I broke this assignment into 4 parts. The first two are completely new, and the last two are updated versions of older code. The two made from scratch are Sprites.java and Background.java. The two updated classes are SimpleFrameworkTemplate.java and BoundingShapes.java (an updated version of vector objects).

Sprites.java

When making Sprites.java I was attempting to finish this assignment first, and think about scalability second. Given my knowledge of how images are received in java, my only recourse was to hard code the hitboxes which lends my program to have a bit of strict usability for only this assignment. However, as I was polishing my code I was able to notice areas where scalability could be easily implemented, and commented as such within the code.

Sprites.java handles all of the Character based sprites and how they interact with both each other and with the background and background objects. This also handles all of the logic for player input and how they deal with the sprites. The last major thing is it deals with states of the Sprite for such things as falling, jumping, standing on an object, or hitting another sprite.

Variables

private final int restPos = 5; // Resting position for each sprite

private BufferedImage SpriteL; // Loads Sprites for left side movement

private BufferedImage SpriteR; // Loads Sprites for right side movement

private BufferedImage SpriteE; // Loads Sprites if non specific movement set is given

private ArrayList<BufferedImage> SpritesLeft = new ArrayList<BufferedImage>(); // Images for left side movement

private ArrayList<BufferedImage> SpritesRight = new ArrayList<BufferedImage>(); // Images for Right side movement

private ArrayList<BufferedImage> SpritesBoth = new ArrayList<BufferedImage>(); // Images for non specific movement

private Vector2f SpriteLoc = new Vector2f(0, 0); // Upper left of sprite location

private Vector2f SpriteLocL = new Vector2f(1, -1); // Bottom right of sprite location

private BoundingShapes hitbox = new BoundingShapes(); // Stores the hitbox

private AffineTransform aT = new AffineTransform(); // Used to transform the image

public int currentAnim = 4; // Determines which Sprite gets played for animation

private int sType = 0; // Checks what type of sprite is being loaded

private int lastMoved = 0; // checks what direction the sprite was last facing

private int degR = 0; // checks how many degrees are being moved

private int imgW; // Width of an image

private int imgH; // Height of an image

private int jumpC = 0; // Counter for number of jumps

private float counter = 0; // counter for switching animations

private float jumpA = 0; // Acceleration of current jump speed

private boolean Jumping = false; // Flag for if jumping

private boolean Falling = false; // Flag for if falling

public boolean Hit = false; // Flag for if hit

private boolean Pipe = false; // Flag for if on pipe

public boolean eMove = false; // Flag for if the enemy can move

Methods

* Constructor
  + Takes an int that tells it which sprite sheet to load
  + Loads said sprite sheet and updates the hitboxes accordingly
  + Also creates subImages
* Render
  + Takes Graphics g, int canvasW, int canvasH, Matrix3x3f view, boolean toggle, boolean moving
  + Renders the animations onto the screen
  + Switches animations based upon time as well as input and state of movement
  + Renders hitboxes
* Updateworld
  + boolean Left, boolean Right, boolean jump, float delta
  + Handles the logic for movement
  + Updates hitboxes
* BoundsCheck
  + BoundingShapes bounds
  + Checks if the sprite is out of bounds, and resets it to inside of the bounds
* CollisionCheck
  + BoundingShapes bounds
  + Checks if a sprite is within a circle hitbox
  + Return true or false
* ObjectCheck
  + BoundingShapes bounds
  + Checks if a sprite is within an objects rectangle hitbox
  + Return values are only to force the end of the logic
* Rotate
  + BufferedImage currentImg
  + Handles the AffineTransformation for rotation
  + Returns the sprite
* getHitBox
  + returns the Hitbox

Background.java

Background.java handles any object that I considered to be background. This includes not only the background image that my sprites exist on, but also includes any objects that I would spawn and must be static such as a pipe or a block. With this class I assumed that each given background would have the same exact dimensions which does limit how well this class will scale with future projects. It is rather light however, so have to re design it should not be a big loss.

