**Name:** Amay Dixit

**ID Number:** 12340220

**Email ID:** amayd@iitbhilai.ac.in

**Date:** 19/11/2023

**Title of the Project:** Interference of light waves and YDSE simulation

**Abstract:** Interference of light is the phenomena of multiple light waves interacting with one another under certain circumstances, causing the combined amplitudes of the waves to either increase or decrease. This project explores the interference of light waves, through a simulation of Young's Double Slit Experiment (YDSE), an important experiment in the field of wave optics.

**Key words:** Interference; wave optics; Young's Double Slit Experiment; simulation.

1. **Introduction**

Interference, a captivating phenomenon in wave optics, helps in determining the nature of waves as they interact and overlap. A profound understanding of interference is essential for studying the complexities of light behaving as a wave, and the Young's Double Slit Experiment (YDSE) serves as an illustrative example of this phenomenon.  
  
The Young's Double Slit Experiment, performed by Thomas Young in the early 19th century, remains an important phenomenon in the study of interference patterns. YDSE is an example of interference of light waves through division of wavefronts. In this experiment, a coherent light source is directed toward two closely spaced slits, creating two separate wavefronts which are observed on the screen. As these waves intersect, regions of constructive and destructive interference emerge, giving rise to a pattern of alternating bright and dark fringes on the screen.  
  
To analyse the physics of interference and to visualize the patterns generated in the Young’s Double Slit Experiment, this project aims provides a simulation this project embarks on a simulation journey. Through computational models and virtual environments, researchers and students can explore the interference of waves, observe the patterns, and develop insights and a deep understanding of the principles of wave optics.

1. **Principle**

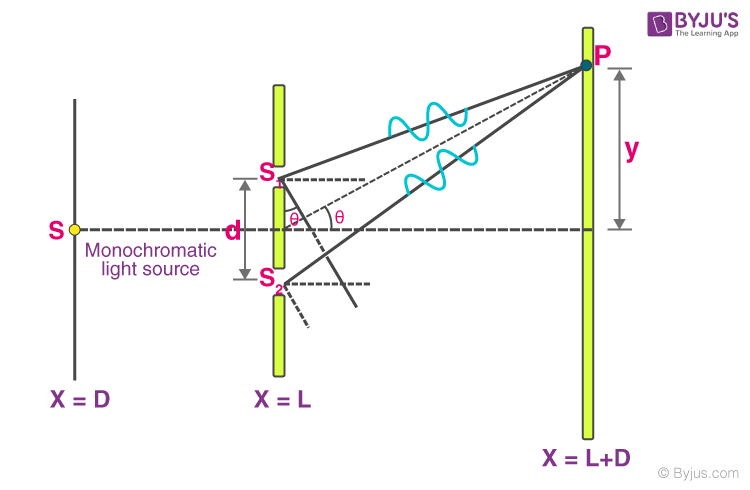
Double-Slit Experiment: The double-slit experiment is a classic demonstration of light interference. In this setup, a beam of light passes through two closely spaced slits, creating an interference pattern on a screen. The resulting pattern depends on factors such as wavelength and slit separation. The project explores the mathematical expressions governing interference patterns and their implications in various scenarios. When waves converge, whether they be light or another form of wave, they create interference patterns—bright fringes where peaks align (constructive interference) and dark fringes where peaks meet troughs (destructive interference). The YDSE simulation allows us to virtually recreate this experiment, offering a platform to explore the impact of different variables such as slit separation and wavelength on the resulting interference pattern.

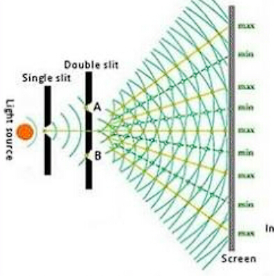
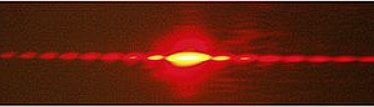
**Young’s Double slit experiment**Each source can be considered a source of coherent [light waves](https://byjus.com/light-waves-and-color-formula/). At any point on the screen at a distance ‘y’ from the centre, the waves travel different distances to create a path difference of Δx at the point. The point approximately subtends an angle of θ at the sources (since the distance D is large, there is only a very small difference between the angles subtended at sources).

