Modern Tetris Implementation: Object-Oriented Design & Documentation

By CS23B1023 Athidathan K

November 11, 2024

Contents

1 Overview

This documentation provides an overview of the Tetris project, explaining its structure, specifications, and how various Object-Oriented Programming (OOP) concepts are incorporated, including encapsulation, abstraction, inheritance, polymorphism, operator overloading, and plans for future use of templates and virtual functions.

The Tetris project is implemented in C++ using the Raylib graphics library. The game simulates the classic Tetris gameplay, where different shaped blocks fall into a grid, and the player manipulates them to form complete rows.

2 Project Structure

```
.vscode/
build/
Font/
    Handjet.ttf
lib/
    raylib.h
src/
    block.cpp
    block.h
    blocks.cpp
    colors.cpp
    colors.h
    game.cpp
    game.h
    grid.cpp
    grid.h
    main.cpp
    position.cpp
    position.h
Makefile
Documentation.tex
main.code-workspace
```

3 Classes and Files

3.1 `Position` Class (`src/position.h`, `src/position.cpp`)

Represents the position of a block in the grid.

```cpp
class Position {
public:
    int row;
    int column;
    Position(int row, int column);
    Position operator+(const Position& other) const ;
};
```

3.2 Block Class Hierarchy (src/block.h, src/block.cpp,src/blocks.cpp)

3.2.1 Base Block Class

Encapsulation: The Block class encapsulates the properties and behaviors of a Tetris block.

```
class Block {
public:
    Block();
    void Draw(int offsetX, int offsetY);
    void Move(int rows, int columns);
    void Rotate();
    void UndoRotation();
    std::vector<Position> GetCellPositions();
protected:
    int id;
    int rotationState;
    int rowOffset;
    int columnOffset;
    std::map<int, std::vector<Position>> cells;
    int cellSize;
    std::vector<Color> colors;
};
```

Example: The Block class manages its state internally and provides public methods for interaction, demonstrating encapsulation.

3.3 Derived Block Classes (src/blocks.cpp)

Inheritance: Multiple classes (LBlock, JBlock, IBlock, OBlock, SBlock, TBlock, ZBlock) inherit from the Block class.

Example:

```
// src/blocks.cpp

#include "block.h"

// LBlock class inheriting from Block
class LBlock : public Block {
public:
    LBlock() {
        id = 1;
        cells[0] = {Position(0, 2), Position(1, 0), Position(1, 1), Position(1, 2)};
        cells[1] = {Position(0, 1), Position(1, 1), Position(2, 1), Position(2, 2)};
        cells[2] = {Position(1, 0), Position(1, 1), Position(1, 2), Position(2, 0)};
        cells[3] = {Position(0, 0), Position(0, 1), Position(1, 1), Position(2, 1)};
        Move(0, 3);
    }
};

// Other block classes (JBlock, IBlock, OBlock, SBlock, TBlock, ZBlock) defined
    similarly
```

Each derived class initializes its specific shape by setting up the cells member variable inherited from the Block class.

3.4 Grid Class (src/grid.h, src/grid.cpp)

Represents the game grid where the blocks are placed. Manages the grid state, drawing, and clearing full rows.

```cpp
// src/grid.h

#pragma once
#include <vector>
#include <raylib.h>

class Grid{
public:
    Grid();
    void Initialize();
    void Print();
    void Draw();
    bool IsCellOutside(int row, int column);
    bool IsCellEmpty(int row, int column);
    int ClearFullRows();
private:
    int grid[20][10];
    int numRows;
    int numCols;
    int cellSize;
    std::vector<Color> colors;
    bool IsRowFull(int row);
    void ClearRow(int row);
    void MoveRowDown(int row, int numRows);

friend class Game;
};
```

3.5 Game Class (src/game.h, src/game.cpp)

Controls the overall game logic.

