Michael Chaves

Professor Ellertson

GIMM 110

31 Oct 2018

Individual Game Rhetorical Analysis

Soon, home assistant robots will be the hottest thing to have in the family. AICorp., the largest company in the world, debuted the Buddy Bot last Spring, and now are hours away from loading the first shipments into their cargo trucks. On the eve of delivery, hackers hacked into AICorp.’s main server and reset the robots to exit the warehouse and destroy mankind. Only one person is in their way: Dr. Gimmstein. Equipped with his prototype laser gun, and his trusty warehouse bot, Dr. Gimmstein must not allow an evil Buddy Bot to exit the building and terrorize the city. It is time to die, AI!

“Die, AI!” is a modified version of the “Shoot” game. Like ShotGame.as, it extends the MovieClip class. It imports similar libraries, adding new libraries such as Greensock, Timer, Text, and SoundMixer. The DieAIDoc is where you will find the games main methods, this is, the methods that are executed first. The first function in this public class its constructor, where it will call the function createStartMenu() and where you see the scale mode of the game window to resize at any size. The next function creates a new start menu, add the child to the stage, and a mouse event listener to begin the intro screen handler method when the start button is clicked down on. The startIntroHandler() removes the current child, in this case, the start menu movieclip, and also removes the start button listener and all sound. It will also call the createIntro() function. When you are taken to the intro screen of the game, the screen rotates 360 degrees as if you are being taken forward in time. This is where I applied my TweenLite.to() greensock. CreateIntro() will also create a new intro screen, add it to the stage, and add the button listener. Excluding the greensock, this process continues in the startGameHandler() and createGame() functions were the game variable is a new game using the DieAIGame class.

The Particle class was kept, but modified mostly in its update() function, where I removed rotation in respect to gravity and air resistance. “Shoot” had a Balloon class file, intended to house the methods to begin and destroy its enemy, mice in balloons. I renamed it to Robot.as and the main difference you will see here is I added a health bar to my enemy. When the enemy’s current health is less than 0, it will explode.

The class that holds most game functions is my DieAIGame.as file. Lines 16-43 are where I declare most of my variables. It contains Booleans, uints, timer, textfields, strings, as well as additional sprites that separate from the original ShootGame class. Once the game object is instantiated in the DieAIDoc class, the first method being called is makeLevelOne(), which instantiates the background Sprite on line 37 to a new level one object, and adds it to the stage. My enemy spawn difficulty and delay are declared, and my laser and robots arrays are instantiated. Next my hero, the player, is also instantiated and given position on the stage before being added. The class’ update event listener is also found in the constructor, which will continue to update the action between pointing a target on the stage and the player, as well as dispatching and removing robots and lasers by inheriting the update method from the Particle class. In the constructor, the exit object is instantiated and positioned on the stage. Then you have three additional Sprites, lasersLayer, robotsLayer, and touchLayer. The touchLayer takes two arguments. To shoot the laser, the touchLayer will add and event listener, and take the argument of a mouse event, and the shootLaser() function found on line 121. The touchLayer will add another event listener which takes the movePlayer() function as the argument. Then finally added to the front of the stage, I call on the timerBox() and scorebox() functions. You will find that I have a touchLayer that take KeyboardEvent and altShooLaser arguments. This was a failed attempt at making it possible to destroy all the enemies if you hit the spacebar. I wanted my son to play, and he has special needs. He’s great with the mouse but cannot yet press specific keys on the keyboard. The space bar is the largest key, and one he hits a lot, so I wanted to make that work. I left it in the game to refer to it at another time, when I will continue to incorporate gameplay for all people to enjoy.

A little after the constructor, you will see two keyboard functions, keyPressedDown() and keyPressedUp(). This takes my booleans and determine if a key is pressed, will the moveRight or moveLeft boolean be true, and then my moveRight and moveLeft variables can be used in my movePlayer() function on line 353. The keys used to move the player left and right are A and D on the keyboard, using their respective key codes in the if statements. The next set of functions make and shoot my lasers, like the arrow functions in “Shoot,” with an exception that I do not use a barrage. On lines 137 and 138, I set my laser equal to the location of my player, with a slight offset to position my laser close enough to where the player’s gun tip is located. My robot functions for the game class follow. To have enemies coming from both sides of the stage, I use two for() loops in the makeRobot() function on line 165. To lose this level, it’s quite simple. If the robots reach the exit doorway object, it’s over. Line 189 is where I created a function to handle that. For each robot in the robots array, if they hit the exit object, then the level and sound are removed, and an instantiated game over object is added. Next in class are my purge methods, which remove the laser and robot from stage once y is equal to zero. The hitTest() method afterwards will inherit the destroy() method from the Robot class, removing the robot if hit by the laser, and adding a score of 10 to the scorebox. It will also remove the laser once the laser reaches the enemy robot. The next eight functions are new from the “Shoot” game. ScoreBox() uses my embedded font, DieAIfont(), and positions it in the upper right of the stage. TimerBox() positions to the left of the stage, but also has an event listener to begin my timerHandler() method. In the timeHandler() method, if seconds and minutes equal 0, the background and sound are removed, and createYouWin() is called. I gave this level a time of 2 minutes to complete, found on line 41. The last function in my Score and Timer section is instantiates a new WinGame() object and add child. This function is used in timerHandler(). MovePlayer() handles my keyPressed Boolean instructions, so if the player moveRight is true, then he moves forward, and if moveLeft is true, then he moves backward. The speed is faster when moving forwards, as seen on line 360, and slower when moving backward, seen on line 370. This play in speed was to make it look like the warehouse bot the player is standing on travels around the warehouse faster one way than the other, for safety reasons. The final physics element I add is keeping the player within the screen boundary. The last function of the DieAIGame class is the update(), as described in the fourth paragraph of the analysis.

The biggest challenges I faced when making this game were taming the audio, and getting the game to end when the enemy robots reach the exit object. Sometimes, the second and third audio clips from the intro start up again through gameplay. Other times, they don’t, which leaves me to wonder if there is an Animate glitch. The player is supposed to lose when the robots reach the exit. Most of the time it works, so I am also puzzled as to when sometimes it does not. I have spent an equal amount of time attempting to solve issues like these as building the game itself. That is part of the game development process, and I would not have it any other way. I plan on fixing this simple game someday and adding new levels and uploading it for free on a flash game site to see if others enjoy it as well. One thing I could have done differently was include more interaction between the enemies and the player. Perhaps the enemies could have shot back at the player, giving the player its own health bar. I originally wanted a larger, stronger enemy robot with more powerful weapons to spawn from time to time, but scrapped that idea when I was having lots of issues running my game. These ideas would have drastically improved gameplay.