Mega's Nuclear Waste Dump

- Readme File-

Thank you for downloading Mega’s Nuclear Waste Dump, a single-player puzzler written in C using the Raylib library. I hope you enjoy this game and continue to enjoy any future games and projects I indulge in.

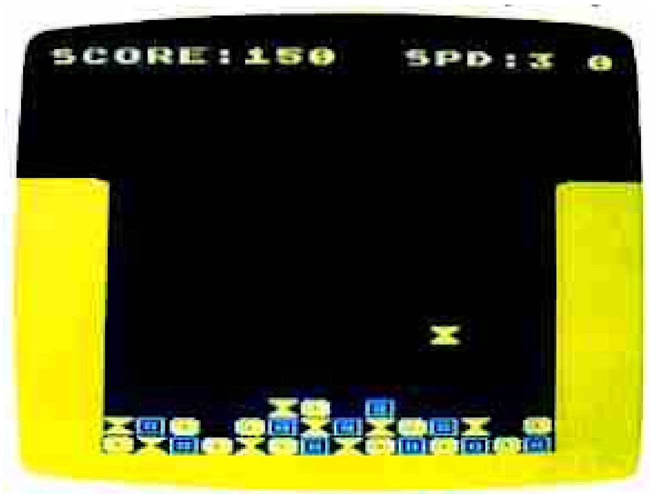
Installation

1. Download the game’s ZIP file from your source of choice. Preferably itch.io.
2. Extract the ZIP file to your desired location.
3. Double-click Mega's Nuclear Waste Dump.exe to start playing.

What Is This Game?

Mega’s Nuclear Waste Dump is a puzzler game based on an 8-bit Atari game, called Uncle Henry’s Nuclear Waste Dump, written by James Hague and published by Antic Publishing in December of 1986. It was a type-in listing for Antic Magazine Vol. 5 No. 8, which, if you don’t know, means that the source code was printed in a home computer magazine or book. Readers were meant to manually enter the code via keyboard and then save it to a cassette tape or floppy disk. Most type-in programs were written in either BASIC or machine code. UHNWD was one of the ones written in BASIC.

According to the magazine page, the game follows an "old cuss" named Uncle Henry who's trying to make a quick buck. He recently discovered that thousands of corporations have tons of nuclear waste…for some reason, and they need somewhere to dispose of it. So, after six months, Uncle Henry opened a nuclear waste dump behind the local Sloppy Joe Hut. However, he needed some employees with all the waste being sent his way. That's where you—*yeah, you!*—come in.



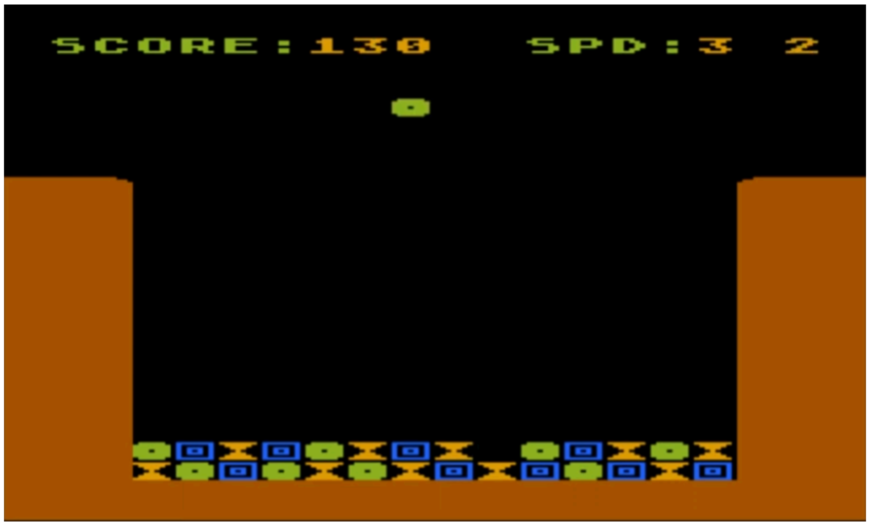
That falling can of nuclear waste may land safely this time, but don't count on safe landings for long. Better get used to bombastic explosions.

The waste containers are represented as blocks of different shapes and colors falling from the top of the screen. They hover at the top for a few seconds while the player selects which column to drop them into. When time runs out, the container of waste falls.

Despite being written in BASIC, the game itself is actually pretty elaborate. While many puzzle games revolve around matching colors or clearing rows while avoiding a full board, this game flips the formula. Instead, the goal is to *avoid* matching colors. If two containers of the same color touch, they explode, and you lose, and then you die. The objective is to strategically fill the play area to the top without triggering a meltdown.

