**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**



**Subject Name: Fundamentals of Programming**

**Semester Project**

**Submitted To:**

**Professor Jahan Zeb/ LE Sana Fatima**

**Submitted By:**

**Usman Awan REG # 480836**

**DE-45/SYN:A DEPT: Computer Engineering**

**Submission Date: 22/05/2024**

**Title: Tetris Game**

**Hardware/Software Required:**

**Hardware: PC**

**Software: C++, mingw compiler, VS Code IDE**

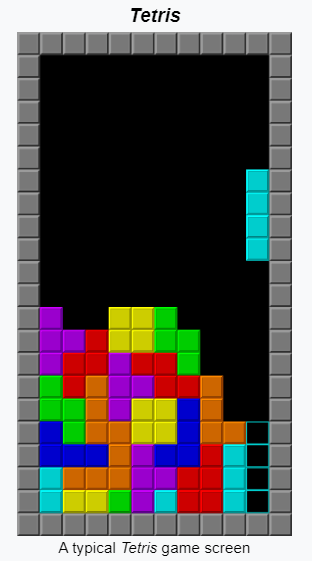
**Table of contents:**

|  |  |  |
| --- | --- | --- |
| **S. no.** | **Contents** | **Page** |
| 1. | Introduction to Tetris | 3 |
| 2. | Libraries | 4 |
| 3. | Global assets and functions | 5 |
| 4. | Assets | 7 |
| 5. | Game loop   * Loop assets * Game timing * Input handling * Block falling and Collision | 8 |
| 6. | Game Field and Graphics | 12 |
| 7. | Scoring and file Handling | 14 |
| 8. | Conclusion and Credits | 15 |

# **Introduction to Tetris:**

Tetris, the iconic puzzle game created in 1985 by Russian Software Engineer Alexey Pajitnov, tasks the player with arranging geometric shapes, called Tetriminos, into horizontal lines. As the lines clear, new Tetriminos fall faster and faster, testing the player’s ability to think strategically and react quickly. With its intuitive gameplay and timeless entertainment over the decades, Tetris is a timeless classic that continues to be enjoyed by millions around the world.

Over the years, many variants of This game have been made. In my Project, I intend to replicate the Nintendo’s version of Tetris from The Nintendo Entertainment System(NES) released in 1989.



**Libraries used:**

The C++ class libraries are modular components of reusable code. Using class libraries, you can integrate blocks of code that have been previously built and tested.

The following libraries have been used in this Project.

#include <iostream>

#include <stdio.h>

#include <Windows.h>

#include <thread>

#include <vector>

using namespace std;

**Global assets and Functions:**

**Global assets:**  The following global assets have been used in this project.

// global arrays/variables.

wstring tetrablocks[7];       // array to store tetris block shapes.

int Field\_Width = 12;       // width of the game's Field.

int Field\_Height = 18;        // height of the game's Field.

unsigned char\* pField = nullptr;  // store all elements of the game's Field as unsigned characters.

int Screen\_Width = 120;       // Console screen size X (columns).

int Screen\_Height = 30;       // Console screen size Y (rows).

**Functions:** The following functions have been used in this project.

// declare prototypes.

int Rotate\_Block(int px, int py, int r);  // to rotate tetris blocks.

int Check\_Block\_Fitting(int tetrablock\_index, int Block\_Rotation, int posX, int posY);  // check if active tetrablock fits the screen bottom.

---Snip---

// define functions.

int Rotate\_Block(int px, int py, int r) {

  switch (r % 4)

  {

  case (0):

    return py \* 4 + px;       // 0 degrees.

    break;

  case (1):

    return 12 + py - (px \* 4);    // 90 degrees.

    break;

  case (2):

    return 15 - (py \* 4) - px;    // 180 degrees.

    break;

  case (3):

    return 3 - py + (px \* 4);     // 270 degrees.

    break;

  }

  return 0;

}

int Check\_Block\_Fitting(int tetrablock\_index, int Block\_Rotation, int posX, int posY) {

  for (int px = 0; px < 4; px++)

    for (int py = 0; py < 4; py++) {

      // Get ID into tetrablock.

      int pi = Rotate\_Block(px, py, Block\_Rotation);

      // Get ID into Game Field.

      int fi = (posY + py) \* Field\_Width + (posX + px);

      // implement collision detection.

      if (posX + px >= 0 && posX + px < Field\_Width) {

        // if tetrablock index = # and field block != 0(0 = empty), then tetrablock is colliding with something

        if (tetrablocks[tetrablock\_index][pi] == L'#' && pField[fi] != 0)

          return false; // fail on first hit.

