**Q1: Overview:**

This program consists of multiple classes that represent different components of a stage performance. The following are the implemented functionalities and the purpose of each class:

Lighting: This class represents a light source within the stage setup. It contains properties such as color, position, direction, intensity, light type, and spread. It provides methods to assign and retrieve the attributes of the light, as well as calculate the illumination at specific points.

LightGroup: This class represents a collection of lights. It offers methods to add lights to the group and assign attributes to all lights in the group simultaneously.

SmokeMachine: This class symbolizes a smoke machine used in the stage setup. It encompasses properties like position, direction, intensity, and neighborhood type. It provides methods to emit smoke particles and disperse them based on the machine's attributes.

Prop: This class signifies a prop placed on the stage. It possesses attributes such as position and shape. It offers methods to retrieve the position and shape of the prop.

BandMember: This class represents a member of a band performing on stage. It inherits from the Prop class and introduces additional attributes like direction and speed. It provides a method to move the band member.

Backdrop: This class represents the backdrop of the stage. It includes attributes like color and image. It provides methods to set the color or load an image as the backdrop. Additionally, it offers methods to determine if the image is monochrome and apply lighting effects to the image.

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

Implemented Features:

This program allows users to create and manipulate multiple lights, a smoke machine, props, band members, and a backdrop. Users can customize lights with various attributes such as color, position, direction, intensity, light type, and spread. The program also enables the calculation of illumination at specific points based on the attributes of the lights.

To facilitate organization, lights can be grouped together, and attributes can be assigned to all lights within a group simultaneously.

The smoke machine is capable of emitting smoke particles and dispersing them according to its specified attributes.

Props and band members can be generated with specific attributes like position, shape, direction, and speed. Additionally, band members can be moved during the performance.

The backdrop can be customized by setting a color or loading an image. Users can also apply lighting effects to the image, and determine if it is monochrome.

The choreography class handles the loading of scene configurations and commands from files. It executes the commands based on their timing, ensuring a synchronized stage performance.

**Q2: User Guide for the Simulation:**

Simulation Overview:

The simulation program consists of several classes that represent different elements of a performance stage. These classes, namely Light, LightGroup, SmokeMachine, Prop, BandMember, Backdrop, and Choreography, offer precise control and manipulation of their respective attributes and behavior.

Executing the Simulation:

To execute the simulation, please follow the step-by-step instructions provided below:

Step 1: Import the Required Modules

To begin, import the necessary modules into your program. Please include the following lines of code:import numpy as np

from PIL import Image # Required for the Backdrop class

import json

import time

Step 2: Create and Configure the Components

Create instances of the required classes and configure their attributes based on your specific scene requirements. Here is an example configuration:# 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") # Adjust the color or specify an image path as needed

# Create choreography and load the scene

choreography = Choreography()

choreography.load\_scene("scene.json") # Specify the path to the scene file

2 / 2

Q2: User Guide for the Simulation:

Simulation Overview:

The simulation program consists of several classes that represent different elements of a performance stage. These classes, namely Light, LightGroup, SmokeMachine, Prop, BandMember, Backdrop, and Choreography, offer precise control and manipulation of their respective attributes and behavior.

Executing the Simulation:

To execute the simulation, please follow the step-by-step instructions provided below:

Step 1: Import the Required Modules

To begin, import the necessary modules into your program. Please include the following lines of code:

python

Copy code

import numpy as np

from PIL import Image # Required for the Backdrop class

import json

import time

Step 2: Create and Configure the Components

Create instances of the required classes and configure their attributes based on your specific scene requirements. Here is an example configuration:

python

Copy code

# 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") # Adjust the color or specify an image path as needed

# Create choreography and load the scene

choreography = Choreography()

choreography.load\_scene("scene.json") # Specify the path to the scene file

Step 3: Define and Load the Commands

Define a set of commands that outline the choreography of the performance. Each command specifies an object, a method to execute, the execution time, and optional arguments. Here's an 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

Step 4: Execute the Simulation

Execute the defined choreography by calling the execute\_commands method of the Choreography class. This will simulate the performance based on the specified choreography.choreography.execute\_commands(commands)

Parameter Sweep (if applicable):

If you wish 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 distinct scenario or configuration. Load and execute the commands for each scenario to observe the effects of different parameter values.Additional Considerations:

Ensure that you provide the correct 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 (scene.json) should contain the configuration of lights, smoke machines, props/band members, and backdrops. Modify the JSON structure in the scene file to define the attributes of each component. Specify the color, position, direction, intensity, and any other relevant parameters for each element. Here's an example:{

"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],

"speed": 2

}

],

"backdrop": {

"color": "black"

}

}

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

{

"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

}

]

By customizing the attributes, timings, and actions in the scene and commands files, you can create various simulations with different lighting effects, smoke patterns, prop movements, and backdrop changes. This allows you to tailor the virtual stage experience to your specific needs and achieve unique performances.