Abstraction: Provides simplified methods to handle game mechanics like input handling and block movement.

```cpp
// src/game.h

#pragma once
#include "grid.h"
#include "block.h"
#include "raylib.h"

class Game {
public:
    Game();
    ~Game();
    void Draw();
    void HandleInput();
    void MoveBlockDown();
    bool gameOver;
    int score;
private:
    void MoveBlockLeft();
    void MoveBlockRight();
    void RotateBlock();
    void LockBlock();
    bool IsBlockOutside();
    bool BlockFits();
    void Reset();
    void UpdateScore(int linesCleared, int moveDownPoints);
    Grid grid;
    Block currentBlock;
    Block nextBlock;
    std::vector<Block> blocks;
};
```

4 OOP Concepts Incorporated

4.1 Encapsulation

Classes manage their own data and expose functionality through public methods. Access specifiers `public` and `private` are used to enforce access control, separating the interface from the implementation.

Example:

In the `Block` class (`src/block.h`), data members are declared as `protected` or `private`, while member functions that form the interface are declared as `public`.

```
// src/block.h

class Block {
public:
    Block();
    void Draw(int offsetX, int offsetY);
    void Move(int rows, int columns);
    void Rotate();
    void UndoRotation();
    std::vector<Position> GetCellPositions();
protected:
    int id;
    int rotationState;
    int rowOffset;
    int columnOffset;
    std::map<int, std::vector<Position>> cells;
    int cellSize;
    std::vector<Color> colors;
};
```

The implementation of these member functions is separated in the corresponding `.cpp` file (`src/block.cpp`), keeping the interface clean and promoting encapsulation.

4.2 Abstraction

Abstraction is achieved by using header files to define class interfaces. The implementation details are hidden in the `.cpp` files, allowing users of the classes to interact with them without needing to understand the internal workings.

Example:

The `Game` class interface is defined in `src/game.h`, abstracting the game mechanics and providing a clear interface for interaction.

```cpp
// src/game.h

class Game {
public:
    Game();
    ~Game();
    void Draw();
    void HandleInput();
    void MoveBlockDown();
    bool gameOver;
    int score;
private:
    // Private member functions and variables
};
```

4.3 Inheritance

Inheritance is used to create specialized block types from a base `Block` class. This allows for code reuse and establishes a hierarchical class structure.

Class Hierarchy:

- `Block` (base class)
  - `IBlock`
  - `JBlock`
  - `LBlock`
  - `OBlock`
  - `SBlock`
  - `TBlock`
  - `ZBlock`

Example:

```cpp
// src/blocks.cpp

class TBlock : public Block {
public:
    TBlock() {
        id = 6;
        cells[0] = {Position(0, 1), Position(1, 0), Position(1, 1), Position(1, 2)};
        // other rotation states...
        Move(0, 3);
    }
};
```

4.4 Polymorphism

Polymorphism allows objects to be treated as instances of their base class rather than their actual derived class. This project demonstrates both compile-time and run-time polymorphism.

4.4.1 Compile-Time Polymorphism

Function Overloading (used):
Function overloading is utilized in the project to allow functions with the same name to have different parameter lists.

```cpp
// src/block.h
class Block {
public:
    void Move(int rows, int columns);

    // Overloaded Move function
    void Move(const Position& pos);
};
```

Operator Overloading (used):
Operator overloading is implemented in the `Position` class to simplify position arithmetic.

```cpp
// src/position.cpp

Position Position::operator+(const Position &other) const{
    return Position(row + other.row, column + other.column);
}
```

This allows positions to be added using the `+` operator.
Templates (not currently used):
Templates are not explicitly used in this project as the data types are consistent, and there's no immediate need for generic programming. However, templates could be implemented for generic data structures or algorithms if needed in future enhancements.

4.4.2 Run-Time Polymorphism

Virtual Functions (planned for future use):
Virtual functions are intended to be used in future extensions, such as when adding a `Player` class to extend multiplayer functionality.
Example:
In future development, a `Player` class could be defined with virtual methods that are overridden by derived classes for different player types (e.g., human player, AI player).

5 Example Code Snippets

5.1 Drawing a Block

```cpp
// src/block.cpp

void Block::Draw(int offsetX, int offsetY) {
    std::vector<Position> tiles = GetCellPositions();
    for (const Position& item : tiles) {
        DrawRectangle(
            item.column * cellSize + offsetX,
            item.row * cellSize + offsetY,
            cellSize - 1, cellSize - 1,
            colors[id]
        );
    }
}
```

Demonstrates encapsulation by handling drawing internally.