      }

    }

  return true;

}

**Assets:**

Assets provide resources to our programs. I have stored the different shapes of Tetris blocks as assets for the Project.

int main()

{

  // create assets(shapes of tetris blocks).

  tetrablocks[0].append(L"..#.");   // shape A.

  tetrablocks[0].append(L"..#.");

  tetrablocks[0].append(L"..#.");

  tetrablocks[0].append(L"..#.");

  tetrablocks[1].append(L"..#.");   // shape B.

  tetrablocks[1].append(L".##.");

  tetrablocks[1].append(L".#..");

  tetrablocks[1].append(L"....");

  tetrablocks[2].append(L".#..");   // shape C.

  tetrablocks[2].append(L".##.");

  tetrablocks[2].append(L"..#.");

  tetrablocks[2].append(L"....");

  tetrablocks[3].append(L"....");   // shape D.

  tetrablocks[3].append(L".##.");

  tetrablocks[3].append(L".##.");

  tetrablocks[3].append(L"....");

  tetrablocks[4].append(L"..#.");   // shape E.

  tetrablocks[4].append(L".##.");

  tetrablocks[4].append(L"..#.");

  tetrablocks[4].append(L"....");

  tetrablocks[5].append(L"....");   // shape F.

  tetrablocks[5].append(L".##.");

  tetrablocks[5].append(L"..#.");

  tetrablocks[5].append(L"..#.");

  tetrablocks[6].append(L"....");   // shape G.

  tetrablocks[6].append(L".##.");

  tetrablocks[6].append(L".#..");

  tetrablocks[6].append(L".#..");

**Game loop:**

The Game loop is the heart of a Game. In video games, a game loop is the core mechanism that keeps the game running continuously. It performs game timings, input handling, logic implementation, scoring etc. Tzhe game loop of this program contains:

1. **Loop assets: Resources for the loop.**

// variables/arrays for Game logic.

  bool GameOver = false;  // Game loop flag.

  int Current\_Piece = 0;

  int Current\_Rotation = 0;

  int CurrentX = Field\_Width / 2;

  int CurrentY = 0;

  bool Key[4];            // Game keys: left arraw, right arrow, down arrow, rotate.

  bool Rotate\_Hold = false; // Check if rotate key is on hold. False by default.

  int Fall\_Speed = 20;    // Fall speed of blocks. On level 1, it takes 20 game ticks for block to reach ground.

  int Fall\_Speed\_Counter = 0; // Counter to increase Fall speed as levels progress.

  bool Push\_Down = false;     // when no. of game ticks = game speed, this bool becomes true and block is pushed down.

  int Piece\_count = 0;    // no. of blocks used up.

  int score = 0;        // score.

  vector<int> Lines;      // Vector to store lines to be removed.

**(b) Game Timing: Single time unit of game is called a Tick.**

while (!GameOver) {

    // Set Game timing.

    this\_thread::sleep\_for(25ms); // single game tick = 25ms.

    Fall\_Speed\_Counter++; // increase speed as game goes on

**(c) Input handling: Implement behaviour of user-input.**

// Handle user input.

    for (int k = 0; k < 4; k++)               // R, L,  D, Z

      Key[k] = (0x8000 & GetAsyncKeyState((unsigned char)("\x27\x25\x28Z"[k]))) != 0;

    // Goes through the keys array, checks the pressed key by GetAsyncKeyState fn.

    // x27 hex for right arrow,x25 for left,x28 for down,Z for z.

---Snip---

// Implement behaviour for keys

    if (Key[1]) {   // left arrow.

