Overview.docx

by

Overview:

The program consists of several classes that represent different elements in a stage performance. Here are the implemented features and the purpose of each class:

Light: Represents a light source in the stage setup. It has attributes such as color, position, direction, intensity, light type, and spread. It provides methods to set and get the attributes of the light and calculate the illumination at a given point.

LightGroup: Represents a group of lights. It provides methods to add lights to the group and set attributes for all the lights in the group simultaneously.

SmokeMachine: Represents a smoke machine in the stage setup. It has attributes such as position, direction, intensity, and neighborhood type. It provides methods to emit smoke particles and diffuse the smoke particles based on the machine's attributes.

Prop: Represents a prop on the stage. It has attributes such as position and shape. It provides methods to get the position and shape of the prop.

BandMember: Represents a member of a band on the stage. It inherits from the Prop class and adds attributes such as direction and speed. It provides a method to move the band member.

Backdrop: Represents the backdrop of the stage. It has attributes such as color and image. It provides methods to set the color or load an image as the backdrop. It also provides methods to check if the image is monochrome and apply lighting effects to the image.

Choreography: Represents the choreography of the stage performance. It loads the scene configuration from a file, loads commands from another file, and executes the commands based on their timing.

Implemented Features:

The program can create and manipulate multiple lights, a smoke machine, props, band members, and a backdrop.

The lights can be set with various attributes such as color, position, direction, intensity, light type, and spread. The illumination at a given point can be calculated based on the light's attributes.

Lights can be grouped together, and attributes can be set for all lights in the group simultaneously.

The smoke machine can emit smoke particles and diffuse them based on its attributes.

Props and band members can be created with specific attributes such as position, shape, direction, and speed. Band members can also move.

The backdrop can be set with a color or an image, and lighting effects can be applied to the image.

The choreography class can load the scene configuration and commands from files and execute the commands based on their timing.

Q2 User Guide for the Simulation:

Overview:

The simulation program consists of multiple classes: Light, LightGroup, SmokeMachine, Prop, BandMember, Backdrop, and Choreography. These classes represent different components of a performance stage, such as lights, smoke machines, props/bands, backdrop, and choreography. Each class has specific attributes and methods to control and manipulate its behavior.

Simulation Execution:

To execute the simulation, follow these steps:

2.1. Import the required modules:

```
import numpy as np

from PIL import Image  # Required for the Backdrop class

import json

import time
```

2.2. Create and configure the components:

Create instances of the required classes and configure their attributes as per your scene requirements. For example

```
# Create lights
```

```
light1 = Light("red", [0, 0, 0], [1, 0, 0], 10, "solid", 90)
light2 = Light("blue", [100, 0, 0], [-1, 0, 0], 10, "solid", 90)

# Create a light group and add lights to it
group = LightGroup([light1, light2])
group.set_intensity(11)

# Create a smoke machine
smoke_machine = SmokeMachine(position=[0, 0, 0], direction=[1, 0, 0], intensity=5,
neighborhood_type='Moore')

# Create props and band members
prop = Prop(position=[10, 5, 0], shape="Guitar")
band_member = BandMember(position=[0, 0, 0], shape="Singer", direction=[1, 0, 0], speed=2)

# Create a backdrop
backdrop = Backdrop(color="black")  # Set color or image path as desired

# Create choreography and load the scene
choreography = Choreography()
choreography.load_scene("scene.json")  # Specify the path to the scene file
```

2.3. Define and load the commands:

Define a set of commands that define the choreography of the performance. Each command specifies an object, a method to execute, execution time, and optional arguments. For example:

```
commands = [
   {"object": "backdrop", "method": "set_color", "time": 0, "args": ["red"]},
   {"object": "smoke_machine", "method": "emit_smoke", "time": 2},
   {"object": "light0", "method": "set_intensity", "time": 1, "args": [15]},
   {"object": "prop_band0", "method": "move", "time": 3},
  # Add more commands as needed

]


# Load the commands

choreography.load_commands("commands.json")  # Specify the path to the commands file
```

2.4. Execute the simulation:

Execute the commands in the choreography by calling the execute_commands method of the Choreography class. This will simulate the performance based on the defined choreography. choreography.execute_commands(commands) choreography.execute_commands(commands) Parameter Sweep (if applicable):

If you want to perform a parameter sweep, modify the commands file to include different sets of commands with varying parameter values. Each set of commands represents a different scenario or configuration. Then, load and execute the commands for each scenario to observe the effects of different parameter values.