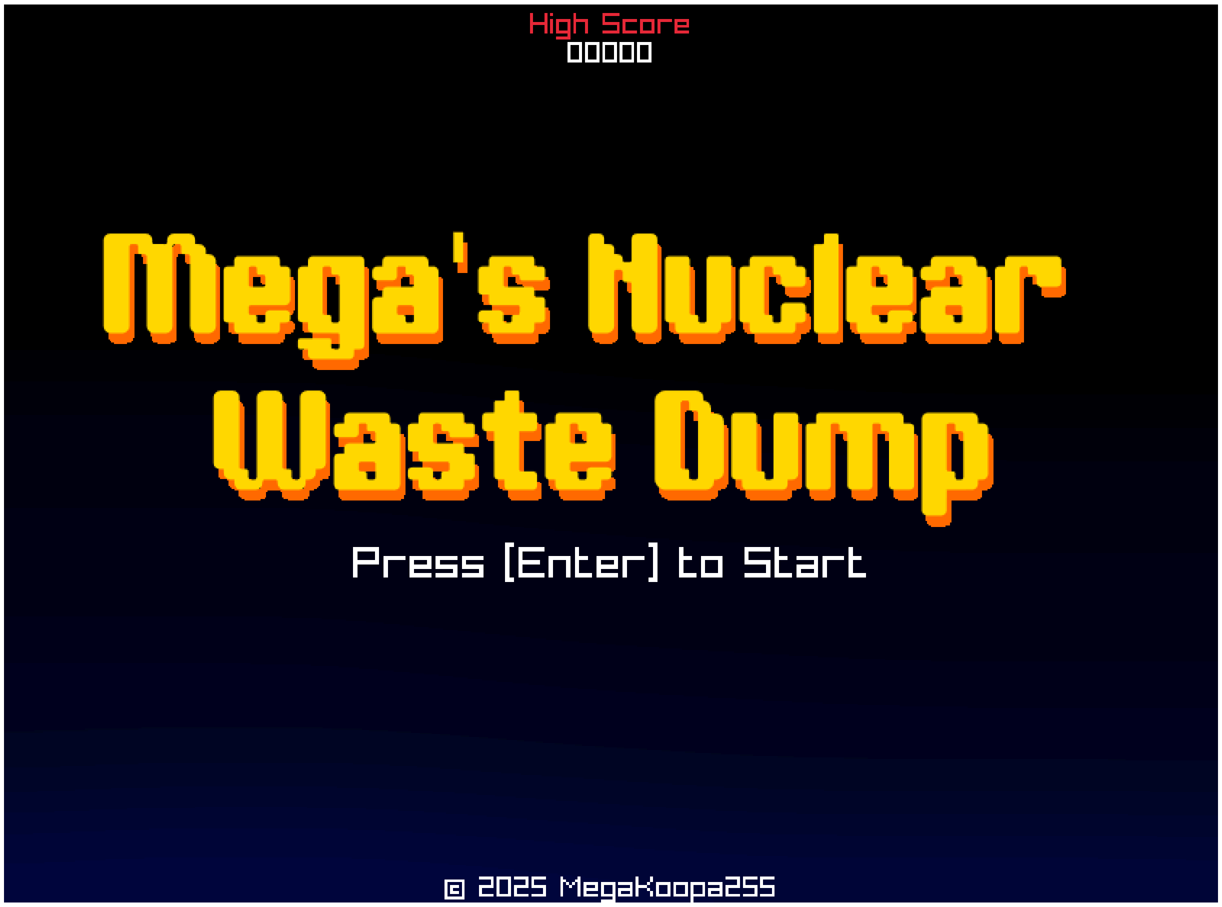
But wait, there's more! When you stack a block on top of another, it doesn't just stay there. If there's an open space at the bottom left, the block will roll down into that spot. The same applies to the bottom right. This adds an extra layer of strategy when deciding where to place each block, and it's a really unique take on the formula.

Hague would go on to work on other games and software and is still active in the gaming industry today. He also later wrote a book called Halcyon Days, in which he interviewed classic game programmers and discussed several other Atari 8-bit games.



In 2015, Hague briefly mentioned Uncle Henry's Nuclear Waste Dump, stating that he made the game in just five days one summer without ever knowing what Tetris—which had come out two years prior—even was. He also claimed, "That's probably the only thing interesting about this game." I find that a bit sad to hear because I think the game is pretty impressive for what it is. Even after all these years, I haven't seen many other games adopt a similar premise in terms of gameplay. With some touch-ups, dusting, and some WD-40, I believe this game could genuinely be a fun addition to a party or friendly gathering.

Since the original game is very difficult to play nowadays—being a type-in listing from an old magazine for a system that very few people still own—I was inspired to create a semi-remake. My goal was to introduce more people to its unique mechanics, allow them to engage with the original game's ideas in a modern platform, and potentially learn about the original Atari game.



High Score

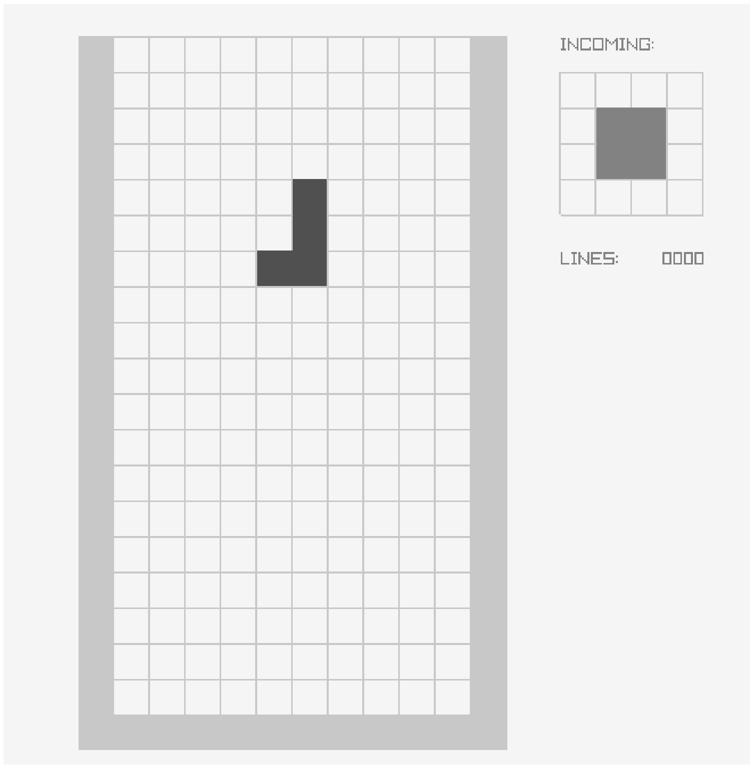
□□□□□

Mega's Nuclear Waste Dump

Press [Enter] to Start

© 2025 MegaKoopa255

Development



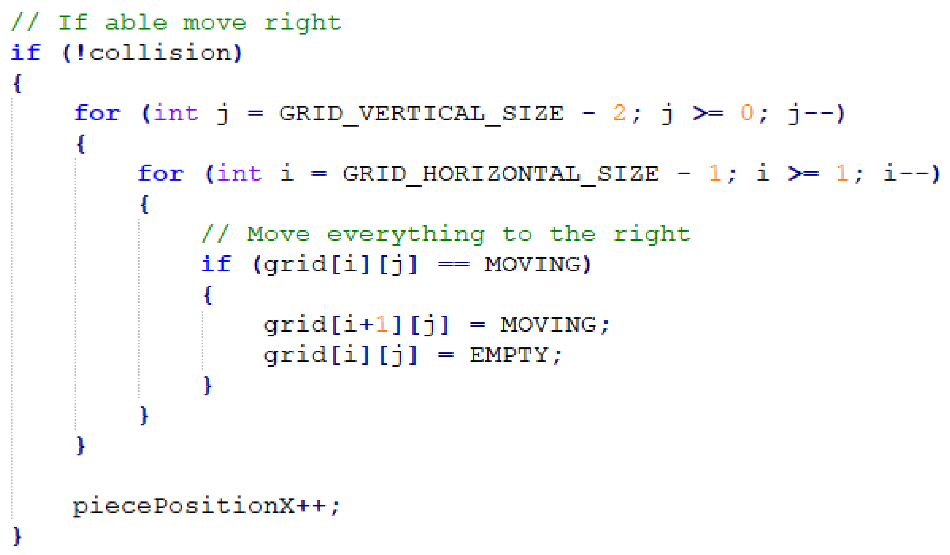
On the Raylib website, there are several templates for different classic games that anyone can use and modify to create their own, including Breakout, a general space shooter, Asteroids, and Tetris. My game is a modified version of the Tetris template, but beyond just aesthetics, I had to make several changes to the game's logic—many of which aren't as obvious as one might expect.

