Object-Oriented Programming in C++
Tetris

Clea Dronne
10475323

May 22, 2022

Abstract

The final project that was chosen is the famous video game Tetris. The aim was to create a terminal based game, using Object-Oriented Programming with all the main functionalities of the existing version of the game. This was successfully achieved, a Tetris game was implemented where pieces fall, move and rotate, lines disappear when full and game over is reached when no more pieces can appear on the screen.

1 Introduction

1.1 Why Tetris?

Tetris is a puzzle game that was first created in 1984 by Alexey Pajitnov, a Russian-American video game designer. Since then, it has become a classic video game and been developed across multiple platforms [1].

1.2 Rules of Tetris

The aim of Tetris is to fill lines using geometric shapes called "tetrominos", which progressively move down on the screen. When they've reached the bottom of the screen or another piece, they stop moving, and a new piece gets generated at the top of the screen. Pieces can move right or left, and can rotate. There is also a functionality to speed a piece down. When a line is full, it disappears and pieces above move down where that line used to be. The game is over when the pieces that get generated at the top of the board cannot move down anymore. Pieces fall increasingly fast as the game progresses and higher levels are reached [2].

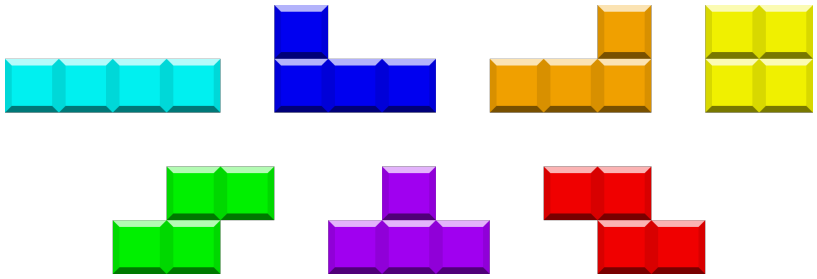

Figure 1: Tetrominos. They are ordered alphabetically and are respectively named "I", "J", "L", "O", "S", "T" and "Z".

2 Code Design and Implementation

The code was ran using the Mingw-w64 compiler on Windows and on the Fedora virtual machine: it is therefore compatible on both operating systems. The game was implemented in the terminal. The *ncurses* library was installed on Linux [3], and an equivalent version *PDcurses* was downloaded for Windows [4]. When building the project on Windows, a warning appears, which can be disregarded. It is recommended to run the game in an external terminal, in order to ensure the board is fully visible.

Compatibility on different operating systems was prioritised over aesthetics. Clearing the terminal using the system("clr") command and moving pieces using the GetAsyncKeyState functionality on Windows, would have given a cleaner output and easier manipulation of the falling pieces. However, printing empty lines to clear the screen and using the curses library for compatibility was chosen. A function to print empty lines was written using the static keyboard, as it is declared in multiple files.

```
// Clears screen by printing empty lines - static function as implemented
// in different files
static void clear_screen()
{
    std::cout << std::string(30, '\n' );
}
```

The different classes used were split across multiple header files for ease of access. The class hierarchy is described with UMLs (Unified Modelling Language) in Figure 2. The use of both these tools allowed for better visualization of the code.

