



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

IMY 771

WASM Tutorial

Name:

Graeme Coetzee

Student Number:

17005168

October 23, 2020

Contents

1	Introduction	2
1.1	Purpose	2
1.2	What is Web Assembly?	2
1.3	Code	2
2	Environment Setup	2
2.1	Technologies Used	2
2.2	Python	3
2.3	C++	4
2.4	Emscripten	5
2.5	VS Code	5
2.6	XAMPP	6
3	Tetris and WASM	7
3.1	How To Tetris	7
3.2	How To WASM	13
3.3	Tying It All Together	14
4	Conclusion	19

1 Introduction

1.1 Purpose

The purpose of this assignment is to provide a tutorial on the basics of web assembly. In order to do this I will create a simple Tetris game in C++, compile it into web assembly code. This web assembly code will then be used on a server where Javascript will interact with it in order to display the game with HTML.

1.2 What is Web Assembly?

Web assembly is a new type of code that can be run in modern web browsers. It is a low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages such as C/C++, C and Rust with a compilation target so that they can run on the web. This allows us to improve performance of code running on the web.

1.3 Code

All of the code shown in this tutorial can be found in my GitHub Repo: https://github.com/GraemeCoetzee/tetris_wasm

2 Environment Setup

Environment setup is an important step in order to follow this tutorial. In order to get the same results as we move through the different steps, it is crucial to set up your environment in a way ensure your project will work.

2.1 Technologies Used

In this tutorial I have made use of a few technologies. Firstly, I am setting up an environment for Windows 10, 64 bit system. The main language I will be using is C++, which will be installed through WinGW. For the web assembly compilation and generation I will be using Emscripten compiler. Emscripten requires that Python is installed, so I will also be exemplifying how to do that. Finally, I will also be giving an introduction into VS Code, which is my IDE, and some useful extensions that I made use of to make the project easier.

2.2 Python

For this project, I installed the latest version of Python, which can be found here:
<https://www.python.org/downloads/windows/>

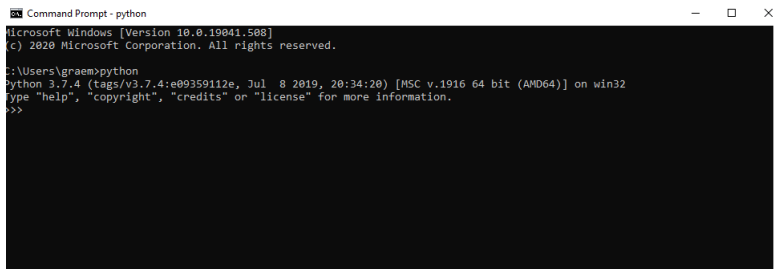
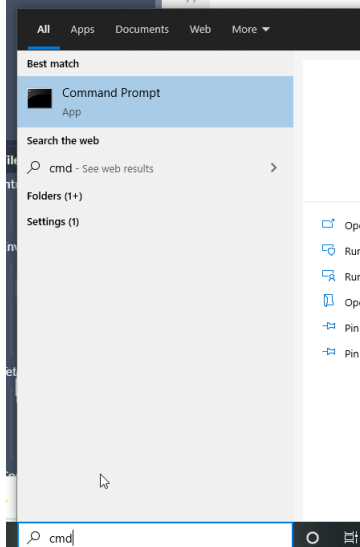


Once on the website select Python version 3.9.0, select the 64 bit Windows executable installer. Once downloaded, run the installer, remember to click the "Add Python 3.9 to PATH" button in order to add Python to your environment variables and then click "Install Now". Follow the steps until it is installed.



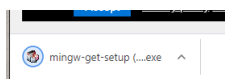
To test that it is installed use command prompt by typing "cmd" in your windows

search bar and open it. Once command prompt is open type "python" and then press the enter button. If you see the following then your Python is correctly installed.

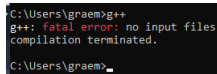


2.3 C++

As mentioned earlier I made use of WinGW, which is a tool for installing c++ on Windows. WinGW Installation will be similar to Python. Find the installer here: <https://osdn.net/projects/mingw/downloads/68260/mingw-get-setup.exe/>. The installer will automatically be downloaded to your system.



