CLASSIC VIDEO GAME JAM

Final Presentation/Essay Outline

Introduction(JR):

  -Goals?

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.

Overview(JR)

-Libraries?

-Resources?

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.

We had a few changes to our original proposal for this project. Due to the fact we lost some team members and more importantly we didn’t know our limitations or how difficult things would be. For our first game we switched it to a trivia game. Our second game it was the same idea, just we added a coin for the player to collect. Game 3 we weren’t able to add obstacles with a moving background so we just stuck with moving sprites that we had to dodge or shoot.

Game1 (JR)

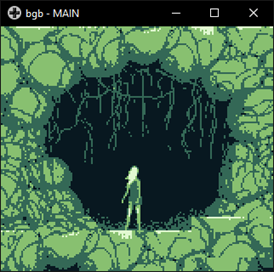
-game?

-Text-based game

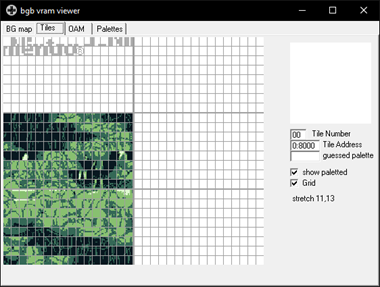
-issues:

-

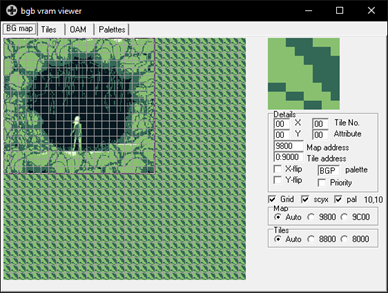
The first game requirements were a title screen and a simple text game, this could be a number-guessing game or something simple like that, at the beginning we wanted to do a cave adventure game where the player could choose to go to the right or left path, but after some time we opted for a true and 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, converted it to grayscale and ran 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.

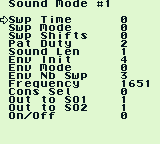


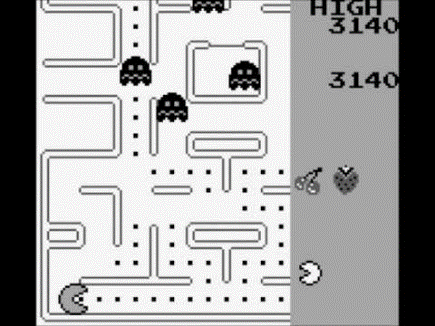
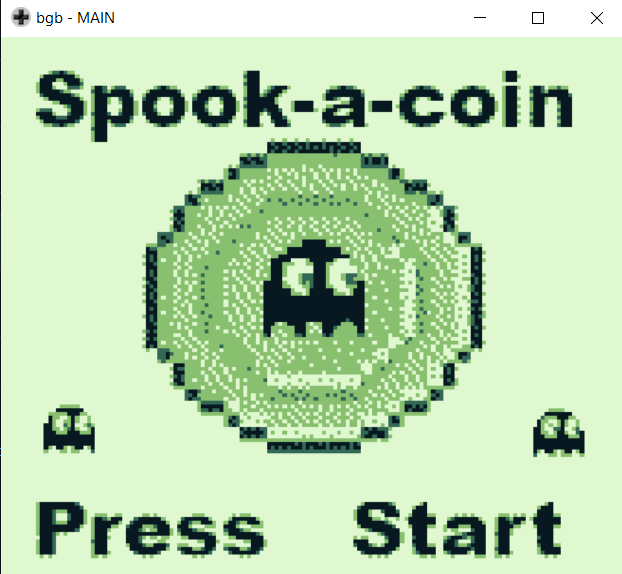
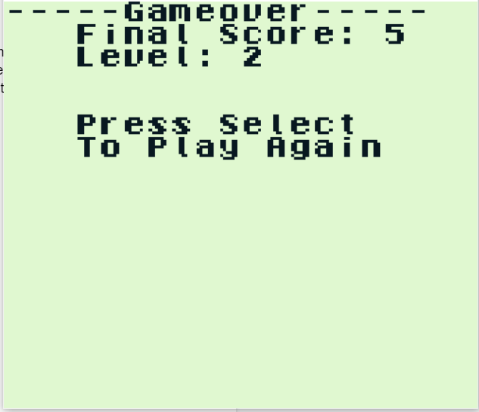
After pressing the start button we print the instructions in the screen and the user has to choose True or False by pressing A or B, the question pool is selected randomly from a questions array, that way the player gets a different experience every time it plays the game, some of the issues we had is 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 a question wrong, a screen with the player’s score is printed.



