Abhishek Nand Kumar

Project Report: Meteorophobia

This is and individual report for CS1830. I'll be covering my game which was created using python. In the first section I'll be going through the user manual.

Message to marker: I've attached my python game as a ".py" and I've linked my codeSkulptor file, there isn't a cap on the framerate in the codeSkulptor file because they didn't support some libraries. However the py file framerate is fine, locked at 30fps.

http://www.codeskulptor.org/#user42_wlbxorxa2Y5VCZQ_4.py

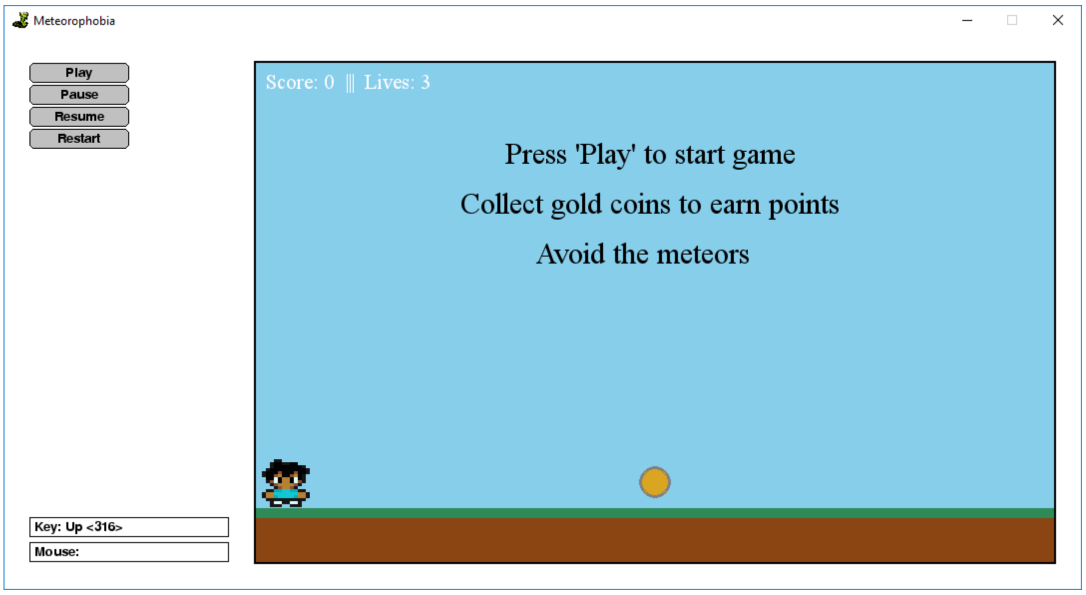
User manual:

INTRO SCREEN

This is the first screen when you open the python game. (1) All instructions are displayed on the front screen. The basic rules of the game are:

- Avoid getting hit by the meteors
- Collect the gold coins to earn a point

(2) Simply press play to start the game.



Score: 0 ||| Lives: 3

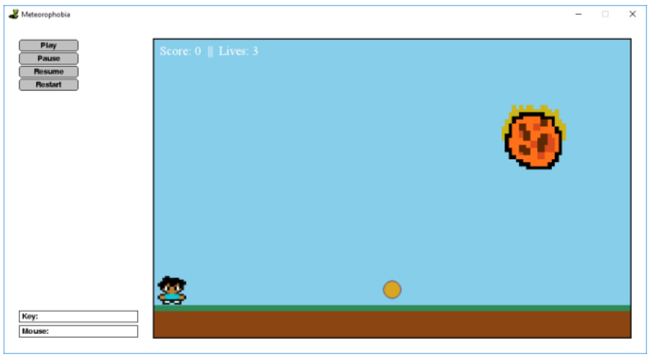
Press 'Play' to start game

Collect gold coins to earn points

Avoid the meteors

PLAYING THE GAME

After pressing “play” the text on screen will disappear and the game will start. The meteor will start falling from the sky and the player can now move.



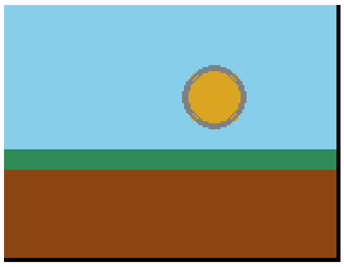
<u>Simply use the arrow keys to move the player:</u>

UP KEY: Makes the player jump

DOWN = make the player face the screen

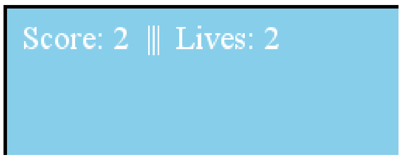
LEFT KEY: Makes the player run left

RIGHT KEY: Makes the player run right



The golden coins will spawn randomly on the floor (one at a time). The player must run into it to pick it up.