      // check if tetrablock fits if pushed left.

      if (Check\_Block\_Fitting(Current\_Piece, Current\_Rotation, CurrentX - 1, CurrentY)) {

        CurrentX = CurrentX - 1;

      }

    }

    if (Key[0]) {   // right arrow.

      // check if tetrablock fits if pushed right.

      if (Check\_Block\_Fitting(Current\_Piece, Current\_Rotation, CurrentX + 1, CurrentY)) {

        CurrentX = CurrentX + 1;

      }

    }

    if (Key[2]) {   // down arrow.

      // check if tetrablock fits if pushed down.

      if (Check\_Block\_Fitting(Current\_Piece, Current\_Rotation, CurrentX, CurrentY + 1)) {

        CurrentY = CurrentY + 1;

      }

    }

    if (Key[3]) {   // z = rotate.

      // check if tetrablock fits if pushed down.

      if (!Rotate\_Hold && Check\_Block\_Fitting(Current\_Piece, Current\_Rotation + 1, CurrentX, CurrentY)) {

        Current\_Rotation = Current\_Rotation + 1;

        Rotate\_Hold = true;

      }

      else Rotate\_Hold = false;

**(d) block falling and collision detection: Blocks fall in Tetris but freeze upon collision with bottom or other blocks. And when a line forms, it is erased and blocks above the line are pushed down.**

// Implement Tetrablocks fall.

    if (Push\_Down) {

      // increase game speed after every 40 blocks.

      Fall\_Speed\_Counter = 0;

      Piece\_count++;

      if (Piece\_count % 40 == 0)

        if (Fall\_Speed >= 10)

          Fall\_Speed--;

      // Check if piece can be pushed down.

      if (Check\_Block\_Fitting(Current\_Piece, Current\_Rotation, CurrentX, CurrentY + 1)) {

        CurrentY++;   // If fits, push down.

      }

      else {              // If doesn't fit, then:

        // Lock current piece in the field(deactivate it).

        for (int px = 0; px < 4; px++) {

          for (int py = 0; py < 4; py++) {

            // if current position of tetrablock is colliding with bottom, increment its Field value to 1.

            // 0 = empty space, 1 = inactive block in our Field.

            if (tetrablocks[Current\_Piece][Rotate\_Block(px, py, Current\_Rotation)] == L'#') {

              pField[(CurrentY + py) \* Field\_Width + (CurrentX + px)] = Current\_Piece + 1;

            }

          }

        }

        // Check if a horizontal line is made.

        for (int py = 0; py < 4; py++) {            // Checking near only where last block fell as line could have only appeared there.

          if (CurrentY + py < Field\_Height - 1) {

            bool Line = true;                // By default, horizontal line is assumed to be made.

            for (int px = 1; px < Field\_Width - 1; px++) { // Restricting check within boundary.

              // bitwise\_and field blocks and line underchecking, if any block is empty(resulted in 0), Line becomes false.

              Line &= (pField[(CurrentY + py) \* Field\_Width + px]) != 0;

            }

            // If Line true then:

            if (Line) {

              // Set horizantal line to "=".

              for (int px = 1; px < Field\_Width - 1; px++) {

                pField[(CurrentY + py) \* Field\_Width + px] = 8;

                // Remove horizontal line by destroying it and pushing the row above line downwards.

                Lines.push\_back(CurrentY + py);

              }

            }

          }

        }

**Game Field and Graphics:**

A game contains of a plane consisting of blocks which is managed by game logic. This plane is called “The Game Field”. Graphics pertains to rendering the game field by means of a Graphics Engine. I have used the <Windows.h> header for rendering.

// initialize the game's Field.

  pField = new unsigned char[Field\_Width \* Field\_Height]; // buffer for the game's Field.

  for (int x = 0; x < Field\_Width; x++) {         // Field boundary.

    for (int y = 0; y < Field\_Height; y++) {

      pField[y \* Field\_Width + x] = (x == 0 || x == Field\_Width - 1 || y == Field\_Height - 1) ? 9 : 0;

      // Field blocks are all empty(position = 0) unless block is at the border(position = 9).

    }

  }

  // initialize game's console screen.

  // creates an array of blank wchars for the screen and give a handle to the screen's buffer and activates it.

