Stephanie Cho

Technical Specifications for Chase’s Game!

Players are stick figures

Enemies are dragons

Obstacles are cool lava rocks

Player avoids lava rocks vertically and enemies both horizontal and vertical.

Player: move left, move right, and jump.

Jumping has acceleration value (not necessarily constant).

Move left will turn the direction of the player left (subtracting the acceleration value from speed).

Move right will turn the direction of the player right (adding the acceleration value from speed).

Does not need any libraries or pre written code.

Player Controls

Left Arrow (“LeftArrow”): Turns the player character left and moves

Right Arrow (“RightArrow”): Turns the player character right and moves

Space (“ ''):

F (“f”): Fires a small horizontal projectile with limited range (About 1⁄3 to 1⁄2 of the screen) in the direction that the player character is facing

D (“d”): Fires a wide, wave-like projectile vertically. There should be a limited number of these (Maybe 3 or 5, depending on game balance).

Player attributes

*playerPositions[]*: This is an array of (x,y) coordinates which shows how the player character should be drawn (global)

x: the x-coordinate of the player

y: the y-coordinate of the player

*playerSpeed*: an int that is the current speed of the player. (global)

negative when the player goes left.

positive when the player goes right

decelerate/approaches should only occur when the player character is neither going right nor left.

*playerScale*: the scale at which the player character should be drawn according to positions[] (global)

*projectiles[]*: an array which will keep track of the horizontal projectiles fired by the player character. It is initially empty. (local)

*health*: an int that keeps track of the health of the player character and is initially set to three (3) (local)

*numOfKills*: an int that starts at zero and increments every time an enemy is destroyed (local)

*numOfDodges*: an int that starts at zero and increments every time an obstacle is dodges (local)

Player-Generated Projectiles

Horizontal Projectile: Speed is twice as fast as the player’s maximum speed. Rectangular/circular collider

Vertical Wave: Speed is half as fast as the player’s maximum speed. Rectangular collider

Player methods

turnLeft()

turnRight()

fireHorizontally()

fireVertically()

getNumOfKills() // returns number of destroyed enemies

getNumOfDodges() // returns number of dodged obstacles

hit() //lose one health

fallInLava() //set health to zero

\* \*All the same behaviors, attributes, and methods apply to the second player.

Add second to all attributes and methods\*\*

Player Difficulty

All the basic actions of moving left and right should be straightforward. However, the speed variable that makes the slide motion moving left and right will extremely complicated. Additionally, the acceleration aspect of jumping will be difficult to map out and code. The horizontal and vertical shots should be easier to figure out after the speed variable is down.

Enemies

An enemy will jump if there is a large positive difference between the y-coordinate of the player character and the enemy

An enemy will approach the x-coordinate of the player character

An enemy will not walk into lava, but may accidentally jump into it

An enemy cannot be hit by obstacles

Spawning Enemies: Enemies spawn at ground level y-coordinate and x-coordinate either 0 or the right edge of the game screen. Spawn 1 enemy every so often

Enemy Attributes

*EnemyPositions[]*: This is an array of (x,y) coordinates which shows how the enemy should be drawn (global)

*x*: the x-coordinate of the enemy

*y:* the y-coordinate of the enemy

*EnemySpeed*: an int that is the current speed of the enemy (global)

negative when the enemy goes left.

positive when the enemy goes right

decelerate/approaches should only occur when enemy is neither going right nor left.

*EnemyScale:* the scale at which the enemy should be drawn according to *positions[]* (global)

*initializeEnemy:* a boolean value which is set to true. (local)

*enemySpawnFreq:* an int that determines how often the enemy will spawn. (global)

*Ω projectiles[]:* an array which will keep track of the horizontal projectiles fired by the enemy. It is initially empty. (local)

Enemy Methods

spawnEnemy()

approach()

jump()

die() // disappears and adds on to numOfKills

Enemy Difficulty

This part is more simple than the last. Spawning and moving the enemy is clear but knowing when to jump is more difficult.

Obstacles

Obstacles fall at a constant speed, although there should be some randomness in assigning this value when the individual obstacles is instantiated

Once an obstacle touches the ground (Not the bottom of the screen!) it is removed (use the splice() function) from the array

Spawning Obstacles: Obstacles spawn in waves (4-7 at a time depending on difficulty). Give slight, random intervals between each one spawning and long intervals between waves

Position: Each obstacle will spawn at y-coordinate 0 and a random x-coordinate between 0 and the width of the screen

Obstacle Attributes

*ObstaclePositions[]*: This is an array of (x,y) coordinates which shows how the obstacle should be drawn (global)

*x*: the x-coordinate of the obstacle

*y:* the y-coordinate of the obstacle

*ObstadleScale:* the scale at which the enemy should be drawn according to *positions[]* (global)

*initializeObstacle:* a boolean value which is set to true. (local)

*obstacleSpawnFreq:* an int that determines how often the obstacle will fall. (global)

*Ω projectiles[]:* an array which will keep track of the horizontal projectiles fired by the enemy. It is initially empty. (local)

Obstacle methods

spawnObstacle()

fall()

splice() //disappears and adds on to numOfDodges

Obstacle Difficulty

Obstacles shouldn’t cause any major issues. The random variable is not very complicated.

Priority

* Player (without accounting speed and acceleration, might run out of time)
* Enemy
* Obstacles
* Add in speed and acceleration factors to player’s movement

Feasibility

If it weren’t for the speed and acceleration of the players movements, I would say this game is fairly feasible. I feel that if it is not able to be finished in 3 days that these variables should be ignored. The game and objective will still function the same.

Almost all the functions are independent from one another but should be kept constant throughout each level unless told otherwise. Each character that moves in the game is slightly different. The only overlap is when an enemy or obstacle is hit then it should affect the number count variables and the health of the player.

Chunks

Day 1 and 2

* Create player with movement and jump

Day 3

* Create platforms, ground and background

Day 4

* Create player projectiles (horizontal and vertical)

Day 5

* Create obstacles and controllers

Day 6 and 7

* Create enemies and controllers

Day 8

* Create colliders

Day 9

* Create overlay

Day 10

* Level progression

Day 11 and 12

* Create more levels