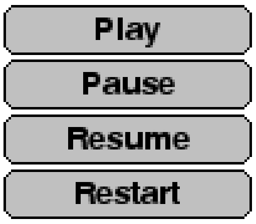
The player starts with three lives. Every time the player gets hit by a meteor, they will lose a life.



By using the arrow keys the player can pick up the gold coins to earn a point. Their updated score and lives remaining will be shown on the top-left corner of the game window.

BUTTONS

<u>On the left are all the buttons within the game:</u>

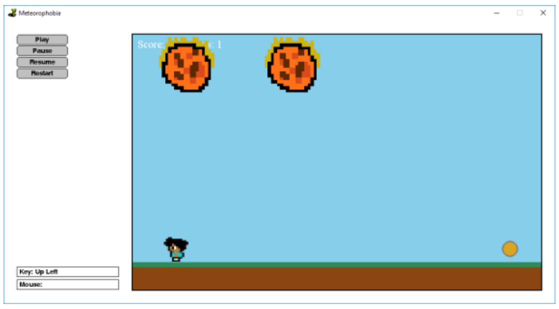


PLAY: This will start the game whilst on the “intro screen”.

PAUSE: this will pause the game whilst it’s running

RESUME: This will resume the game after it has been paused.

RESTART: This will restart the game when the player has died.



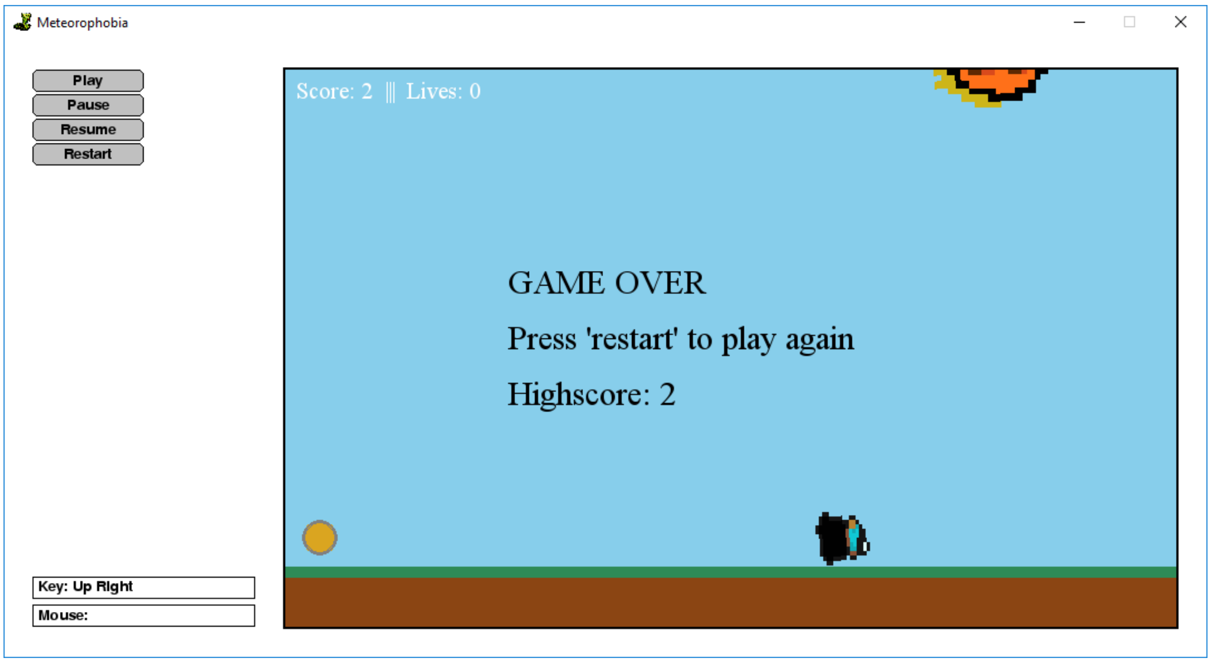
Key: Up Left

Mouse:

The game progressively gets harder, once the player gets more than five coins there will be two meteors that fall from the sky at the same time and they fall faster.

At 11 coins there will be three meteors that fall from the sky at higher speeds.

