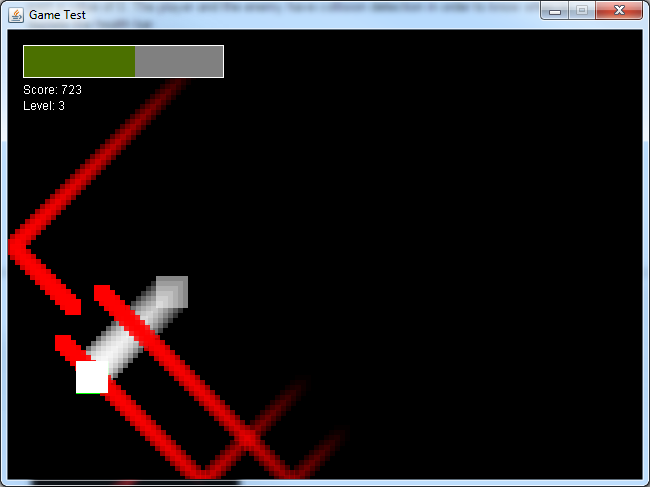
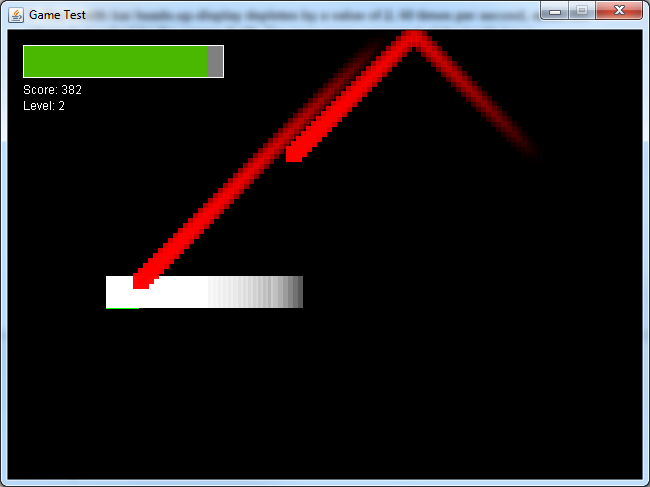
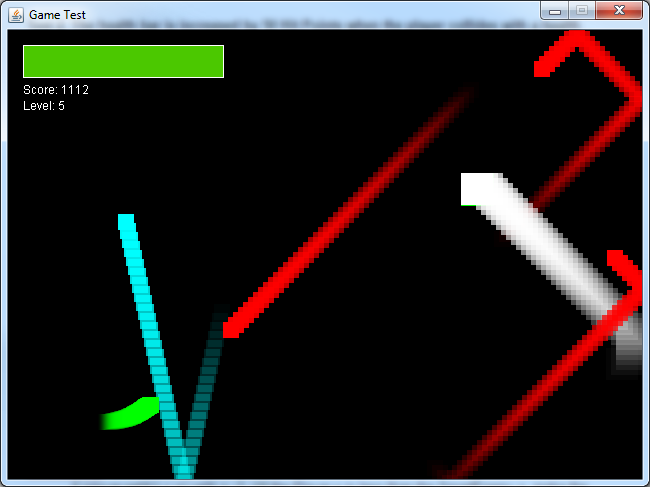
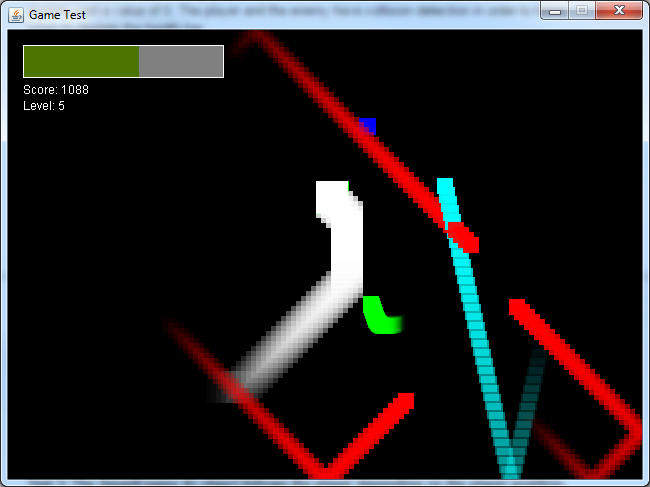
Testing Documentation | ICS4U | Ian Webster | 5/29/17

**Test 1. The health bar heads-up-display depletes by a value of 2, 60 times per second, when the player is touched by the enemy (ball).** The inner green component of the health bar is “clamped” to the surrounding grey frame. This allows the value of the health bar to never deplete past a value of 0. The player and the enemy have collision detection in order to know when to deplete the health bar. (*Before and after-enemy-collision pictures below).*



**Test 2. The health bar is increased by 50 Hit Points when the player collides with a health item.**

**Expected and Actual Results:**



**Test 3: The SmartEnemy AI object follows the player depending on the player position. The AI system took 3 separate methods to fully develop for smooth movement and player tracking.**