Additional Considerations:

Make sure to provide the appropriate file paths for the scene file and commands file.

Customize the scene file and commands file to suit your specific simulation requirements. The scene file should contain the configuration of lights, smoke machine, props/bands, and backdrop. The commands file should define the sequence of actions to be performed by each component at specific times.

For the scene file (scene.json), modify the JSON structure to define the attributes of each component. Specify the color, position, direction, intensity, and other relevant parameters for lights, smoke machine, props/bands, and backdrop.

Example scene.json: {

```
 "lights": [

  {

   "color": "red",

   "position": [0, 0, 0],

   "direction": [1, 0, 0],

   "intensity": 10,
  "light_type": "solid",
  "spread": 90
 },
 {
  "color": "blue",
  "position": [100, 0, 0],
  "direction": [-1, 0, 0],
  "intensity": 10,
  "light_type": "solid",
  "spread": 90
 }
],
"smoke_machine": {
 "position": [0, 0, 0],
 "direction": [1, 0, 0],
 "intensity": 5,
 "neighborhood_type": "Moore"
},
"props_bands": [
 {
  "position": [10, 5, 0],
  "shape": "Guitar"
 },
 {
  "position": [0, 0, 0],
  "shape": "Singer",
  "direction": [1, 0, 0],
  "velocity": 2
 }
 ],
 "backdrop": {
  "color": "black"
 }
}
```

For the commands file (commands.json), define the sequence of actions to be performed by each component at specific times. Specify the object (e.g., light, smoke machine, prop/band, backdrop), the method to execute, the execution time, and any required arguments.

```
[
 {
  "object": "backdrop",
  "method": "set_color",
  "time": 0,
  "args": ["red"]
 },
 {
  "object": "smoke_machine",
  "method": "emit_smoke",
  "time": 2
 },
 {
  "object": "light0",
  "method": "set_intensity",
  "time": 1,
  "args": [15]
 },
 {
  "object": "prop_band0",
  "method": "move",
  "time": 3
 }
```

] Customize the attributes, timings, and actions in the scene and commands files to match your desired simulation scenario. Add or remove components, modify their properties, and define different sequences of actions to create unique performances.

By customizing the scene file and commands file, you can create various simulations with different lighting effects, smoke patterns, prop movements, and backdrop changes, providing a versatile and interactive virtual stage experience.

Q3

Feature	**Code Reference(s)**	**Test Reference(s)**	Completion Date
1	Light class (choreography.py)	Light class tests (choreography_tests.py)	
2	SmokeMachine class (smoke.py)	SmokeMachine class tests (smoke_tests.py)	
3	Prop class (props.py) BandMember class (props.py)	Prop and BandMember class tests (props_tests.py)	
4	Backdrop class (backdrop.py)	Backdrop class tests (backdrop_tests.py)	
5	Choreography class (choreography.py)	Choreography class tests (choreography_tests.py)	

Q4:

Q5 To set up and compare the simulations for the showcase, you can follow these steps:

Introduction: Provide a brief description of how you have set up and compared the simulations. Mention any specific commands or input files used to reproduce the results.

"""

In this showcase, I have implemented a choreography simulation using Python classes. The simulation includes lights, smoke machine, props, band members, backdrop, and choreography commands.

To set up the simulation, I have created several classes representing different components of the choreography. Each class has relevant methods and attributes to perform the desired actions.

I will demonstrate three different scenarios in the showcase, including initializing the lights and testing the illumination, emitting and diffusing smoke particles, and moving band members with props.

Let's proceed to the code and showcase the simulation.