**Q3**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Code Reference(s)** | **Test Reference(s)** | Completion Date |
| 1 | Light class (choreography.py) | Light class tests (choreography\_tests.py) | 10/05/2023 |
| 2 | SmokeMachine class (smoke.py) | SmokeMachine class tests (smoke\_tests.py) | 15/05/2023 |
| 3 | Prop class (props.py) BandMember class (props.py) | Prop and BandMember class tests (props\_tests.py) | 17/05/2023 |
| 4 | Backdrop class (backdrop.py) | Backdrop class tests (backdrop\_tests.py) | 20/05/2023 |
| 5 | Choreography class (choreography.py) | Choreography class tests (choreography\_tests.py) | 23/05/2023 |

**Q4: Overview of Classes in the Choreography System:**

Light Class: Within the choreography.py file, the Light class is designed to handle the management of lighting effects throughout the choreography. This class provides methods for modifying properties such as color and intensity, allowing for precise control over the lights. It also includes an initialization method to set default values and additional functionality to adjust these values as needed. Rigorous testing of the Light class is conducted in the choreography\_tests.py file to ensure reliable performance.

SmokeMachine Class: Implemented in the smoke.py file, the SmokeMachine class oversees the operation of a smoke machine throughout the choreography. Its methods enable the activation and deactivation of the smoke, as well as the adjustment of its density and potentially its color. Thorough testing of the SmokeMachine class is performed in the smoke\_tests.py file to validate its functionality.

Prop and BandMember Classes: Found in the props.py file, the Prop and BandMember classes play vital roles in representing various elements on the stage. The Prop class encapsulates properties such as position, size, and specific behaviors associated with the props. Similarly, the BandMember class holds information about band members, including their instruments, stage positions, and available actions. The proper functioning of these classes is thoroughly evaluated through testing in the props\_tests.py file.

Backdrop Class: Within the backdrop.py file, the Backdrop class is responsible for managing the visual appearance and changes of the backdrop during the choreography. This includes the ability to modify the image, color, or other visual aspects associated with the backdrop. Extensive testing of the Backdrop class is conducted in the backdrop\_tests.py file to ensure its reliable performance.

Choreography Class: The Choreography class, located in the choreography.py file, serves as the central orchestrator of all elements within the choreography system. It maintains instances of the Light, SmokeMachine, Prop, BandMember, and Backdrop classes, effectively managing their interactions to produce the desired choreography. Rigorous testing in the choreography\_tests.py file guarantees the seamless integration and synchronized execution of all components within the choreography system.

Visualization of Relationships: In a visual representation, each class (Light, SmokeMachine, Prop, BandMember, Backdrop) would be depicted in its own distinct box, representing its attributes and methods. The Choreography class would be connected to these classes, indicating that it contains instances of them. The relationships between the Choreography class and the other classes would typically be depicted as compositions, denoted by solid diamonds on the side of the Choreography class, as it controls the lifecycle of these instances.

**Q5: Setting up and Comparing Simulations for the Showcase:**

Introduction:

In this showcase, I have developed a choreography simulation using Python classes. The simulation encompasses various components, including lights, a smoke machine, props, band members, backdrop, and choreography commands. In the following steps, I will outline how the simulations were set up and demonstrate three different scenarios to showcase their functionalities.

Scenario 1: Initializing Lights and Testing Illumination

To begin, we initialize the Light class to represent individual lights on the stage. We create instances of the Light class, such as light1 and light2, with different attributes like color, position, direction, intensity, light type, and spread. We use the appropriate methods to retrieve and display these attributes. Additionally, we showcase the utilization of the LightGroup class to group lights and set the intensity for all lights in the group simultaneously. Finally, we calculate and present the illumination at specific points using the illuminate() method.

Scenario 2: Emitting and Diffusing Smoke Particles

Next, we shift our focus to the SmokeMachine class, responsible for managing the operation of a smoke machine. We instantiate a smoke machine object and set its attributes, including position, direction, intensity, and neighborhood type. Our demonstration includes the emission of smoke particles and their diffusion over multiple timesteps. We retrieve and display the current state of the smoke particles after each timestep by employing the appropriate methods.

Scenario 3: Moving Band Members with Props

In this scenario, we utilize the Prop and BandMember classes to represent props and band members on the stage. We create a Prop object with specific attributes like position and shape, and we showcase the retrieval and display of these attributes. Additionally, we create a BandMember object with attributes such as position, shape, direction, and speed. By utilizing the move() method, we simulate the movement of the band member and present the updated position and shape.

Conclusion:

To conclude, the showcased simulations demonstrate the capabilities of the implemented classes, including light control, smoke emission and diffusion, prop representation, and band member movement. By incorporating this code into your showcase and executing it, you can observe the output and explore the functionalities of the choreography simulation.

