```
""" lander.py
   lunar lander program using gameEngine
"""

import pygame, gameEngine, random

class Lander(gameEngine.SuperSprite):
   def __init__(self, scene):
      gameEngine.SuperSprite.__init__(self, scene)
      self.thrust = .1
      self.sideThrust = .05
      self.setImage("lander.gif")
      self.setAngle(90)
      self.inFlight = True

   def checkEvents(self):
      self.checkKeys()

   def checkKeys(self):
      keys = pygame.key.get_pressed()

      if keys[pygame.K_UP]:
         self.dy -= self.thrust
         self.inFlight = True

      if keys[pygame.K_LEFT]:
         self.dx += self.sideThrust
         self.inFlight = True

      if keys[pygame.K_RIGHT]:
         self.dx -= self.sideThrust
         self.inFlight = True

      self.updateVector()

class Platform(gameEngine.SuperSprite):
   def __init__(self, scene):
      gameEngine.SuperSprite.__init__(self, scene)
      self.setImage("platform.gif")
      self.reset()

   def reset(self):
      #pick random position on screen
      self.x = random.randint(0, self.screen.get_width())

      #make sure it's in bottom half of screen
      screenHeight = self.screen.get_height()
      self.y = random.randint(screenHeight/2, screenHeight)

class Game(gameEngine.Scene):
   def __init__(self):
      gameEngine.Scene.__init__(self)
      self.setCaption("Lunar Lander - arrow key to begin")
```

```
        self.lander = Lander(self)

        self.platform = Platform(self)

        self.lblInfo = gameEngine.Label()
        self.lblInfo.center = (320, 20)
        self.lblInfo.size = (300, 30)

        self.sprites = [self.lander, self.platform, self.lblInfo]
        self.gravity = .02

    def update(self):
        #add force of gravity
        if self.lander.inFlight == True:
            self.lander.addDY(self.gravity)
        self.checkLanding()

        self.updateInfo()

    def checkLanding(self):
        #check collisions
        if self.lander.collidesWith(self.platform):
            #check for good landing
            if self.lander.dx < .5:
                if self.lander.dx > -.5:
                    if self.lander.dy >= 0:
                        if self.lander.dy < 1:
                            print "nice landing"
                        else:
                            print "too much vertical velocity"
                    else:
                        badDY = self.lander.dy
                        print "must approach from top %.2f" % badDY
                else:
                    print "going too fast to left"
            else:
                print "going too fast to right"

            self.lander.dx = 0
            self.lander.dy = 0
            self.lander.updateVector()
            self.lander.inFlight = False

    def updateInfo(self):
        info = "dx: %.2f, dy: %.2f" % (self.lander.dx, self.lander.dy)
        self.lblInfo.text = info

def main():
    game = Game()
    game.start()

if __name__ == "__main__":
    main()
```

```
/* simpleGame.js
   a very basic game library for the canvas tag
   loosely based on Python gameEngine
   and Scratch
   expects an HTML5-compliant browser
   includes support for mobile browsers

   Main code and design: Andy Harris - 2011/2012
   Animation and tile elements by Tyler Mitchell
*/


//variable holding key being pressed
var currentKey = null;
var keysDown = new Array(256);
var virtKeys = false;

function Sprite(scene, imageFile, width, height){
    //core class for game engine
  this.canvas = scene.canvas;
  this.context = this.canvas.getContext("2d");
  this.image = new Image();
  this.image.src = imageFile;
  this.animation = false; // becomes Animation Class
  this.width = width;
  this.height = height;
  this.cHeight = parseInt(this.canvas.height);
  this.cWidth = parseInt(this.canvas.width);
  this.x = 200;
  this.y = 200;
  this.dx = 10;
  this.dy = 0;
  this.imgAngle = 0;
  this.moveAngle = 0;
  this.speed = 10;
  this.camera = false;
  this.visible = true;
  this.boundAction = WRAP;

  this.changeImage = function(imgFile){
    this.image.src = imgFile;
  } // end this.changeImage

  this.setImage = function(imgFile){
    //set and change image are the same thing.
    this.image.src = imgFile;
  } // end this.setImage

  this.setPosition = function(x, y){
    //position is position of center
    this.x = x;
    this.y = y;
```

```
} // end setPosition function

this.setX = function (nx){ this.x = nx; }
this.setY = function (ny){ this.y = ny; }
this.setChangeX = function (ndx){ this.dx = ndx; }
this.setChangeY = function (ndy){ this.dx = ndx; }
this.setDX = function(newDX){
  this.dx = newDX;
}
this.setDY = function(newDY){
  this.dy = newDY;
}
this.changeXby = function(tdx){ this.x += tdx};
this.changeYby = function(tdy){ this.y += tdy};
this.hide = function(){this.visible = false; }
this.show = function(){this.visible = true; }

this.draw = function(){
  //draw self on canvas;
  //intended only to be called from update, should never
  //need to be deliberately called by user
  ctx = this.context;

  ctx.save();
     //The following lines are for Tyler's code. Removed for now
     //if( this.camera ){ ctx.translate(this.x - this.camera.cameraOffsetX, this.y - this.camera.cameraOffsetY); }
     //else{ ctx.translate(this.x, this.y); }

    //transform element
    ctx.translate(this.x, this.y);
    ctx.rotate(this.imgAngle);

    //draw image with center on origin
      if( this.animation != false ){
        this.animation.drawFrame(ctx);
      }
      else{
        ctx.drawImage(this.image,
          0 - (this.width / 2),
          0 - (this.height / 2),
          this.width, this.height);
      }
  ctx.restore();

} // end draw function

this.update = function(){
  this.x += this.dx;
  this.y += this.dy;
  this.checkBounds();
  if (this.visible){
    this.draw();
  } // end if
} // end update
```

```
this.setBoundAction = function(action){
  this.boundAction = action;
} // end setBoundAction

this.checkBounds = function(){
  //behavior changes based on
  //boundAction property

  camX = 0;
  camY = 0;
  if(this.camera){ camX = this.camera.cameraOffsetX; camY = this.camera.cameraOffsetY; }
  rightBorder = this.cWidth + camX;
  leftBorder = camX;
  topBorder = camY;
  bottomBorder = this.cHeight + camY;

  offRight = false;
  offLeft = false;
  offTop = false;
  offBottom = false;

  if (this.x > rightBorder){
    offRight = true;
  }

  if (this.x < leftBorder){
    offLeft = true;
  }

  if (this.y > bottomBorder){
    offBottom = true;
  }

  if (this.y < 0){
    offTop = true;
  }

  if (this.boundAction == WRAP){
    if (offRight){
      this.x = leftBorder;
    } // end if

    if (offBottom){
      this.y = topBorder;
    } // end if

    if (offLeft){
      this.x = rightBorder;
    } // end if

    if (offTop){
      this.y = bottomBorder;
    }
```

```
   } else if (this.boundAction == BOUNCE){
     if (offTop || offBottom){
       this.dy *= -1;
       this.calcSpeedAngle();
       this.imgAngle = this.moveAngle;
     }

     if (offLeft || offRight){
       this.dx *= -1;
       this.calcSpeedAngle();
       this.imgAngle = this.moveAngle;
     }

   } else if (this.boundAction == STOP){
     if (offLeft || offRight || offTop || offBottom){
       this.setSpeed(0);
     }
   } else if (this.boundAction == DIE){
     if (offLeft || offRight || offTop || offBottom){
       this.hide();
       this.setSpeed(0);
     }

   } else {
     //keep on going forever
   }
  } // end checkbounds

  this.loadAnimation = function (imgWidth, imgHeight, cellWidth, cellHeight){
    this.animation = new Animation(this.image, imgWidth, imgHeight, cellWidth, cellHeight);
       this.animation.setup();
  }

  //animation methods
  this.generateAnimationCycles = function(slicingFlag, framesArray){
    //Default: assume each row is a cycle and give them names Cycle1, Cycle2, ... , CycleN
       //SINGLE_ROW: all the sprites are in one row on the sheet, the second parameter is either a number saying each
cycle is that many frames or a list of how many frames each cycle is
       //SINGLE_COLUMN: all the sprites are in one column on the sheet, the second parameter is either a number
saying each cycle is that many frames or a list of how many frames each cycle is
       //VARIABLE_LENGTH: How many frames are in each cycle. framesArray must be defined.
       cWidth = this.animation.cellWidth;
       cHeight = this.animation.cellHeight;
       iWidth = this.animation.imgWidth;
       iHeight = this.animation.imgHeight;
       numCycles = 0;
       nextStartingFrame = 0;
         if(typeof framesArray == "number" || typeof slicingFlag == "undefined"){
           if( slicingFlag == SINGLE_COLUMN ){ numCycles = (iHeight/cHeight)/framesArray; }
              else if( typeof slicingFlag == "undefined" ){ numCycles = (iHeight/cHeight); framesArray = iWidth/cWidth;
}
           else{ numCycles = (iWidth/cWidth)/framesArray; }
              for(i = 0; i < numCycles; i++){
                cycleName = "cycle" + (i+1);
```

```
            this.specifyCycle(cycleName, i*framesArray, framesArray);
          }
        }
        else{
          numCycles = framesArray.length;
          for(i = 0; i < numCycles; i++){
            cycleName = "cycle" + (i+1);
            this.specifyCycle(cycleName, nextStartingFrame, framesArray[i]);
            nextStartingFrame += framesArray[i];
          }
        }
      this.setCurrentCycle("cycle1");
  }

  this.renameCycles = function(cycleNames){ this.animation.renameCycles(cycleNames); }
  this.specifyCycle = function(cycleName, startingCell, frames){ this.animation.addCycle(cycleName, startingCell,
frames); }
  this.specifyState = function(stateName, cellName){ this.animation.addCycle(stateName, cellName, 1); }
  this.setCurrentCycle = function(cycleName){ this.animation.setCycle(cycleName); }
  this.pauseAnimation = function(){ this.animation.pause(); }
  this.playAnimation = function(){ this.animation.play(); }
  this.resetAnimation = function(){ this.animation.reset(); }
  this.setAnimationSpeed = function(speed){ this.animation.setAnimationSpeed(speed); }

