Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:		Student ID:	
Other name(s):			
Unit name:		Unit ID:	
Lecturer / unit coordinator:		Tutor:	
Date of submission:		Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: [illegible] Date of signature:

(By submitting this form, you indicate that you agree with all the above text.)

1. Overview (2 marks) describe your program's purpose and implemented features.

The program simulates a stage that has moving lights, moving smoke machines, moving images(called objects) and a background. The light interacts with the smoke, objects and background. Objects and Background are imported png's. Everything can be controlled from an Excel file with several sheets.

2. User Guide (2 marks) how to use your simulation

Dependencies:

- matplotlib 3.7.1
- numpy 1.24.3
- opencv-python(also known as cv2) 4.7.0.72
- pandas 2.0.1

The program is controlled entirely by an excel file with several sheets. There must be a sheet labeled "init" that sets up each element of the simulation. To create something, add the type to the "Type" column, give it a name, then fill out the initial conditions. The types don't have to be grouped but it is more legible when they are. Any row without a type won't be read. Image Location and Background Image Location should be string paths to the relevant image.

For each element of the simulation except the stage there must be a sheet that's named exactly what's written as the name of the element. This sheet controls what the object does in the simulation. The first column is the instruction. It must be accompanied by an end time, which is the time that the instruction ends). Each instruction will run at least once.

The available instructions are:

- Move To: Will move to a specific state, reaching it when end time is reached. All columns except loop index must be filled out. Everything except color is smoothly interpolated. To move instantly set an end time of 0
- Loop To: Go to and execute a specific line. The internal time will be set to the endtime of the previous instruction(0 if first instruction). The loop index is two less than the excel row. End time is irrelevant
- Hold: The element will not change until the end time is elapsed.
- Stop: The element will not change again.
- End: Ends the entire program

To run the file, either rename it to be "Choreography.xlsx" and put it in the same folder as the program, or edit the program to change choreographyLocation to be the path of the new choreography file.

3. Traceability Matrix (10 marks) of features, implementation and testing of your code. The matrix should be a table with columns for: i. Feature - numbered for easy referencing ii. Code reference(s) –

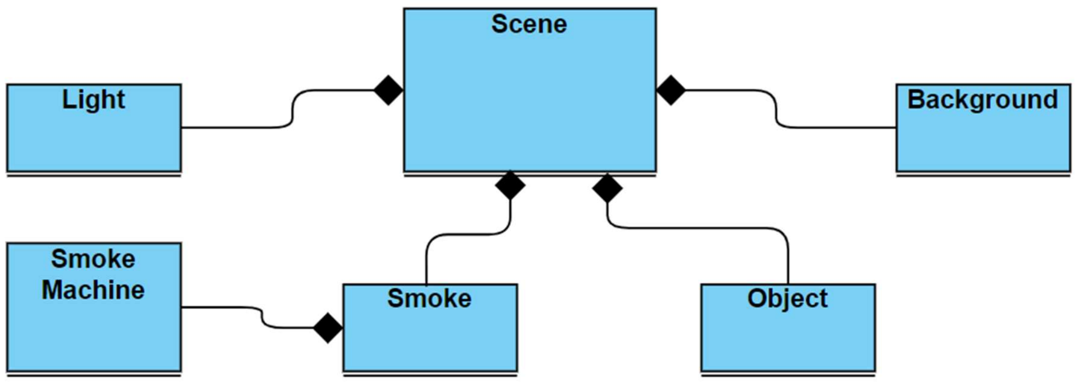
reference to files/classes/methods or snippets of code only, do not put the whole program in the report iii. Test reference(s) – test code or describe how you tested your feature was correctly implemented iv. Completion date - N/A if not implemented

