Introduction

Our game is Eradication Man. The idea behind it is supposed to be a mix of the old school beat ‘em ups and shoot ‘em ups. So it is designed as a 2D-side scroller, in which the background itself and the characters, sometimes, are the moving aspect of the game. Similar to games as Shovel Knight, Contra, Mario and the Teenage Mutant Ninja Turtles.

Rules

The rules are pretty basic for the game.

1. Defeat the viruses by using your syringe and dart gun.
2. Viruses grant currency on defeat.
3. Contact the friendly bunny, he will give you upgrades for a price.
4. Collect new powerups to defeat bosses and armored viruses.

An example would be: Once you enter the level, you always start off without any ammo in your dart gun, but you use the syringe to defeat the viruses which also drop essential DNA currency to trade with the bunny, who will help you get better powerups and strengthen yourself in the process. As you progress in any level, defeated viruses have the chance to drop ammo for the dart gun, which will let you defeat the viruses at a distance.

Design

- Controls- Simple controls such as using the JavaScript event key codes to implement the movement of left and right, as well as when the space bar is pressed, the character jumps. Using the event key codes also lets us use the letter X as our way to shoot and the letter R as the reset button.

- Animation Mechanisms- The main animation is done though the creation of gifs. Once we have a certain gif created, we create a variable based on the image we want, such as **var EnemyImg**. Then we use the **loadImage()** function as part of p5 to load the gif to the variable. Then depending on what the movement is being done, or if the characters are just standing, a specific gif is being loaded by the **image()** function.

- Win State/Lose state- This portion itself is partially connected to the variable of health. Currently in our build there is no Win state, but there is a lose state, in which the character`s health reaches 0. If hit too many times by the virus, it is game over.

- Visual Representations of game states- The use of game states in our game is again fairly simple. The game always starts in the 0-game state or the menu. This is until the player hits the enter key, in which the game begins and enters game state 1. When the characters health reaches 0, that is when the game state is changed to 3 or the lose state. We had saved the 2 for the win state but never finished.

- Expected Skills the player must have- For this game, there is no previous requirement to be able to play the game effectively. Newcomers are welcome, though if you do have experience with these types of games that will help.

Software Architecture Detail

- State
  - This has the responsibility of being each game state such as the win, lose or game state.
  - It collaborates with the draw function.
  - We set each game state or the variable **state** to either 0, 1, or 3. 0 being the menu, 1 being the game, and 3 being the lose state. 2 was supposed to be the win state. This was done in the draw function. When state = 0, we would have the title screen, but we would also put certain variables back to their original position to act at part of the reset. When someone hits the enter button in this state, it changes state to 1. When state is 1, it runs the game function as well as having the reset button in the background so if someone does hit R, the values are reset. During this state if the players health reaches 0, they state is set equal to 3 or the lose state and just indicates to hit R to restart.

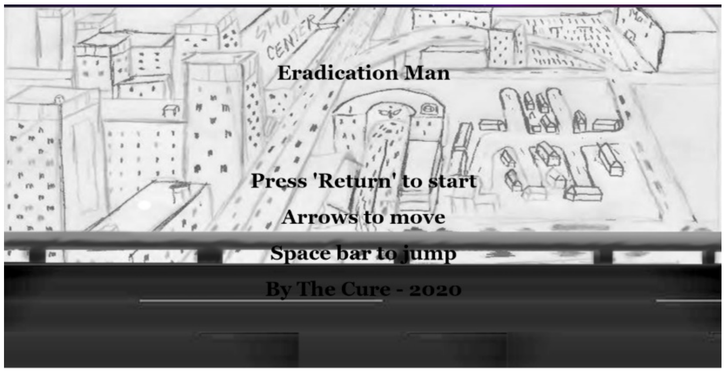
- Player
  - Responsibility of being the playable character
  - Works with the game function and the keydown functions to facilitate movement. As well as work with the enemy to see about the lower health. Since it also works with the State, it may change the state based on the player health.
  - The player did have a couple variables to it. Such as the p1x, p1y and pWidth and pHeight. It also had other variables that were attributed to the multimedia aspect such as having the sprite show. The collision with the virus and boxes were done separately. The virus collision was done below the code to show the virus sprite. The boxes collisions were done in the platform function. The player

also had its own function that is called during the game function. It is supposed to determine which sprite to use when looking left, right and when shooting as we did not have a jumping animation.

- Virus
    - Works with the player to determine if the player is hit.
    - This was done simple with about 8 lines of code to show the sprite and the collision with the virus. Though the shooting collision was not implemented due to time constraints.