  this.calcVector = function(){
    //used throughout speed / angle calculations to
    //recalculate dx and dy based on speed and angle
    this.dx = this.speed * Math.cos(this.moveAngle);
    this.dy = this.speed * Math.sin(this.moveAngle);
  } // end calcVector

  this.calcSpeedAngle = function(){
    //opposite of calcVector:
    //sets speed and moveAngle based on dx, dy
    this.speed = Math.sqrt((this.dx * this.dx) + (this.dy * this.dy));
    this.moveAngle = Math.atan2(this.dy, this.dx);
  }

  this.setSpeed = function(speed){
    this.speed = speed;
    this.calcVector();
  } // end setSpeed

  this.getSpeed = function(){
    //calculate speed based on current dx and dy
    speed = Math.sqrt((this.dx * this.dx) + (this.dy * this.dy));
    return speed;
  } // end getSpeed

  this.changeSpeedBy = function(diff){
    this.speed += diff;
    this.calcVector();
  } // end changeSpeedBy
```

```
this.setImgAngle = function(degrees){
   //offset degrees by 90
   degrees = degrees - 90;
  //convert degrees to radians
  this.imgAngle = degrees * Math.PI / 180;
} // end setImgAngle

this.getImgAngle = function(){
  //imgAngle is stored in radians.
  //return it in degrees
  //don't forget we offset the angle by 90 degrees
  return (this.imgAngle * 180 / Math.PI) + 90;
}

this.changeImgAngleBy = function(degrees){
  rad = degrees * Math.PI / 180;
  this.imgAngle += rad;
} // end changeImgAngle

this.setMoveAngle = function(degrees){
  //take movement angle in degrees
  // offset degrees by 90
  degrees = degrees - 90
  //convert to radians
  this.moveAngle = degrees * Math.PI / 180;
  this.calcVector();
} // end setMoveAngle

this.changeMoveAngleBy = function(degrees){
  //convert diff to radians
  diffRad = degrees * Math.PI / 180;
  //add radian diff to moveAngle
  this.moveAngle += diffRad;
  this.calcVector();
} // end changeMoveAngleBy

//convenience functions combine move and img angles
this.setAngle = function(degrees){
  this.setMoveAngle(degrees);
  this.setImgAngle(degrees);
} // end setAngle

this.changeAngleBy = function(degrees){
    this.changeMoveAngleBy(degrees);
    this.changeImgAngleBy(degrees);
} // end changeAngleBy

this.turnBy = function(degrees){
  //same as changeAngleBy
  this.changeAngleBy(degrees);
}

this.addVector = function(degrees, thrust){
  //Modify the current motion vector by adding a new vector to it.
```

```
  //offset angle by 90 degrees
  degrees -= 90;
  //input angle is in degrees - convert to radians
  angle = degrees * Math.PI / 180;

  //calculate dx and dy
  newDX = thrust * Math.cos(angle);
  newDY = thrust * Math.sin(angle);
  this.dx += newDX;
  this.dy += newDY;

  //ensure speed and angle are updated
  this.calcSpeedAngle();
} // end addVector

this.collidesWith = function(sprite){
  //check for collision with another sprite

  //collisions only activated when both sprites are visible
  collision = false;
  if (this.visible){
    if (sprite.visible){
      //define borders
      myLeft = this.x;
      myRight = this.x + this.width;
      myTop = this.y;
      myBottom = this.y + this.height;
      otherLeft = sprite.x;
      otherRight = sprite.x + sprite.width;
      otherTop = sprite.y;
      otherBottom = sprite.y + sprite.height;

      //assume collision
      collision = true;

      //determine non-colliding states
      if ((myBottom < otherTop) ||
         (myTop > otherBottom) ||
         (myRight < otherLeft) ||
         (myLeft > otherRight)) {
           collision = false;
      } // end if

    } // end 'other visible' if
  } // end 'I'm visible' if

  return collision;
} // end collidesWith

this.distanceTo = function(sprite){
    //get centers of sprites
    myX = this.x + (this.width/2);
    myY = this.y + (this.height/2);
```

```
    otherX = sprite.x + (sprite.width/2);
    otherY = sprite.y + (sprite.height/2);
    diffX = myX - otherX;
    diffY = myY - otherY;
    dist = Math.sqrt((diffX * diffX) + (diffY * diffY));
    return dist;
  } // end distanceTo

  this.angleTo = function(sprite){
    //get centers of sprites
    myX = this.x + (this.width/2);
    myY = this.y + (this.height/2);
    otherX = sprite.x + (sprite.width/2);
    otherY = sprite.y + (sprite.height/2);

    //calculate difference
    diffX = myX - otherX;
    diffY = myY - otherY;
    radians = Math.atan2(diffY, diffX);
    degrees = radians * 180 / Math.PI;
    //degrees are offset
    degrees += 90;
    return degrees;
  } // end angleTo

  this.setCameraRelative = function( cam ){ this.camera = cam; }

  this.report = function(){
    //used only for debugging. Requires browser with JS console
    console.log ("x: " + this.x + ", y: " + this.y + ", dx: "
              + this.dx + ", dy: " + this.dy
              + ", speed: "  + this.speed
              + ", angle: " + this.moveAngle);
  } // end report
} // end Sprite class def

function Scene(){
  //Scene that encapsulates the animation background

  //determine if it's a touchscreen device
  this.touchable = 'createTouch' in document;

  //dynamically create a canvas element
  this.canvas = document.createElement("canvas");
  this.canvas.style.backgroundColor = "yellow";
  document.body.appendChild(this.canvas);
  this.context = this.canvas.getContext("2d");

  this.clear = function(){
    this.context.clearRect(0, 0, this.width, this.height);
  }

  this.start = function(){
    //set up keyboard reader if not a touch screen.
```

```
  if (!this.touchable){
    this.initKeys();
    document.onkeydown = this.updateKeys;
    document.onkeyup = this.clearKeys;
  } // end if
  this.intID = setInterval(localUpdate, 50);
  document.onmousemove = this.updateMousePos;
}

this.stop = function(){
  clearInterval(this.intID);
}

this.updateKeys = function(e){
  //set current key
  currentKey = e.keyCode;
  console.log(e.keyCode);
  keysDown[e.keyCode] = true;
} // end updateKeys

this.clearKeys = function(e){
  currentKey = null;
  keysDown[e.keyCode] = false;
} // end clearKeys

this.initKeys = function(){
  //initialize keys array to all false
  for (keyNum = 0; keyNum < 256; keyNum++){
        keysDown[keyNum] = false;
  } // end for
} // end initKeys

this.setSizePos = function(height, width, top, left){
  //convenience function.  Cals setSize and setPos
  this.setSize(height, width);
  this.setPos(top, left);
} // end setSizePos

this.setSize = function(width, height){
  //set the width and height of the canvas in pixels
  this.width = width;
  this.height = height;
  this.canvas.width = this.width;
  this.canvas.height = this.height;
} // end setSize

this.setPos = function(left, top){
  //set the left and top position of the canvas
  //offset from the page
  this.left = left;
  this.top = top;

  //CSS3 transform to move elements.
  //Cross-browser compatibility would be awesome, guys...
```

```
    this.canvas.style.MozTransform = "translate(" + left + "px, " + top + "px)";
    this.canvas.style.WebkitTransform = "translate(" + left + "px, " + top + "px)";
    this.canvas.style.OTransform = "translate(" + left + "px, " + top + "px)";

  } // end setPos

  this.setBG = function(color){
    this.canvas.style.backgroundColor = color;
  } // end this.setBG

  this.updateMousePos = function(e){
    this.mouseX = e.pageX;
    this.mouseY = e.pageY;
  } // end function

  this.hideCursor = function(){
    this.canvas.style.cursor = "none";
  }

  this.showCursor = function(){
    this.canvas.style.cursor = "default";
  }

  this.getMouseX = function(){
    //incorporate offset for canvas position
    return document.mouseX - this.left;
  }

  this.getMouseY = function(){
    //incorporate offset for canvas position
    return document.mouseY - this.top;
  }

  this.hide = function(){
    this.canvas.style.display = "none";
  }

  this.show = function(){
    this.canvas.style.display = "block";
  }

  this.setSize(800, 600);
  this.setPos(10, 10);
  this.setBG("lightgray");

} // end Scene class def

function Sound(src){
  //sound effect class
  //builds a sound effect based on a url
  //may need both ogg and mp3.
  this.snd = document.createElement("audio");
  this.snd.src = src;
  //preload sounds if possible (won't work on IOS)
```

```
  this.snd.setAttribute("preload", "auto");
  //hide controls for now
  this.snd.setAttribute("controls", "none");
  this.snd.style.display = "none";
  //attach to document so controls will show when needed
  document.body.appendChild(this.snd);

  this.play = function(){
    this.snd.play();
  } // end play function

  this.showControls = function(){
    //generally not needed.
    //crude hack for IOS
    this.snd.setAttribute("controls", "controls");
    this.snd.style.display = "block";
  } // end showControls

} // end sound class def

function Joy(){
  //virtual joystick for ipad
  //console.log("joystick created");
  //when activated, document will have the following properties
  //mouseX, mouseY: touch read as mouse input
  //diffX, diffY: touch motion read as a joystick input
  //if virtKeys is set true
  //joystick inputs will be read as arrow keys

  //properties
  SENSITIVITY = 50;
  diffX = 0;
  diffY = 0;
  var touches = [];
  var startX;
  var startY;

  //define event handlers
  this.onTouchStart = function(event){
    result = "touch: ";
    touches = event.touches;
    startX = touches[0].screenX;
    startY = touches[0].screenY;
    result += "x: " + startX + ", y: " + startY;
    //define mouse position based on touch position
    this.mouseX = startX;
    this.mouseY = startY;
    //console.log(result);
  } // end onTouchStart

 this.onTouchMove = function(event){
    result = "move: "
    event.preventDefault();
    touches = event.touches;
```

```
  //map touch position to mouse position
  this.mouseX = touches[0].screenX;
  this.mouseY = touches[0].screenY;
  this.diffX = touches[0].screenX - startX;
  this.diffY = touches[0].screenY - startY;
  result += "dx: " + this.diffX + ", dy: " + this.diffY;

  //manage virtual keys if enabled
  if (virtKeys){
    THRESHHOLD = 10;
    if (this.diffX > THRESHHOLD){