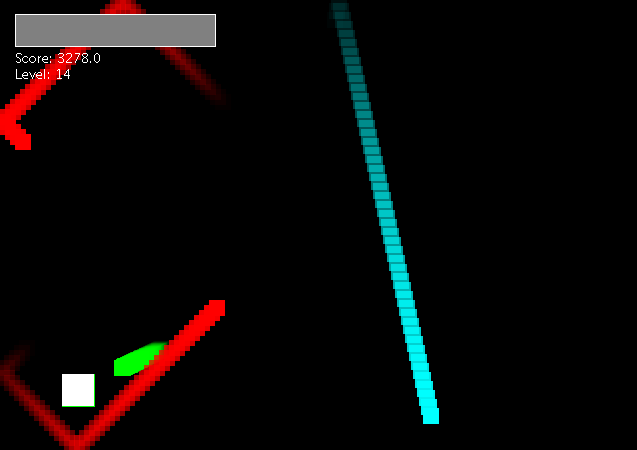
**Expected and Actual Result:**

**Method 1: Involved setting individual *if* statements within the *SmartEnemy* class. (I went through multiple tests to ensure that this method worked**

if (player.getX() > x) velX = 2; //If the Player x is more than the SmartEnemy x, make the SmartEnemy follow the Player.

if (player.getX() < x) velX = -2; //If the Player x is less than the SmartEnemy x, make the SmartEnemy follow the Player. (negative velocity because player is behind the enemy)

if (player.getX() == x) velX = 0; //If x of player is equal to x of SmartEnemy, then stop the SmartEnemy from moving.

if (player.getY() > y) velY = 2; //Same as above but for y-position of the player and SmartEnemy.

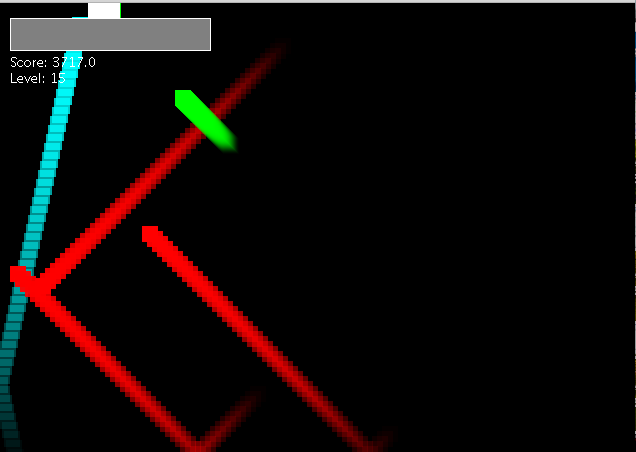
if (player.getY() < y) velY = -2;

if (player.getY() == y) velY = 0;

From the screenshot, it is visible that the *SmartEnemy* (green), is moving with very direct motion. The enemy is only able to find the most direct route to the player, and therefore moves in a very straight path. The next step in developing the AI was to enable it to move in a smoother, less rigid path.

**Method 2: Involves an algorithm to calculate the distance between the player and the enemy. Due to the limitations of integers, the initial code did not work. In order to prevent having to change all of the variables in the game to float, I simply multiplied the final velocity values by 3. The movement remains quite rigid in this method, and lacks smoothness.**

float diffX = x - player.getX() - 8;

float diffY = y - player.getY() - 8;

float distance = (float) Math.sqrt((x - player.getX())\*(x - player.getX()) + (y-player.getY()) \* (y-player.getY()));

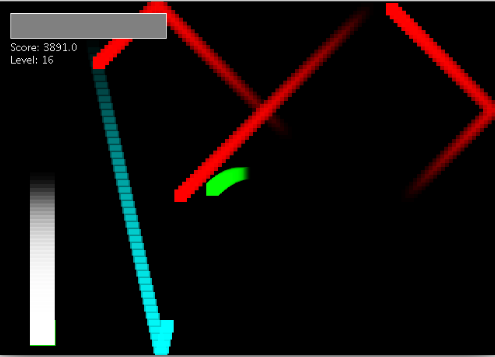
velX = (int) ((-1.0/distance) \* diffX \* **3**); //Velocity of enemy is set based on distance and direction from the player (EX: If the enemy is far from the player, it will move fast. If the enemy is close to the player, it will move slowly.)

velY = (int) ((-1.0/distance) \* diffY \* **3**);

**Method 3: Involves using the same algorithm as above (Without the \*3 in the velocity code), but every integer in the game has to be changed to a float. This allows for the most optimal precision due to the preciseness of the float values.**

float diffX = x - player.getX() - 8;

float diffY = y - player.getY() - 8;

float distance = (float) Math.sqrt((x - player.getX())\*(x - player.getX()) + (y - player.getY()) \* (y - player.getY()));

velX = (float) ((-1.0/distance) \* diffX);

velY = (float) ((-1.0/distance) \* diffY);

**Optimal precision allows for precise movement (see right).**

**Test 4: Game run test. On occasion (1 out of 10 trials), the game will freeze with the following error. Attempts have been made to implement a different game run method, but the results remain unchanged.**

**Expected Result:** Game should run normally without stalling.

**Actual Result:** Game stalls approximately 1/10 trials.

