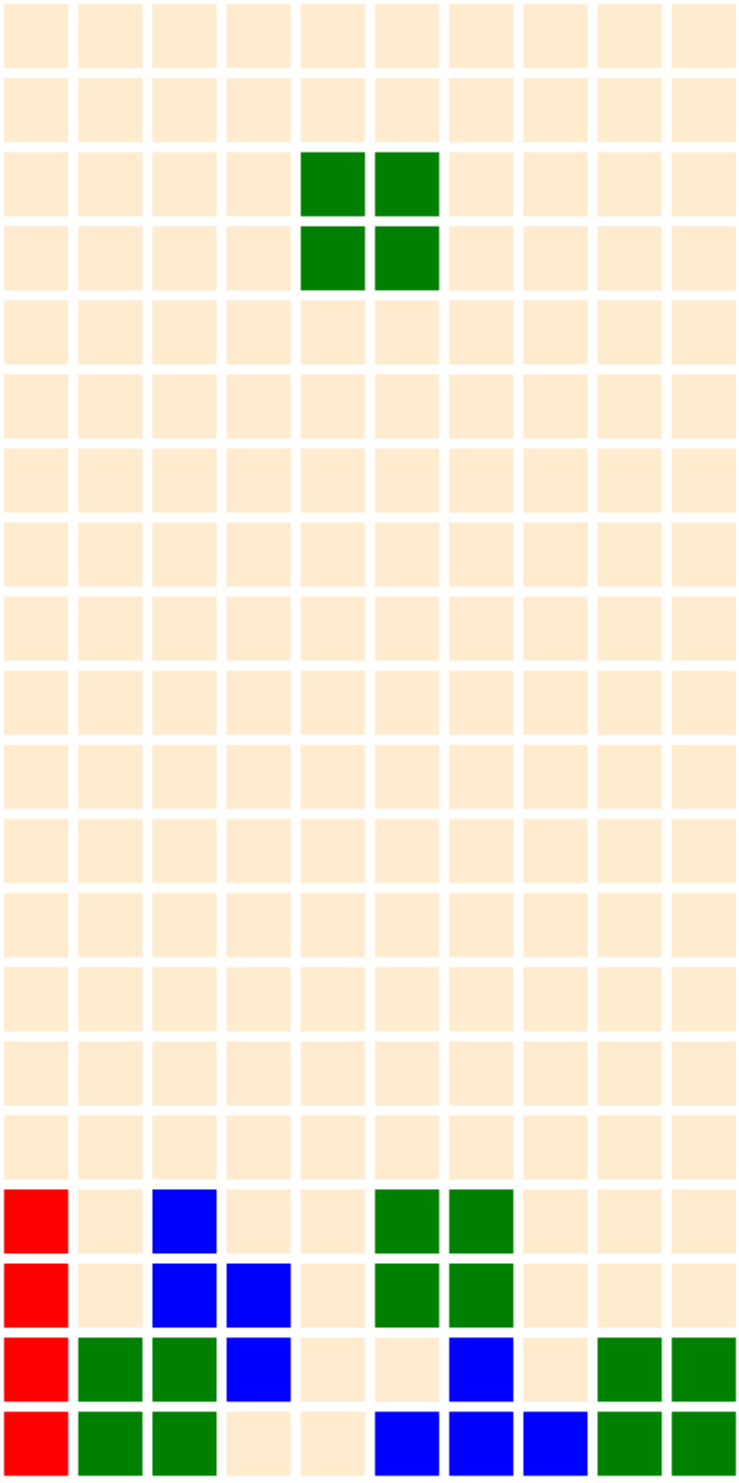
Tetris using WebAssembly



Start Reset Left Right Rotate

Project Description

In this project, we will implement the popular Tetris game as a practical example of how to use WebAssembly language. Our game logic will be written in C++, and then compiled to a WebAssembly module, which will then be utilized by our client-side Javascript and HTML files in order to host our Tetris game in a web browser. Our HTML files will then be hosted by a simple http server. For this project, all setup steps assume you are developing within a Windows 10 environment.

Environment Setup

Python Prerequisite

On Windows 10, Python version 2.1.12 or newer is needed in order to run the Emscripten SDK from GitHub.

Emscripten Setup

As we will be writing our game logic in C++, we will need an SDK that is able to compile C++ code to WebAssembly, or corresponding .wasm files. There are various compilers that are capable of doing this. In this project, we will be using the popular Emscripten SDK.