      keysDown[K_RIGHT] = true;
    } else {
      keysDown[K_RIGHT] = false;
    } // end if

    if (this.diffX < -THRESHHOLD){
      keysDown[K_LEFT] = true;
    } else {
      keysDown[K_LEFT] = false;
    } // end if

    if (this.diffY > THRESHHOLD){
      keysDown[K_DOWN] = true;
    } else {
      keysDown[K_DOWN] = false;
    } // end if

    if (this.diffY < -THRESHHOLD){
      keysDown[K_UP] = true;
    } else {
      keysDown[K_UP] = false;
    } // end if

  } // end if

} // end onTouchMove

this.onTouchEnd = function(event){
  result = "no touch";
  touches = event.touches;
  this.diffX = 0;
  this.diffY = 0;

  //turn off all virtual keys
  if (virtKeys){
    keysDown[K_LEFT] = false;
    keysDown[K_RIGHT] = false;
    keysDown[K_UP] = false;
    keysDown[K_DOWN] = false;
  }
} // end onTouchEnd

// add utility methods to retrieve various attributes
```

```
  this.getDiffX = function(){
    //compensate for possible null
    if (document.diffX == null){
      document.diffX = 0;
    } // end if
    return document.diffX;
  }
  this.getDiffY = function(){
    //compensate for possible null
    if (document.diffY == null){
      document.diffY = 0;
    } // end if
    return document.diffY;
  }

  this.getMouseX = function(){return document.mouseX;}
  this.getMouseY = function(){return document.mouseY;}

  //add event handlers if appropriate
  touchable = 'createTouch' in document;
  if (touchable){
    document.addEventListener('touchstart', this.onTouchStart, false);
    document.addEventListener('touchmove', this.onTouchMove, false);
    document.addEventListener('touchend', this.onTouchEnd, false);
  } // end if

} // end joy class def

function Accel(){
  //virtual accelerometer

  //properties
  var ax;
  var ay;
  var az;

  var rotX;
  var rotY;
  var rotZ;

  if (window.DeviceMotionEvent==undefined){
      console.log("This program requires an accelerometer");
  } else {
    window.ondevicemotion = function(event){
      this.ax = event.accelerationIncludingGravity.x;
      this.ay = event.accelerationIncludingGravity.y;
      this.az = event.accelerationIncludingGravity.z;

      rotation = event.rotationRate;
      if (rotation != null){
        this.rotX = Math.round(rotation.alpha);
        this.rotY = Math.round(rotation.beta);
        this.rotZ = Math.round(rotation.gamma);
      } // end if
```

```
    } // end event handler
  } // end if

  //return values with utility methods

  this.getAX = function(){
    if (window.ax == null){
      window.ax = 0;
    }
    return window.ax;
  } // end getAx

  this.getAY = function(){
    if (window.ay == null){
      window.ay = 0;
    }
    return window.ay;
  } // end getAx

  this.getAZ = function(){
    if (window.az == null){
      window.az = 0;
    }
    return window.az;
  } // end getAx

  this.getRotX = function(){return rotX;}
  this.getRotY = function(){return rotY;}
  this.getRotZ = function(){return rotZ;}

} // end class def

function Timer(){
  //simple timer

    this.reset = function(){
        this.date = new Date();
        this.startTime = this.date.getTime();
        this.elapsedTime = 0;
    } // end reset


    this.getCurrentTime = function(){
        this.date = new Date();
        return this.date.getTime();
    } // end getCurrentTime

    this.getElapsedTime = function(){
        current = this.getCurrentTime();
        return (current - this.startTime) / 1000;
    } // end getElapsedTime

    //make alias functions for animations...
    this.start = this.reset;
```

```
    this.getTimeElapsed = this.getElapsedTime;

    this.reset();
} // end Timer def

var AnimTimer = function()
{
  //special timer for animations
  this.date = new Date();
  this.lastTime = 0;
  this.currentTime = 0;

  this.start = function(){
       this.currentTime = Date.now();
  }

  this.reset = function(){
       this.currentTime = Date.now();
  }

  this.getTimeElapsed = function(){
    this.lastTime = this.currentTime;
       this.currentTime = Date.now();
       return (this.currentTime - this.lastTime);
  }
}


function localUpdate(){
    //will be called once per frame
    //calls the update function defined by
    //the user
    update();
} // end localUpdate

/* tile and event stuff added by Tyler */

function GameButton(label){
    /*
       This object creates a button that can be sized
       and positioned wherever you wish. The label will
       be displayed, but can be complete HTML (including
       an image tag if you wish.)  Use isClicked() to
       get the current status of the button (true or false.)
       Responds to touch events on mobile devices.
    */

    this.clicked = false;
    this.button = document.createElement("button");
    this.button.setAttribute("type", "button");
    this.button.innerHTML = label;
    this.button.style.position = "absolute";
    this.button.style.left = "0px";
    this.button.style.top = "0px";
```

```
    this.button.onmousedown = function(){
        this.clicked = true;
    } // end mousedown

    this.button.ontouchstart = function(){
        this.clicked = true;
    } // end touchstart

    this.button.onmouseup = function(){
        this.clicked = false;
    } // end onmouseup

    this.isClicked = function(){
        return this.button.clicked;
    } // end isClicked

    this.setPos = function(left, top){
        this.button.style.left = left + "px";
        this.button.style.top = top + "px";
    } // end setPos

    this.setPosition = function(left, top){
        //utility alias for setPos
        this.setPos(left, top);
    }

    this.setSize = function(width, height){
        this.button.style.width = width + "px";
        this.button.style.height = height + "px";
    } // end setSize

    document.body.appendChild(this.button);
} // end gameButton class def

function Animation(spriteSheet, imgWidth, imgHeight, cellWidth, cellHeight){
  //Animation class by Tyler Mitchell
  //for simplicity, all cells must be the same width and height combination
  this.sheet = spriteSheet;
  this.imgWidth = imgWidth;
  this.imgHeight = imgHeight;
  this.cellWidth = cellWidth;
  this.cellHeight = cellHeight;
  this.animationLength = 1000;
  this.changeLength = false;
  this.cycles = new Array();
  this.currentCycleName = "";
  this.currentCycle = null;
  this.cyclePlaySettings = new Array( PLAY_LOOP, PLAY_LOOP, PLAY_LOOP, PLAY_LOOP );
  this.changeAnimation = false;
  this.timer = new AnimTimer();
  this.framesPerRow = 0;
  this.framesPerColumn = 0;
  this.totalCycleTime = 0;
```

```
  this.fps = 0;
  this.isPaused = false;

  this.setup = function(){
    this.timer.start();
    this.framesPerRow = this.imgWidth / this.cellWidth;
    this.framesPerColumn = this.imgHeight / this.cellHeight;
  }

  this.addCycle = function(cycleName, startingCell, frames){
    cycle = new Array(cycleName, startingCell, frames);
       this.cycles.push(cycle);
  }

  this.drawFrame = function(ctx){//most of the math in this function could be done only once if we want to make it
faster
    this.fps += 1;
    if( !this.isPaused ){ this.totalCycleTime += this.timer.getTimeElapsed(); }
    if(this.changeAnimation == true){// find the correct animation in
         for( i = 0; i < this.cycles.length; i++ ){
           if( this.cycles[i][0] == this.currentCycleName ){
                this.currentCycle = this.cycles[i];
              }
         }
       }
       if( this.changeAnimation || this.changeLength ){
         this.frameDelta = this.animationLength / this.currentCycle[2]; // this will be how much time should pass at a
minimum before switching to the next frame
         this.changeAnimation = false;
         this.changeLength = false;
         this.fps = 0;
       }
       //console.log("Cycletime: " + this.totalCycleTime);
       //console.log("Frame Delta: " + this.frameDelta);
       //I think the following line is the trouble spot
       //currentFrame = Math.floor( (this.totalCycleTime % this.animationLength) / this.frameDelta );
       elTime = this.totalCycleTime % this.animationLength;
       currentFrame = Math.floor(elTime / this.frameDelta);
       console.log(elTime);

       //document.getElementById("FPS").innerHTML = this.animationLength;//for debugging
       row = Math.floor( ( this.currentCycle[1] + currentFrame ) / this.framesPerRow );
       col = (this.currentCycle[1] + currentFrame) - (row * Math.floor(this.imgWidth / this.cellWidth));
       frameY = row * this.cellHeight;
       frameX = col * this.cellWidth;

       ctx.drawImage(this.sheet, frameX, frameY, this.cellWidth, this.cellHeight, 0 - (this.cellWidth / 2), 0 -
(this.cellHeight / 2), this.cellWidth, this.cellHeight);
  }


  this.setCycle = function(cycleName){
    this.currentCycleName = cycleName;
       this.changeAnimation = true;
```

```
      this.totalCycleTime = 0;
  }

  this.renameCycles = function(cycleNames){
    for(i = 0; i < cycleNames.length; i++){
        number = parseInt( this.cycles[i][0].slice(5) );
        if(this.currentCycleName == this.cycles[i][0]){ this.currentCycleName = cycleNames[number-1]; }
        this.cycles[i][0] = cycleNames[number-1];
      }
  }

  this.play = function(){
    this.isPaused = false;
      this.timer.reset();
  }

  this.pause = function(){
    this.isPaused = true;
  }

  this.reset = function(){
    this.totalCycleTime = 0;
      this.timer.reset();
  }

  this.setAnimationSpeed = function( animLength ){//animLength is in milliseconds
    if( animLength <= 50 ){ animLength = 50; }
      this.animationLength = animLength;
      this.changeLength = true;
  }

}// end of Animation class

/*
The following classes are experimental, and are not yet
tested for widespread use.
They provide tile-based worlds and a camera
All are by Tyler Mitchell



function Camera(scene){
  this.canvas = scene.canvas;
  this.context = this.canvas.getContext("2d");
  this.cHeight = parseInt(this.canvas.height);
  this.cWidth = parseInt(this.canvas.width);
  this.cameraOffsetX = 0;
  this.cameraOffsetY = 0;
  this.target = false;
  this.waitX = 0;
  this.waitY = 0;