DEATH SCREEN



Score: 2 ||| Lives: 0

GAME OVER

Press 'restart' to play again

Highscore: 2

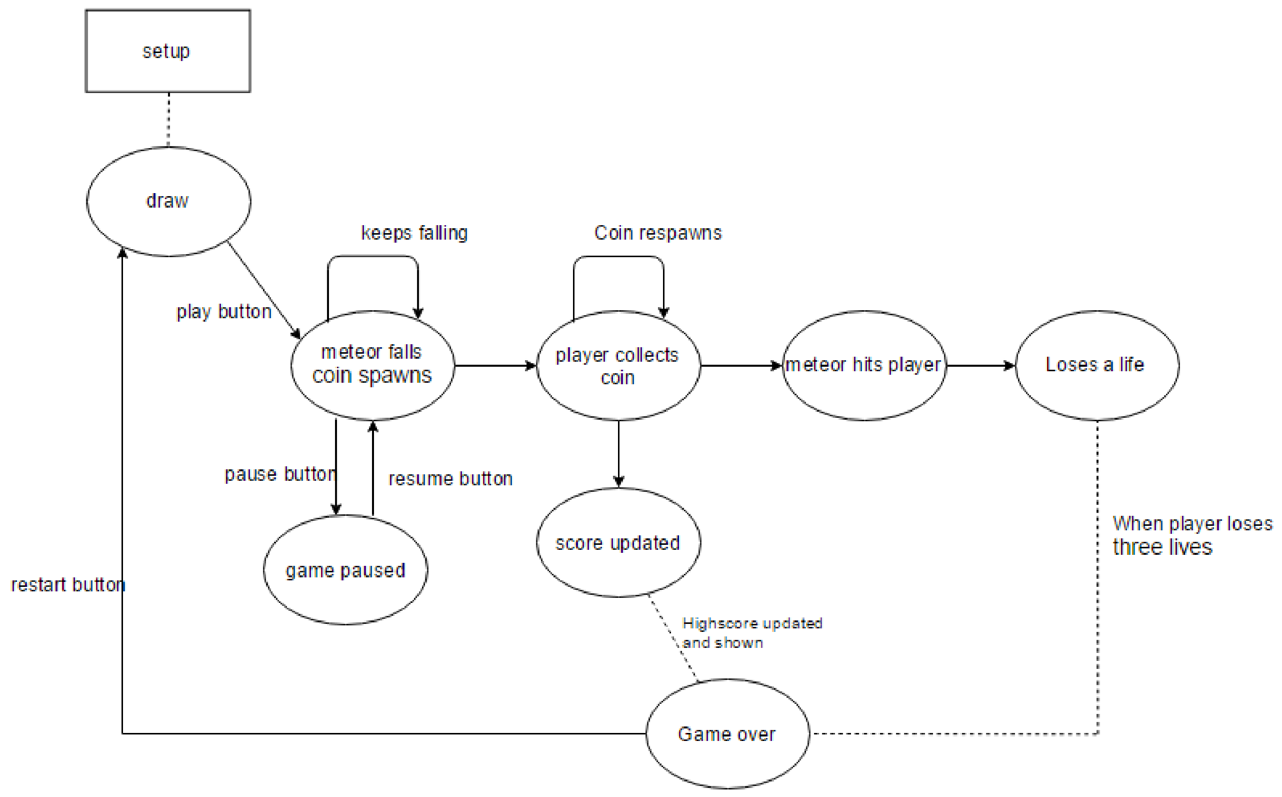
Key: Up Right

Mouse:

Above is the screen when the player has run out of lives. It gives a message saying “game over”, a message telling the player how to restart and the highest score that the player has accumulated.

Simply press restart to restart the game, this will reset the player lives and the score, it will also reduce the difficulty to one meteor falling at a time.

State Diagram:



Abhishek Nand Kumar

Game code:

```
def keydown(key):
    global right_pressed
    global left_pressed
    global up_pressed
    global down_pressed

    if key == simplegui.KEY_MAP['right']:
        right_pressed = True
    elif key == simplegui.KEY_MAP['left']:
        left_pressed = True
    elif key == simplegui.KEY_MAP['up']:
        up_pressed = True
    elif key == simplegui.KEY_MAP['down']:
        down_pressed = True


def keyup(key):
    global right_pressed
    global left_pressed
    global up_pressed
    global down_pressed

    if key == simplegui.KEY_MAP['right']:
        right_pressed = False
    elif key == simplegui.KEY_MAP['left']:
        left_pressed = False
    elif key == simplegui.KEY_MAP['up']:
        up_pressed = False
    elif key == simplegui.KEY_MAP['down']:
        down_pressed = False
```