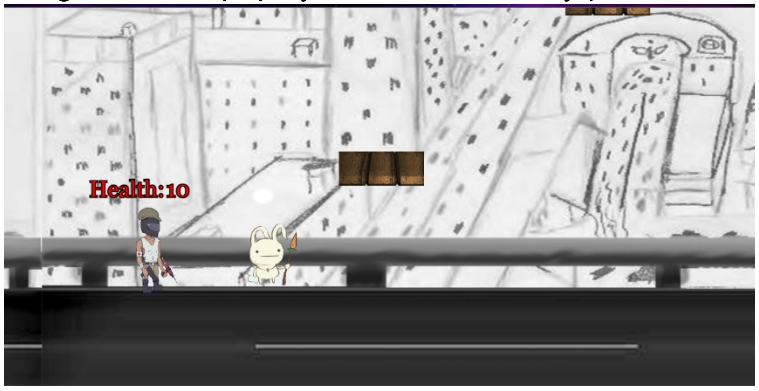
- NPC
    - So far works by itself and only collaborates with the keypress function.
    - The only thing we have for the NPC is the NPC being displayed and running away from the main character, as the idea is that he does not know who he is and not yet saved him.
- Platforms
    - Works with the player collision so the player can stand on it.
    - This had its own function called platforms in which the sprites were called, but as images and the collision code was used.
- Tranquilizer
    - The shooting code gave us the most trouble, so we put that in the back burner for a while, we tried implementing a beat`em up style use of the syringe but were unable to. So, we went back to the dart gun. The code for this is done within the movement code, since we needed to make sure that when the player was looking right, he was able to shoot, but not when he was looking left. Though this was the 1st iteration of our code.
- KeyPressed
    - Work with the state, player, virus, bunny and tranquilizer.
    - Determines which way the player is moving, if it goes from the menu to the game, if the player is to shoot and if the virus is moving toward the player and when the bunny runs away.
    -

Game Demonstration Visuals

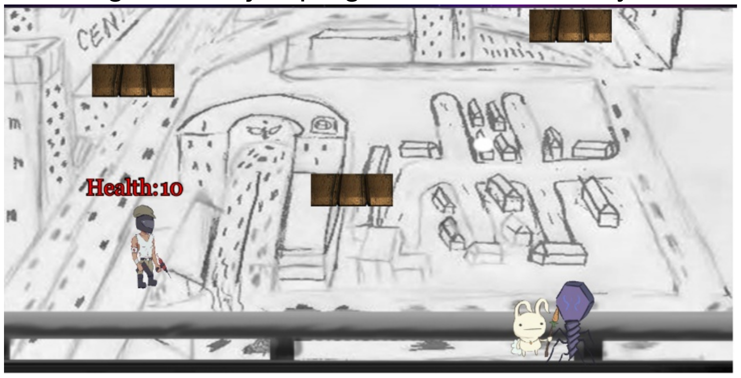


This is the main Start screen.

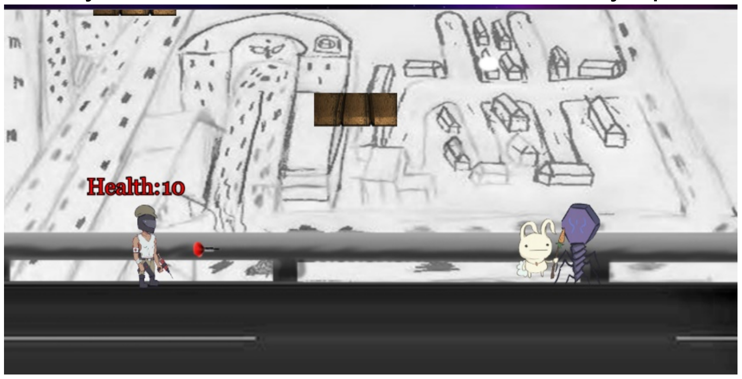
During the creation of the game there were no big algorithms used. The simple algorithms were done to determine which animation would be happening when the character is moving either left or right, when he is shooting, as that is all we had so far. As for the organization of the code, We used comments to show what section is for what. The beginning of the code is first commented with game control, which is the game states. Then we have one that is the players x and y coordinates as well as height and width, commented as player. All the variables are separated based on what they were going to be used for. We also commented the section for the preload, where we would load an image to a variable, specifically a variable that was part of the multimedia section. The setup function was used to create the canvas as well as to center align the images and text as well as begin the bullet array. In the draw function was used to indicate the 3 states of the game we have at the moment. Below the draw function is where we placed all the other functions that we created and used to make sure the images came up, player information, key presses, and box collision.



After the Draw function, we have the platform function. This used the image function to call on the platform variable and initialize its dimensions and its placement. The collision was also done here. So the idea is that when the player is less than have of the width of the box then he was would standing on it by moving the players y position slightly above that of the center of the boxes y position. We would also indicate that since he was standing, he can jump again and his velocity would be 0.



The game function is next. Here we used the image function to call multiple backdrops and initialize the dimensions and placement, but we needed an offset in the x-axis so it can look like the background is moving. We also initialize the camera positions and the zoom it has. Here we also call the player1 function and platform function and set up the enemy and its collision as well as the bunny npc and call the keypressed function.



After we have the gravity function, that is just that. We would have max and min boundaries that the player can go as well as include the velocity when the player is coming down.

Bibliography

https://molleindustria.github.io/p5.play/

https://p5js.org/

https://www.dreamstime.com/stock-photos-tranquilizer-dart-regular-metal-red-feathered-tail-isolated-white-background-image36574043 - Tranquilizer sprite

https://www.youtube.com/watch?v=Ouza_4SsbLc

All sprites made by Tianzheng Li

Notes about bibliography.

The coding train video was used toward the end of the project, but was too late to see there were better ways of doing things.

We used a stack overflow question that someone asked about scrolling backgrounds, and we sort of understood what they were saying, but was in C# and the link is 404.

We also used a reddit post to where they were talking about the collisions in p5 but has since been taken down.