#	Feature	Code Reference	Testing	Completion Date
1	Light object generates light screen	Light.getNewLightScreen()	Used plt.im_show() to display lightscreen for a few manually entered lights	8-5-2023 Optimised to run ~3000x faster(~3s to ~0.001s) on 15-5-2023
2	Light obeys choreography	Light.update()	Created a stage with only a light and ensure it did what it was told to	20-5-2023
3	Smoke Machine emits smoke	SmokeMachine.setSmoke()	Used plt.im_show() to display smoke intensity after machine emitted	12-5-2023 Removed on 18-5-2023 as ran slowly, didn't spread effectively, and direction couldn't be controlled
4	Smoke Machine emits smoke particles	SmokeMachine.getNewSmoke()	Printed out smoke particle details, ensuring they are in the expected ranges	18-5-2023
5	Smoke Machine Obeys Choreography	SmokeMachine.update()	Created a stage with only a smoke machine and ensured it did what it was told to	21-5-2023
6	Smoke Controller generates velocity screen	Smoke.getNewVelocityScreen()	Used plt.im_show() to display the x value and then the y value of the velocity	18-5-2023
7	Smoke Controller moves particles, gets new particles, and calculates	Smoke.updateSmokeScreen()	Used plt.im_show() to display the intensity of smoke at each point in the screen with a smoke machine to keep generating more smoke	18-5-2023
8	Object generates correct object screen with itself rotated, rescaled and positioned	Object.getObjectScreen()	Used plt.im_show() to show display the object screen to ensure everything is correct.	13-5-2023
9	Object obeys choreography	Object.update()	Created a stage with only an object and ensure it did what it was told to	20-5-2023
10	Lights, Smoke Machines and Objects can be added to and	Scene.addLight()	Created a stage and added a light to it, then printed out the list of lights, the list of objects	14-5-2023

	stored by the stage		and the list of smoke machines	
11	Stage gets all the information from the lights and generates the light display	Scene.getLightPatchCollection()	Created a stage with a few lights and then used plt.add_collection() to display the light display	22-5-2023
12	Stage can get and store a backdrop	Scene.setBackground()	Created a stage, set the backdop, displayed with plt.imshow()	17-5-2023
13	Stage gets all the relevant screens, combines them, and generates the output	Scene.render()	Created a stage with lights and smoke machines and a backdrop and	19-5-2023
14	Choreography file path is read in from the command line	N/A	N/A	N/A
15	Stage and stage elements setup based on choreography	spinal-tap.py Lines 20 to 85	Gave a variety of input files and displayed outputs to ensure it set each element up correctly	22-5-2023
16	Constantly update and display a new frame, with elements obeying choreography	spinal-tap.py Line 90 to 115	Gave a variety of input files and displayed outputs to ensure it set each element up correctly	22-5-2023

4. Discussion (10 marks) of implemented features (referring to the Traceability Matrix), explaining how they work and how you implemented them. UML Class Diagrams should be included for objects and their relationships.

The program works through generating and combining screens. A screen is a 2d array of values or another array. The first two dimensions represent the horizontal and vertical position. It then contains information about that point on the screen, such as the intensity of smoke or the colour of light present there. Screens are always numpy arrays so elementwise operations can be done very efficiently.