5.2 Handling Block Rotation

```cpp
// src/block.cpp

void Block::Rotate() {
    rotationState = (rotationState + 1) % cells.size();
}
```

Shows how the block manages its rotation state.

5.3 Moving Blocks Down Automatically

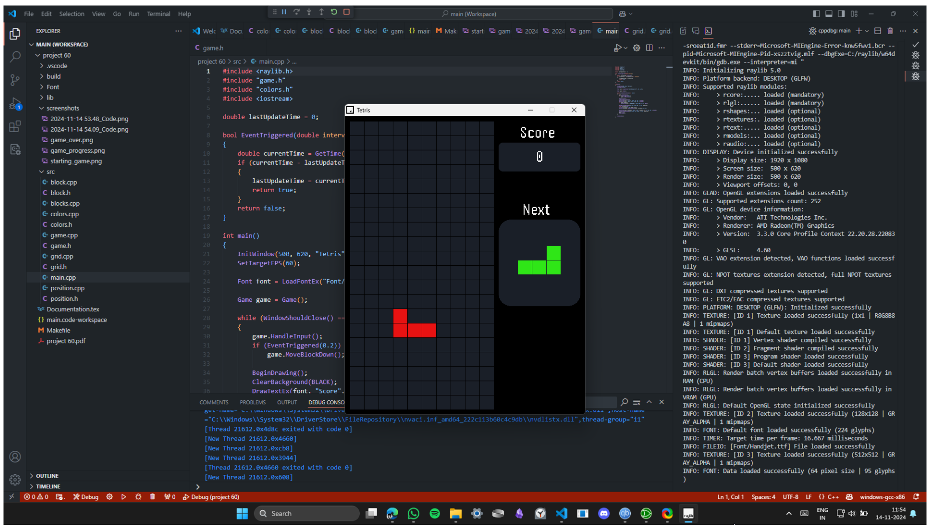
```cpp
// src/main.cpp

if (EventTriggered(0.2)) { // Game speed is 0.2 seconds
    game.MoveBlockDown();
}
```

Demonstrates how the game logic moves the current block down at regular intervals.