This is the code that looks for key presses. it picks up four inputs: up, down, down left and right. When a key is pressed, it is set to “true”. This is how I control what happens within the game. For example:

```
def draw(canvas):
. . .
        if right_pressed:
                 if frameIndex != (1, 0):
                 cordX += STEP
                 frameIndex = (1, 0)
```

This is within the draw handler. So, when right has been pressed I change the sprite image (this will animate the sprite when it’s moving), the if statement is if the sprite frame is not the first one then go to the second frame in the sprite sheet. The next line says to increase the y coordinates of the sprite by “STEP” amount (STEP is 7 which was defined at the beginning, the sprite will move forward 7 pixels every frame when the right button is pressed). And then finally return the sprite frame to the first one, this makes it look like the sprite is running, creating the running animation.

To create the sprites I used all the things provided in the slides on moodle.

<u>Buttons</u>

```
frame.add_button("Play", play, 100)
frame.add_button("Pause", pause, 100)
frame.add_button("Resume", resume, 100)
frame.add_button("Restart", restart, 100)
```

```

def pause():
    global STEP
    global meteorSTEP

    STEP = 0
    meteorSTEP = 0


def resume():
    global STEP
    global meteorSTEP

    STEP = 7
    meteorSTEP = 10


def restart():
    global life
    global STEP
    global meteorSTEP
    global score
    global angle

    life = 3
    score = 0
    meteorSTEP = 10
    STEP = 7
    angle = 0
```

This is the code that controls the buttons. I used frame.add_button to create the buttons and it leads to their designated functions, for example if you press the pause button, it will set STEP (the variable that it used to increase the y coordinate of the player) to zero which means the player wouldn’t be able to move it does the same thing for the meteor (meteorSTEP).

<u>Difficulty</u>

```
if score > 5:
    if meteorSTEP != 0:
        meteorSTEP = 15
        canvas.draw_image(meteor2,
                          (MframeWidth * MframeIndex[0] + MframeCentreX,
                           MframeHeight * MframeIndex[1] + MframeCentreY),
                          (MframeWidth, MframeHeight),
                          (xMeteor2, yMeteor),
                          meteorDims,
                          angle)
        if yMeteor == -100:
            xMeteor2 = random.randint(0, 800)
```

This is within the draw() function. It’s how I set the game difficulty. When the play reaches a score of 5, it increases the speed of the falling meteors by increase the STEP it does each frame. It then draws a second meteor and spawns it randomly with the game window using the “random” library in python.

Meteor hits player/coin collision:

```
Def draw(canvas):
. . .

        if collisions.coinCollision(xCoin, cordX):
            xCoin = random.randint(0, 800)
            score += 1

        if collisions.meteorCollision(xMeteor, cordX, yMeteor, cordY):
            if life > 0:
                life -= 1
                yMeteor = -20
                xMeteor = random.randint(0, 800)
```

```
class collisions:
    def coinCollision(x1, x2): #coin, cordx
        if x2 > x1 - 20 and x2 < x1 + 20:
            return True

    def meteorCollision(x1, x2, y1, y2): #xMeteor, cordX, yMeteor, cordY
        if y2 < y1 + 70 and y2 > y1 - 70:
            if x2 < x1 + 70 and x2 > x1 - 70:
                return True
```

This is how I check for collisions between the player and the coins and the player and the meteors.

Coin collision: I send the x coordinate of the coin and the x coordinate of the player to the coin collision function in the collision class. In the collision function, it checks to see if the player is within range of the diameter of the coin, if the player is, then it returns true. This will then increase the score of the player and respawn the coin on another x coordinate of the game.

Meteor collision: For this I must check if the player is within the diameter of the meteor if the player is then it returns true. This then reduces the life by one and it respawns the meteor at the top of the game at a random x coordinate using the random library.

Death/Game Over:

```
if gameOver(life):
    frameIndex = [2, 1]
    angle = 300
    STEP = 0
    canvas.draw_text("GAME OVER", (200, 200), 30, "black")
    canvas.draw_text("Press 'restart' to play again", (200, 250), 30, "black")
    meteorSTEP = 0

    if score > highScore:
        highScore = score

    canvas.draw_text("Highscore: " + str(highScore), (200, 300), 30, "black")
```

```
def gameOver(x):
    if x <= 0:
        return True
```

This checks to see if the game should end. The life is sent to the gameOver() function and if it is below or equal to zero then it returns true, this means that the game should end. When the game ends, it sets the player sprite to the death sprite and it rotates it sideways to make it look like he's dead. It then sets all movement to zero (meteors and player movement). it displays a message on screen showing the players high score (and setting the high score) and it shows a message saying "game over".