For one, I had to change the Tetrominoes into 1x1-sized blocks. On top of that, I had to assign colors to them randomly as they spawned. That wasn't too difficult to do since I had a decent amount of resources and online assistance. One issue I had, though, was that when the blocks stopped moving, they turned into the default gray blocks again, and having them remain the same color wasn't as easy as adding an extra line of code to make non-moving blocks colored. Trying that would just cause the non-moving blocks to change into the color of the currently falling one.

To amend this, I had to store the color data in a matrix called grid[][]. So, when a block fell, the specific place it fell in corresponded to a coordinate in the grid matrix, and its color was stored in that location, reflecting on the game board. So, I got the colors to load and stay loaded properly, which was mostly simple.

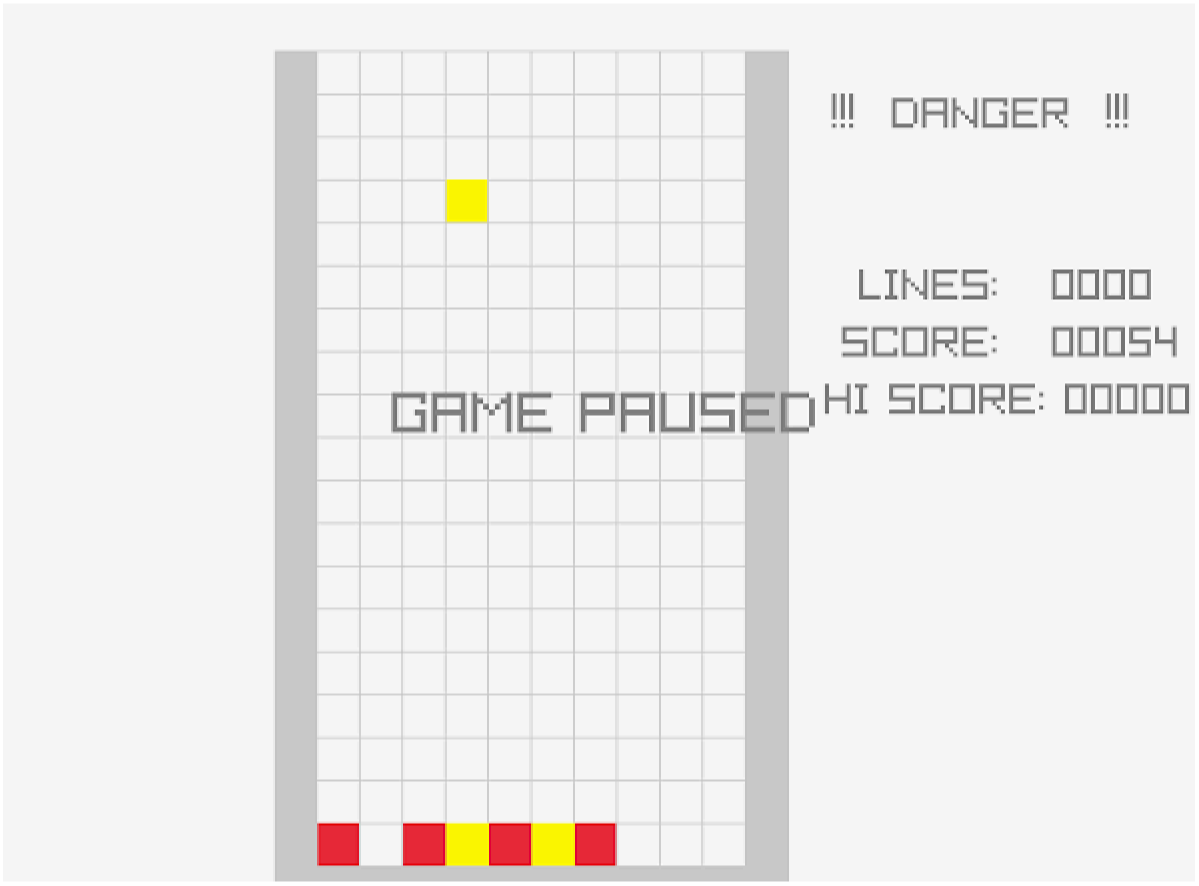
After I assigned colors to the blocks, I then had to modify the game logic to give the player a game over if two of the same colored blocks touched one another. Since the grid was just implemented in the previous paragraph, adding a method that checks the color of the adjacent tile was a relatively simple procedure, and without much error, I got it to work. The next thing I wanted to do was make it so that when a block was stacked on top of another block, it fell to the side in the same way it does in the Atari game.

I initially hoped that it would be as simple as checking the grid and implementing something akin to "if (grid[i-1][j-1] == EMPTY) {grid[i][j] = grid[i-1][j-1]} and if (grid[i+1][j-1] == EMPTY) {grid[i][j] = grid[i+1][j-1]}." …It was not that simple at all. I'm not sure what I was doing wrong because at first, the blocks would move over regardless of whether the adjacent tiles were empty or not. To make things worse, instead of rolling down, they'd defy the laws of physics and roll up! I tried tinkering with the values, but they'd keep on just rolling up into the sky. Eventually, the tiles would eventually just start duplicating sometimes as well and breaking outside of the playfield. It was stressful. I don’t even know what I did to fix the problem because eventually, after some tinkering, it just started working as intended.



```
// If able move right
if (!collision)
{
    for (int j = GRID_VERTICAL_SIZE - 2; j >= 0; j--)
    {
        for (int i = GRID_HORIZONTAL_SIZE - 1; i >= 1; i--)
        {
            // Move everything to the right
            if (grid[i][j] == MOVING)
            {
                grid[i+1][j] = MOVING;
                grid[i][j] = EMPTY;
            }
        }
    }

    piecePositionX++;
}
```

After that, though, I ran into another problem—I neglected to account for how to handle blocks falling from greater heights. If there were multiple available spaces for a block to roll down to, it would fall one tile below and then stop. Fortunately, this wasn’t too difficult to figure out—I simply turned the previous algorithm into a while loop.