6 Screenshots

Below are some screenshots demonstrating the gameplay of the Tetris project.



```
#include <raylib.h>
#include "game.h"
#include "colors.h"
#include <iostream>

double lastUpdateTime = 0;

bool EventTriggered(double interv
{
    double currentTime = GetTime(
    if (currentTime - lastUpdateT
    {
        lastUpdateTime = currentT
        return true;
    }
    return false;
}

int main()
{
    InitWindow(500, 620, "Tetris"
    SetTargetFPS(60);

    Font font = LoadFontEx("Font/

    Game game = Game();

    while (WindowShouldClose() ==
    {
        game.HandleInput();
        if (EventTriggered(0.2))
            game.MoveBlockDown();

        BeginDrawing();
        ClearBackground(BLACK);
        DrawTextEx(font, "Score".
```

```
="C:\\Windows\\System32\\DriverStore\\FileRepository\\nvaci.inf_amd64_222c113b60c4c9db\\nvdlistx.dll",thread-group="i1"
[Thread 21612.0x4d8c exited with code 0]
[New Thread 21612.0x4660]
[New Thread 21612.0xcb8]
[New Thread 21612.0x3944]
[Thread 21612.0x4660 exited with code 0]
[New Thread 21612.0x608]
```

```
-sroeat1d.fmr --stderr=Microsoft-MIEngine-Error-knw5fwv1.bcr --
pid=Microsoft-MIEngine-Pid-xszztvig.mlf --dbgExe=C:/raylib/w64d
evkit/bin/gdb.exe --interpreter=mi "
INFO: Initializing raylib 5.0
INFO: Platform backend: DESKTOP (GLFW)
INFO: Supported raylib modules:
INFO:     > rcore:..... loaded (mandatory)
INFO:     > rlgl:...... loaded (mandatory)
INFO:     > rshapes:... loaded (optional)
INFO:     > rtextures:. loaded (optional)
INFO:     > rtext:..... loaded (optional)
INFO:     > rmodels:... loaded (optional)
INFO:     > raudio:.... loaded (optional)
INFO: DISPLAY: Device initialized successfully
INFO:     > Display size: 1920 x 1080
INFO:     > Screen size:  500 x 620
INFO:     > Render size:  500 x 620
INFO:     > Viewport offsets: 0, 0
INFO: GLAD: OpenGL extensions loaded successfully
INFO: GL: Supported extensions count: 252
INFO: GL: OpenGL device information:
INFO:     > Vendor:   ATI Technologies Inc.
INFO:     > Renderer: AMD Radeon(TM) Graphics
INFO:     > Version:  3.3.0 Core Profile Context 22.20.28.22083
0
INFO:     > GLSL:     4.60
INFO: GL: VAO extension detected, VAO functions loaded successf
ully
INFO: GL: NPOT textures extension detected, full NPOT textures
supported
INFO: GL: DXT compressed textures supported
INFO: GL: ETC2/EAC compressed textures supported
INFO: PLATFORM: DESKTOP (GLFW): Initialized successfully
INFO: TEXTURE: [ID 1] Texture loaded successfully (1x1 | R8G8B8
A8 | 1 mipmaps)
INFO: TEXTURE: [ID 1] Default texture loaded successfully
INFO: SHADER: [ID 1] Vertex shader compiled successfully
INFO: SHADER: [ID 2] Fragment shader compiled successfully
INFO: SHADER: [ID 3] Program shader loaded successfully
INFO: SHADER: [ID 3] Default shader loaded successfully
INFO: RLGL: Render batch vertex buffers loaded successfully in
RAM (CPU)
INFO: RLGL: Render batch vertex buffers loaded successfully in
VRAM (GPU)
INFO: RLGL: Default OpenGL state initialized successfully
INFO: TEXTURE: [ID 2] Texture loaded successfully (128x128 | GR
AY_ALPHA | 1 mipmaps)
INFO: FONT: Default font loaded successfully (224 glyphs)
INFO: TIMER: Target time per frame: 16.667 milliseconds
INFO: FILEIO: [Font/Handjet.ttf] File loaded successfully
INFO: TEXTURE: [ID 3] Texture loaded successfully (512x512 | GR
AY_ALPHA | 1 mipmaps)
INFO: FONT: Data loaded successfully (64 pixel size | 95 glyphs
)
```