Run the installer, follow the steps until it is installed. Once installed, follow the same process to open command prompt, by searching for "cmd" and running the program. Type "g++" and hit enter. If you get the following output then WinGW has successfully installed c++:



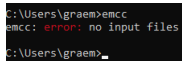
```
C:\Users\graem>g++  
g++: fatal error: no input files  
compilation terminated.  
C:\Users\graem>
```

2.4 Emscripten

To install Emscripten it is highly recommended that you follow the tutorial that I followed: https://www.tutorialspoint.com/webassembly/webassembly_installation.htm

You will have to download the repository, extract it, open the extracted folder and open command prompt from within the folder. From here you will be able to run the set up and installation commands that are specified in the tutorial linked above. Make sure that your Python was correctly installed before you move onto this step, as Emscripten requires Python to be installed prior.

Once you have installed Emscripten, open command prompt using methods specified above, type "emcc" and hit enter. If you get the following output, then you have installed Emscripten correctly:

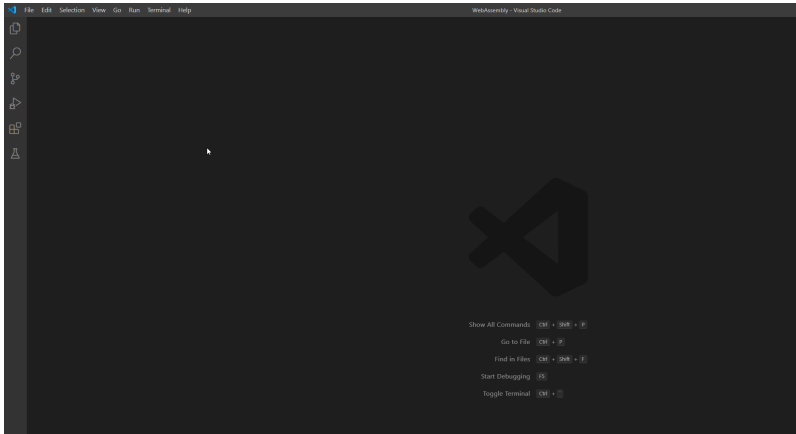


```
C:\Users\graem>emcc  
emcc: error: no input files  
C:\Users\graem>
```

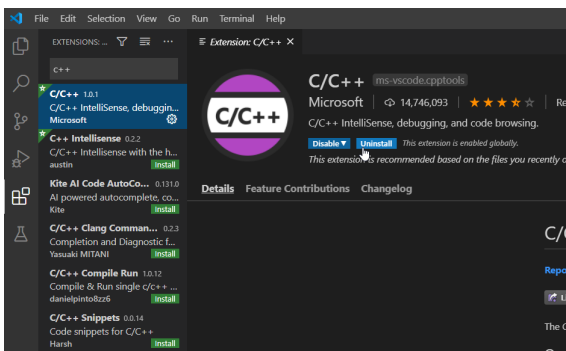
2.5 VS Code

Download and install VS Code from the following link: <https://code.visualstudio.com/>

Once installed, run the program, it should look like this:



Find the extension table along the left side, search for c++, and install the first result:



This extension will make coding in c++ easier, and more visually pleasing to look at.

2.6 XAMPP

An important tool, that will be needed for the end of this tutorial is Xampp. Xampp is a tool for hosting servers locally and will be required to run the files generated in this tutorial.

Xampp can be found here: [https://xampp-windows.en.softonic.com/download?](https://xampp-windows.en.softonic.com/download?utm_source=SEM&utm_medium=paid&utm_campaign=EN_UK_DSA&gclid=CjwKCAjw_sn8BRBrEiwAnUG)

[utm_source=SEM&utm_medium=paid&utm_campaign=EN_UK_DSA&gclid=CjwKCAjw_sn8BRBrEiwAnUG](https://xampp-windows.en.softonic.com/download?utm_source=SEM&utm_medium=paid&utm_campaign=EN_UK_DSA&gclid=CjwKCAjw_sn8BRBrEiwAnUG)

BwE Once installed, run, and press the start button on the First row, for "Apache"
Press the explore button, navigate to the htdocs folder and copy the .html, .wasm and .js files there, open a browser and type "localhost" and press enter.

3 Tetris and WASM

Once you have managed to get your environment set up, the fun stuff can start. Firstly I am going to discuss making the Tetris game, using C++, I will then compile the code using emscripten, and finally use JavaScript to use the web assembly to make my HTML game work.