Variables

private BufferedImage background; // the background image

private BoundingShapes bounds = new BoundingShapes(); // hitboxes for the bounds of the screen

private Vector2f SpriteLoc = new Vector2f(0, 0); // Upper left of sprite location

private Vector2f SpriteLocL = new Vector2f(1, -1); // Bottom right of sprite location

private String path; // String for the path

private int oType; // Type of background object

Methods

* Constructor
  + String location
  + Int type
* Initialize
  + Gets the png at the passed path
  + Creates proper hitboxes for the given object
* UpdateWorld
  + Updates the hitbox
* Render
  + Renders the background objects and the hitboxes if toggled
* getHitBox
  + returns the hitbox for the background

BoundingShapes

This is an extension of VectorObjects, created for this assignment specifically in order to handle hitboxes and how they work. As this is a growing amalgamation of code, it can handle a wide variety of situations, however it may not be the most efficient.

Variables

private int colorSelection; // Color Selection index

public ArrayList<Vector2f> points; // Arraylist for Points

public ArrayList<Vector2f> circle; // Arraylist for Circles

public Matrix3x3f pointmat; // Point matrix

public Vector2f center; // Center of Arraylists

private float deltar = 0; // change in rotation

public float mover = 0; // total movement of rotation

private float deltax = 0; // change in x values

private float deltay = 0; // change in y values

public float movex = 0; // total x movement

public float movey = 0; // total y movement

private Color[] COLORS = { Color.BLUE, Color.RED, Color.GREEN, Color.BLACK, Color.YELLOW }; // Array for color

// choice

Methods

* Constructor
  + Initializes the ArrayLists and Vector2fs
* All of the same Methods from Vector Object
  + addPoint was changed to AddPointLine, works exactly the same
* setpoint
  + changes point x to given Vector2f
* setCirclePoint
  + changes circlepoint x to given Vector2f
* addCirclePoint
  + adds Vector2f p to circle
* setCircleDim
  + Is identical to addCirclePoint, just used to visual differentiation
* renderHitbox
  + reinvents the wheel for drawRect, draws the hitbox in the passed viewport
* bounds
  + same as renderHitbox without the reinvention
* renderMatrix
  + renders points based upon their matrix, used for shapes with multiple points
* renderLine
  + draws a line from point 0 to point 1 for point using world coordinates
  + Used primarily for testing
* renderOval
  + draws an oval

SimpleFrameWorkTemplate

When I modified this SimpleFrameWorkTemplate my goal was to use this template to gather user input in ‘A’, ‘D’, ‘B’, and ‘Space’. It also handles creating each object and making sure they run, as well as handle the basic code of viewports and canvas creation.

Variables

private Sprites Mario = new Sprites(0); // Creates a new sprite for Mario

private Sprites Enemy = new Sprites(1); // Creates a new sprite for the goomba

private BackGround background = new BackGround("background", 0); // Creates a new background

private BackGround pipe = new BackGround("pipe", 1); // Creates a pipe Object

private Matrix3x3f viewmul; // Creates the viewport matrix

private Boolean left = false; // Flag for moving left

private Boolean right = false; // Flag for moving right

private Boolean jump = false; // Flag for jumping

private Boolean togglehitbox = false; // Flag for toggling hitboxes

private Boolean moving = false; // Flag for if moving

private float counter = 0; // Counter for delta time

Methods

* Constructor
  + Creates the basis of the Canvas
* Initialize
  + Super.intialize
* processInput
  + Checks if the input keys were pressed
* updateObjects
  + updates each object, and resets their movement after passing to the sprites
  + Checks for collision detection for the bounds, objects and other sprites for the main sprite
  + Tells the hit sprite when to reset itself
* Render
  + Basic render function, renders each object based upon the viewport matrix as well as their current status
  + Resets moving so that the next frame will be current
* Terminate
  + Super.terminate
* Main
  + Launches the App

Sprites were found:

Pipe : <https://www.spriters-resource.com/nes/supermariobros/sheet/52571/>

Goomba: <http://www.mariouniverse.com/images/sprites/gb/sml/enemies-2.gif>

Mario: <http://www.mariouniverse.com/images/sprites/gb/sml/mario-3.gif>

Background: <https://www.spriters-resource.com/game_boy_advance/wariowareincmegamicrogames/sheet/59246/>

Each sprite sheet was curated in order to grab only sprites needed using GIMP