Consider a monochromatic light source ‘S’ kept at a considerable distance from two slits: s1 and s2. S is equidistant from s1 and s2. s1 and s2 behave as two coherent sources as both are derived from S.

The light passes through these slits and falls on a screen which is at a distance ‘D’ from the position of slits s1 and s2. ‘d’ is the separation between two slits.

When the slit separation (d) and the screen distance (D) are kept unchanged, to reach *P,* the light waves from s1 and s2 must travel different distances. It implies that there is a path difference in Young’s double slit experiment between the two light waves from s1 and s2.



**Simulation and its Code**

The Scope of this project is to provide a simple simulation to understand the interference of the waves through the Young’s double slit experiment.

1. **Installation :**

* **Prerequisites**
  + Python installed on your system (version 3.6 or higher).
  + Pygame library installed (pip install pygame).
* **Steps to Install:**
  + Download the code - Save the provided code in a file with a .py extension, for example, YDSE\_Simulation.py.
  + Install Pygame – open the terminal and run the command ‘pip install pygame’Run
* **Run the simulation**
  + Navigate to the directory where you saved the ‘YDSE\_Simulation.py’ file using the terminal or command prompt
  + Run the command ‘python YDSE\_Simulation.py’
  + The simulation window should open, allowing you to interact with the Young's Double Slit Experiment.

1. **Interacting with the Simulation**

* **Changing parameters**
  + Click on text boxes to change parameters like slit distance (D), wavelength (λ), and slit width (d).
  + Use the backspace key to clear values before entering new ones
* **Moving the Slits**
  + Click on the black screen to change the distance between slits (D)
* **Back to default**
  + Click "Back to Default" to reset the parameters to their default values.
* **Read Instructions**
  + Click "Read Instructions" for a pop-up guide on using the simulation effectively.

**Source Code:**

import pygame, tkinter as tk, time

from sys import exit

from math import pi, sin, asin

def gradientRect(window, left\_colour, right\_colour, target\_rect):

    colour\_rect = pygame.Surface((2, 2))

    pygame.draw.line(colour\_rect, left\_colour, (0,0), (1,0))

    pygame.draw.line(colour\_rect, right\_colour, (0,1), (1,1))

    colour\_rect = pygame.transform.smoothscale(colour\_rect, (target\_rect.width, target\_rect.height))

    window.blit(colour\_rect, target\_rect)

def wavelength\_to\_rgb(wavelength):

    gamma = 0.8

    intensity\_max = 255

    if 380 <= wavelength < 440:

        R = -(wavelength - 440) / (440 - 380)

        G = 0.0

        B = 1.0

    elif 440 <= wavelength < 490:

        R = 0.0

        G = (wavelength - 440) / (490 - 440)

        B = 1.0

    elif 490 <= wavelength < 510:

        R = 0.0

        G = 1.0

        B = -(wavelength - 510) / (510 - 490)

    elif 510 <= wavelength < 580:

        R = (wavelength - 510) / (580 - 510)

        G = 1.0

        B = 0.0

    elif 580 <= wavelength < 645:

        R = 1.0

        G = -(wavelength - 645) / (645 - 580)

        B = 0.0

    elif 645 <= wavelength <= 750:

        R = 1.0

        G = 0.0

        B = 0.0

    else:

        R = 255

        G = 255

        B = 255

    # Adjust intensity

    if 380 <= wavelength < 645:

        factor = 1.0

    else:

        factor = 0.3 + 0.7 \* (750 - wavelength) / (750 - 645)

    if 380 <= wavelength <= 750:

        R = int(intensity\_max \* (R \*\* gamma) \* factor)

        G = int(intensity\_max \* (G \*\* gamma) \* factor)