  this.focalPointX = 0;
  this.focalPointY = 0;
```

```
  this.moveCamera = function(x, y){
    this.cameraOffsetX += x;
      this.cameraOffsetY += y;
  }

  this.followSprite = function(sprite, waitX, waitY){// wait rectangle currently not working
      this.target = sprite;
      if( typeof waitX != "undefined" ){
        this.waitX = waitX;
        this.waitY = waitY;
      }
  }

  this.update = function(){
    // center the camera on the sprite
      this.focalPointX = this.cameraOffsetX + this.cWidth/2;
      this.focalPointY = this.cameraOffsetY + this.cHeight/2;
      if(this.target && !this.checkFocusBounds() ){
        this.cameraOffsetX = this.target.x + (this.target.width/2) - (this.cWidth/2) + this.waitX;
        this.cameraOffsetY = this.target.y + (this.target.height/2) - (this.cHeight/2) + this.waitY;
      }
  }

  this.checkFocusBounds = function(){
    centerX = this.target.x + (this.target.width/2);
      centerY = this.target.y + (this.target.height/2);
    if( Math.abs(this.focalPointX - centerX) >= this.waitX ){ return false; }
      if( Math.abs(this.focalPointY - centerY) >= this.waitY ){ return false; }
      else{ return true; }
  }
}

function Tile( mapX, mapY, x, y, type ){
  this.x = x;
  this.y = y;
  this.mapX = mapX;
  this.mapY = mapY;
  this.isCollidable = false;
  this.collisionCallback = false;
  this.type = type;
  this.isAnimated = false;
  this.isCollidable = false;
  this.isClickable = false;
  this.clickCallback = false;
  this.animationPlaying = false;

  this.setCollision = function( callBack ){
    this.collisionCallback = callBack;
      this.isCollidable = true;
  }

  this.setAnimation = function(){
    this.isAnimated = true;
  }
```

```
  this.setClick = function( callBack ){
    this.isClickable = true;
      this.clickCallback = callBack;
  }

  this.checkCollision = function( sprite, w, h ){
    shw = sprite.width/2;
      shh = sprite.height/2;
      scx = sprite.x + shw;
      scy = sprite.y + shh;
      thw = w/2;
      thh = h/2;
      tcx = this.x + thw;
      tcy = this.y + thh;
    if( Math.abs( scx - tcx ) < (thw + shw) ){
        if( Math.abs( scy - tcy ) < (thh + shh) ){
          this.collisionCallback(this);
        }
      }
  }
}

function TileMap(scene){
  this.tileSheet = new Image();
  this.tiles = new Array();
  this.symbolImageMap = new Array();
  this.tileAnimations = new Array();
  this.specificTileAnimations = new Array();
  this.mapData = false;
  this.tileWidth = 0;
  this.tileHeight = 0;
  this.sheetWidth = 0;
  this.sheetHeight = 0;
  this.camera = new Camera(scene);

  this.loadTileSheet = function(tileWidth, tileHeight, sheetWidth, sheetHeight, tileSheet, tileSymbols){
    this.tileSheet.src = tileSheet;
      this.tileWidth = tileWidth;
      this.tileHeight = tileHeight;
      this.SheetWidth = sheetWidth;
      this.SheetHeight = sheetHeight;
      numRows = Math.floor(this.SheetWidth/this.tileWidth);
      numCols = Math.floor(this.SheetHeight/this.tileHeight);
      for(i = 0; i < numRows; i++){
        for(j = 0; j < numCols; j++){
          if( (i*numCols)+j < tileSymbols.length ){
            this.symbolImageMap[(i*numCols)+j] = new Array( j*this.tileWidth, i*this.tileHeight,
tileSymbols[(i*numCols)+j] );
              }
        }
      }
  }
```

```
  this.loadMapData  = function(mapArray){// mapArray must be a 2-dimensional Array
    this.mapData = new Array();

    for(i = 0; i < mapArray.length; i++){
        this.mapData.push( new Array() );
        temp = new Array();
        for(j = 0; j < mapArray[i].length; j++){
          k = 0;
            notConverted = true;
          while( notConverted && k < this.symbolImageMap.length ){
              if( mapArray[i][j] == this.symbolImageMap[k][2]){ this.mapData[i][j] = k; notConverted = false; } //
convert tile symbols to integers for faster comparisons
              k++;
            }
            temp[j] = new Tile(j, i, j*this.tileWidth, i*this.tileHeight, k);// k = tile type
        }
        this.tiles.push(temp)
      }
  }

  this.drawMap = function(){//this could be WAY faster
    this.camera.update();
    ctx = this.camera.context;
    for(i = 0; i < this.mapData.length; i++){//for each row
        for(j = 0; j < this.mapData[i].length; j++){ //for each column of each row
          drawX = this.tiles[i][j].x - this.camera.cameraOffsetX;
            drawY = this.tiles[i][j].y - this.camera.cameraOffsetY;
            if( 0 < drawX < this.camera.cWidth && 0 < drawY < this.camera.cHeight ){//don't draw any tiles that will
not be in the camera's view
            ctx.save();
            sheetX = this.symbolImageMap[ this.mapData[i][j] ][0];
              sheetY = this.symbolImageMap[ this.mapData[i][j] ][1];
              ctx.translate(drawX, drawY);
              if( this.tiles[i][j].animationPlaying ){ this.drawTileAnimation(this.tiles[i][j], ctx); }
              else{
                  ctx.drawImage(this.tileSheet, sheetX, sheetY, this.tileWidth, this.tileHeight, 0, 0, this.tileWidth,
this.tileHeight);
                  ctx.restore();
              }
            }
        }
      }
  }

  this.addTileCollision = function( collisionCallback, typeOrX, y ){// accept tile type or coordinates
    if( typeof y == "undefined" ){ // then the first argument is a tile type
        for( i = 0; i < this.tiles.length; i++ ){
          for( j = 0; j < this.tiles[i].length; j++ ){
              if( this.tiles[i][j].type == typeOrX ){
                this.tiles[i][j].setCollision( collisionCallback );
              }//end if
            }//end for j
        }//end for i
      }//end if type
```

```
    else{ // then a tile coordinate was passed in
      this.tiles[typeOrX][y].setCollision( collisionCallback );
    }
  }

  this.loadCollisionMap = function( collisionMap ){// tile Symbol and collision Callback - - NOTE: This function will overwrite specific Collision Callbacks
    //convert collisionMap symbols to their associated integers
    for( l = 0; l < collisionMap.length; l++ ){
      c = 0;
      notConverted = true;
      while( c < this.symbolImageMap.length && notConverted ){
        if( this.symbolImageMap[c][2] == collisionMap[l][0] ){
          collisionMap[l][0] = c+1;
            notConverted = false;
        }
          c++;
      }
    }
    //set collision callback for each tile
   for( i = 0; i < this.tiles.length; i++ ){
      for( j = 0; j < this.tiles[i].length; j++ ){
        k = 0;
          notAssigned = true;
        while( k < collisionMap.length && notAssigned ){
          if( this.tiles[i][j].type == collisionMap[k][0] ){
              this.tiles[i][j].setCollision( collisionMap[k][1] );
                notAssigned = false;
            }
            k++;
          }
      }
    }
  }

  this.mapScroll = function( dx, dy ){ this.camera.moveCamera(dx, dy); }
  this.cameraFollowSprite = function(sprite, waitX, waitY){ this.camera.followSprite(sprite, waitX, waitY); }

  this.loadZOrderMap = function( zMap ){}

  this.addTileAnimation = function( imgWidth, imgHeight, cellWidth, cellHeight, tileName, animSheet ){
   animation = new Animation(animSheet, imgWidth, imgHeight, cellWidth, cellHeight);
    animation.setup();
    for( i = 0; i < this.symbolImageMap.length; i++ ){ // find the tile number that corresponds to the tile name.
          if( this.symbolImageMap[i][2] = tileName ){
                this.tileAnimations[i] = animation;// i = tileNumber, animation
          }
    }
  }

  this.addSpecificTileAnimation = function(imgWidth, imgHeight, cellWidth, cellHeight, tileX, tileY, animSheet){
   animation = new Animation(animSheet, imgWidth, imgHeight, cellWidth, cellHeight);
    animation.setup();
    this.specificTileAnimations[tileX][tileY] = animation;
```

```
  }

  this.drawTileAnimation = function( tile, ctx ){
        notSpecific = true;
        if (typeof this.specificTileAnimations[tile.mapX][tile.mapY] !== 'undefined' &&
this.specificTileAnimations[tile.mapX][tile.mapY] !== null) {
            notSpecific = false;
            this.specificTileAnimations[tile.mapX][tile.mapY].reset();
            this.specificTileAnimations[tile.mapX][tile.mapY].drawFrame(ctx);
        }
        if (typeof this.tileAnimations[tile.type] !== 'undefined' && this.tileAnimations[tile.type] !== null &&
notSpecific) {
          this.tileAnimations[tile.type].reset();
            this.tileAnimations[tile.type].drawFrame(ctx);
        }
  }

  this.playTileAnimation = function( tile ){ tile.animationPlaying = true; }
  this.stopTileAnimation = function( tile ){ tile.animationPlaying = false; }

  this.checkCollisions = function(sprite){ //check for collisions between sprite and tile
    tileCoordX = Math.floor( sprite.x/this.tileWidth );
      tileCoordY = Math.floor( sprite.y/this.tileHeight );
      checkRowsBegin = tileCoordX - 1;
      checkRowsEnd = tileCoordX + 2;
      checkColsBegin = tileCoordY - 1;
      checkColsEnd = tileCoordY + 2;
      if( tileCoordX > -1 && tileCoordY > -1 && tileCoordY < this.mapData.length && tileCoordX <
this.mapData[tileCoordY].length ){// if sprite is in a tile
        if( tileCoordX == 0 ){ checkRowsBegin = 0; }
        if( tileCoordX == (this.mapData[tileCoordY].length-1) ){ checkRowsEnd = this.mapData.length; }
        if( tileCoordY == 0 ){ checkColsBegin = 0; }
        if( tileCoordY == (this.mapData.length-1) ){ checkColsBegin = this.mapData[tileCoordY].length; }
        for( i = checkColsBegin; i < checkColsEnd; i++ ){
          for( j = checkRowsBegin; j < checkRowsEnd; j++ ){
              if( this.tiles[i][j].isCollidable ){
                this.tiles[i][j].checkCollision(sprite, this.tileWidth, this.tileHeight);
              }
            }
        }
      }
  }

  this.makeSpriteMapRelative = function(sprite){ sprite.setCameraRelative( this.camera ); }

  this.setPosition = function(){}
}

End of experimental classes
*/

//keyboard constants
K_A = 65; K_B = 66; K_C = 67; K_D = 68; K_E = 69; K_F = 70; K_G = 71;
K_H = 72; K_I = 73; K_J = 74; K_K = 75; K_L = 76; K_M = 77; K_N = 78;
```

