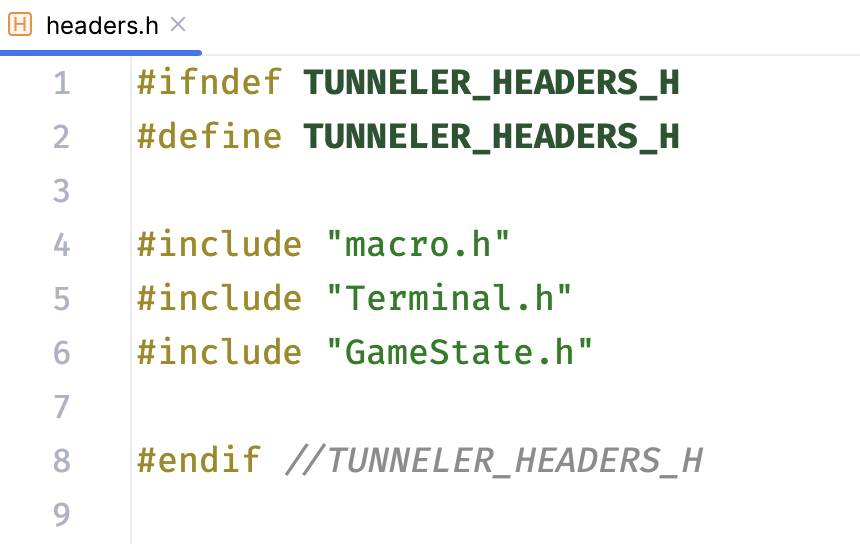


It can find the player on the screen.



The ***Listen\_Event()*** Function can detect a ***Keyboard-Arrow-Key-Press*** and as per the Key-Press-Event, the ***Read\_Key()*** Function can uniquely identify the Key and return the result to the ***Listen\_Event()*** Function so that it can register the final Key-Press into the member variable key of the ***GameState*** class.

**headers.h**



**Game.h**



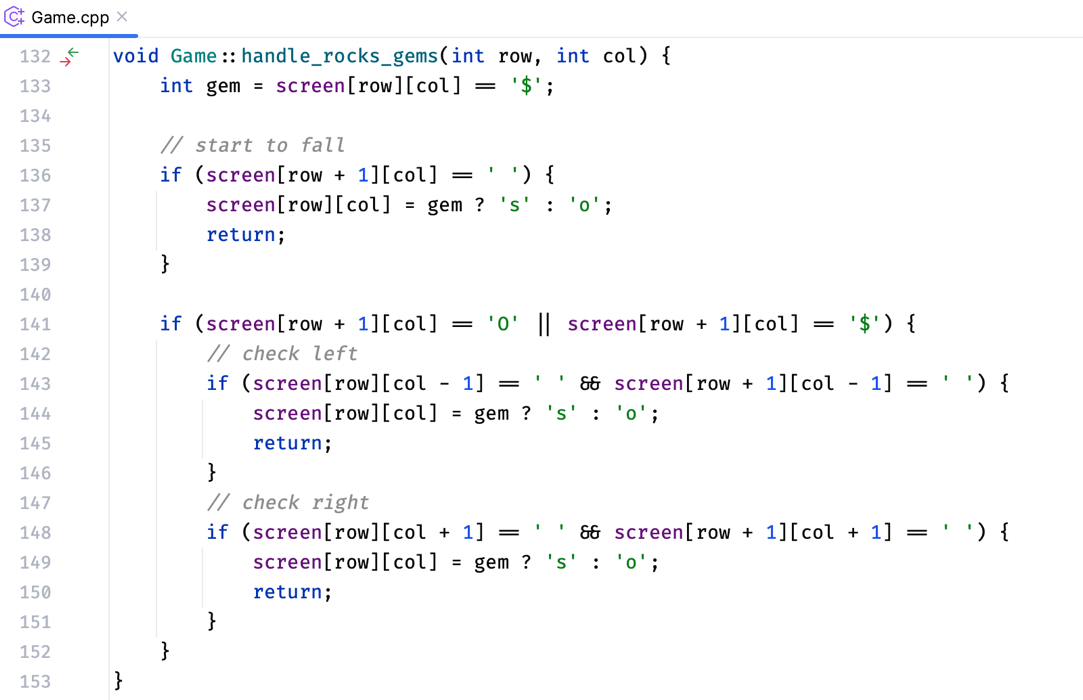
This is the dedicated class for the game.