Figure 1: Starting the Tetris game

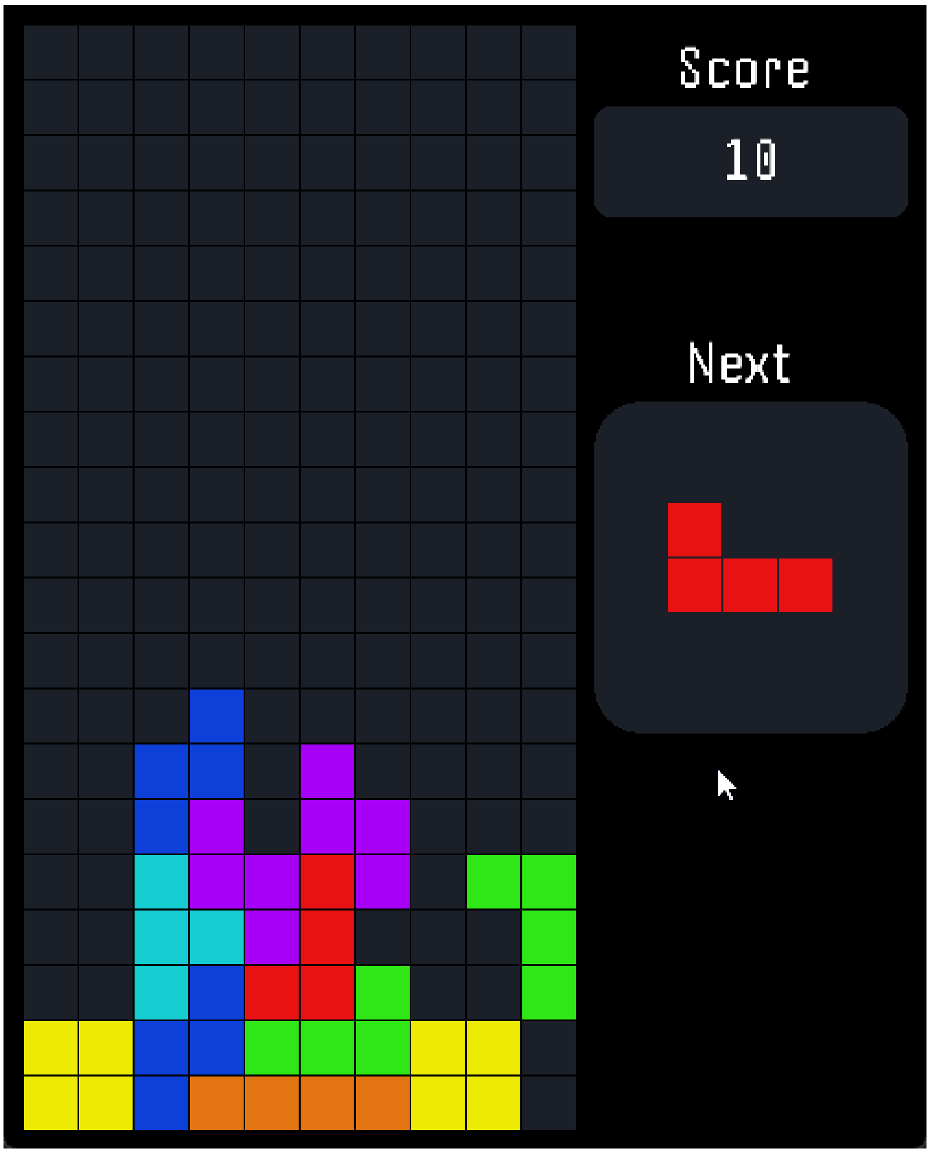


Figure 2: Gameplay in progress

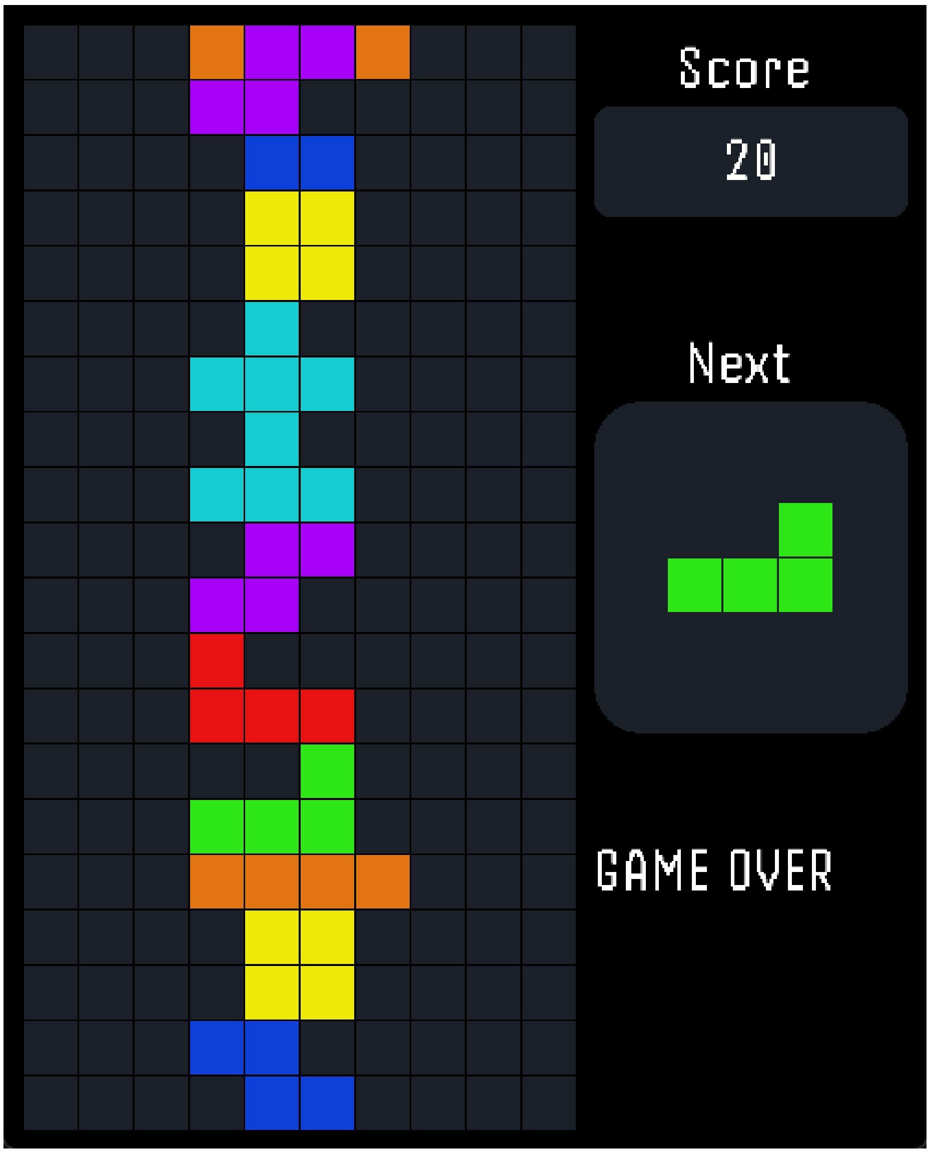


Figure 3: Game Over screen

7 Building the Project

The project is developed in C++ using the Raylib graphics library, and the instructions are tailored for Windows users. this project was developed on Windows 10 using MinGW-w64 as the compiler. The instructions assume a similar setup, but the project can be adapted for other platforms with minor modifications. The project can also be downloaded from my github `https://github.com/Athidathan/OOPtetris/`

7.1 Requirements

- **Operating System**: Windows 10 or higher
- **Compiler**: MinGW-w64 (GCC for Windows) or Microsoft Visual Studio
- **Library**: Raylib (version 4.0 or higher)
- **Make Build system**: MinGW-w64 includes `mingw32-make` for building the project using the provided `Makefile`.

7.2 Instructions

1. Install the required compiler and Raylib library.
2. Open the Command Prompt and navigate to the project directory:

```
    cd path\to\your\project
```

3. Compile the project using the provided `Makefile`:

```
    mingw32-make
```

4. Run the game executable:

```
    game.exe
```

`make`

This command will compile the source files and link the necessary libraries to produce the final executable.

8 Future Plans

Looking ahead, there are several enhancements planned to expand the functionality and educational value of this Tetris project:

- **Local Multiplayer**: Implementing a local multiplayer mode where two players can play simultaneously on the same machine. This feature will introduce new dynamics and while incorporating polymorphism.
- **Additional Game Modes**:
  - **Versus Mode**: Players compete against each other, with mechanics such as sending garbage lines to the opponent when clearing multiple lines.
  - **Challenge Mode**: Players face progressively increasing difficulty levels with unique challenges.
  - **Time Attack Mode**: Players aim to score as high as possible within a limited time frame.

9 Conclusion

This Tetris project serves as a practical example of implementing OOP concepts in C++. Through encapsulation, abstraction, inheritance, polymorphism, and operator overloading, the project provides a comprehensive introduction to OOP in a game development context.

The project demonstrates how to structure a C++ application using classes and inheritance, how to manage game state and logic, and how to integrate a graphics library (Raylib) to handle rendering and input.

With future enhancements, especially when the game is extended to support multiplayer functionality. The project offers ample opportunities to delve into advanced topics such as networking, artificial intelligence, and improved user interfaces, further enriching the learning experience.

References

[1] Raylib Official Website. `https://www.raylib.com/`

[2] Raylib Learning Resources. `https://github.com/raysan5/raylib/wiki`

[3] Tetris Game. `https://en.wikipedia.org/wiki/Tetris`