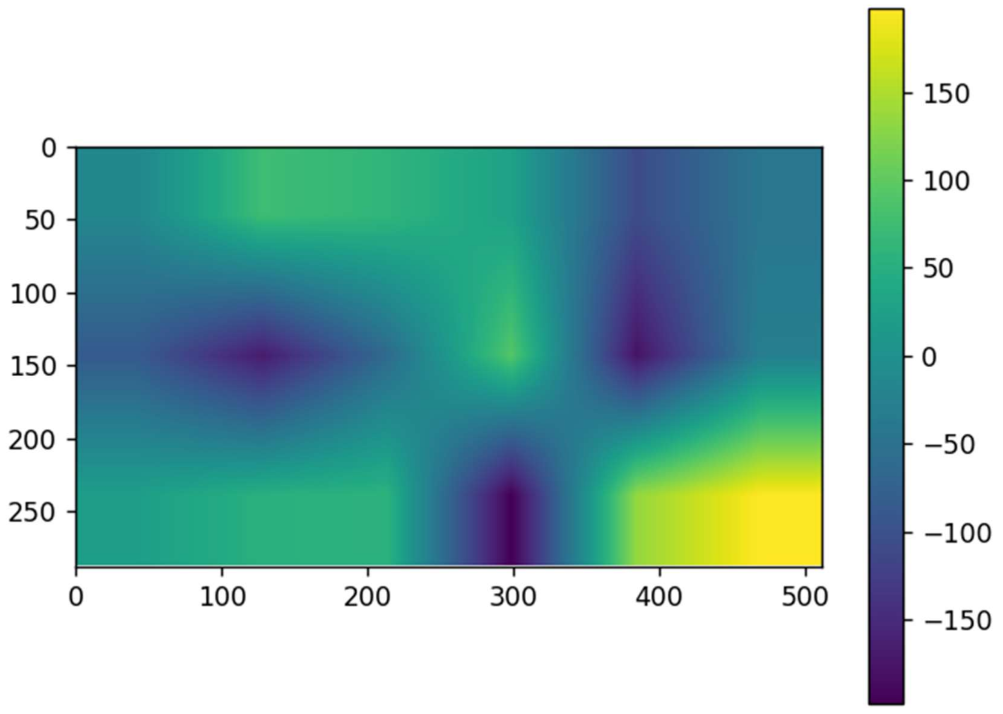


Simply put, the stage gets light screens (what light is where), a smoke screen (how much smoke is where), a background/ screen and an object screen. These are then merged together to produce the final frame.

The program starts by generating all the light screens (one per light), which contain the colour of light at each pixel using (1). It uses the spread angle and direction of each light to calculate how much the light cone edges should move to the left and right for each pixel down it travels. It then stores the left and right edge position, and loops through each row, setting the pixels between the left and right edge to be the lights colour (everything is initially black (0, 0, 0)) and spreading the left and right edges out. Originally (pre 15-5-2023) it checked the angle of every pixel individually to the light, which ran very slowly (around 3s per frame for a few lights). The current method is much faster (around 0.001 for same number of lights) because it does an entire row at once and has less computationally intensive math (addition instead of trigonometry). All the light screens are then summed elementwise to produce a cumulative light screen.

Next a smoke screen is generated. The smoke screen is an array of the intensity of smoke at each pixel. Originally an area around the smoke machine in the previous smoke screen was set to an intensity (3). The intensities where then averaged using an extended Moore neighbourhood. This also ran extremely slowly, spread out uniformly, had no directional control and looked bad. This was replaced with a particle based system.

The first step is to generate a velocity screen (6), which contains a velocity representing the airflow at each point in the room. First a much smaller array is generated completely randomly. Then cv2 is used to resize (with cv2.resize()) it and smoothly interpolate between each point generating a smooth but random set of velocities.



Screenshot of x values of velocity screen. Y values are similar.

Using (4) a set of particles are generated for each smoke machine, then stored in the smoke object. They are then moved around with (5). Each frame they lose a little bit of their previous velocity and gain some from the velocity screen, and some from randomly generated values. Any particles that move outside the screen are deleted. Particle intensities are also slowly reduced from 1 to 0 over ten seconds.

The smoke screen is initiated with values of 0. Everywhere there is a particle, the surrounding cells have the intensity of the particle added to them. The intensity of the smokescreen is then diffused with Moore neighbourhoods using cv2.blur() which runs much faster and does the same thing as the blurring function used previously. The smoke screen intensities are then clipped to values between 0 and 1.

Next the background screen is gotten. Since the background is a png, the background screen is the image with the alpha mask stripped such that it’s opaque. This is then merged with the object screens. which has the object in the correct position on the screen with its alpha channel. The alpha channel is separated, multiplied with the object screen, then one subtract the alpha channel is multiplied with the background screen, then the two are added together. This is repeated for each object, creating a screen with the backdrop and all the objects.