K_O = 79; K_P = 80; K_Q = 81; K_R = 82; K_S = 83; K_T = 84; K_U = 85;
K_V = 86; K_W = 87; K_X = 88; K_Y = 89; K_Z = 90;
K_LEFT = 37; K_RIGHT = 39; K_UP = 38;K_DOWN = 40; K_SPACE = 32;
K_ESC = 27; K_PGUP = 33; K_PGDOWN = 34; K_HOME = 36; K_END = 35;
K_0 = 48; K_1 = 49; K_2 = 50; K_3 = 51; K_4 = 52; K_5 = 53;
K_6 = 54; K_7 = 55; K_8 = 56; K_9 = 57;

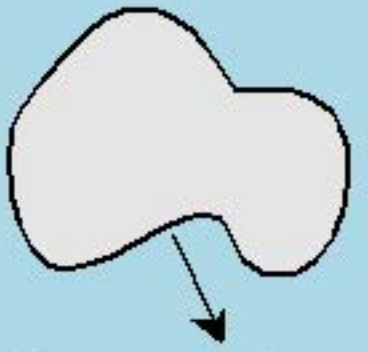
//Animation Constants
SINGLE_ROW = 1; SINGLE_COLUMN = 2; VARIABLE_LENGTH = 3;
PLAY_ONCE = 1; PLAY_LOOP = 2;

