The project goal is to work on systems with limited hardware resources while using modern tools and joining communities that have worked with these systems to gain knowledge about techniques to develop classic video games using high-level languages.

The project is divided into four parts in which we developed four different games with different features and challenges that would help us understand how games were made for these consoles with the final goal being the fourth game where we would have to apply all the things we learned from the three previous games to make a more complete game.

For our project, we were required to work with a system from the 16-bit generation or earlier, basically, we could work with anything before the Super NES.

We decided to work with the Game Boy which is an 8-bit handheld game console developed by Nintendo in 1989. As mentioned earlier the Game Boy has an 8-bit CPU and has color support of 2-bit which are 4 shades of light to dark olive green.

Despite being an old console we found some communities dedicated to developing homebrew Gameboy games which were mostly for the Gameboy Color, but we thought there would be a lot of resources for us to learn about.

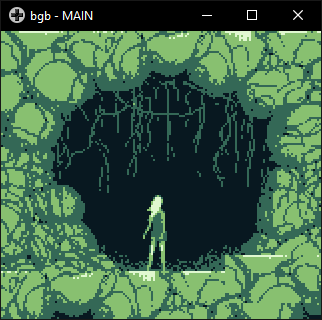
To code our games we used the Gameboy Developers Kit (GBDK) which is a set of tools that grant us the ability to develop our games in C. The set of libraries allowed us to compile our C code into 8-bit assembly generating .gb files that can be used on a real Gameboy or with an emulator like bgb64 or VGB.

We also used Visual Studio Code as our IDE because it can open terminals in the ide which let us compile our games to see changes from the IDE. And we used BGB which is an emulator for the Gameboy that allowed us to see the VRAM, memory banks, and other useful information when debugging the games.

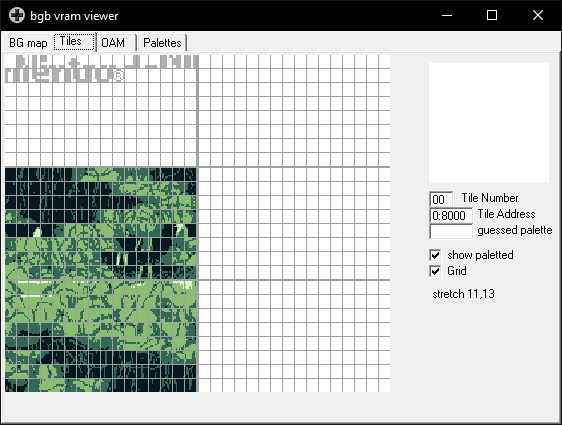
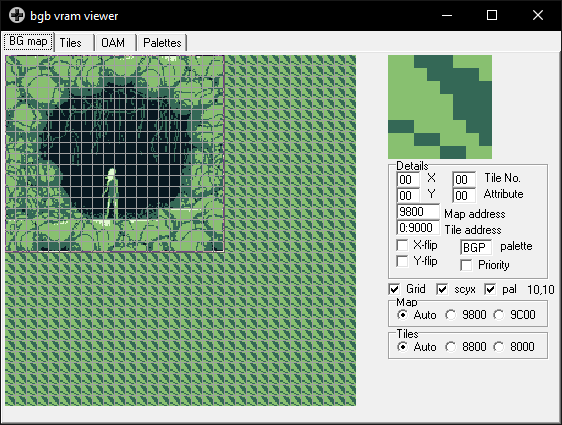
To design the sprites and maps we used Harry Mulder’s Gameboy Development: Gameboy Tile Designer and Map Builder. The tile designer is a program that we used to design tiles in 8x8, and the map builder allowed us to use these tiles to draw in a canvas the backgrounds of our Gameboy games.

The major reason we used this to build our maps instead of using other methods is that once we build the maps and the tiles we have the option to export the design directly to RGBDS assembly-source and GBDK C-source. This will provide us the tiles and maps that we drew as arrays of hexadecimal data that the GBDK header file can interpret.

The first game requirements were a title screen and a simple text game, this could be a number-guessing game or something simple like that, at the beginning we wanted to do a cave adventure game where the player could choose to go to the right or left path, but after some time we opted for a true and false game since we thought only going left or right without any information to take decisions would be stressful for the player when losing. The game would start with a splash screen of a girl looking at a cave.

