Reid Nelson

My game, Spook ‘Em Up, is a Halloween themed shoot ‘em up based on the Shoot and Survive examples provided, following a ghost as he tries to collect points in order to purchase a new Halloween costume. All in all, there are four main class files that were created to put the game together, along with the varied assets I created for the visuals of the game.

My gameplay choices mainly took inspiration from the Touhou series and related shoot ‘em ups, with vertical scrolling and shooting as well as descending enemies. My main departures from those games, however, come in the way the points and health are set up, as well as the lack of enemy bullets. If descending skulls are successfully shot, the player’s score increases by 100 and the player’s health increases by 2, whereas if they pass the screen the score decreases by 65 and the health decreases by 11. In the end, it set up as sort of a double score system, but was ultimately necessary to build a solid difficulty level and encourage the player to hit every skull they could. Once the score passes 10000, a win screen triggers, showing the player exactly the costume they’ve won for the ghost, whereas if the health goes beneath zero, a failure screen appears showing the final score and an encouragement to try again. Overall, the gameplay is pretty straightforward, and I feel is suited well to a small five to ten minute game.

The design of the game assets were based a lot in more cutesy stylings of Halloween, moreso going for a comfy feel rather than something overtly creepy. The skulls faces still are aggressive enough to warrant firing at, and the background features small references to popular horror tropes and stories. Both the player’s and the enemies’ animations are hand drawn, giving a more fluid style of movement than static characters would. For the ghost, I was going for a sort of billowing sheet feel, whereas the skulls were based on the chattering teeth type toys with their slow chomping as they move across the screen. Bullets do not have much flair to them, which, when considering the number that can be onscreen at once, works well in that they do not distract from the game itself.

The scripts of the game are broken into four class files, with one main one housing the game itself, one for the start menu, one for the enemy dispatchment, and one for the particle dispatchment. The main class file, spookEmUpBase, begins by importing various code libraries from inside of flash, as well as the greensock library and my particle and enemy classes. Next come my variables, setting my directional velocities as integers, my bullets and skulls as arrays, my various layers as sprites, my timers, my score and health fields, and the setup for the scrolling background. The first function, spookEmUpBase, is based mainly on the Survive game’s first function, calling to makeLevelOne to set up the game space, setting the specific values for the directional velocities, the arrays, layers, and adding event listeners for my functions that are integral to the gameplay. Next are the DownKey and UpKey functions, based on the basic player movement tutorial and setting player movement speed based on whether or not the arrow keys are pressed, as well as starting the animations within the player object for moving in each direction. The Fire function sets up an if statement wherein if the “Z” key is pressed, the spawnBullet function is run. The OnEnterFrame function adds the velocity to the player. The Framing function next adds the boundaries for the player, based again on the basic player movement tutorial and comparing the player height and width to the stage height and width to prevent the character from moving offscreen. The setupTouchLayer function places the layer on which the objects will be placed, as well as adding the event listeners for the DownKey, UpKey, and Fire functions and setting the player object’s size and spawn position. The makeLevelOne function beings by setting referential names for the objects that will be added in the function, as well as adding the background layer for the scrolling while calling the parallax function and the particle and enemy layers atop that. The player, scoreboard, and health are all added to the layer, in their desired positions, and the specific parameters for the health and score are set as well. The enemy spawning is also set up in this function, with two separate timers spawning in the enemies in a staggered manner. The spawnEnemy function holds the first of my greensock animations, as I could not properly get enemy movement to work without it. The skulls spawn in on a set y position and random x position, tweening to a set y and random x across the screen from them. These skulls have a purge event listener added to them, and the rip and success functions for win and loss conditions are checked at the end of each spawn. The spawnBullet function runs in much the same way, running the bullets on a greensock straight upwards from the player position. The purgeSkullHandler, purgeSkull, purgeBulletHandler, and purgeBullet functions serve to remove the skulls and bullets upon call, removing them from the layer as well as from the array, both based on the Shoot game’s purge functions. The parallax and align functions serve to set the scrolling background and ensure it stays in position while repeating itself. The hitTest function’s main addition was adding score and health upon a successful hit, setting up the overall system for the game. The rip and success functions were both based on the germaphobia example’s infested function, setting win and loss conditions and spawning in the correct images while removing everything from the stage, as well as offering a restart button based on my startEmUp class’s functions startGameHandler and createGame, both based on the starting functions from the Shoot game. Finishing off the spookEmUpBase class is the update function, based again on the Shoot game, but varying in the loss of health and score when the skull is below a certain y value. Due to the nature of setting it up like that, for every moment the skull is beneath the y value, it runs the function, hence why the health and score reduce by 11 and 65 respectively rather than just the 1 and 5 listed in the function.

The startEmUp class is based largely on the starting function for the shoot game, setting my desired position for the start screen and giving a means for starting the game itself.

The Enemy class is based on the Balloon class from the Shoot game, mainly differing in the addition of a full death animation in the destroy function rather than a single death frame as well as the y value after which the skull is purged.

The Particle class’s main use in the game is in the removal of the bullets and the reference from which to remove skulls as well, based on the Shoot game once again.

Overall, I feel that for a first game, it went pretty smoothly, though with plenty of frustrations along the way that are inherent to coding itself, and a couple bugs that I wanted to solve but could not quite figure out how.. I definitely am happy with how transformative it was from the examples provided, and feel a lot more confident coding and figuring out how things work within actionscript.