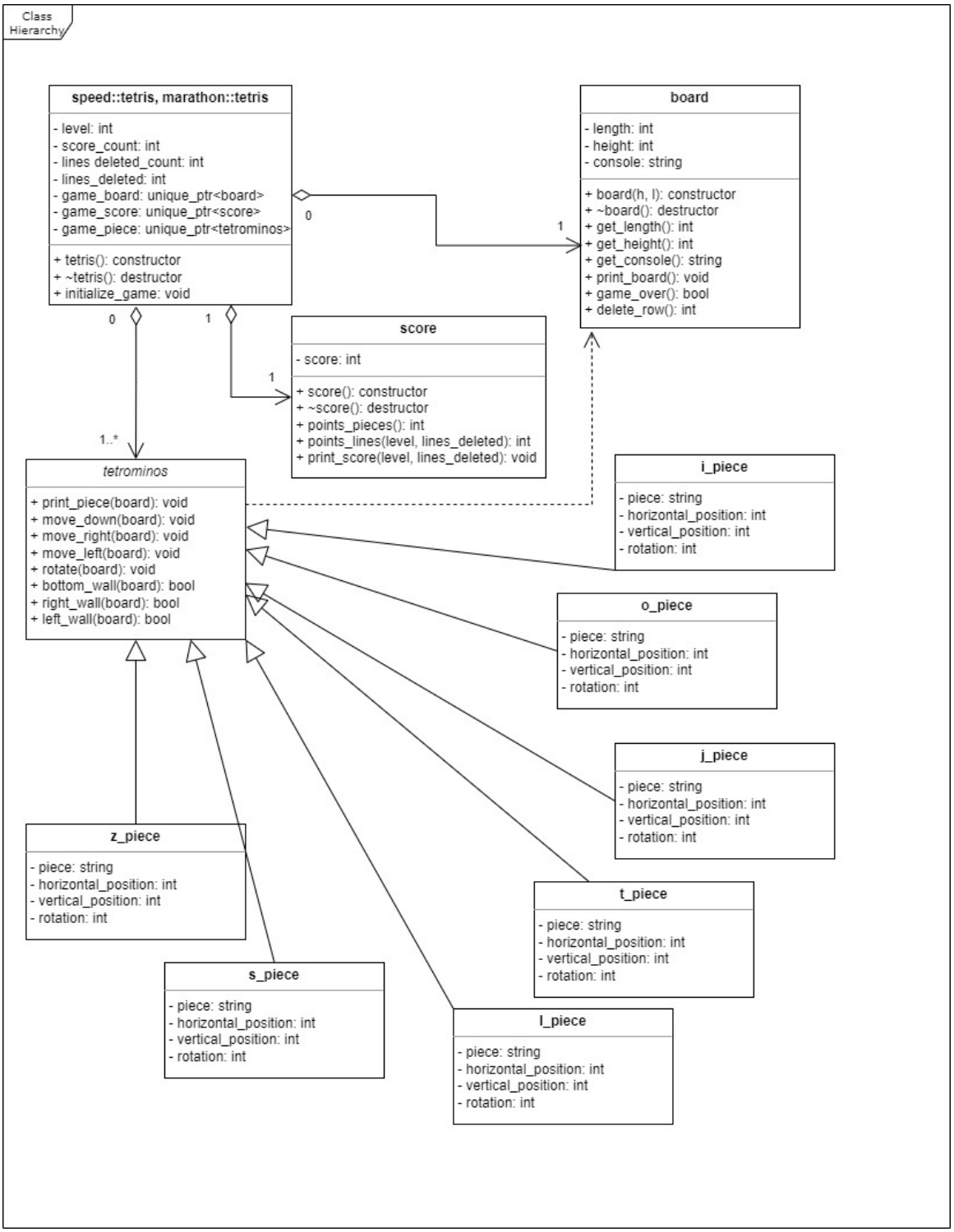

Figure 2: Class Hierarchy

2.1 Initializing Game

The game starts by asking the user whether they want to play a "Speed" or "Marathon" version of the game. The Speed version has a 1 minute timer, during which the player needs to score as many points as they can. The Marathon version runs until game over.

```
template <typename T> bool is_it_correct(T game_mode, T input_1, T input_2)
{
    if (game_mode!=input_1 & game_mode!=input_2) {
        return false;
    } else {
```

```
        return true;
    }
}
```

This part of the code showcases the use of input validation, as the player needs to press 1 or 2. If something else is inputted, they are prompted to choose again. A template is used to improve flexibility and versatility of the code: the chosen input validation could easily be changed, for example to "m" for marathon and "s" for speed.

Once the user has chosen its game mode, a tetris object is generated with the use of the smart unique pointer. The smart pointer avoids memory leaks, as the object gets automatically deleted when it goes out of scope. Smart pointers are imported in the memory header.

The Speed and Marathon games are defined using namespaces speed::tetris and marathon::tetris. The use of namespaces increases readability of the two different game versions, and avoids having two distinct classes.

```
namespace marathon {
    class tetris
    {
    private:
        int level {};
        int lines_deleted {};
        int lines_deleted_count {};
        int score_count {};
        std::unique_ptr<board> game_board;
        std::unique_ptr<tetrominos> game_piece;
        std::unique_ptr<score> game_score;

    public:
        tetris()=default;
        ~tetris();
        void initialize_game();
    };
}
```

In the speed namespace, the initialize_game differs by having a timer. This functionality was imported with the header ctime. When the function gets called, the current time is saved. There is an if statement later in the function, to break out of the game loop if the timer has exceeded 1 minute.

```
void speed::tetris::initialize_game()
{
    clear_screen();
    clock_t time;
    time = clock();
    while (!game_board->game_over()) {

        int generator = random_number(1,7);
        switch(generator)
        ...
    ...
    time = clock();
    if ((float)time/CLOCKS_PER_SEC > 60.0) {
        std::cout << "You played a 1 minute game" << std::endl;
        break;
    }
```

2.2 Board, Piece and Score

The first object that gets instantiated when tetris::initialize_game is called, is the board object. The board object has three attributes: its height, length and a string array named console, which contains all its elements and updates when new pieces enter the board. This version of the game

has 12 rows and 12 columns, but the code works with different board sizes as well. Each of these attributes has a corresponding "getter" function to access the corresponding element.