  // Windows handles are the pathway between user space and the OS. Various attributes have been assigned to our handle. eg, read/write

  wchar\_t \*screen = new wchar\_t[Screen\_Width \* Screen\_Height];

  for (int i = 0; i < Screen\_Width \* Screen\_Height; i++) screen[i] = L' ';

  HANDLE hConsole = CreateConsoleScreenBuffer(GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CONSOLE\_TEXTMODE\_BUFFER, NULL);

  SetConsoleActiveScreenBuffer(hConsole);

  DWORD dwBytesWritten = 0;

---Snip---

// Render output.

    // display or draw the field of the game.

    for (int x = 0; x < Field\_Width; x++)

      for (int y = 0; y < Field\_Height; y++)

        screen[(y + 2) \* Screen\_Width + (x + 2)] = L" ABCDEFG=X"[pField[y \* Field\_Width + x]];

    // display or draw current or active piece.

    for (int px = 0; px < 4; px++)

      for (int py = 0; py < 4; py++)

        if (tetrablocks[Current\_Piece][Rotate\_Block(px, py, Current\_Rotation)] == L'#')

          screen[(CurrentY + py + 2) \* Screen\_Width + (CurrentX + px + 2)] = Current\_Piece + 65;

    // draw score.

    swprintf\_s(&screen[2 \* Screen\_Width + Field\_Width + 6], 16, L"Score: %8d", score);

    // Draw the frame of the game and refresh it continuously.

    if (!Lines.empty()) {

      // Draw frame.

      WriteConsoleOutputCharacter(hConsole, screen, Screen\_Width \* Screen\_Height, { 0, 0 }, &dwBytesWritten);

      this\_thread::sleep\_for(400ms);  // delay a bit before screen refresh.

      // Remove line and push the row above downwards.

      for (auto &v : Lines)

        for (int px = 1; px < Field\_Width - 1; px++) {

          for (int py = v; py > 0; py--)

            pField[py \* Field\_Width + px] = pField[(py - 1) \* Field\_Width + px];

          pField[px] = 0;

        }

      Lines.clear();

    }

    // Display Frame.

    WriteConsoleOutputCharacter(hConsole, screen, Screen\_Width \* Screen\_Height, { 0, 0 }, &dwBytesWritten);

  }

**Scoring:**

Scoring is rewarding mechanism of a game. Players are rewarded based on the amount of scores they rack up.

  int score = 0;        // score.

---Snip---

// update score per block placed.

        score += 25;

        if (!Lines.empty())

          score += (1 << Lines.size() - 1) \* 10;

---Snip---

// draw score.

    swprintf\_s(&screen[2 \* Screen\_Width + Field\_Width + 6], 16, L"Score: %8d", score);

---Snip---

  // Close program and display score and save it.

  CloseHandle(hConsole);

  cout << "Game Over! Score: " << score << endl; cout << "\nPrevious scores:\n";

  // Open file in append mode (ios::app) to add the new score

  ofstream outfile("example.txt", ios::app);

  if (!outfile) {

    cerr << "Error: Unable to open the file." << endl;

    return 1;

  }

  outfile << score << endl;

  outfile.close();

  // Reopen the file in read mode to display the scores

  ifstream infile("example.txt");

  if (!infile) {

    cerr << "Error: Unable to open the file." << endl;

    return 1;

  }

  string line;

  while (getline(infile, line)) {

    cout << line << endl;

  }

  infile.close();

  system("pause");

  return 0;

}

**Conclusion:**

In making this project, all the programming fundamentals learnt throughout the semester were revitalized in our minds. We also learned many new concepts like Game basics, vectors, threads, standard libraries, etc.

Overall, we enjoyed making this group project together.

Special Thanks,

Professor. Jahan Zeb,

LA Sana Fatima

References:

<https://en.wikipedia.org/wiki/Tetris>

<https://docs.oracle.com/cd/E19957-01/806-3569/Libintro.html>

<https://www.youtube.com/watch?v=8OK8_tHeCIA>

<https://github.com/OneLoneCoder/Javidx9/tree/c9ca5d2e5821f2d2e07f07f388803c185a68d13a/SimplyCode>