There is now three screens, light, smoke and backdrop/object. In the real world, light could either be reflected by the smoke or the backdrop, but all light eventually goes somewhere. It is assumed that all the light reflects then goes to the camera. This isn't true in the real world, but it's a fair assumption since effectively a fraction based on distance is reflected into the camera and the stage is close to the same distance from the camera everywhere. When light is reflected off of a coloured surface, the reflected light is the incident light subtract the absorbed light. This can be approximated by multiplying the incident light with the colour of the surface, because all colours are stored as values between 0 and 1. Smoke can be assumed to be white, so reflects the incident light exactly.

Hence, to produce the output image, the smoke and light screens are multiplied together to produce a lit smoke screen. The remaining light is one take the smoke screen multiplied with the light screen. This is then multiplied with the backdrop/object screen to produce to produce a lit backdrop/object screen. Finally, the lit backdrop and lit smoke screen are summed together to produce the output screen.

Sometimes this can result in a colour value greater than one, mostly when two lights of the same colour overlap on a white or smoky section. In real life this is solved by the pupil contracting so everything gets darker. This can be represented by dividing the output by the maximum output colour value (provided its greater than 1). This is the final output frame.

To get the light display uses matplotlib patches. For all the lights it gets the details and sets up a sector using the wedge patch. It also creates a parallelogram for the laser.

5. Showcase (10 marks) of code output, including three different scenarios: a. Introduction: Describe how you have chosen to set up and compare the simulations for the showcase. Include commands, input files – anything needed to reproduce your results. b. Discussion: Show and discuss the outputs/results of each scenario.

Included in the zip are three mp4 recordings of different situations, Choreography, Choreography2 and Choreography3. Each is paired with a .xlsx of the same name that contains the instructions that the program read to generate what is shown in the video.

Choreography is a demonstration of basically every element, every instruction.

Choreography2 is a demonstration of specifically the smoke machine and smoke, showing how it looks.

Choreography3 is a demonstration of specifically the lights, showing how they interact with each other.

6. Conclusion (2 marks) reflection on your assignment with respect to the specification

While the program isn't perfect and doesn't implement every feature, it does implement a wide array of features effectively. Through efficient application of object-oriented programming, bugs have been minor and limited to new features. Furthermore, it has remained straightforward to add new features. I have personally gained an understanding of a variety of new libraries that haven't been taught to me explicitly.

7. Future Work (2 marks) further investigations and/or extensions that could follow.

Currently the laser width is based on its horizontal width. This means as the laser becomes closer to horizontal, it looks thinner. This could be resolved by dynamically setting the horizontal width of the laser based on the desired width and current angle.

Theoretically lights could have a width and a spread angle, this isn't tested. It would likely work in the main view, but the light view would likely display incorrectly. This could be resolved with a few additional checks to determine if its needed and how to do it.

Movement currently has no acceleration, so is very jerky. Smoothing could be added to look more natural.

Currently the file to read the choreography from is hard coded into the program, it could be entered as a command line argument or a text input.

Currently only supports png inputs, could be made to support other formats of images.

Lights can't fade between colours, can't be told to follow another light, can only be given one color at a time and only display as a cone. All of these could be implemented.

The lights, smoke machine and objects could all inherit from a more general object that contains things like the position and much of the choreography are the same across each of them so should be put into a parent class.

8. References (2 marks)

- backdrop.png – HDFootageStock.com, 2016, “Spine health care abstract dark colorful background. Human spine anatomy with red color flashing circles at the middle of the spine and transparent hexagons at the background.” HDFootageStock. https://hdfootagestock.com/video/21011863/spine-health-care-abstract-dark-colorful-background-human-spine-anatomy-with-red-color-flashing-circles-at-the-middle-of-the-spine-and-transparent-hexagons-at-the-background
- guitar.png – 365psd.com, n.d. “Flying V guitar free vector by 365psd.com.” freeimages.com, https://www.freeimages.com/vector/flying-v-guitar-5406261
- drum.png – pngtree.com, n.d. “drum PNG Image, Vector Drums, Drum Clipart, Vector Instrument.” pinterest.com.au, https://www.pinterest.com.au/pin/696298792365765081/