Space Defense: The Coding Phase

Before the game even starts, there's a lot going on in the coding. First off, we've taken the rain program and completely repurposed it into our own version of 'Missile Command'. The original structures used to generate raindrops are now what’s used to create the incoming bombs, or enemy missiles. This code was also copied and reversed to allow the user to fire projectiles back at the incoming bombs/missiles from a spaceship. This spaceship is also repurposed code from the original rain.c example. The umbrella in now the player controlled spaceship which can fire projectiles upward. And after several global variables are defined, the game will move into the main function where it will initialize. The game will load new sound effects, and images, and call the PromptPlayer() function.

When players first start Space Defense, they will be greeted by a menu. From This menu, the game will prompt the player to either start a new game or quit the program. This is handled by the PromptPlayer() function which checks for what the player enters. If the player enters 'q' or '0', then the function will end the game.
However, if the user selects '1' (or 'new game' when in a more finished state) they'll go deeper into the PromptPlayer() function by giving a 1 or 2 player option. The user will then be given a choice between three difficulty settings. After the PromptPlayer() function completes, the game will have a variable defined which will determine how many players there are, and how many lives the player has. As of now, the 2nd player option isn't complete but ideally will be done by duplicating the structure used to make the first player object, copying the linked list used to make the incoming missiles and 1st player projectiles, and finding a 2nd set of controls that don't conflict with the 1st player controls. Whether or not these 2nd player properties get used is determined in the PromptPlayer() function.

Some of the biggest changes to the code are found in the render() and physics() functions. In the render() function, the original appearance of the raindrops are replaced with a custom image of a bomb, which will be the new target for players to hit.

The projectile that the player fires, originally a simple red rectangle, is now an image of a missile.

An image of an explosion was also added for when a bomb collides with any object in game. Early in coding, only one explosion could be rendered on screen at a time. However this has been improved by replacing the responsible variables with integer arrays to allow for any desired maximum number of rendered explosions.
The Umbrella has been replaced with a really cool looking space ship.

And the purple mountain background is now an image of space.

In the Physics() function, a lot of real time checks and updating occurs. The code used to simulate gravity drawing raindrops down the screen was copied and modified in order to move the player-fired projectiles up the screen. The collision between the umbrella and rain was also modified to detect collisions between the
incoming missiles and outgoing projectiles by using a series of comparisons on the individual node positions. When a collision is detected, a new function showExplosion() is called which displays an explosion at the coordinates of the collision, and then move the 'exploded' objects away from the viewable play space.

We're also adding additional heads up display features to inform player about remaining lives, current score, and missile count. And hopefully, we'll soon finish a 'game-win' and 'game-lose' condition to make the Space Defense a fully functional game.