        B = int(intensity\_max \* (B \*\* gamma) \* factor)

    return (R, G, B)

class Wavefronts:

    def \_\_init\_\_(self, startx, starty):

        self.startx, self.starty = startx, starty

        self.boxes = [0]

        self.rect = pygame.Rect(startx, starty, 40, 40)

        self.prev\_D = 800

        self.wavefront\_seperation = 5

        self.wavefront\_seperation\_input = str(self.wavefront\_seperation)

        self.propogate = True

    def production(self, color = 'white'):

        pygame.draw.arc(screen, color, self.rect, -1.5, 1.5, width = 1)

    def propogation(self, color = 'white'):

        self.wavefront\_seperation\_input = str(self.wavefront\_seperation)

        for i in self.boxes[::-1]:

            self.rect = pygame.Rect(self.startx, self.starty-i, 2\*i, 2\*i)

            self.production(color)

            if self.boxes.index(i) <= 10:

                time.sleep(0.001)

        if self.propogate:

            self.boxes.insert(0,self.boxes[0] + self.wavefront\_seperation)

    def animate(self, scr\_pos):

        for i in range(self.startx + 2\*self.boxes[-1], scr\_pos ,self.wavefront\_seperation):

            pass

class Main:

    def \_\_init\_\_(self):

        self.default()

    def draw(self):

        self.s.propogation(wavelength\_to\_rgb(self.L))

        pygame.draw.line(screen, 'white', (10,0), (10,800))

        pygame.draw.rect(screen ,'black', pygame.Rect(self.D,0,800,800))

        pygame.draw.line(screen, 'white', (self.D, 0), (self.D, 395 - self.d/2), width = 5)

        pygame.draw.line(screen, 'white', (self.D, 405 - self.d/2), (self.D, 395 + self.d/2), width = 5)

        pygame.draw.line(screen, 'white', (self.D, 405 + self.d/2), (self.D, 800), width = 5)

        if (self.D - self.s.startx)/2 <= self.s.boxes[0]:

            self.s1.propogation(wavelength\_to\_rgb(self.L))

            self.s2.propogation(wavelength\_to\_rgb(self.L))

        pygame.draw.rect(screen, 'white', pygame.Rect(800,0,200,800))

        pygame.draw.rect(screen, 'white', pygame.Rect(0, 770, 1000, 30))

        if self.s1.rect.midright[0] > 800:

            self.scr()

        self.text\_surface = text\_font.render('D = ' + str(self.actual\_D) + ' cm', False, 'black')

        self.text\_rect = self.text\_surface.get\_rect(center = (self.D + (800-self.D)/2, 780))

    def change\_D(self):

        if self.s.prev\_D != self.D:

            self.s.prev\_D = self.D

            self.s1.startx = self.D

            self.s2.startx = self.D

            self.restart(1)

        self.actual\_D = 2\*(800 - self.D)/10

    def change\_d(self):

        self.s1.starty = 400 - self.d/2

        self.s2.starty = 400 + self.d/2

    def change\_L(self):

        self.s1.wavefront\_seperation = self.L/115

        self.s2.wavefront\_seperation = self.L/115

        self.s.wavefront\_seperation = self.L/115

        self.restart(lm[1]%2)

    def restart(self, cond):

        if cond:

            self.s1.boxes.clear()

            self.s2.boxes.clear()

            self.s.boxes.clear()

            self.s.boxes.append(0)

            self.s1.boxes.append(0)

            self.s2.boxes.append(0)

    def default(self):

        self.d = 50

        self.actual\_d = self.d/10

        self.d\_input\_text = str(self.actual\_d)

        self.D = 300

        self.actual\_D = 2\*(800 - self.D)/10

        self.source = (10,400)

        self.L = 600

        self.L\_input\_text = str(self.L)

        self.B = (self.L)\*(self.actual\_D/self.d)\*(10\*\*-4)

        self.actual\_B = self.L \* (self.D/100)/self.actual\_d

        self.B\_input\_text = str(self.B)

        self.s = Wavefronts(\*self.source)

        self.s.prev\_D = self.D

        self.slit1 = (self.D, 400 - self.d/2)

        self.slit2 = (self.D, 400 + self.d/2)

        self.s1 = Wavefronts(\*self.slit1)

        self.s2 = Wavefronts(\*self.slit2)

    def update(self):

        self.change\_D()

        self.change\_d()

        self.change\_L()

        self.draw()

        self.ui()

        self.actual\_D = 800 - self.D

        self.actual\_d = self.d/10

        self.actual\_B = self.L \* (self.actual\_D/100)/self.actual\_d

        self.B\_input\_text = str(self.B)

        screen.blit(self.text\_surface, self.text\_rect)

