Project Report: Canopy Simulation

Name: Yasiru Fernando

Student Number: 22345563

Date: 2024/10/14

Contents

1.0 Overview

The Canopy Simulation program shows normal and thermal imaging of a particular landscape. The programme depicts how temperature changes in the set landscape according to different times of the day.

Taking the COMP1005 Practest3 code into inspiration, a full RGB landscape view is built, complete with forests, parks, houses and apartments. Each of these objects possesses its unique thermal properties, and the programme depicts how their temperature changes according to these unique thermal properties and external influences such as atmospheric temperature cooling and heat diffusion.

The programme is built to be user-friendly, with proper commenting for easy understanding throughout the code. A proper error handling system is implemented throughout the code, ensuring the code functions properly despite running into an error. Command-line arguments are built into the program, allowing users to access a specific scenario with pre-set configurations easily. At the end of each scenario, the user will be provided with several graphs and a table consisting of data about temperature values and their behaviours.

The three scenarios depicted in this program are:

1. Normal map
2. Rain map
3. Snow map

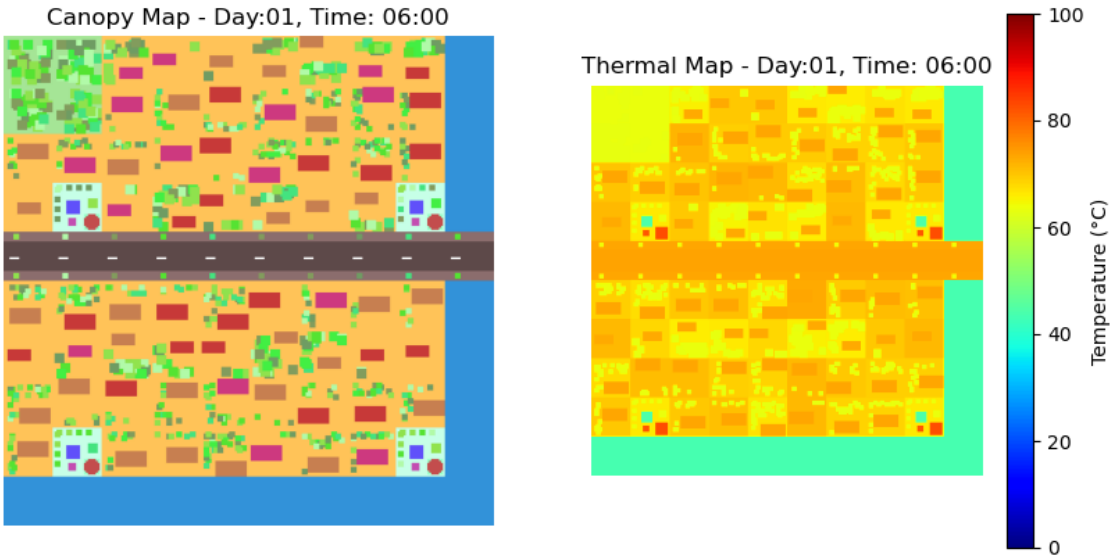


Figure 1: Canopy map simulation

2.0 User Guide

2.1 Requirements

The Canopy Simulator requires the following four files to run. These four files must be placed in the same directory:

1. map.py
    - The main code file consists of code for simulations and data plotting.

2. object_placement.py
    - The file contains code for block and item placement in the map, some hard-coded, some randomly.

3. object.py
    - The file containing all the classes used in the program.

4. colours.csv
    - File containing colour values for trees and houses.

Moreover, a collection of Python libraries will be required for the program to function properly. The said libraries are as follows:

1. Random module
2. Numpy
3. Matplotlib
4. Math
5. Argparse
6. CSV
7. OS

It is important to ensure that these libraries, modules, and Python and pip are installed globally or within a virtual environment.