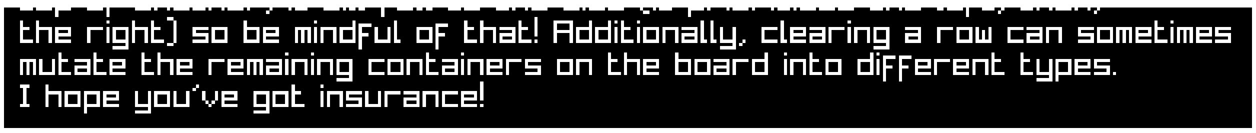


The next issue I encountered was with the "Upcoming Piece" preview. The blocks still appeared as gray, as per the template, and unlike the board tiles, I couldn't refer to the matrix. Adding a color value to the upcoming pieces just made them appear as the same color as the piece currently falling on the board. Honestly, I didn't have the patience to fix this, so I removed the preview altogether and replaced it with a sign reading "!!! DANGER !!!"

With that, most of the core algorithm was coded and designed. However, I decided to make a few changes to the original game. The first difference was that rather than requiring the player to fill the play area, you simply have to make rows, as I wanted it to function more like an endless game. I also wasn't sure how fun it would be if filling the entire area was the sole objective.

The second change was that instead of having the pieces hover at the top before falling into the chosen column, they now immediately start falling from the top, and the player can move them mid-air—similar to Tetris. Like Tetris, the rate at which they fall also increases as you clear more lines.

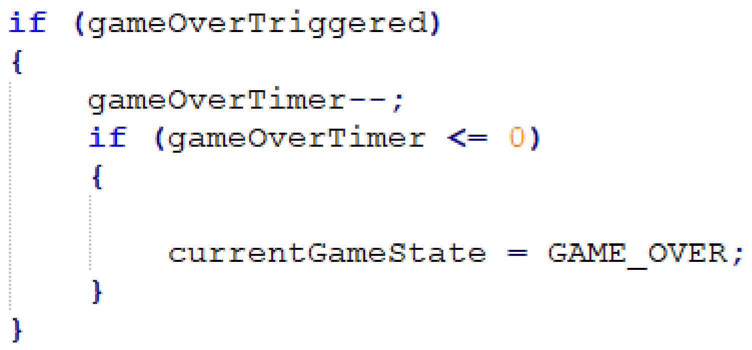
One bug I noticed was that, for whatever reason, when you made a match, some blocks had a chance of changing colors. I didn't catch this until very late in development, but it wasn't a big enough issue for me to bother fixing, and it honestly seemed like an interesting mechanic to keep players on their toes. So, I took the classic "it's not a bug, it's a feature" approach and left that in.



the right] so be mindful of that! Additionally, clearing a row can sometimes mutate the remaining containers on the board into different types.
I hope you've got insurance!

The last problem I had was that the transition to the Game Over screen was way too abrupt. This issue applied to all the Raylib templates—if you died, the game would immediately boot you to the Game Over screen on the same frame. In a game like this, it's hard for the player to even notice what caused them to lose. So, this was a problem that, rather than keeping it in to keep players on their toes, I went out of my way to fix. It required me to modify how the Game Over function worked.

What I needed to do was add two variables: a boolean called gameOverTriggered, which is set to false by default, and an int called gameOverTimer, which was set to 120, but later 100 for the additional explosion sprite. Then, I set them up in the following method: when two blocks of the same color touched, the trigger would be set to true. Setting the trigger to true causes the timer to count down every frame. Since the game runs at 60 FPS, the game will stall for 100/60 = $1.\overline{6}$ seconds before sending the player to the Game Over screen.



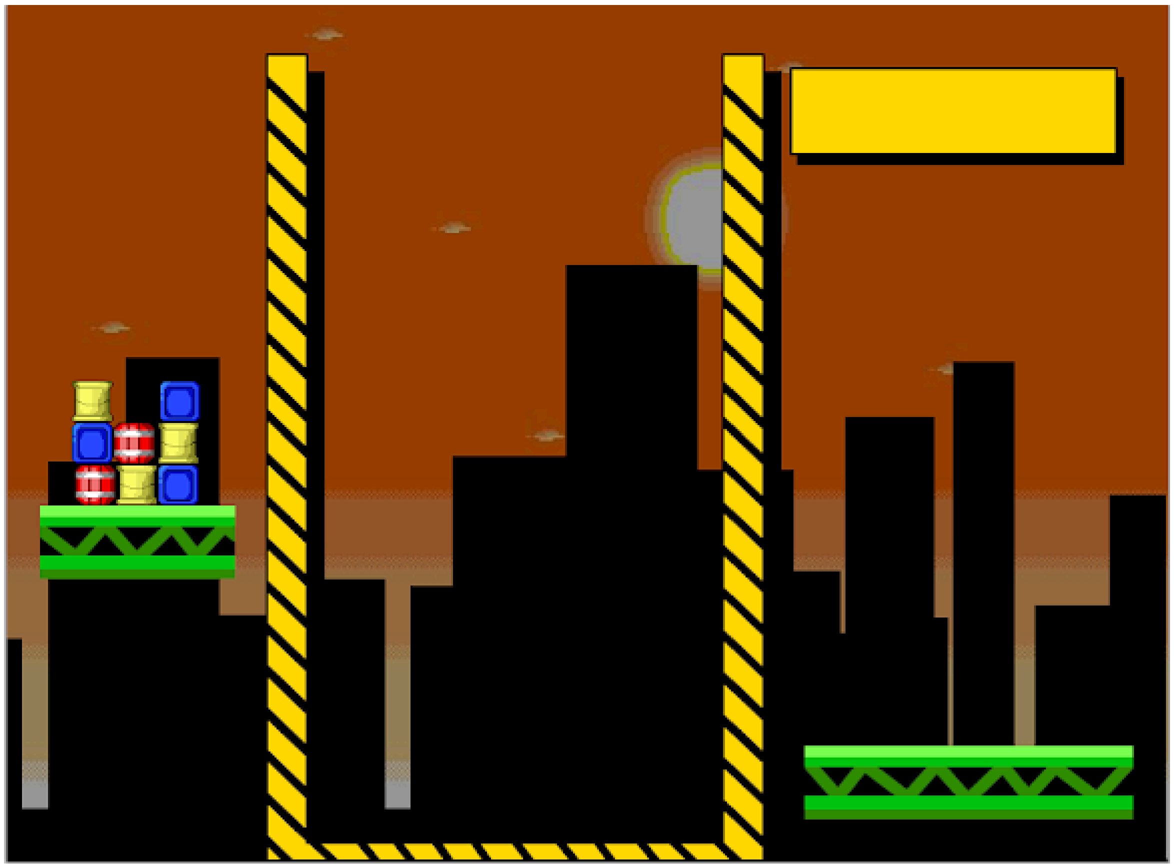
```
if (gameOverTriggered)
{
    gameOverTimer--;
    if (gameOverTimer <= 0)
    {

        currentGameState = GAME_OVER;
    }
}
```