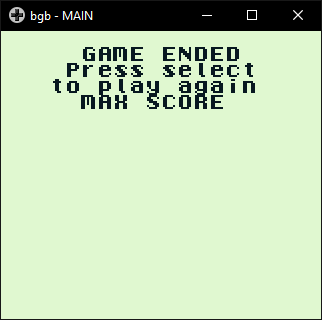
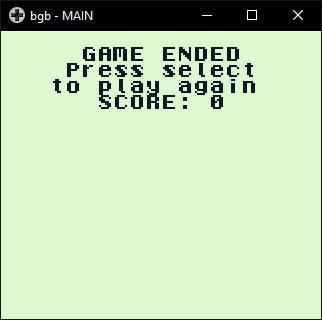


We did the title screen in GIMP and after that we resized it to 160px by 144px, convert it to grayscale and run an Indexed Color Conversion with a maximum of 4 colors because the Gameboy only has 2 bits for color support. After that we got the image shown above but to actually include it in our game we used a GameBoyPngConverter that we found on gingemonster’s GitHub who is a developer of homebrew Gameboy games. The converter takes the png and outputs two arrays, one with the tiles that are used in the background and another with the map.

And to display it we first load the tiles in the VRAM of the Gameboy  
  
and then with the map array we show the tiles in the desired order.  


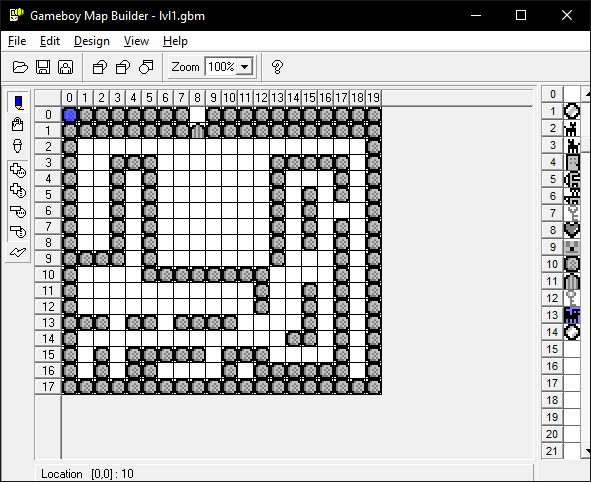
After learning this principle we were able to load the tiles needed to show the questions for the actual game.

After pressing the start button we print the instructions in the screen and the user has to choose True or False by pressing A or B, the question pool is selected randomly from a questions array, that way the player gets a different experience every time it plays the game, some of the issues we had is that while using the gbdk header files we could not use some of the C built in libraries like time.h and therefore we had to get the seed for the random order by measuring how long the user holds the start button after the title screen. After the player either wins or loses by answering wrong a question a screen with the player’s score is printed.



GAME 4

The first thing we worked on in this game was the background collisions, to make things easier we made it so that the characters would move 8 pixels, this way it could move only from tile to tile. This trick is used in a lot of classic adventure games like Pokémon where every character could only by tiles and not be between them. Then we made a function that would convert the x and y location to the corresponding tile in the background, and after that we would compare that tile in the background array to see if it was an empty space.



Then we called that function in every input function to check if the tile were the player wanted to move was an empty space. This method allowed us to build maps and don’t have to worry about the collision information because those functions were reusable for all the maps we were working with.

Then we coded the movement of all the enemies, at the beginning we wanted to have every character to be able to see you, to make the player hide to avoid being seen but after we coded the movement of all the enemies, we noticed that coding that type of collision for all of the enemies would take a lot of the memory from the rom, that’s why we made it so that the player would only have to avoid being hit by the ghost and hide from one dog in each level.

The memory constraints of the Gameboy system were more noticeable in this game because we had a lot of enemies and those enemies had a lot of information in their codes, we wanted to add more sounds and music like in the previous games but in the last weeks we noticed that we were running out of memory, we tried methods like using different banks in the rom or setting a lot of variables to constants to free some space but we were not able to squeeze more levels and details in the last game.