//Boundary action constants
WRAP = 0; BOUNCE = 1; STOP = 3; DIE = 4; CONTINUE = 5;

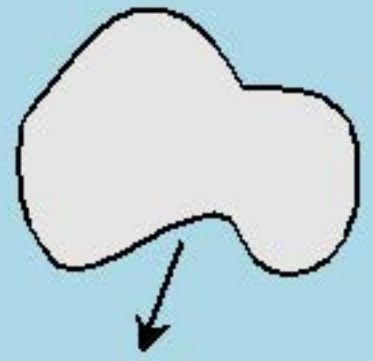
Mail Pilot plan

Lives: 5 Points 1200

simple scorekeeping.
If lives <= 0, end game
keep high score and re-enter game



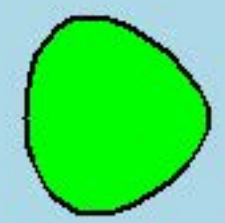
3 or more clouds
move independantly



each time a cloud enters
it has a random x, random
drift left to right, random
downward speed

tiled ocean background
moves downward for
the illusion of forward
motion. Constant speed
(same as island)

player flies over island
for points



when island hits bottom of
screen or plane, it is regenerated
at random position at top of screen

island always moves
straight down at a
constant speed

sound effects:
engine noise
cloud-plane collision
island-plane collision

layers:
clouds
plane
island
background

when a cloud moves off
the bottom of the screen
a new cloud is randomly
generated at the top

mouse controls plane's side-to-side motion
plane stays constant in y, but scrolling
background and elements give illusion of
forward motion.