On that note, the last core task I needed to do was complete the title screen and a tutorial screen. To do this, I had to create an array that held four different game states: TITLE_SCREEN, TUTORIAL, PLAYING, and GAME_OVER. Whenever the player performed an action—such as pressing Enter on the title screen or losing in-game—the game would switch to the appropriate state.

Each of these screens is pretty self-explanatory. I moved the main game logic into the PLAYING state and modified the template's game-over condition into the GAME_OVER state. In addition, for now, the title and tutorial screens are just placeholder text.

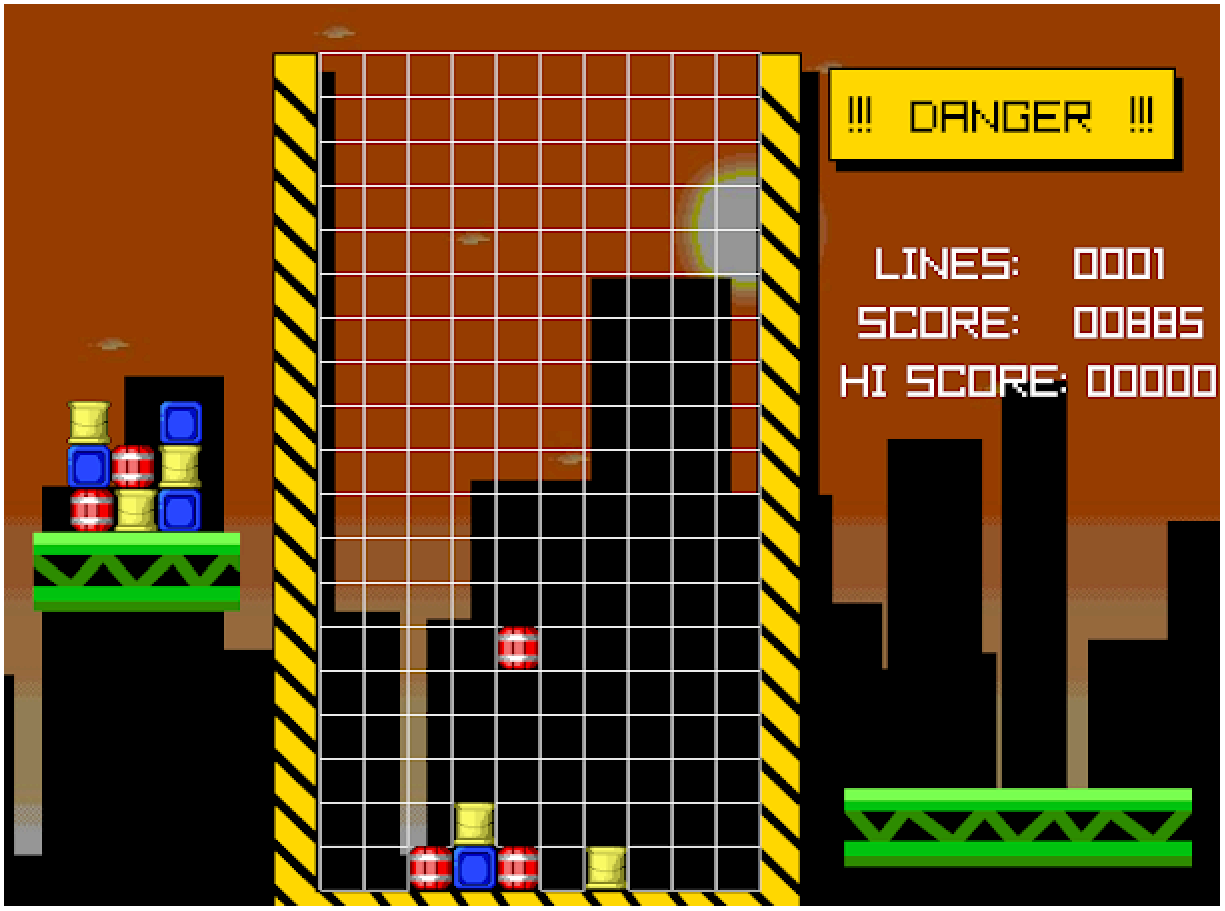
When it came to art, I aimed for a 16-bit style similar to many Super Famicom games. However, creating 1:1 sprites was tricky since the SFC screen resolution is 256 × 224 pixels, whereas my game window was 840 × 620. To make the sprites accurate, I would have needed to upscale them… I did not do this.

Anyway, I started by designing the game screen. I already had an idea in mind for the background—I took the Oil Ocean zone background from Sonic 2 and modified it by adding buildings and other details. Then, I drew the main arena, which was essentially just a big "U" with barricade tape. After this, the screen still looked a bit empty, so I added two small platforms—one holding empty containers and another with a mascot character cheering you on, similar to Patrako from the Cleopatra's Fortune games.