3.1 How To Tetris

Firstly we create our C++ file which we will be working in. For this project I am going to call it "tetris.cpp"

In my new file, I'm going to start by importing some libraries that I will need at some stage during the project. The most important library being the Emscripten library, which we need in order to define our functions in a way that our JavaScript can interact with it later. I will import these libraries:

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <vector>
#include "emscripten.h"

using namespace std;
```

Any functions I need to use in our Javascript application need to be defined inside a "C wrapper" and be defined with the EMSCRIPTEN-KEEPALIVE attribute. This will allow us to export it later and use it in the next application:


```
int main() {  
    return 0;  
}  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
void EMSCRIPTEN_KEEPALIVE test_function() {  
    //code goes here  
}  
  
#ifdef __cplusplus  
}  
#endif
```

Any other helper or regular function that we do not want to access directly from anywhere outside, does not have to be defined like this, the regular c++ definition will do fine.

Now that we know how to make regular and Emscripten functions, let's start making our game. Firstly, we will start by initializing our separate pieces, also known as tetrominoes. Here I will define 7 different shapes in string format, where a letter will represent a valid block in the shape and, a dash will represent an empty space.

```

void EMSCRIPTEN_KEEPALIVE initializeGame() {
    resetPiece();
    gameDone = false;
    score = 0;

    piece[0] = "--R-
    ^
    --R-
    ^
    --R-
    ^
    --R-"; // Straight Line
    piece[1] = "--O-
    ^
    --OO-
    ^
    --O-
    ^
    ----"; // T Shape
    piece[2] = "--YY--YY-----"; // Square shape
    piece[3] = "--GG--G---G-----"; // LShape
    piece[4] = "--BB--B---B-----"; // Inverted L Shape
    piece[5] = "--P--PP--P-----"; // Z Shape
    piece[6] = "--M---MM---M-----"; // Inverted Z Shape

    tetrisField = new char[maxWidth * maxHeight];

    for (int x = 0; x < maxWidth; x++)
        for (int y = 0; y < maxHeight; y++)
            tetrisField[y * maxWidth + x] = (x == 0 || x == maxWidth - 1 || y == maxHeight - 1) ? 9 : 0;

    tempGameState = new char[maxWidth * maxHeight];
    for (int i = 0; i < maxWidth * maxHeight; i++) {
        tempGameState[i] = ' ';
    }
}

```

Here I show you how 2 shapes would look if they were in 2D representation, the rest are shown as normal strings and how we will be working with them for the rest of the project. After initializing our shapes, I set up the field that the shapes will be placed on, along with the borders of the playing field.

Now that we have our pieces and playing field set up, we need to be able to move a piece, and with moving a piece, we also need to be able to tell if the place it is moving to, is valid. We need to determine the validity of the move before we actually move the piece.

```

64 void performMove(int keyPressed) {
65     if(keyPressed % 4 == 0) {
66         if(testValidMove(rotationDegree, currentXPos - 1, currentYPos)) {
67             currentXPos--;
68         }
69     } else if(keyPressed % 4 == 1) {
70         if(testValidMove(rotationDegree, currentXPos + 1, currentYPos)) {
71             currentXPos++;
72         }
73     } else if(keyPressed % 4 == 2) {
74         if(testValidMove(rotationDegree, currentXPos, currentYPos + 1)) {
75             currentYPos++;
76         }
77     } else if(keyPressed % 4 == 3) {
78         if(testValidMove(rotationDegree + 1, currentXPos, currentYPos)) {
79             rotationDegree++;
80         }
81     }
82 }
83

```

Different keys represent different moves, the valid moves are left, right, down and rotate. Each move requires a test for a valid move which is done as follows:

```

43 bool testValidMove(int newRotationDegree, int newXCoord, int newYCoord)
44 {
45     for (int x = 0; x < 4; x++)
46         for (int y = 0; y < 4; y++)
47         {
48             int index = rotatePiece(newRotationDegree, x, y);
49             int tetrisFieldIndex = (newYCoord + y) * maxWidth + (newXCoord + x);
50
51             if (newXCoord + x >= 0 && newXCoord + x < maxWidth)
52             {
53                 if (newYCoord + y >= 0 && newYCoord + y < maxHeight)
54                 {
55                     if (piece[currentPiece][index] != '.' && tetrisField[tetrisFieldIndex] != 0)
56                         return false;
57                 }
58             }
59         }
60     return true;
61 }

```

By doing this, we only allow moves that are valid, which is very important for this game.

One of the important moves, is rotation, which is done as follows. We make use of different equations, that will perform different actions on the pieces, depending on the different rotation degree.