**Game.cpp**

It can control the player’s position and properties based on Key-Press.

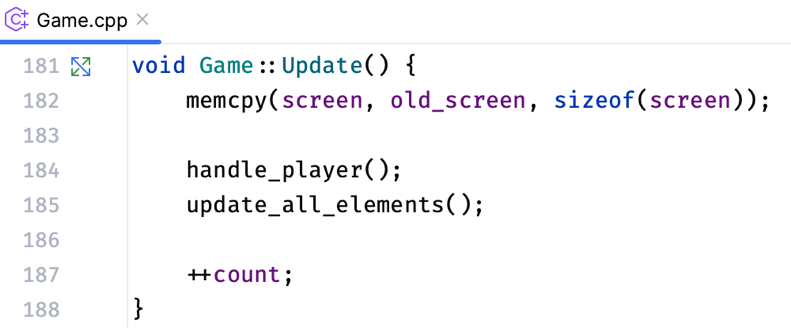


It handles the falling rock’s and gem’s position and properties.



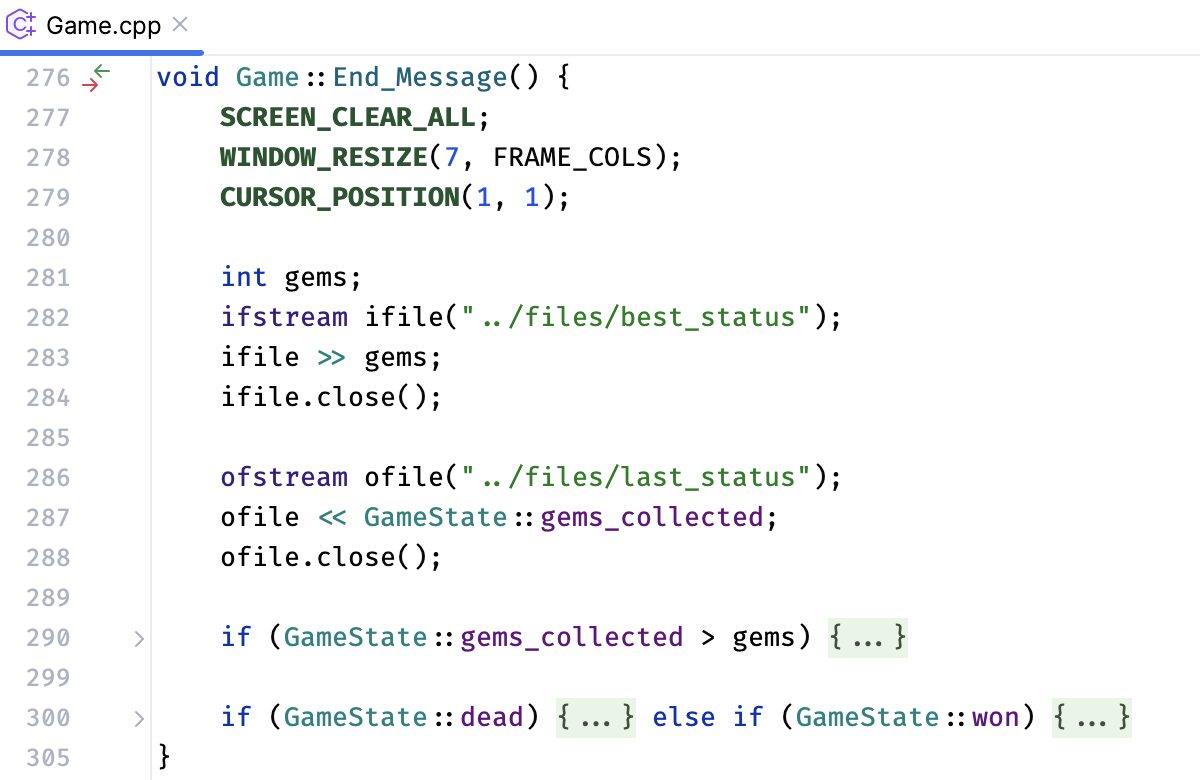
It handles the rock’s and gem’s position and properties, not the falling ones.