when plane hits cloud, lives
decremented and cloud is
regenerated randomly at top
of screen

```
""" mpPlane
    step 1 of mailPilot
    build plane sprite,
    control it with mouse
"""

import pygame
pygame.init()

class Plane(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("plane.gif")
        self.rect = self.image.get_rect()

    def update(self):
        mousex, mousey = pygame.mouse.get_pos()
        self.rect.center = (mousex, 430)

def main():
    screen = pygame.display.set_mode((640, 480))
    pygame.display.set_caption("Mail Pilot! mpPlane.py - creating the Plane sprite")

    background = pygame.Surface(screen.get_size())
    background.fill((0, 0, 255))
    screen.blit(background, (0, 0))
    plane = Plane()

    allSprites = pygame.sprite.Group(plane)
    clock = pygame.time.Clock()
    keepGoing = True
    while keepGoing:
        clock.tick(30)
        pygame.mouse.set_visible(False)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                keepGoing = False

        allSprites.clear(screen, background)
        allSprites.update()
        allSprites.draw(screen)

        pygame.display.flip()

    #return mouse cursor
    pygame.mouse.set_visible(True)
if __name__ == "__main__":
    main()
```

```
""" mpIsland
    step 2 of mailPilot
    add island, move it,
    don't worry about collisions yet
"""

import pygame, random
pygame.init()

screen = pygame.display.set_mode((640, 480))

class Plane(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("plane.gif")
        self.rect = self.image.get_rect()

    def update(self):
        mousex, mousey = pygame.mouse.get_pos()
        self.rect.center = (mousex, 430)


class Island(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("island.gif")
        self.rect = self.image.get_rect()
        self.reset()

        self.dy = 5

    def update(self):
        self.rect.centery += self.dy
        if self.rect.top > screen.get_height():
            self.reset()

    def reset(self):
        self.rect.top = 0
        self.rect.centerx = random.randrange(0, screen.get_width())


def main():
    screen = pygame.display.set_mode((640, 480))
    pygame.display.set_caption("Mail Pilot! mpIsland - adding the Island")

    background = pygame.Surface(screen.get_size())
    background.fill((0, 0, 255))
    screen.blit(background, (0, 0))
    plane = Plane()
    island = Island()

    allSprites = pygame.sprite.Group(island, plane)
    clock = pygame.time.Clock()
```

```
    keepGoing = True
    while keepGoing:
        clock.tick(30)
        pygame.mouse.set_visible(False)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                keepGoing = False

        allSprites.clear(screen, background)
        allSprites.update()
        allSprites.draw(screen)

        pygame.display.flip()

    #return mouse cursor
    pygame.mouse.set_visible(True)
if __name__ == "__main__":
    main()
```

```
""" mpCollision
   step 4 of mailPilot
   add audio and collision
   detection
"""

import pygame, random
pygame.init()

screen = pygame.display.set_mode((640, 480))

class Plane(pygame.sprite.Sprite):
   def __init__(self):
      pygame.sprite.Sprite.__init__(self)
      self.image = pygame.image.load("plane.gif")
      self.image = self.image.convert()
      self.rect = self.image.get_rect()

      if not pygame.mixer:
         print "problem with sound"
      else:
         pygame.mixer.init()
         self.sndYay = pygame.mixer.Sound("yay.ogg")
         self.sndThunder = pygame.mixer.Sound("thunder.ogg")
         self.sndEngine = pygame.mixer.Sound("engine.ogg")
         self.sndEngine.play(-1)


   def update(self):
      mousex, mousey = pygame.mouse.get_pos()
      self.rect.center = (mousex, 430)



class Island(pygame.sprite.Sprite):
   def __init__(self):
      pygame.sprite.Sprite.__init__(self)
      self.image = pygame.image.load("island.gif")
      self.image = self.image.convert()
      self.rect = self.image.get_rect()
      self.reset()

      self.dy = 5

   def update(self):
      self.rect.centery += self.dy
      if self.rect.top > screen.get_height():
         self.reset()

   def reset(self):
      self.rect.top = 0
      self.rect.centerx = random.randrange(0, screen.get_width())
```

```
class Cloud(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("Cloud.gif")
        self.image = self.image.convert()
        self.rect = self.image.get_rect()
        self.reset()

    def update(self):
        self.rect.centerx += self.dx
        self.rect.centery += self.dy
        if self.rect.top > screen.get_height():
            self.reset()

    def reset(self):
        self.rect.bottom = 0
        self.rect.centerx = random.randrange(0, screen.get_width())
        self.dy = random.randrange(5, 10)
        self.dx = random.randrange(-2, 2)

def main():
    screen = pygame.display.set_mode((640, 480))
    pygame.display.set_caption("Mail Pilot! mpCollision.py - collisions and sounds")

    background = pygame.Surface(screen.get_size())
    background.fill((0, 0, 255))
    screen.blit(background, (0, 0))
    plane = Plane()
    island = Island()
    cloud = Cloud()

    allSprites = pygame.sprite.Group(island, plane, cloud)
    clock = pygame.time.Clock()
    keepGoing = True
    while keepGoing:
        clock.tick(30)
        pygame.mouse.set_visible(False)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                keepGoing = False

        #check collisions
        if plane.rect.colliderect(island.rect):
            plane.sndYay.play()
            #island.reset()
        if plane.rect.colliderect(cloud.rect):
            plane.sndThunder.play()
            #cloud.reset()


        allSprites.clear(screen, background)
        allSprites.update()
        allSprites.draw(screen)
```

```
        pygame.display.flip()

    plane.sndEngine.stop()

    #return mouse cursor
    pygame.mouse.set_visible(True)
if __name__ == "__main__":
    main()
```

```
""" mpMultiClouds
    step 6 of mailPilot
    add multiple clouds
"""

import pygame, random
pygame.init()

screen = pygame.display.set_mode((640, 480))

class Plane(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("plane.gif")
        self.image = self.image.convert()
        self.rect = self.image.get_rect()

        if not pygame.mixer:
            print "problem with sound"
        else:
            pygame.mixer.init()
            self.sndYay = pygame.mixer.Sound("yay.ogg")
            self.sndThunder = pygame.mixer.Sound("thunder.ogg")
            self.sndEngine = pygame.mixer.Sound("engine.ogg")
            self.sndEngine.play(-1)

    def update(self):
        mousex, mousey = pygame.mouse.get_pos()
        self.rect.center = (mousex, 430)

class Island(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("island.gif")
        self.image = self.image.convert()
        self.rect = self.image.get_rect()
        self.reset()

        self.dy = 5

    def update(self):
        self.rect.centery += self.dy
        if self.rect.top > screen.get_height():
            self.reset()

    def reset(self):
        self.rect.top = 0
        self.rect.centerx = random.randrange(0, screen.get_width())

class Cloud(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("Cloud.gif")
```

```
        self.image = self.image.convert()
        self.rect = self.image.get_rect()
        self.reset()

    def update(self):
        self.rect.centerx += self.dx
        self.rect.centery += self.dy
        if self.rect.top > screen.get_height():
            self.reset()

    def reset(self):
        self.rect.bottom = 0
        self.rect.centerx = random.randrange(0, screen.get_width())
        self.dy = random.randrange(5, 10)
        self.dx = random.randrange(-2, 2)

class Ocean(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("ocean.gif")
        self.rect = self.image.get_rect()
        self.dy = 5
        self.reset()

    def update(self):
        self.rect.bottom += self.dy
        if self.rect.bottom >= 1440:
            self.reset()

    def reset(self):
        self.rect.top = -960


def main():
    screen = pygame.display.set_mode((640, 480))
    pygame.display.set_caption("Mail Pilot! mpMultiClouds.py - multiple cloud collisions")

    background = pygame.Surface(screen.get_size())
    background.fill((0, 0, 0))
    screen.blit(background, (0, 0))
    plane = Plane()
    island = Island()
    cloud1 = Cloud()
    cloud2 = Cloud()
    cloud3 = Cloud()
    ocean = Ocean()

    friendSprites = pygame.sprite.Group(ocean, island, plane)
    cloudSprites = pygame.sprite.Group(cloud1, cloud2, cloud3)
    clock = pygame.time.Clock()
    keepGoing = True
    while keepGoing:
        clock.tick(30)
        pygame.mouse.set_visible(False)
```