```

int rotatePiece(int newRotationDegree, int x, int y) {
    if(newRotationDegree % 4 == 0) {
        return y * 4 + x;
    } else if(newRotationDegree % 4 == 1) {
        return 12 + y - (x * 4);
    } else if(newRotationDegree % 4 == 2) {
        return 15 - (y * 4) - x;
    } else if(newRotationDegree % 4 == 3) {
        return 3 - y + (x * 4);
    } else {
        return -1;
    }
}

```

Now that we are able to move piece in different directions, rotate and have collision detection (testing if a move is valid) we can now work on the game structure. This

includes, keeping track of a score, removing full lines and generating a string version of the game state which can be used by the JavaScript to represent the game. So let's look at score and full lines:

When we move a piece, and it is a valid move, we just update it's position. That was done in the "performMove function".

```
bool EMSCRIPTEN_KEEPALIVE movePiece(bool movePieceDown, int key) {
    if(!movePieceDown) {
        performMove(key);
    } else {
        if (testValidMove(rotationDegree, currentXPos, currentYPos + 1)) {
            currentYPos++;
        } else {
            {
                setPieceInPlace();
                fullLinesUpdateScore();
                resetPiece();
                gameDone = !testValidMove(rotationDegree, currentXPos, currentYPos);
            }
        }
    }
    return gameDone;
}
```

If there are no more valid moves, it means that our piece has reached the bottom, and we therefore need to set it in place, and prepare the nex piece to start falling, as well as then update the score for successfully having a piece land.

Here we set the piece in place on the final playing field, once it has reached the end:

```
183 void setPieceInPlace() {
184     for (int x = 0; x < 4; x++) {
185         for (int y = 0; y < 4; y++) {
186             if (piece[currentPiece][rotatePiece(rotationDegree, x, y)] != '-') {
187                 tetrisField[(currentYPos + y) * maxWidth + (currentXPos + x)] = currentPiece + 1;
188             }
189         }
190     }
191 }
192
```

Once we have set the piece in place, we have to check if the current piece cause any full lines to occur, and if that is true we need to update our score accordingly and then remove the full line from the playing field. This is done as follows:

```
void fullLinesUpdateScore() {
    for (int y = 0; y < 4; y++) {
        if (currentYPos + y < maxHeight - 1)
        {
            bool lineFlag = true;
            for (int x = 1; x < maxWidth - 1; x++) {
                lineFlag &= (tetrisField[(currentYPos + y) * maxWidth + x]) != 0;
            }

            if (lineFlag)
            {
                fullLines.push_back(currentYPos + y);
            }
        }
    }

    score += 50;

    if (!fullLines.empty()) score += fullLines.size() * 100;

    for (int i = 0; i < fullLines.size(); i++) {
        int line = fullLines[i];
        for (int x = 1; x < maxWidth - 1; x++)
        {
            for (int y = line; y > 0; y--)
                tetrisField[y * maxWidth + x] = tetrisField[(y - 1) * maxWidth + x];
            tetrisField[x] = 0;
        }
    }

    fullLines.clear();
}
```

Once we have our score updated, full lines removed and piece set into place, we get the next piece ready to fall from the top:

```
void resetPiece() {
    currentXPos = maxWidth / 2;
    currentYPos = 0;
    rotationDegree = 0;
    currentPiece = rand() % 7;
}
```

One more thing that we need to do is to check if the new piece is in a valid position as soon as it is dropped, because if it is not, the game is over.

Now that the entire game is set up and should be in a playable state, we need to generate a string version of the game state, that JavaScript can use to visualize the

game:

```

129 const char* EMSCRIPTEN_KEEPALIVE getGameState() {
130     for (int x = 0; x < maxWidth; x++)
131         for (int y = 0; y < maxHeight; y++)
132             tempGameState[(y) * maxWidth + (x)] = " ROYGBPM-W"[tetrisField[y * maxWidth + x]];
133
134     for (int x = 0; x < 4; x++) {
135         for (int y = 0; y < 4; y++) {
136             if (piece[currentPiece][rotatePiece(rotationDegree, x, y)] != '-') {
137                 tempGameState[(currentYPos + y) * maxWidth + (currentXPos + x)] = "ROYGBPM"[currentPiece];
138             }
139         }
140     }
141
142     string gameState = "";
143     for(int i = 0; i < maxWidth * maxHeight; i++) {
144         gameState += tempGameState[i];
145     }
146
147     return cstr(gameState);
148 }

```

We are now done with our C++ and are ready to start compiling.

3.2 How To WASM

To compile our project onto web assembly we are using Emscripten as mentioned above. The command that we will be using looks like this:

```

emcc -o tetris.html tetris.cpp -O1 -s WASM=1 -shell-file html_template/basic_html.html
-s NO_EXIT_RUNTIME=1 -s "EXTRA_EXPORTED_RUNTIME_METHODS=['ccall']"
-s EXPORTED_FUNCTIONS=['_main', '_movePiece', '_getScore',
'_getGameState', '_initializeGame']

```

- emcc is the name of the command we are running
- tetris.html is the output html that will be generated
- html_template/basic_html.html is the template html I am providing the compiler.
- The exported runtime method allows us to use the ccall function in our JavaScript, which will allow us to interact with the wasm code.

- The exported functions are our own functions that we want to be able to use in our JavaScript

Once we run this command, three files will be generated, a tetris.wasm, tetris.js and tetris.html

3.3 Tying It All Together

Now that we are able to interact with our wasm, I will prepare some html and js inorder to represent the game in a browser.

Firstly we add some buttons to stop and start our game, and a div that will contain rows and columns (our cells for the tetris). I also made use of the bootstrap library in order to make styling easier. I also add buttons for me to perform moves such as moving left and right.

```
<div class="container justify-content-center text-center mt-3">
  <button
    id="start"
    type="button"
    class="btn btn-secondary ml-auto mr-1"
    onclick="start();"
  >
    Play
  </button>

  <button
    id="restart"
    type="button"
    class="btn btn-secondary mr-1 mx-1"
    onclick="reset();"
  >
    Start Over
  </button>

  <button
    id="stop"
    type="button"
    class="btn btn-secondary mr-auto ml-1"
    onclick="stop();"
  >
    Stop
  </button>
</div>

<div class="container mt-5" id="tetrisField"></div>

<div class="container justify-content-center text-center mt-3">
  <button
    id="left"
    type="button"
    class="btn btn-secondary ml-auto mr-1"
    onclick="moveLeft()"
  >
    <i class="fa fa-arrow-left" aria-hidden="true"></i>
  </button>

  <button
    id="right"
    type="button"
    class="btn btn-secondary ml-1 mr-1"
    onclick="moveRight()"
  >
    <i class="fa fa-arrow-right" aria-hidden="true"></i>
  </button>
```

Now that we have html structure to work with, let's start populating the playing field.

Here we will make use of the ccall function we exported earlier, to make calls to the wasm. Here I will make a call to the game state function which will return the current game state of the playing field, and I will then populate the playing field in the html, also adding different colours to different blocks, using css classes.

```
function updateGameState() {
  let n = Module.ccall("getGameState", "string", [], []);

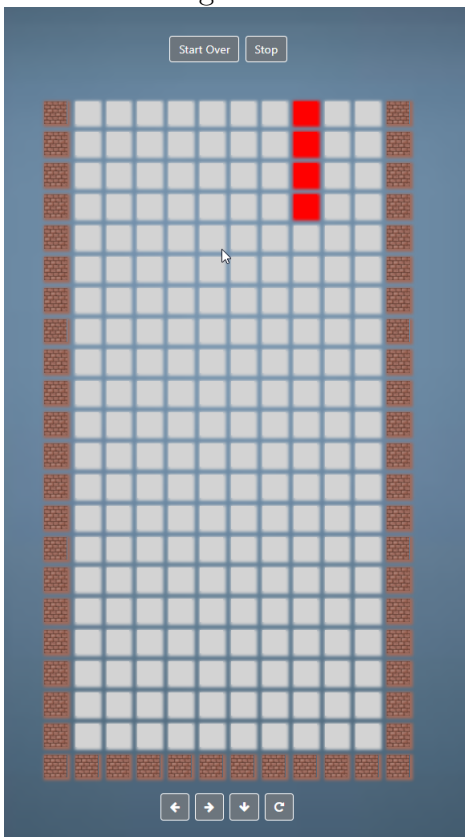
  let tetrisField = $("#tetrisField");
  tetrisField.html("").addClass("mx-auto");

  let index = 0;
  for (let i = 0; i < 22; i++) {
    let row = $("
```



```
.colour-green {  
  background-color: green;  
  box-shadow: 0.1rem 0.1rem 0.25rem 0.15rem green;  
}  
  