Game2 (JV)

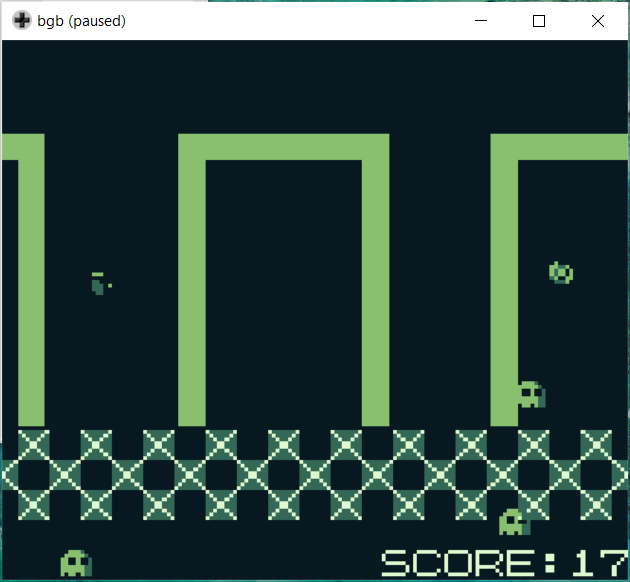
After we learned the system, libraries, and how GBDK worked with our first game experience, we then started with our second game that requires the use of sprite and some form of gameplay. Some other requirements were: a fixed-screen environment, sound, and at least 1 independently moving sprite. For research on this game we got help from a youtube channel named GamingMonsters. He went through the basics on how to set up the environment, get a sprite on the screen, and how to use the different functions and features GBDK provides. For example, set\_sprite\_data(),move\_sprite(),set\_sprite\_tile() and many others. We learned to make our sprites using GBTD and exporting them to .c files for us to use in our program. So after we finally got our sprites on screen, we had to find a way to check for collision detection. So we made a simple function that checked the X,Y coordinates for the player and enemies and checked if they came 8 pixels apart to determine whether or not they collided with each other.

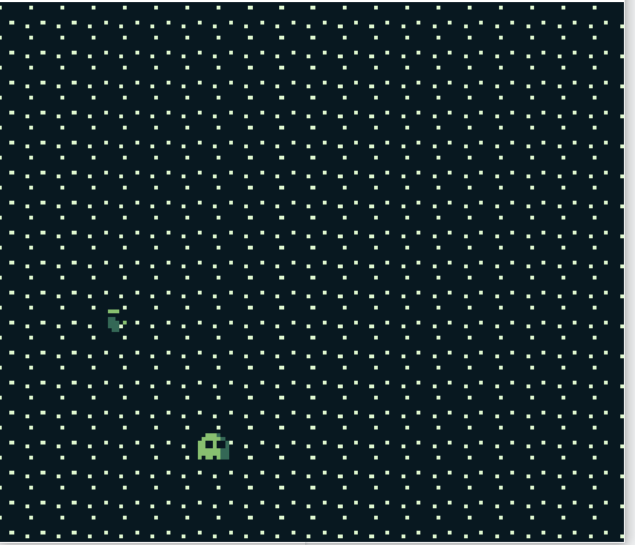
Moving on to our start screen, we designed a start screen using ‘GameBoyPNGConverter’ in which you are able to upload a picture to this program, and it will convert the image into a .c file. In order for the program to work we would first have to desaturate our image using an image editor(GIMP), and then change the color settings so it can be only four colors, just as the GameBoy was limited to. As we discussed earlier, we needed to figure out how to add sound to our game. With the help of Zal0 on GitHub, who made a program using GBDK named ‘GBSound’, that allowed the user to create the different types of sound by changing the registers made it really convenient for us to make sounds for our game. It was still challenging to use and find what we wanted but it worked out it in the end.