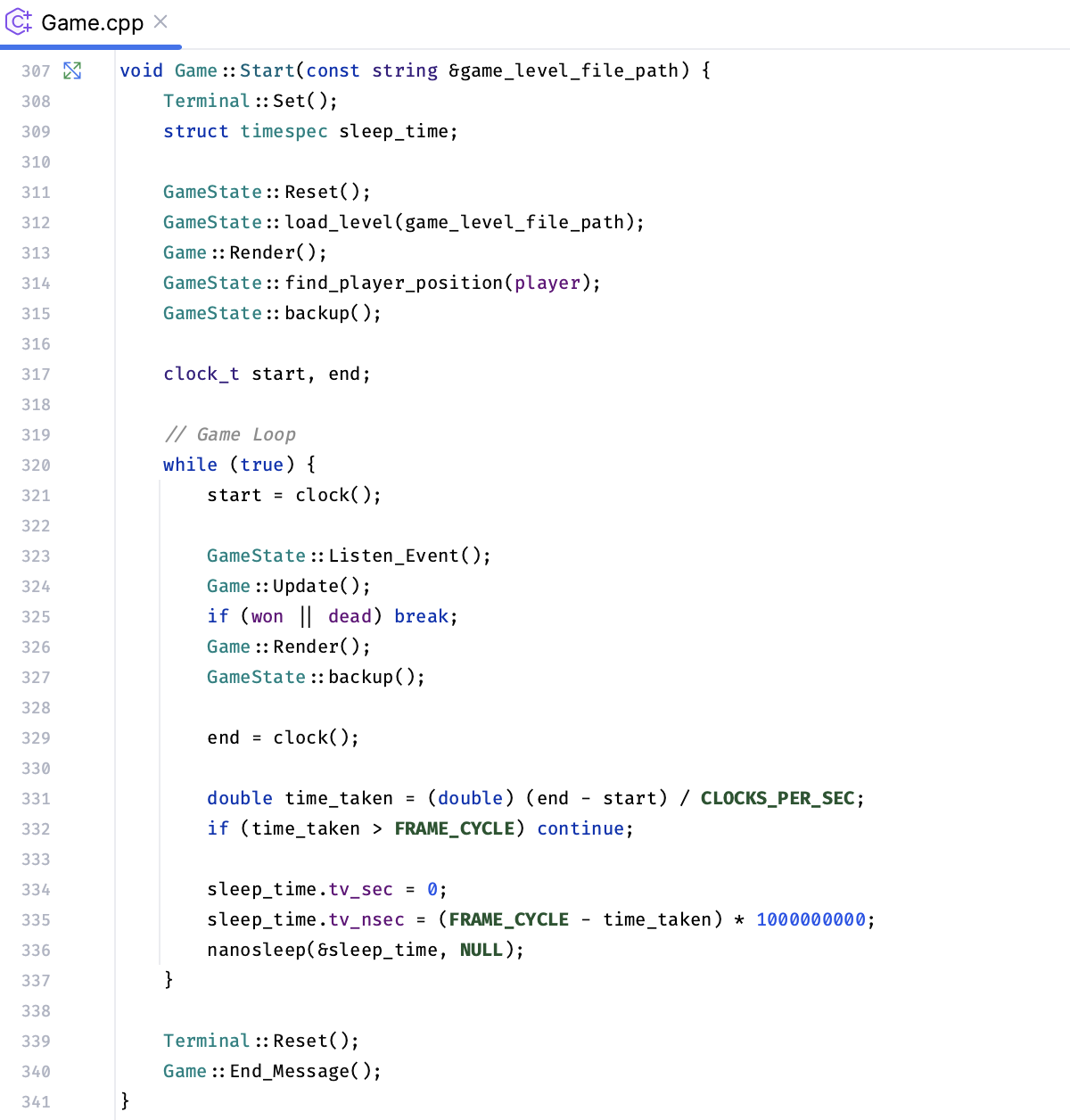
It updated all the elements present on the screen based on expected effects.



It backs up the latest screen. Then, handles the player and all the elements by calling the previously discussed functions.

It can render the screen in the terminal as expected.

It shows an end message while exiting the game.

It starts the game, maintains the game loop, and does all necessary things to keep playing the game till winning or dying.

**And, at last, there’s a main() function to start the program altogether.**

**In conclusion,** learning **OOP** with **C++** equips programmers with a powerful set of tools and concepts that enhance code quality, promote efficient development practices, and contribute to the creation of scalable and maintainable software systems.

**GitHub Repository**

**GitHub Link:**

<https://github.com/ShifatHasanGNS/2107067-OOP-Project.git>

**Or, Scan This QR Code:**

After **October 8, 2023**, you will lose access to this QR Code