.colour-yellow {  
  background-color: yellow;  
  box-shadow: 0.1rem 0.1rem 0.25rem 0.15rem yellow;  
}  
  
.colour-orange {  
  background-color: orange;  
  box-shadow: 0.1rem 0.1rem 0.25rem 0.15rem orange;  
}  
  
.colour-purple {  
  background-color: purple;  
  box-shadow: 0.1rem 0.1rem 0.25rem 0.15rem purple;  
}
```

After receiving the game state, and populating all of the cells and some styling, our HTML Tetris game looks something like this:



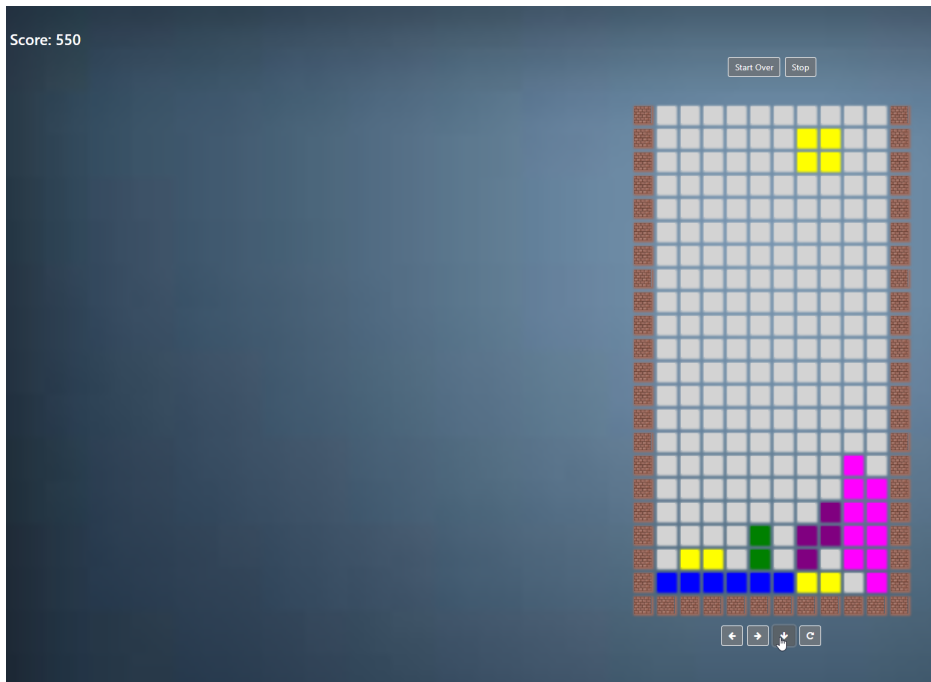
Now we need to add some game mechanics, such as the game timer, that automatically moves the piece down, after a certain amount of time. This is simple and will look something like this:

```
251 let timer;
252
253 function startTimer() {
254   timer = setInterval(function () {
255     let n = Module.ccall(
256       "movePiece",
257       "boolean",
258       ["boolean", "number"],
259       [true, 2]
260     );
261
262     if (n) {
263       endGame();
264       return;
265     }
266
267     updateGameState();
268     updateScore();
269   }, 1000);
270 }
```

Now our piece automatically moves down one block every second. Now we need to add functionality to the move buttons so that we can move the piece ourselves. That will look like this:

```
181
182 function moveLeft() {
183   let n = Module.ccall(
184     "movePiece",
185     "boolean",
186     ["boolean", "number"],
187     [false, 0]
188   );
189
190   if (n) {
191     endGame();
192     return;
193   }
194
195   updateGameState();
196 }
197
198 function moveRight() {
199   let n = Module.ccall(
200     "movePiece",
201     "boolean",
202     ["boolean", "number"],
203     [false, 1]
204   );
205
206   if (n) {
207     endGame();
208     return;
209   }
210
211   updateGameState();
212 }
213
214 function moveDown() {
215   let n = Module.ccall(
216     "movePiece",
217     "boolean",
218     ["boolean", "number"],
219     [false, 2]
220   );
221
222   if (n) {
223     endGame();
224     return;
225   }
226
227   updateGameState();
228 }
229
```

It seems our game is working, I am able to move and rotate pieces, the lines disappear, and my score is updating



4 Conclusion

This was a fun project and really nice to figure out. Even though this could all have been done in JavaScript alone, the speed and efficiency given by C++ makes it worth it, even more so for far more complex projects.