    def scr(self):

        if self.d and self.D > 4:

            self.B = ((800-self.D) \* sin(asin(73/700)) \* 50 \* self.L / self.d)/600

        i = 0

        while (i\*self.B < 500):

            gradientRect(screen, wavelength\_to\_rgb(self.L), (0,0,0), pygame.Rect(800, 400 - i\*self.B, 200, self.B/2))

            gradientRect(screen, (0,0,0), wavelength\_to\_rgb(self.L), pygame.Rect(800, 400 - i\*self.B - self.B/2, 200, self.B/2))

            gradientRect(screen, wavelength\_to\_rgb(self.L), (0,0,0), pygame.Rect(800, 400 + i\*self.B, 200, self.B/2))

            gradientRect(screen, (0,0,0), wavelength\_to\_rgb(self.L), pygame.Rect(800, 400 + i\*self.B + self.B/2, 200, self.B/2))

            i+=1

    def ui(self):

        pygame.draw.rect(screen, 'white', pygame.Rect(0, 800, 1000, 200))

        pygame.draw.rect(screen, 'white', pygame.Rect(800, 770, 200, 230))

        B\_text = text\_font.render(f"β = {(self.actual\_B)/5} µm", False, 'black')

        B\_text\_rect = B\_text.get\_rect(topleft = (850, 800))

        screen.blit(B\_text, B\_text\_rect)

        d\_text = text\_font.render(f"d = {self.actual\_d} mm", False, 'black')

        d\_text\_rect = d\_text.get\_rect(topleft = (10, 800))

        screen.blit(d\_text, d\_text\_rect)

        pygame.draw.rect(screen, 'black', pygame.Rect(10, 819, 80, 16), width = 1)

        d\_input = text\_font.render(self.d\_input\_text, False, 'black')

        d\_input\_rect = d\_input.get\_rect(center = (45, 828))

        screen.blit(d\_input, d\_input\_rect)

        L\_text = text\_font.render(f"λ = {self.L} nm", False, 'black')

        L\_text\_rect = L\_text.get\_rect(topleft = (100, 800))

        screen.blit(L\_text, L\_text\_rect)

        pygame.draw.rect(screen, 'black', pygame.Rect(100, 819, 80, 16), width = 1)

        L\_input = text\_font.render(self.L\_input\_text, False, 'black')

        L\_input\_rect = L\_input.get\_rect(center = (135, 828))

        screen.blit(L\_input, L\_input\_rect)

        default\_text = text\_font.render('Back to Default', False, 'red')

        default\_text\_rect = default\_text.get\_rect(topleft = (300, 815))

        screen.blit(default\_text, default\_text\_rect)

        pygame.draw.rect(screen, 'black', pygame.Rect(295, 810, 120, 20), width = 1)

        instructions\_text = text\_font.render('Read Instructions', False, 'red')

        instructions\_text\_rect = instructions\_text.get\_rect(topleft = (450, 815))

        screen.blit(instructions\_text, instructions\_text\_rect)

        pygame.draw.rect(screen, 'black', pygame.Rect(445, 810, 150, 20), width = 1)

    def pop\_up(self):

        greet = tk.Label(text = '''Welcome to the simulation for YDSE!!!

This simulation is made by AMAY DIXIT

Following is the basic guide on how to operate the program\n\n

1. Click on the text boxes below the variables (eg. d, λ) to change them

 2. Once you click on them, the option to enter values is active, press backspace to erase the current value completely and only then enter the new value

3. Do not forget to click anywhere else on the screen after you're done entering the values, not doing so can cause bugs\n

4. Click on the black screen to change the values of D, you need not drag the slit surface,

   just click on the black screen once and the slit surface will follow the cursor, click again to stop it, you can see the values of D changing

''')

        greet.pack()

        window.mainloop()

# pygame setup

pygame.init()

screen = pygame.display.set\_mode((1000,850))

pygame.display.set\_caption("Amay's Double Slit Experiment")

SCREEN\_UPDATE = pygame.USEREVENT

pygame.time.set\_timer(SCREEN\_UPDATE, 150)

clock = pygame.time.Clock()

text\_font = pygame.font.Font(None, 22)

main = Main()