**Q6: Simulation Program for a Performance Stage**

The task involved developing a simulation program for a performance stage, encompassing various elements such as lights, smoke machines, props, band members, backdrops, and choreography. The program was implemented using several classes that represent these elements and provide specific functionalities.

The Light class serves to represent stage lights, with attributes including color, position, direction, intensity, light type, and spread. It provides methods to set and retrieve these attributes, as well as a calculation method for determining illumination at specific points.

For smoke effects, the SmokeMachine class is introduced, responsible for emitting and diffusing smoke particles based on attributes like position, direction, intensity, and neighborhood type. It offers methods to emit smoke, diffuse particles, and retrieve the current state of the smoke particles.

Props on the stage are represented by the Prop class, featuring attributes for position and shape. Methods are available to access these attributes.

Band members are modeled by the BandMember class, which includes attributes for position, shape, direction, and speed. It provides methods for band member movement and retrieving their position and shape.

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

To coordinate the stage elements, the Choreography class is introduced. It can load a scene configuration from a file, including information about lights, smoke machines, props/band members, and backdrops. It can also load commands from a file, specifying actions to be executed at specific times. Methods are available to execute the commands and manipulate the objects accordingly.

In the main function, instances of each class are created, and relevant methods are called to showcase the program's functionality. The program demonstrates the initialization of lights, a smoke machine, props/band members, and a backdrop. It then executes a series of commands based on a choreography file, manipulating the stage objects accordingly.

Overall, the program offers a flexible and extensible framework for simulating a performance stage, enabling the creation and manipulation of various elements. It provides the ability to define choreography and execute actions at specific times.

To further enhance the program, additional features and functionalities can be incorporated into existing classes, or new classes can be introduced to represent other elements of a performance stage.

The program's utilization allows users to easily simulate and visualize different stage setups, experiment with lighting and smoke effects, and create dynamic choreographies. It proves to be a valuable tool for artists, stage designers, and event planners in planning and visualizing their stage productions.

**Q7: Future Enhancements:**

Shadows: An intriguing expansion for the simulation would be the incorporation of shadow casting. Shadows contribute depth and realism to the scene, and implementing shadow algorithms like shadow mapping or shadow volumes can achieve this effect. By allowing lights and props to cast dynamic and interactive shadows, the simulation's visual fidelity can be significantly enhanced.

Light Falloff: Presently, the illumination calculation assumes a linear falloff of light intensity with distance. However, in reality, light intensity follows an inverse square law, diminishing as the square of the distance. Implementing a more accurate light falloff model based on the inverse square law would improve the authenticity of the lighting simulation.

Advanced Smoke Simulation: The smoke simulation can be elevated by considering additional factors such as temperature, air density, and wind. Incorporating these factors can render the behavior of smoke particles more realistic, resulting in a more immersive simulation. Exploring advanced smoke simulation techniques, like fluid dynamics-based simulations, can provide precise and lifelike smoke behavior.

Interaction and Controls: Adding user interaction and controls to the simulation can enhance its interactivity and engagement. For instance, enabling users to modify light properties, adjust the direction and speed of props and band members, or control the smoke machine would provide a more interactive experience. Implementing a user interface or integrating with input devices would enable real-time manipulation of the scene.

Physics-Based Prop Animations: Currently, prop and band member movements follow simple linear paths. Enhancing animations with physics-based simulations can yield more realistic and dynamic movements. By leveraging physics engines or algorithms, it becomes possible to simulate realistic collisions, gravity, and other physical interactions, thereby infusing vitality into the scene.

Multi-threading and Performance Optimization: As the simulation's complexity grows, optimizing performance becomes crucial. Implementing multi-threading techniques can distribute computational load across multiple threads or processors, leading to improved overall performance and responsiveness. Profiling and optimizing critical code sections can further enhance the simulation's efficiency.

Scene Editing and Customization: Offering users tools or interfaces to create and customize their own choreographies and scenes would expand the application's functionality. Allowing users to import their own props, lights, and backdrops, as well as defining choreography through a visual editor or scripting language, would empower users to create unique and personalized simulations.

Advanced Rendering Techniques: Exploring advanced rendering techniques like global illumination, ray tracing, or physically-based rendering can elevate the simulation's visual quality. These techniques provide more realistic lighting and shading effects, resulting in visually captivating and lifelike scenes.

Sound and Music Integration: Integrating sound effects and music into the simulation would enhance the overall experience. Synchronizing the choreography with audio cues can create a captivating audio-visual spectacle. Integration with audio libraries or tools to handle audio playback and synchronization would enable the simulation to dynamically respond to audio inputs.

Virtual Reality (VR) Support: Expanding the simulation to support virtual reality would offer an immersive and interactive experience. Users could enter the virtual environment and interact with scene elements using VR controllers. Implementing VR support would enable users to explore the choreography from various perspectives and intuitively interact with the simulation.

These future enhancements would further elevate the simulation program, providing users with more realistic, interactive, and customizable experiences