The elements of the console are retrieved using an index system console(a,b), where a is the row number ranging from 1 to length and b the column number ranging from 1 to height. This corresponds to the index function in the code, and to the overloading of the () operator.

At each turn in the tetris class, the board::game_over and board::delete_rows functions are called, to respectively check if the game is over and if any rows are full and need to be deleted.

A score object is also instantiated. The score object has a score_count attribute and keeps track of the score in the game. The functions points_pieces and points_lines are called at every turn by the tetris object.

2.3 Tetrominos

The last object that gets instantiated is a tetrominos object. Tetrominos is the abstract class from which the 7 different game pieces are derived from.

```
class tetrominos
{
public:
    virtual void print_piece(board &board)=0;
    virtual void move_down(board &board)=0;
    virtual void move_right(board &board)=0;
    virtual void move_left(board &board)=0;
    virtual void rotate(board &board)=0;
    virtual bool bottom_wall(board &board)=0;
    virtual bool right_wall(board &board)=0;
    virtual bool left_wall(board &board)=0;
};
```

It fully controls the movements of all the pieces. The pieces classes are:

- i_piece,
- j_piece,
- l_piece,
- o_piece,
- s_piece,
- t_piece,
- z_piece.

All tetrominos have the same functions, but these differ in their implementation due to the various shapes. All functions start with a switch case to check the piece's current rotation number. The O piece only has 1 rotation number; I, S and Z have 2 rotation numbers; and T, L and J have in 4 rotation numbers.

The pieces can move down, right and left using the corresponding functions. The bottom_wall, right_wall and left_wall functions check if a tetromino can move in the desired direction, depending on whether it has reached a wall or another piece. They get called in the move_down, move_right and move_left functions, before a piece gets displaced.

The rotation function is the most complex of all. There are if statements and try catch statements for each of the rotation switch cases, to check if the tetromino is allowed to rotate.

```
void t_piece::rotate(board &board)
{
    std::string* game_console = board.get_console();
    int height {board.get_height()};
    int length {board.get_length()};

    switch (rotation)
    {
```

```
        case 0:
            if (vertical_position>1) {
                try {
                    if (game_console[board.index(horizontal_position+1,
                        vertical_position-1)]==" ") {

                        for (int i{};i<3;i++) {
                            game_console[board.index(horizontal_position+i,
                                vertical_position)]=" ";
                        }
                        game_console[board.index(horizontal_position+1,
                            vertical_position+1)]=" ";

                        vertical_position-=1;
                        for (int i{};i<3;i++) {
                            game_console[board.index(horizontal_position+1,
                                vertical_position+i)]=piece[i];
                        }
                        game_console[board.index(horizontal_position,
                            vertical_position+1)]=piece[3];
                        rotation=1;
                    }
                } catch (std::out_of_range& except) {}
            }
            break;
```

The type of tetromino that gets instantiated at each turn in the game is randomly generated by a lambda function in the tetris class. This is achieved using the uniform distribution from the random library.

```
auto random_number = [] (auto initial, auto final)
{
    std::random_device dev;
    std::mt19937 rng(dev());
    std::uniform_int_distribution<std::mt19937::result_type> distribution(initial,final);
    return distribution(rng);
};
```

The number generated then goes into a switch case, and a tetrominos is instantiated. After this happens, the piece gets added onto the board object and the board is printed in the terminal. At this point, the curses library (PDcurses on Windows) is used, and the user can press certain keys on their keyboard to make the piece move: "a" to move left, "d" to move right, "w" to rotate the piece and "s" to accelerate a piece downwards.

The speed at which pieces fall is controlled by the use of the "Sleep" function from the window.h library or unistd.h for linux. Pieces move on the screen following the function $\exp(-l/5.0)$. l corresponds to the current level of the game: each game starts at level 0 and this gets incremented every 3 rows deleted. Therefore at level 0, pieces move down every 1 second.