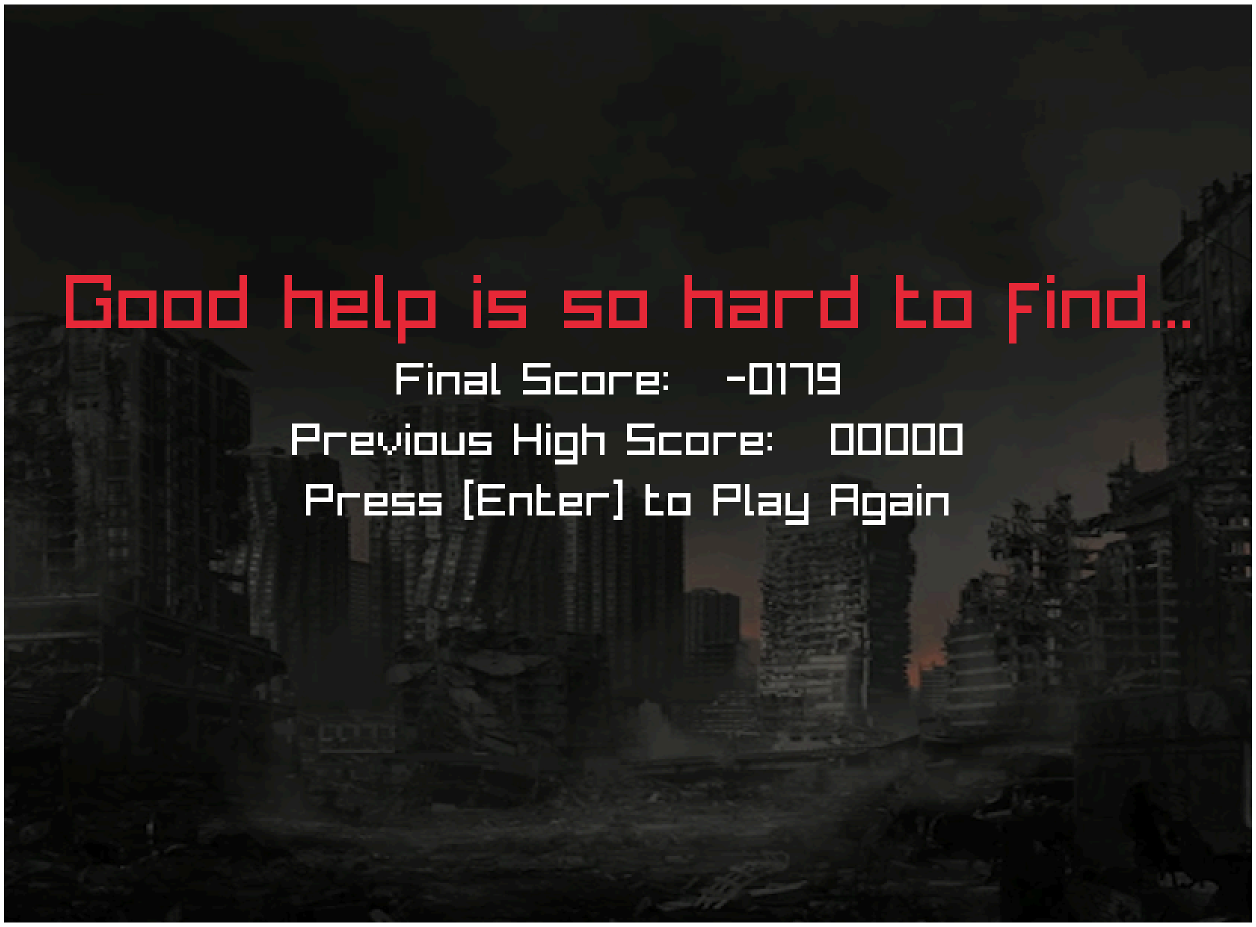


After this, I implemented the background into the game, and it looked great. Next on the docket, though was replacing the blocks with the sprites I made. I made three sprites: the yellow container, which comes from the oil barrel from Donkey Kong; the red container, which is a barrel from Donkey Kong; and the blue container, a modified block from Super Mario World. Please don't think about the logistics behind putting nuclear waste in something that looks like a red wooden barrel.

Replacing the block graphics was much harder than I expected, though. Since I had to essentially rework how the blocks spawned. After finding out the method, I still ran into several errors like blocks spawning as invisible or the game crashing outright. I messed around with the code for a while but just couldn't figure it out. After considering throwing in the towel for the night and getting ready to go to bed, I realized that I hadn't saved the barrel graphics in the right folder. When I moved the graphics over, they spawned in perfectly fine…

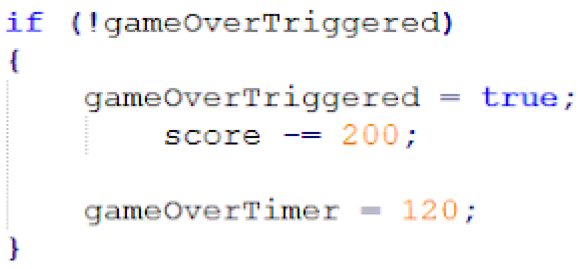


I also designed a Game Over screen to replace the placeholder. For this, I just looked for a stock image of a ruined city and put Uncle Henry’s iconic quote, “Good help is so hard to find,” followed by your score and the high score. It’s not the most aesthetically pleasing game over screen, but it’s a game over screen nonetheless.


Good help is so hard to find...
Final Score: -0179
Previous High Score: 00000
Press [Enter] to Play Again

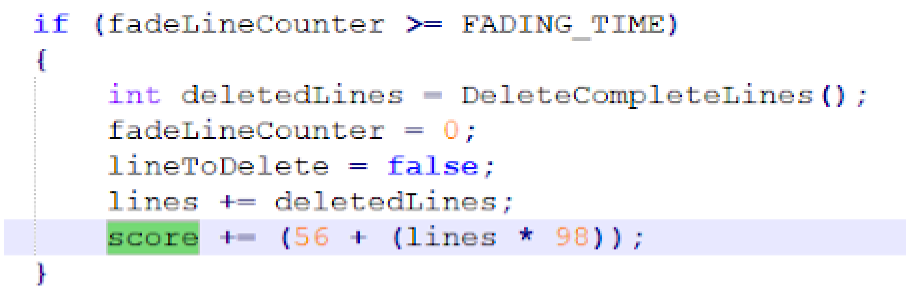
One feature that I added later on in development, though, was changing the “Final Score” text to yellow if you beat the previous high score.

I also made the white grid lines a bit fainter, changed the in-game text to standard case instead of all uppercase, and adjusted how the game records your score using the following formulae.



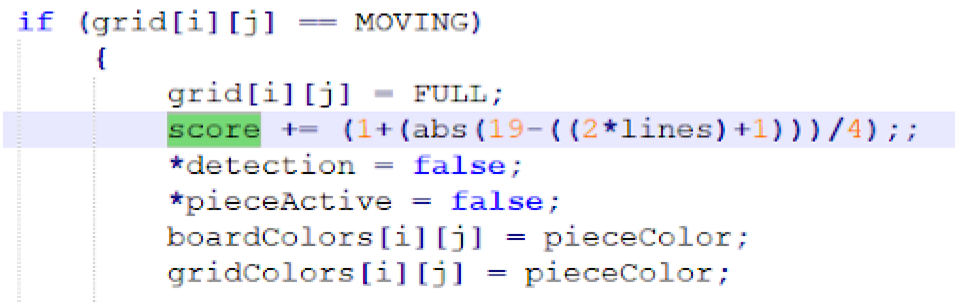
```
if (!gameOverTriggered)
{
    gameOverTriggered = true;
        score -= 200;

    gameOverTimer = 120;
}
```

You lose 200 points upon a Game Over.



```
if (fadeLineCounter >= FADING_TIME)
{
    int deletedLines = DeleteCompleteLines();
    fadeLineCounter = 0;
    lineToDelete = false;
    lines += deletedLines;
    score += (56 + (lines * 98));
}
```

For every line you make, you get 56 points plus the number of lines you currently have times 98. I wanted the scoring system to be more momentum-based.