Before I discuss what our game is I would like to give some examples on what other games we could have made with these requirements, games with the same features would be games like: pacman, space invaders, snake, asteroids, etc. So for our game we came up with a game we named Spook-a-Coin, where the goal of the game is to dodge falling ghosts and at the same time collect as many coins without dying. The player only has one life, and the game has up to 5 levels in which the difficulty gets more and more challenging, and the speed of the ghosts gets faster as the player collects more coins. We also added a score and level counter to the screen, and a game results background when the player loses.  We also were able to add sound into this game. There are three sounds in total, when the player collects a coin, levels up, or loses the game. Each sound is different and distinct to one another.

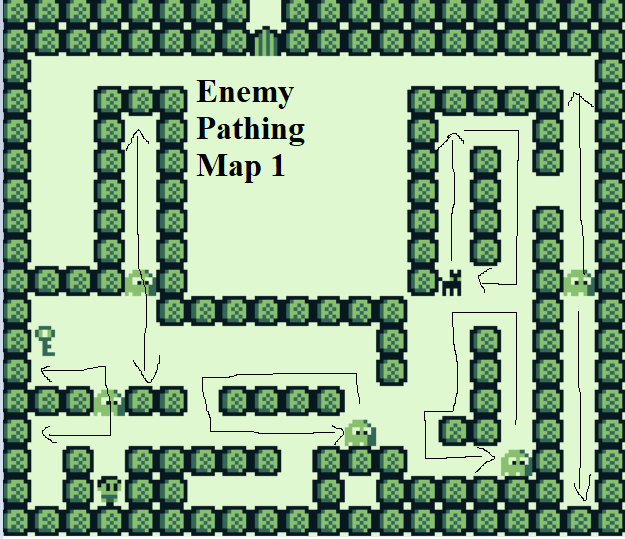
For this game we had no major issues, but some minor issues we came across, but were able to fix were: how to get the letters on the screen while still displaying the background in which the gameplay runs. We solved this issue by creating each letter and number in a tile, uploading the letter/number background, and calling it at the bottom of the screen. An issue we weren’t able to solve was to use more than four sounds for our game. There are four different sound modes: NR10\_REG - NR14\_REG, NR20\_REG - NR24\_REG,NR30\_REG - NR34\_REG, and NR40\_REG - NR44\_REG. After using all four, when we would try to add a different sound using one of the used sound modes, we weren’t able to. There wasn’t much info on how these sound registers work and we weren’t able to solve this issue.

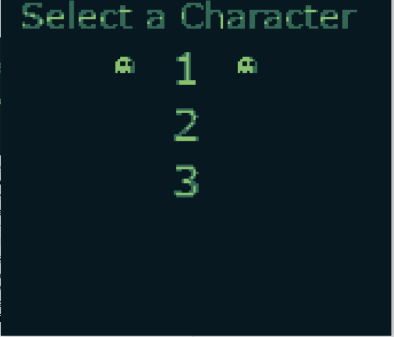
Game3 (JV)