1. Clone the Emscripten GitHub repository (https://github.com/emscripten-core/emsdk.git) into a folder on your local machine.
   This can be done with the GitHub Desktop application, or by the following command in terminal environments supporting Git commands:
   `git clone https://github.com/emscripten-core/emsdk.git`
2. Open a Powershell terminal window as an Administrator.
3. Enable running unsigned scripts, as this is needed to run the Emscripten setup scripts.
   This can be done with the following command:
   `set-executionpolicy remotesigned`
4. In the Powershell terminal, navigate into the folder containing the Emscripten GitHub repository on your local machine, and pull the latest version of the Emscripten SDK.
   This can be done with the GitHub Desktop application, or by the following command in terminal environments supporting Git commands: `git pull`
5. Download and install the latest versions of the Emscripten SDK tools - this might take a while.
   This can be done with the following command:
   `emsdk install latest`
6. Activate the latest versions of the Emscripten SDK tools that you downloaded and installed in the previous step.
   This can be done with the following command:
   `emsdk activate latest`
7. Set PATH and environment variables for Emscripten ease of use. This makes Emscripten more accessible from folders outside of the SDK repo itself.
   This can be done with the following command:
   `emsdk_env.bat`

Creating The Game

The written code will be divided between 4 files - **index.html**, **style.css**, **script.js** and **tetris.cpp**.

index.html

The HTML file contains the grid that we will use in order to display our Tetris GUI, as well as the buttons that we will use to take input for moving left and right, and rotating the active block.

```
<div class="container">
    <div id="cell-1" class="cell"></div>
    <div id="cell-2" class="cell"></div>
    <div id="cell-3" class="cell"></div>
    <div id="cell-4" class="cell"></div>
    <div id="cell-5" class="cell"></div>
    <div id="cell-6" class="cell"></div>
    <div id="cell-7" class="cell"></div>
    <div id="cell-8" class="cell"></div>
    <div id="cell-9" class="cell"></div>
    <div id="cell-10" class="cell"></div>
    <div id="cell-11" class="cell"></div>
    <div id="cell-12" class="cell"></div>
    <div id="cell-13" class="cell"></div>
    <div id="cell-14" class="cell"></div>
    <div id="cell-15" class="cell"></div>
    <div id="cell-16" class="cell"></div>
    <div id="cell-17" class="cell"></div>
    <div id="cell-18" class="cell"></div>
    <div id="cell-19" class="cell"></div>
    <div id="cell-20" class="cell"></div>
    <div id="cell-21" class="cell"></div>
    <div id="cell-22" class="cell"></div>
    <div id="cell-23" class="cell"></div>
    <div id="cell-24" class="cell"></div>
    <div id="cell-25" class="cell"></div>
    <div id="cell-26" class="cell"></div>
```

```
<div class="button-container">
    <button class="button button-start" onclick="tetrisStart()">Start</button>
    <button class="button button-reset" onclick="resetTetrisGrid()">Reset</button>
    <button class="button button-left" onclick="moveLeft()">Left</button>
    <button class="button button-right" onclick="moveRight()">Right</button>
    <button class="button button-rotate" onclick="moveRotate()">Rotate</button>
    <div id="lostLabel">You lost - game reset.</div>
</div>
```

We also include our CSS styling, as well as both our own JavaScript code and the JavaScript glue code produced by Emscripten in our HTML file.

```
</body>
<script src="script.js"></script>
<script src="tetris.js"></script>
```

style.css

The CSS file contains some basic styling code, such as widths, heights, padding and background colours.
More importantly, it contains the column and row setup for our Tetris grid.
We also create classes for three different colours, to differentiate between different types of Tetris blocks.

```css
.container {
    width: 20rem;
    margin: 2rem;
    display: grid;
    grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr;
    grid-template-rows: 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr 1fr;
    grid-gap: 5px;
}
.cell {
    width: 2rem;
    height: 2rem;
    background-color: blanchedalmond;
    font-size: 1rem;
    text-align: center;
}
```

```css
.red {
    background-color: red;
}

.green {
    background-color: green;
}

.blue {
    background-color: blue;
}
```

script.js

This JavaScript file initializes a set of variables and manipulates our HTML Tetris grid. It also acts as a middleman between functions called by our HTML input buttons and our C++ code (compiled to WASM) that handles our game logic.

Some basic grid manipulation includes clearing our Tetris grid and converting the integer values for each cell received from WASM into a corresponding colour and class present in our CSS file. The JavaScript file also contains our timing interval, which determines the pace at which the game progresses.

```
function resetTetrisGrid() {
    if (alreadyStarted == true) {
        Grid = [];
        for (let count = 0; count < 200; count++) {
            Grid.push('');
        }
        updateTetrisGrid(Grid);
        let lostLabel = document.getElementById("lostLabel");
        lostLabel.style.display = "block";
        clearInterval(gameLoop);
        alreadyStarted = false;
    }
}

function updateTetrisGrid(grid) {
    for (let count = 0; count < grid.length; count++) {
        if (grid[count] == 'r') {
            document.getElementById('cell-' + (count + 1)).className = "cell red";
        } else if (grid[count] == 'g') {
            document.getElementById('cell-' + (count + 1)).className = "cell green";
        } else if (grid[count] == 'b') {
            document.getElementById('cell-' + (count + 1)).className = "cell blue";
        } else if (grid[count] == '') {
            document.getElementById('cell-' + (count + 1)).className = "cell";
        }
    }
};
```