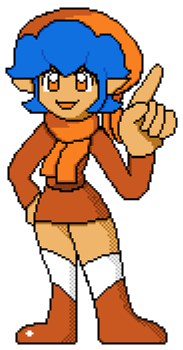


```
if (grid[i][j] == MOVING)
    {
        grid[i][j] = FULL;
        score += (1+(abs(19-((2*lines)+1)))/4);;
        *detection = false;
        *pieceActive = false;
        boardColors[i][j] = pieceColor;
        gridColors[i][j] = pieceColor;
```

For every tile you drop, you get one point, plus the absolute value of 19 - 2 times the number of lines you have, plus one, and all divided by four. I went a bit crazy when I wrote this portion. Aside from that, you also get one point for every container that rolls down from a higher location, so make sure you strategize around that.

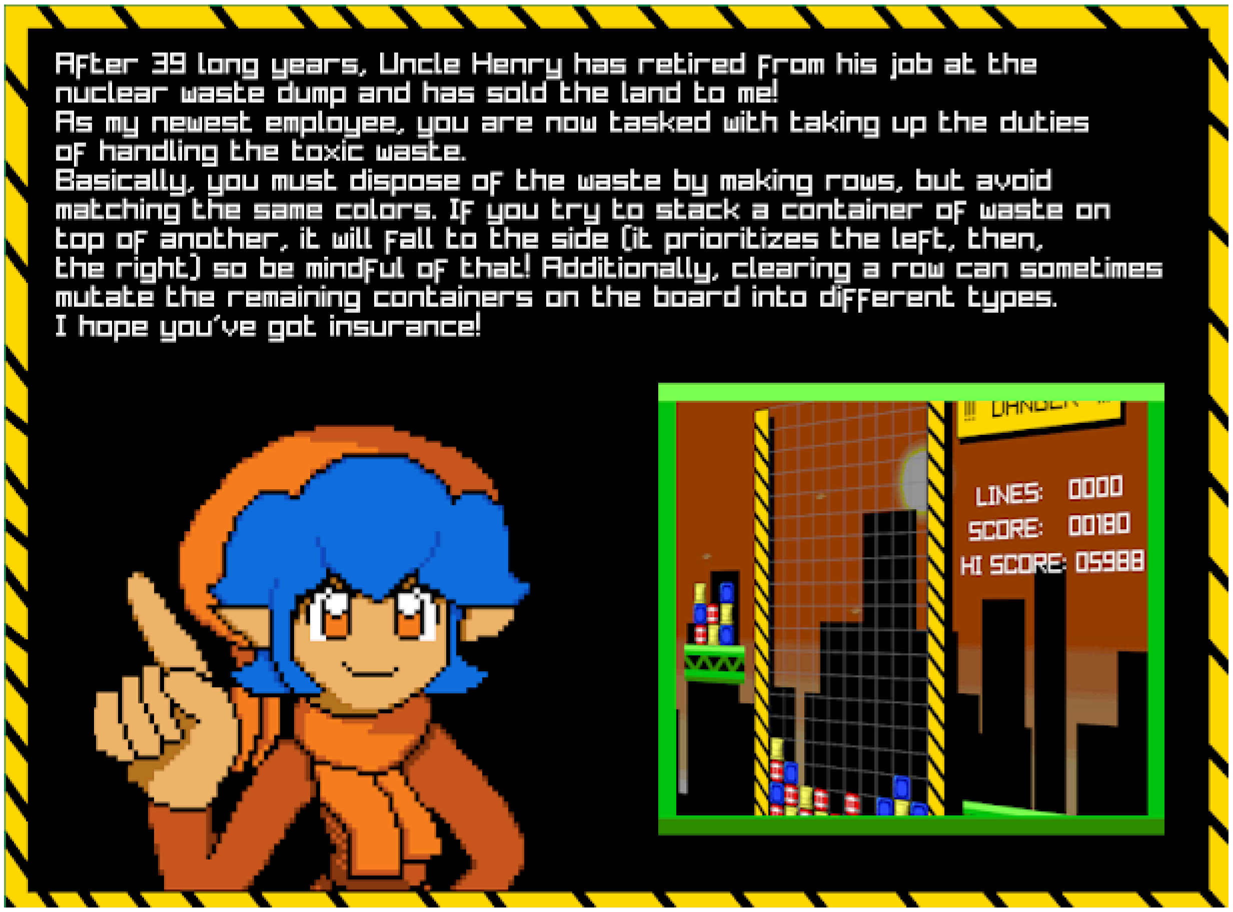
With all those changes, the last few things I needed to do were draw the tutorial screen and my mascot character for the playing and tutorial states. The original design of the character was created by an old friend. The basic design was inspired by many of the middle Pegasus sisters from the Fire Emblem series, like Farina, Thea, Tana, and especially Catria, who was my favorite Fire Emblem character at the time.

I did make some changes to the general idea over time, though. For instance, I drew her with darker skin to make her closer to my actual appearance and gave her pointy elf ears. Aside from that, since this game is set in a more urban setting and not a medieval fantasy, her armor was removed in favor of a standard brown long-sleeve and a miniskirt. For the basic mockup for the sprite, I used a sprite of Patrako from Cleopatra’s Fortune S-Tribute as a heavy reference.



I think the sprite came out pretty well, aside from the huge hands.

Like Patrako, I wanted to make several cheering sprites of the character rooting for the player and reacting to stuff like having two containers of the same color touch one another. As you'd expect, though, that was outside of the scope of this project. Additionally, I wanted to draw a different sprite of Mega-chan for the tutorial screen, but constraints resulted in me simply resizing the original sprite I made previously, mirroring it, and closing her mouth.