For our third game, which required a moving character sprite, independently moving sprite(s), a moving background, and sound; we created our game called Ghost Hunter. For comparison from these requirements, a game like Super Mario Bros  would be an example. So the purpose and story of this game is that the player is in a haunted house with ghosts all over the home, as the player goes down the hallway he is faced with ghosts moving toward him. The goal for this game is to not come in contact with the ghosts. The player can either shoot them for one point, or just dodge them. Once the player shoots 3 ghosts, coins will start to appear and will be 1 point for each collected coin, and if the player misses a coin, he will lose a point. The most impressive thing we did for Ghost Hunter was make an A.I. for the ghosts’ movement. With the help of some research and some great GBDK examples by GitHub user Mike Rombout, we were able to understand and apply this A.I to our game. This was a great discovery since we also used this in our final game, which has multiple different A.I’s compared to just one. We also just used the same A.I array for all 3 ghosts and coin, but used a random number generator to make their start point different each time they pass through the screen. Every sprite moves/paths in a wave simulating a typical ghost’s movement. The player is also able to move around the screen and can also double shot two ghosts at once.

One of the issues we stumbled upon were memory issues, we had been loading in too many tile files that we were not using anymore and just had to delete all the unused ones to make the game compile again. Another issue we had was the sprites not being visible due to a bad background color choice and hard on the eyes when staring at it for too long. We quickly got feedback from our peers and changed it right away. Other than that, we really had no unsolvable issues with this game, which was awesome! If we were able to work on it more than we had time for, we would have added a boss level where a meta-sprite bounces around the screen until you shoot it x amount of times, and also a wave of ghosts that has only 1 8x8 opening in where the player has to fit in that space or else it would get killed by the ghosts. Many great ideas that we could have just transitioned this game to our final game, but we really were intrigued and excited in making this next game called Maze Run.

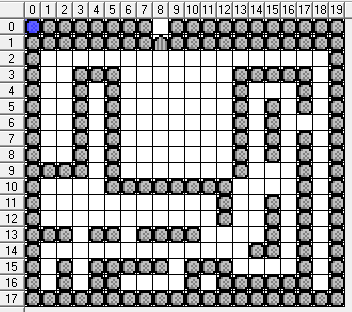
Game4 (Both)

For our final game we created a maze game, Maze Run. This game we were required to take all our knowledge from our previous games and apply it to our final game masterpiece. So we came up with the idea to make a maze game where the player must find their way to the ladder while at the same time avoiding the ghosts touching you and a wolf seeing you. There are four levels currently and you’re only able to access the ladder to move on to the next level until you collect a key in that room. The difference from the wolves and ghosts is, if the wolf sees you directly in his path, he will attack, and the player will lose the game. All enemies move in a fixed pattern using the A.I. we previously used for game 3.

These levels get more difficult as the game progresses, and in the end if the player wins, a message is shared, stating that they made it safely out of the maze. Another feature we added to this game was our start menu which has 1 sprite that is controlled to either select start or select story. We also added a character select with up to 3 different characters the player is able to select. 

code/problems(unsolved, solved)/future?

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 be in 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 where the player wanted to move was an empty space, we also used this method to see when the player would get the key or reach the stairs to the next level. 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 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 as mentioned earlier, 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 in more levels and details in our final game.

Conclusion(JV)

With this project we learned and improved key skills that are definitely  going to help us in future projects and/or future jobs.  Some of these skills are:  team communication, time/project  management, and troubleshooting bugs/issues. For team communication we set up a weekly meeting and we both made it clear that we were available all throughout the day if any of us had any trouble or questions. For time/project management we would assign ourselves with needed tasks for that particular game and later combine our work and approve or give feedback  of each other’s work. Lastly, and more importantly, we learned many ways on how to research our issues whether it was looking at forums, finding a community discord, reading the library documentation, or reading through other people’s game code and figuring out how they solved the particular issue/bug.

Overall this project was a great experience building games with a 30+ year old system, and it gave us great appreciation for games that were created for the GameBoy like: Super Mario Land, the Pokemon series, Donkey Kong, etc. After creating all of our games we acknowledge the time and effort put into these household name games created by Nintendo and these other major companies at a higher level. Whether it’s frontend work with the background image, sprite movement, music and soundtracks, or especially gameplay, we notice all the little things that we may have overlooked in the past when we would look at these games as a gamer, compared to as now as a GameBoy programmer.