# tkinter setup

window = tk.Tk()

main.pop\_up()

k = -1

lm = [0, 0, 0]

while 1:

    key = pygame.key.get\_pressed()

    mouse = pygame.mouse.get\_pos()

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            exit()

        if mouse[0] < 750 and mouse[1] < 800:

            if event.type == pygame.MOUSEMOTION:

                if k%2 == 0:

                    main.D = mouse[0]

            elif event.type == pygame.MOUSEBUTTONDOWN:

                k += 1

        if event.type == pygame.MOUSEBUTTONDOWN:

            lm = [0 for x in lm]

            if mouse[0] in range(10,90) and mouse[1] in range(820,835):

                lm[0] += 1

            if mouse[0] in range(100,180) and mouse[1] in range(820,835):

                lm[1] += 1

            if mouse[0] in range(295,295+120) and mouse[1] in range(810,830):

                main.default()

            if mouse[0] in range(445, 445+150) and mouse[1] in range(810, 830):

                main.pop\_up()

        if lm[0]%2:

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_BACKSPACE:

                    main.d\_input\_text = main.d\_input\_text[:-1]

                else:

                    main.d\_input\_text += event.unicode

                    main.d = float(main.d\_input\_text)\*10

        if lm[1]%2:

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_BACKSPACE:

                    main.L\_input\_text = main.L\_input\_text[:-1]

                else:

                    main.L\_input\_text += event.unicode

                    main.L = float(main.L\_input\_text)

        if lm[2]%2:

            pass

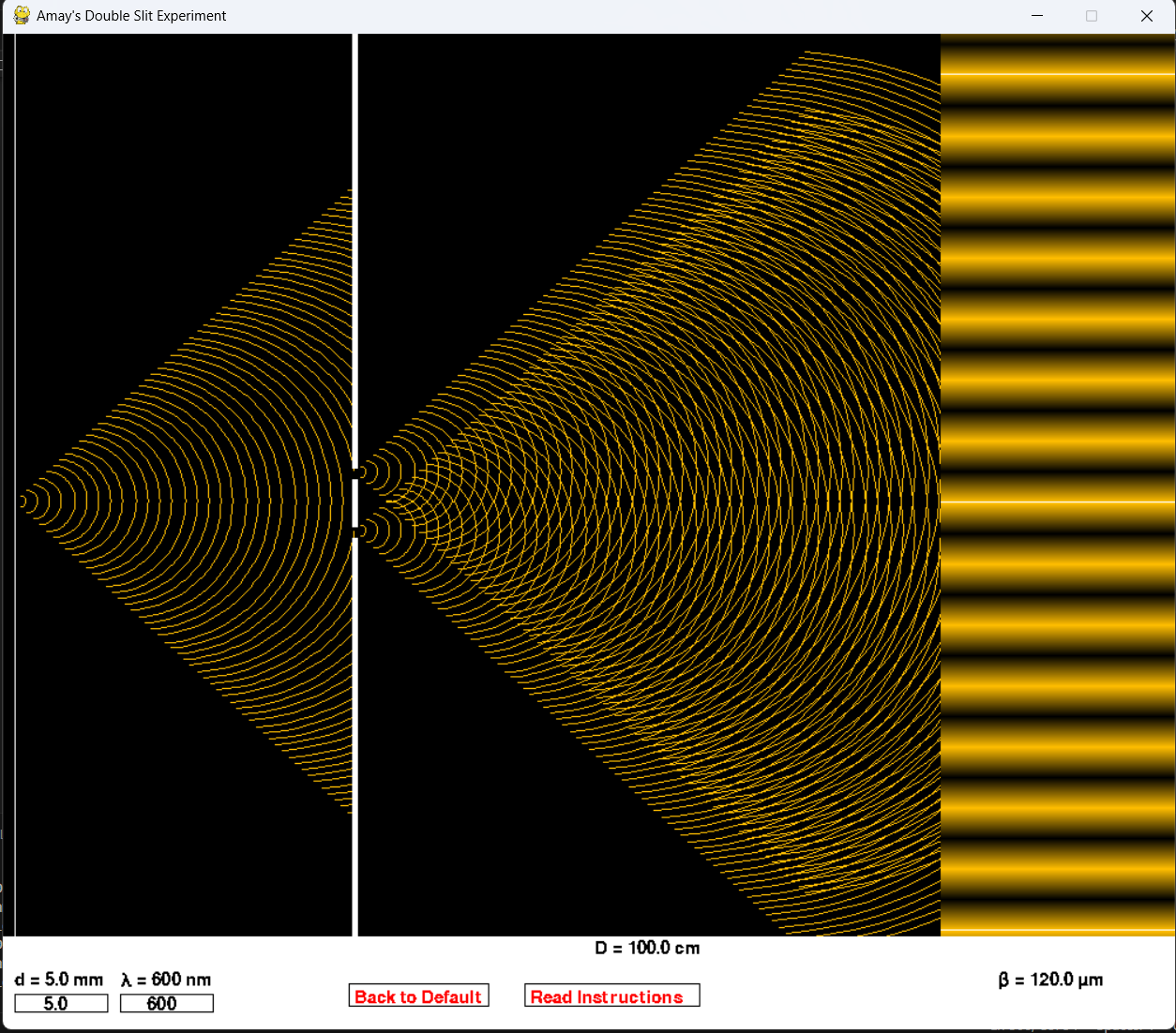
    screen.fill('#000000')