"""

Scenario 1: Initializing Lights and Testing Illumination

"""

```
# Create a light
light1 = Light("red", [0, 0, 0], [1, 0, 0], 10, "solid", 90)
print(f"Light1 color: {light1.get_color()}")
print(f"Light1 position: {light1.get_position()}")
print(f"Light1 direction: {light1.get_direction()}")
print(f"Light1 intensity: {light1.get_intensity()}")
print(f"Light1 light type: {light1.get_light_type()}")
print(f"Light1 spread: {light1.get_spread()}")

# Create another light
light2 = Light("blue", [100, 0, 0], [-1, 0, 0], 10, "solid", 90)

# Create a light group
group = LightGroup([light1, light2])
```

```
group.set_intensity(11)  # Increase the intensity of all lights in the group

print(f"Light1 intensity after group set: {light1.get_intensity()}")
print(f"Light2 intensity after group set: {light2.get_intensity()}")

# Test illumination
print(f"Illumination at [50, 0, 0] by light1: {light1.illuminate([50, 0, 0])}")
print(f"Illumination at [150, 0, 0] by light1: {light1.illuminate([150, 0, 0])}")
print(f"Illumination at [50, 0, 0] by light2: {light2.illuminate([50, 0, 0])}")
print(f"Illumination at [150, 0, 0] by light2: {light2.illuminate([150, 0, 0])}")
"""
Scenario 2: Emitting and Diffusing Smoke Particles
"""
# Create a SmokeMachine
smoke_machine = SmokeMachine(position=[0, 0, 0], direction=[1, 0, 0], intensity=5,
neighborhood_type='Moore')

# Emit smoke particles
smoke_machine.emit_smoke()

# Print initial smoke particles
print("Initial Smoke Particles:")
print(smoke_machine.get_smoke_particles())

# Diffuse smoke particles for 10 timesteps
for i in range(10):
    smoke_machine.diffuse_smoke()
    print(f"Smoke Particles after timestep {i + 1}:")
    print(smoke_machine.get_smoke_particles())
```

```
"""

Scenario 3: Moving Band Members with Props

"""

# Create a Prop

prop = Prop(position=[10, 5, 0], shape="Guitar")


# Print the prop's position and shape

print("Prop Position

"""

# Create a Prop

prop = Prop(position=[10, 5, 0], shape="Guitar")


# Print the prop's position and shape

print("Prop Position:", prop.get_position())

print("Prop Shape:", prop.get_shape())


# Create a BandMember

band_member = BandMember(position=[0, 0, 0], shape="Singer", direction=[1, 0, 0], speed=2)


# Move the band member

band_member.move()


# Print the band member's position and shape

print("Band Member Position:", band_member.get_position())

print("Band Member Shape:", band_member.get_shape())

"""
```

Scenario 3: Moving Band Members with Props

In this part, we create a Prop object representing a prop with a position of [10, 5, 0] and a shape of "Guitar". We then print the prop's position and shape.

Next, we create a BandMember object representing a band member with a position of [0, 0, 0], a shape of "Singer", a direction of [1, 0, 0], and a speed of 2. We move the band member using the move() method.

Finally, we print the band member's updated position and shape.

This scenario demonstrates how band members can be moved and their positions and shapes can be accessed using the BandMember and Prop classes.

You can incorporate this code into your showcase and run it to observe the output.

Q6

The assignment involved creating a simulation program for a performance stage. The program consists of several classes that represent different elements of the stage, including lights, smoke machines, props, band members, backdrops, and choreography.

The Light class represents a light on the stage. It has attributes such as color, position, direction, intensity, light type, and spread. The class provides methods to set and get these attributes, as well as a method to calculate the illumination at a given point.

The SmokeMachine class represents a smoke machine on the stage. It emits and diffuses smoke particles based on its position, direction, intensity, and neighborhood type. The class provides methods to emit smoke, diffuse smoke particles, and get the current smoke particles.

The Prop class represents a prop on the stage. It has attributes for position and shape, and provides methods to get these attributes.

The BandMember class represents a band member on the stage. It has attributes for position, shape, direction, and speed. The class provides methods to move the band member and get its position and shape.

The Backdrop class represents the backdrop of the stage. It can have either a color or an image. The class provides methods to set and get the color or image, check if the image is monochrome, and apply lighting effects to the image.