Our C++ code (compiled to WASM) returns vectors when multiple values need to be retrieved. The JavaScript file therefore also has some utility functionality for converting these vectors to arrays more easily interpreted as our Tetris grid.

```
function convertVectorToGridArray(vector) {
    let gridToReturn = [];
    let valueToAdd;
    let intToInterpret;
    for (let count = 0; count < vector.size(); count++) {
        intToInterpret = vector.get(count);
        if (intToInterpret == 1) {
            valueToAdd = 'r';
        } else if (intToInterpret == 2) {
            valueToAdd = 'g';
        } else if (intToInterpret == 3) {
            valueToAdd = 'b';
        } else {
            valueToAdd = '';
        }
        gridToReturn.push(valueToAdd);
    }
    return gridToReturn;
}
```

tetris.cpp

Our CPP file contains our Tetris game logic, handling the more computationally expensive parts of our game. This functionality is then exposed to our JavaScript code, in order to represent this within our Tetris grid. In order to make our data type conversions from C++ to WASM possible, so our JavaScript code can interpret it, we include some Emscripten header files.

```
#include <emscripten/bind.h>
#include <emscripten/val.h>

using namespace emscripten;
```

In order to make index calculation and rotations less complex, our Tetris blocks are stored as single-dimensional arrays each representing 4x4 cell squares, with integers indicating the colour represented by each cell. We also have some utility functions for cleaner index to coordinate conversions, and vice versa. These are used extensively when doing block rotations.

The CPP file contains a **tetrisStart** function, that initializes our persistent variables and generates our first random Tetris block onto a C++ array representing the entire game grid. Here, we also have a utility function in order to convert these arrays to vectors for communication through our WASM file.

```
std::vector<int> tetrisStart() {
    srand (time(NULL));

    board = new int[200];
    for (int count = 0; count < 200; count++) {
        board[count] = 0;
    }
    gameOver = false;

    addRandomBlock();

    return convertArrayToVector(board);
}
```

Our CPP file also contains the functionality for moving these generated blocks left and right, and rotating these blocks, as well as performing checks ahead-of-time to determine whether these movements would be legal or not. We also have functions that return the gameOver state of the game, which the Javascript calls within each time interval it completes, and a utility function for removing completed rows and moving down all above blocks. These functions are quite lengthy and can all be found within the referenced Git Repository. Below is a sample of the moveDown function, demonstrating the basic flow of our movement functions.

```
std::vector<int> moveDown() {

    if (canBlockMoveDown() == true) {
        int currentIndex = activeIndex + 33;
        int activeIndexCount = 3;

        for (int count = 15; count >= 0; count--) {
            if ((board[currentIndex] > 0) && (partOfActiveBlock(currentIndex) == true)) {
                board[currentIndex + 10] = board[currentIndex];
                board[currentIndex] = 0;
                activeBoardIndices[activeIndexCount] = activeBoardIndices[activeIndexCount] + 10;
                activeIndexCount--;
            }
            if ((count % 4) == 0) {
                currentIndex -= 7;
            } else {
                currentIndex--;
            }
        }

        activeIndex += 10;
    } else {
        removeCompletedRows();
        addRandomBlock();
    }

    for (int firstRowCount = 0; firstRowCount < 10; firstRowCount++) {
        if ((board[firstRowCount] > 0) && (partOfActiveBlock(firstRowCount) == false)) {
            gameOver = true;
        }
    }

    return convertArrayToVector(board);
}
```

Finally, in order to make these functions available for use by our JavaScript code, we need to bind them to JavaScript functions through Emscripten. As we return integer vectors in many of our functions, it is also necessary to explicitly register an integer vector conversion.

```
EMSCRIPTEN_BINDINGS(Module) {
    function("tetrisStart", &tetrisStart);
    function("moveDown", &moveDown);
    function("moveLeft", &moveLeft);
    function("moveRight", &moveRight);
    function("moveRotate", &moveRotate);
    function("gameLost", &gameLost);

    register_vector<int>("vector<int>");
}
```

Building The Project

The building process involves compiling our tetris.cpp file into a corresponding tetris.wasm module, as well as a tetris.js file containing our glue code making the WASM module usage possible within JavaScript. We will use the Emscripten SDK (setup earlier in this tutorial) to do these conversions.

The following commands can be used to compile our tetris.cpp file (note that this command needs to be run from the context of the emsdk repo downloaded earlier, and the paths need to be relative from this folder).

`cd emsdk/upstream/emscripten``

`./emcc --bind -o /path/from/this/folder/to/tetris/repo/tetris.js /path/from/this/folder/to/tetris/repo/tetris.cpp`

Note also that we specify a **tetris.js** file in this command. This tells EMCC to produce the glue code we require within the folder we specify. When creating your own projects, any errors at this step will probably be a result of compilation errors within your CPP file.

Running The Project

After **tetris.js** and **tetris.wasm** have been produced by EMCC, running our project is as simple as serving the **index.html** file on our local machine. This can be done in many ways, such as downloading a Live Server plugin within your IDE, however, below is a set of commands that uses **NPM** to download a Node.js server library, which can then easily be run.

Navigate to the project's git repository and execute the following commands:
`npm install -g http-server`
`http-server`

The URLs where your index.html will be served are then displayed in your terminal.

Source Code

All source code can be found at https://github.com/KevinCoetzee10/WebAssemblyTetris