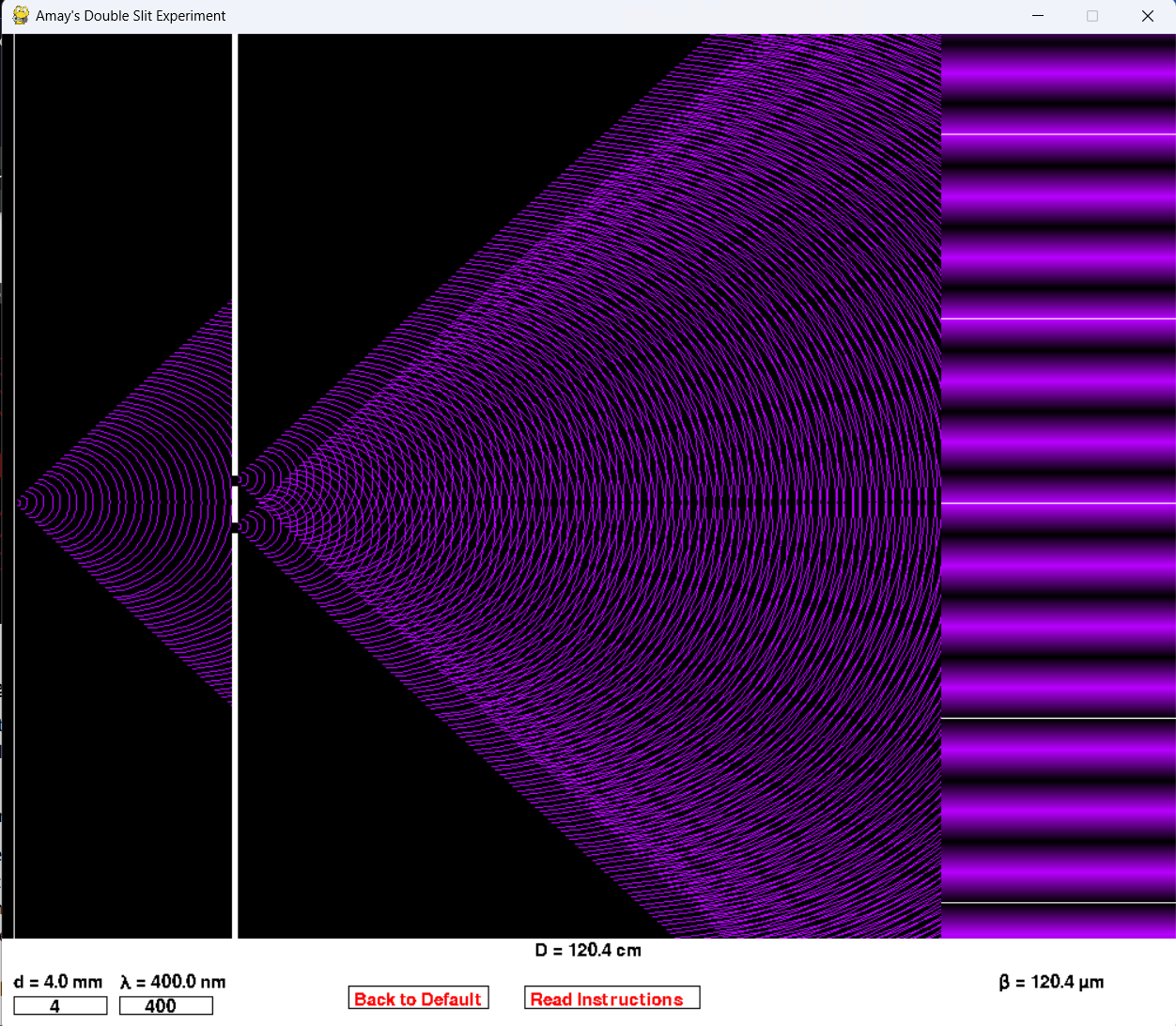
    main.update()