The piece is in a loop until it reaches the bottom. Board functions then check if any lines are full and points get added. Each fallen piece is worth 25 points. A deleted line is worth $(\text{level}+1)\times k$ points, where k depends on how many lines are deleted at once: for 1 line $k=40$, for 2 lines $k=100$, 3 lines $k=300$ and 4 lines $k=1000$.

The piece object then gets deleted, and a new piece gets instantiated. This keeps happening until the game is over.

3 Results

The output in the terminal will now be shown below. In Figure 3, the game starts. We can see what the user is asked whether he wants to play the speed or marathon version of the game.

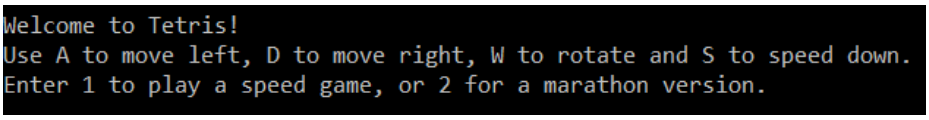

Figure 3: Start

The board then gets printed and pieces are generated. They slowly fall on the board and start accumulating at the bottom. In Figure 4, the bottom line is about to be full and deleted from the board: only the 4th column is missing a piece, and the "T" shape is about to fill the hole.

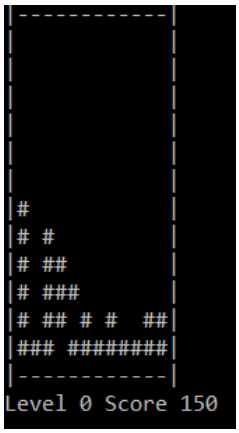

Figure 4: Line about to be full and deleted

On Figure 5, the end game for the marathon version is shown. Pieces have reached the top of the board, no more can fall and the game ends.

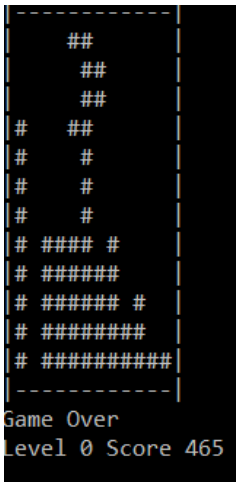

Figure 5: Game Over

For the speed version, the end game is shown in Figure 6. The player has played a 1 minute game and the game has ended.

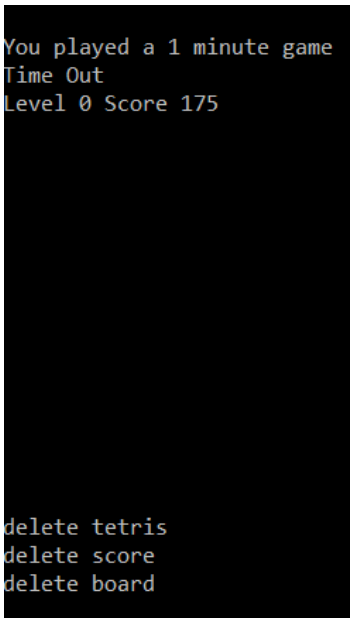

Figure 6: Time out

4 Conclusion

The game Tetris was successfully implemented. All of the game functionalities were coded in: there are 7 different tetrominos that fall, move and rotate, lines disappear and game over is reached. The two different game modes (speed and marathon), an additional functionality, was added.

Object-Oriented features were used, with the use of 3 classes, as well as an additional abstract class and 7 derived classes. The code was split across different files, and advanced features such as templates, lambda closures, try catches and namespaces were used. The game is compatible on both Windows and Linux.

The code could be improved by combining some of the tetrominos functionalities in a single class. Although some criteria would need to stay separate for all 7 pieces, some functions could have been common. Finally, creating a graphical interface (GUI) would have significantly improved the user friendliness of the game.

In terms of game features beyond the project, more pieces using different shapes could have been code. Additionally, some versions of the game have a "joker" option, where they can clear a section of the board and its pieces by placing it. This allows to keep playing the game for longer when the player is close to losing.

Word count: 2000

References

[1] S.L. Kent. *The Ultimate History of Video Games, Volume 1: From Pong to Pokemon and Beyond . . . the Story Behind the Craze That Touched Our Lives and Changed the World.* Ultimate History of Video Games. Crown, 2010. ISBN: 9780307560872.

[2] *About Tetris.* URL: `https://tetris.com/about-us`. (accessed: 21.05.2022).

[3] *ncurses.* URL: `https://jbwyatt.com/ncurses.html`. (accessed: 14.05.2022).

[4] *PDcurses.* URL: `https://pdcurses.org/`. (accessed: 14.05.2022).