After 39 long years, Uncle Henry has retired from his job at the nuclear waste dump and has sold the land to me!
As my newest employee, you are now tasked with taking up the duties of handling the toxic waste.
Basically, you must dispose of the waste by making rows, but avoid matching the same colors. If you try to stack a container of waste on top of another, it will fall to the side (it prioritizes the left, then, the right) so be mindful of that! Additionally, clearing a row can sometimes mutate the remaining containers on the board into different types.
I hope you've got insurance!

I'm not lazy, I swear. I just wanted this game to see the light of day sooner rather than never 😭

I also wrote a little fanfic about Uncle Henry retiring, and added a screenshot of early gameplay. I turned the gameplay image to the left slightly to make it look cooler, I don't know if I succeeded though.

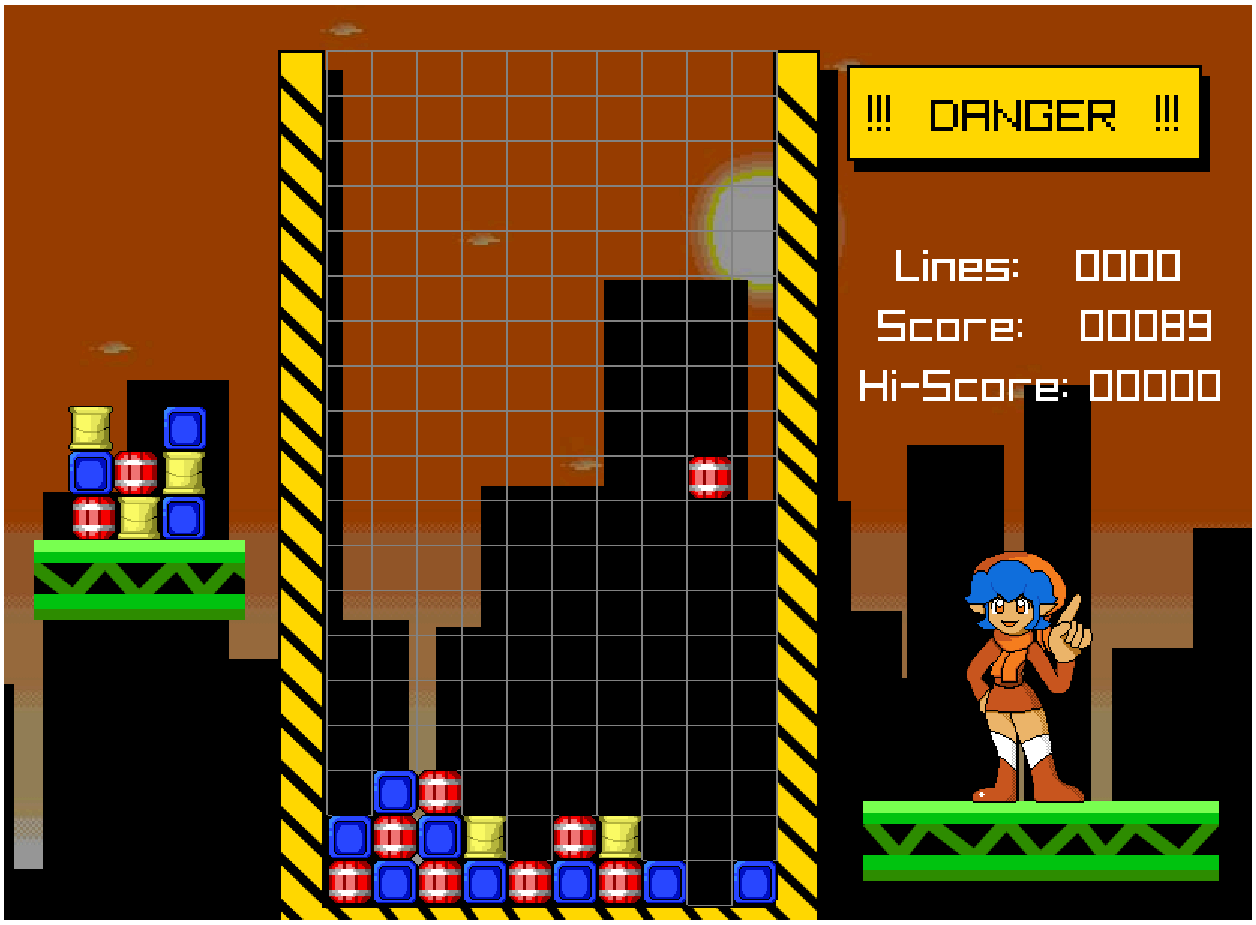
The very last thing I added was the music and sound effects. I knew exactly what song I wanted to add from the get-go. Well, I guess technically, at first, I wanted to play The Wastelands theme from Mario Strikers Charged because, y'know, it's nuclear **waste**. But later, I figured that Crystal Canyon from the same game fit the vibe of the game much better.

I didn't just want to use the song as is, though; I wanted a 16-bit rendition of the song to play during gameplay to fit with the general graphics. This was when I went out and started looking for a midi of the song. A user by the name of PaperMikes uploaded a piano tutorial of this song back in 2019 and provided a downloadable midi of their version of the song in the video description, so I downloaded that and got to work.

I decided to use EarthBound's soundfont, which came out a lot better than I thought it would and was light on the ears. After that, I added some sound effects, like the button confirmations, the sound when a block is placed, line clears, and the explosion sound effects. The button confirmation and placing sound effects worked perfectly as intended, but the explosion sound effect and the line clear sound effects are a bit bugged.

You see, the starting part of each sound effect places rapidly over and over again until the event that triggered it—clearing a line or setting the gameOverTriggered variable to true—is complete. While this is an easy fix, the result of this bug actually sounds better to me than the intended sound effects, they're very reminiscent of the sound effects on the original Atari, so I left both those sound effects as they were.

With the main programming, art, and music completed, I went back to do some final touches, and finally, I had a game on my hands!



How to Play

As I said before, the object of this game is to rack up the highest score possible—but be careful! Matching two pieces of the same color will end your run. Plan your avenues carefully and adapt to the drop patterns to avoid an unlucky game over!

Controls:

- **Enter**: Start game, advance tutorial, and reset after a game over
- **Left/Right Keys**: Move pieces
- **Down Key**: Drop pieces faster
- **Note**: In life, there is no pause button, so there is no pause feature in this game either!

Dependencies

- Raylib (https://www.raylib.com/) — A simple and easy-to-use library to enjoy videogames programming.
- Notepad++ for Raylib