    pygame.display.update()

    clock.tick(90)

The following images show the program window upon running it :

****

****

1. **Results and Discussions**

In this section, we discuss the results obtained from the interference and Young's Double Slit Experiment (YDSE) simulation code. The analysis aims to provide a comprehensive understanding of the simulated scenarios, the impact of different parameters, and the implications of the simulation results.

The Young's Double-Slit Experiment (YDSE) is a classic experiment in physics that demonstrates the wave-particle duality of light and other particles. Here are the key results and conclusions drawn from the YDSE:

* **Interference Pattern:** When light or other particles, such as electrons, pass through two closely spaced slits, an interference pattern is observed on a screen placed behind the slits. This pattern consists of alternating bright and dark fringes. The conclusion drawn from this is a major contributor in modern and quantum physics.
* **Wave-Particle Duality:** The interference pattern observed in the YDSE suggests that light and particles exhibit both wave-like and particle-like properties. This phenomenon is a fundamental aspect of quantum mechanics and is known as wave-particle duality.
* **Superposition of Waves:** The interference pattern arises from the superposition of waves emerging from the two slits. When the waves meet, they interfere constructively (resulting in bright fringes) or destructively (resulting in dark fringes), depending on the phase relationship between the waves.
* **Quantum Nature of Particles:** The YDSE was originally conducted with light, but it has been extended to particles like electrons. The fact that particles exhibit interference patterns implies that they also possess wave-like characteristics, challenging the classical notion of particles as solid, localized entities.
* **Probability Distribution:** The interference pattern observed in the YDSE is related to the probability distribution of finding a particle at a particular position on the screen. The bright fringes correspond to regions where the probability of finding the particle is higher, while the dark fringes correspond to regions of lower probability.
* **Collapse of the Wavefunction:** The act of measuring or detecting the particle's position causes the interference pattern to disappear. This is explained by the collapse of the particle's wavefunction, resulting in the particle behaving more like a classical particle with a definite position.

In summary, The YDSE provides experimental evidence supporting the wave-particle duality of particles and highlights the probabilistic nature of quantum mechanics. The interference pattern and the subsequent collapse of the wavefunction are key features of quantum phenomena, challenging our classical intuitions about the behavior of matter and light.

1. **Conclusion**

In conclusion, the Young's Double Slit Experiment (YDSE) simulation project has effectively reproduced interference patterns with accuracy. Through systematic parameter variations, the project provided insights into the influence of factors such as slit separation and wavelength on interference patterns. Dynamic visualizations and quantitative analyses added depth to the understanding of wave interference.

The project's achievements lie in accurately depicting interference phenomena, conducting a comprehensive sensitivity analysis, and offering practical implications for real-world applications. Moving forward, potential areas for exploration include extending the simulation to multiple slits, integrating experimental data for validation, and leveraging the project for educational purposes. The YDSE simulation stands as a valuable tool for studying and visualizing wave interference dynamics.

**References**

1. <https://www.pygame.org/docs/>
2. https://www.geeksforgeeks.org/pygame-tutorial/
3. <https://byjus.com/jee/youngs-double-slit-experiment/>
4. <https://www.olympus-lifescience.com/en/microscope-resource/primer/lightandcolor/interference/>
5. Optics by Ajoy Ghatak