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                keepGoing = False

        #check collisions

        if plane.rect.colliderect(island.rect):
            plane.sndYay.play()
            island.reset()

        hitClouds = pygame.sprite.spritecollide(plane, cloudSprites, False)

        if hitClouds:
            plane.sndThunder.play()
            for theCloud in hitClouds:
                theCloud.reset()

        #friendSprites.clear(screen, background)
        #cloudSprites.clear(screen, background)
        friendSprites.update()
        cloudSprites.update()
        friendSprites.draw(screen)
        cloudSprites.draw(screen)

        pygame.display.flip()

    #return mouse cursor
    pygame.mouse.set_visible(True)
if __name__ == "__main__":
    main()
```

```
""" carGE.py
    extend SuperSprite to add keyboard input
"""

import pygame, gameEngine

class Car(gameEngine.SuperSprite):
    def __init__(self, scene):
        gameEngine.SuperSprite.__init__(self, scene)
        self.setImage("car.gif")

    def checkEvents(self):
        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            self.turnBy(5)
        if keys[pygame.K_RIGHT]:
            self.turnBy(-5)
        if keys[pygame.K_UP]:
            self.speedUp(.2)
        if keys[pygame.K_DOWN]:
            self.speedUp(-.2)

        self.drawTrace()

def main():
    game = gameEngine.Scene()
    game.background.fill((0xCC, 0xCC, 0xCC))

    car = Car(game)
    game.sprites = [car]

    game.start()

if __name__ == "__main__":
    main()
```

```
""" simpleGe.py
    example of simplest possible
    game engine program
"""

import pygame, gameEngine

game = gameEngine.Scene()
game.start()
```

```
""" asteroids.py
    basic variation of the classic
    using gameEngine
"""

import pygame, gameEngine, random

class Ship(gameEngine.SuperSprite):
    def __init__(self, scene):
        gameEngine.SuperSprite.__init__(self, scene)
        self.setImage("shipSmall.gif")
        self.setSpeed(0)
        self.setAngle(0)

    def checkEvents(self):
        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            self.rotateBy(5)
        if keys[pygame.K_RIGHT]:
            self.rotateBy(-5)
        if keys[pygame.K_UP]:
            self.addForce(.2, self.rotation)
        if keys[pygame.K_SPACE]:
            self.scene.bullet.fire()

class Bullet(gameEngine.SuperSprite):
    def __init__(self, scene):
        gameEngine.SuperSprite.__init__(self, scene)
        self.setImage("bullet.gif")
        self.imageMaster = pygame.transform.scale(self.imageMaster, (5, 5))
        self.setBoundAction(self.HIDE)
        self.reset()

    def fire(self):
        self.setPosition((self.scene.ship.x, self.scene.ship.y))
        self.setSpeed(12)
        self.setAngle(self.scene.ship.rotation)

    def reset(self):
        self.setPosition ((-100, -100))
        self.setSpeed(0)


class Rock(gameEngine.SuperSprite):
    def __init__(self, scene):
        gameEngine.SuperSprite.__init__(self, scene)
        self.setImage("rock.gif")
        self.reset()

    def checkEvents(self):
        self.rotateBy(self.rotSpeed)

    def reset(self):
```

```
        """ change attributes randomly """

        #set random position
        x = random.randint(0, self.screen.get_width())
        y = random.randint(0, self.screen.get_height())
        self.setPosition((x, y))

        #set random size
        scale = random.randint(10, 40)
        self.setImage("rock.gif")
        self.imageMaster = \
           pygame.transform.scale(self.imageMaster, (scale, scale))

        self.setSpeed(random.randint(0, 6))
        self.setAngle(random.randint(0, 360))
        self.rotSpeed = random.randint(-5, 5)

class Game(gameEngine.Scene):
    def __init__(self):
        gameEngine.Scene.__init__(self)
        self.ship = Ship(self)
        self.bullet = Bullet(self)

        self.rocks = []
        for i in range(10):
            self.rocks.append(Rock(self))

        self.rockGroup = self.makeSpriteGroup(self.rocks)
        self.addGroup(self.rockGroup)
        self.sprites = [self.bullet, self.ship]
        self.setCaption("asteroids")

    def update(self):
        rockHitShip = self.ship.collidesGroup(self.rocks)
        if rockHitShip:
            rockHitShip.reset()

        rockHitBullet = self.bullet.collidesGroup(self.rocks)
        if rockHitBullet:
            rockHitBullet.reset()
            self.bullet.reset()

def main():
    game = Game()
    game.start()

if __name__ == "__main__":
    main()
```

```
""" guiDemoGE.py
    demonstrates the GUI objects in
    gameEngine
"""

import pygame, gameEngine

class Game(gameEngine.Scene):
    def __init__(self):
        gameEngine.Scene.__init__(self)
        #initialize background to yellow
        self.background.fill((0xff, 0xff, 0x00))

        self.addLabels()
        self.addButton()
        self.addScroller()
        self.addMultiLabel()

        self.sprites = [self.lblTitle, self.label,
                        self.lblButton, self.button,
                        self.lblScroller, self.scroller,
                        self.multi]

    def addLabels(self):
        self.lblTitle = gameEngine.Label()
        self.lblTitle.text = "GameEngine GUI Demo"
        self.lblTitle.center = (320, 40)
        self.lblTitle.size = (300, 30)

        self.label = gameEngine.Label()
        self.label.font = pygame.font.Font("goodfoot.ttf", 40)
        self.label.text = "Label"
        self.label.fgColor = (0xCC, 0x00, 0x00)
        self.label.bgColor = (0xCC, 0xCC, 0x00)
        self.label.center = (320, 100)
        self.label.size = (100, 50)

    def addButton(self):
        self.lblButton = gameEngine.Label()
        self.lblButton.center = (200, 180)
        self.lblButton.text = "Button"

        self.button = gameEngine.Button()
        self.button.center = (450, 180)
        self.button.text = "don't click me"

    def addScroller(self):
        self.lblScroller = gameEngine.Label()
        self.lblScroller.text = "scroller"
        self.lblScroller.center = (200, 250)

        self.scroller = gameEngine.Scroller()
        self.scroller.center = (450, 250)
```

```
        self.scroller.minValue= 0
        self.scroller.maxValue = 250
        self.scroller.value = 200
        self.scroller.increment = 5

    def addMultiLabel(self):
        self.multi = gameEngine.MultiLabel()
        self.multi.textLines = [
            "This is a multiline text box.",
            "It's useful when you want to",
            "put larger amounts of text",
            "on the screen. Of course, you",
            "can change the colors and font."
            ]
        self.multi.size = (400, 120)
        self.multi.center = (320, 400)

    def update(self):
        if self.button.clicked:
            self.lblButton.text = "Ouch!"
            self.background.fill((0x00, 0x00, 0x00))
            self.screen.blit(self.background, (0, 0))

        self.lblScroller.center = (self.scroller.value, 250)

def main():
    game = Game()
    game.start()

if __name__ == "__main__":
    main()
```