The Choreography class coordinates the elements of the stage. It can load a scene from a file, which includes information about lights, smoke machines, props/bands, and backdrops. It can also load commands from a file, which specify actions to be executed at specific times. The class provides methods to execute the commands and manipulate the objects accordingly.

In the main function, an instance of each class is created, and various methods are called to demonstrate the functionality of the program. The program initializes lights, a smoke machine, props/bands, and a backdrop. It then executes a series of commands based on a choreography file, manipulating the objects on the stage.

Overall, the program provides a flexible and extensible framework for simulating a performance stage. It allows for the creation and manipulation of various elements, such as lights, smoke, props, band members, and backdrops, and provides the ability to define choreography and execute actions at specific times.

The program can be further enhanced by adding more features and functionalities to the existing classes, as well as introducing new classes to represent additional elements of a performance stage.

By using this program, users can easily simulate and visualize different stage setups, experiment with various lighting and smoke effects, and create dynamic choreographies for performances. It can be a valuable tool for artists, stage designers, and event planners to plan and visualize their stage productions.

Q7

Future Work:

Shadows: One possible extension is to introduce shadow casting in the scene. Shadows can add depth and realism to the simulation. This can be achieved by implementing shadow algorithms such as shadow mapping or shadow volumes. Shadows can be cast by lights and props, creating dynamic and interactive shadows in the scene.

Light Falloff: Currently, the illumination calculation assumes a linear falloff of light intensity with distance. However, in reality, light intensity usually follows an inverse square law, where the intensity decreases as the square of the distance. Implementing a more accurate light falloff model can improve the realism of the lighting simulation.

Advanced Smoke Simulation: The smoke simulation can be further enhanced by considering factors such as temperature, air density, and wind. By incorporating these factors, the smoke particles can behave more realistically, creating a more immersive simulation. Advanced smoke simulation techniques, such as fluid dynamics-based simulations, can be explored to achieve more accurate smoke behavior.

Interaction and Controls: Adding user interaction and controls to the simulation can make it more interactive and engaging. For example, allowing users to change the properties of lights, adjust the direction and speed of props/band members, or control the smoke machine can provide a more interactive experience. Implementing a user interface or integrating with input devices can enable users to manipulate the scene in real-time.

Physics-Based Prop Animations: Currently, the movement of props and band members is simple and follows a linear path. Enhancing the animations with physics-based simulations can create more realistic and dynamic movements. Physics engines or algorithms can be used to simulate realistic collisions, gravity, and other physical interactions, adding more life to the scene.

Multi-threading and Performance Optimization: As the complexity of the simulation increases, optimizing performance becomes crucial. Implementing multi-threading techniques can distribute the computational load across multiple threads or processors, improving the overall performance and responsiveness of the simulation. Additionally, profiling and optimizing critical sections of the code can further enhance the simulation's efficiency.

Scene Editing and Customization: Providing tools or an interface for users to create and customize their own choreographies and scenes can extend the application's functionality. Allowing users to import their own props, lights, and backdrops, and defining their choreography through a visual editor or scripting language, can empower users to create unique and personalized simulations.

Advanced Rendering Techniques: To enhance the visual quality of the simulation, exploring advanced rendering techniques such as global illumination, ray tracing, or physically-based rendering can provide more realistic lighting and shading effects. These techniques can improve the overall visual fidelity and create more visually appealing and realistic scenes.

Sound and Music Integration: Adding sound effects and music to the simulation can enhance the overall experience. Syncing the choreography with audio cues can create a synchronized audio-visual spectacle. Integrating with audio libraries or tools to handle audio playback and synchronization can enable the simulation to react dynamically to the audio input.

Virtual Reality (VR) Support: Extending the simulation to support virtual reality can provide an immersive and interactive experience. Users can enter the virtual environment and interact with the scene elements using VR controllers. Implementing VR support can open up new possibilities for exploring the choreography from different perspectives and interacting with the simulation in a more intuitive way.

Overview.docx

ORIGINALITY REPORT

1%	0%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
2	dokumen.pub Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Overview.docx

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14