2.2 Running the programme

To run the programme, open the folder directory with the desired IDE and run the map.py file:

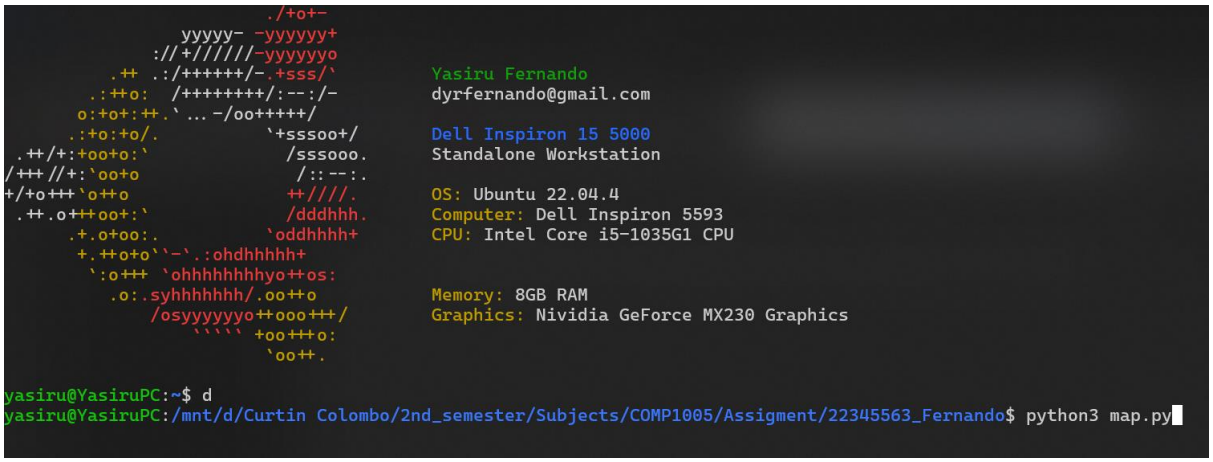


Figure 2: Running the program

This will execute the programme, allowing the user to set up a scenario of their choosing.

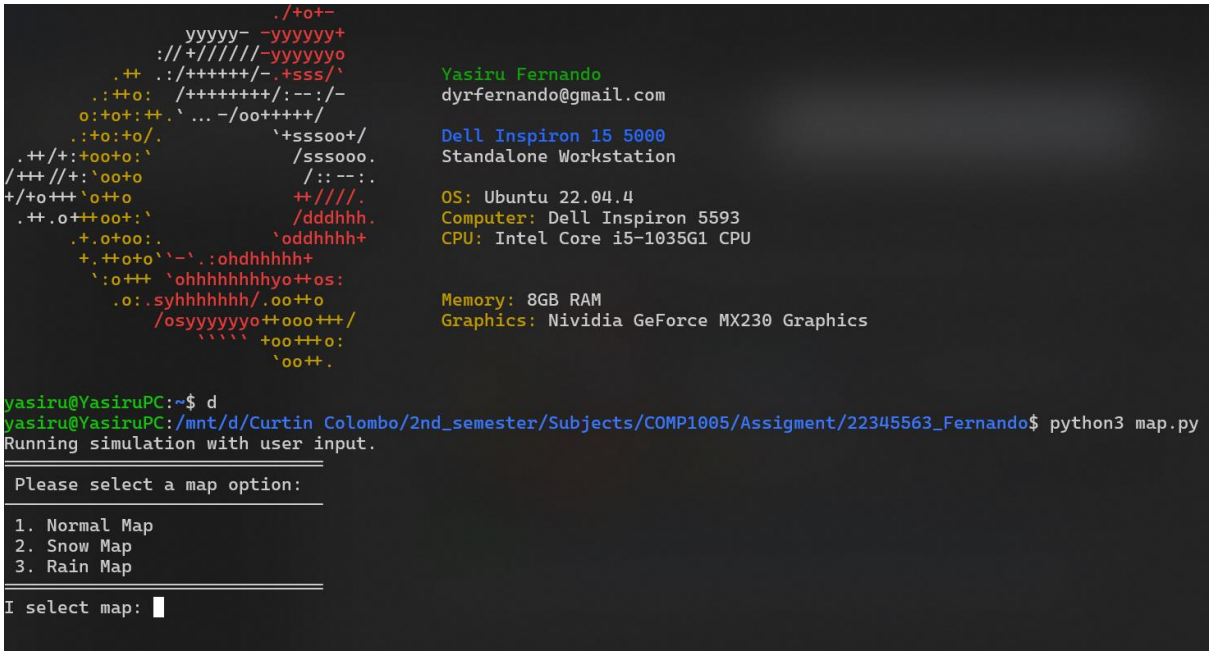


Figure 3: Running the program with user inputs

The user is free to set map configurations as they please, provided they are valid inputs. Error handling is in place to ensure the program does not crash in case of an invalid input.

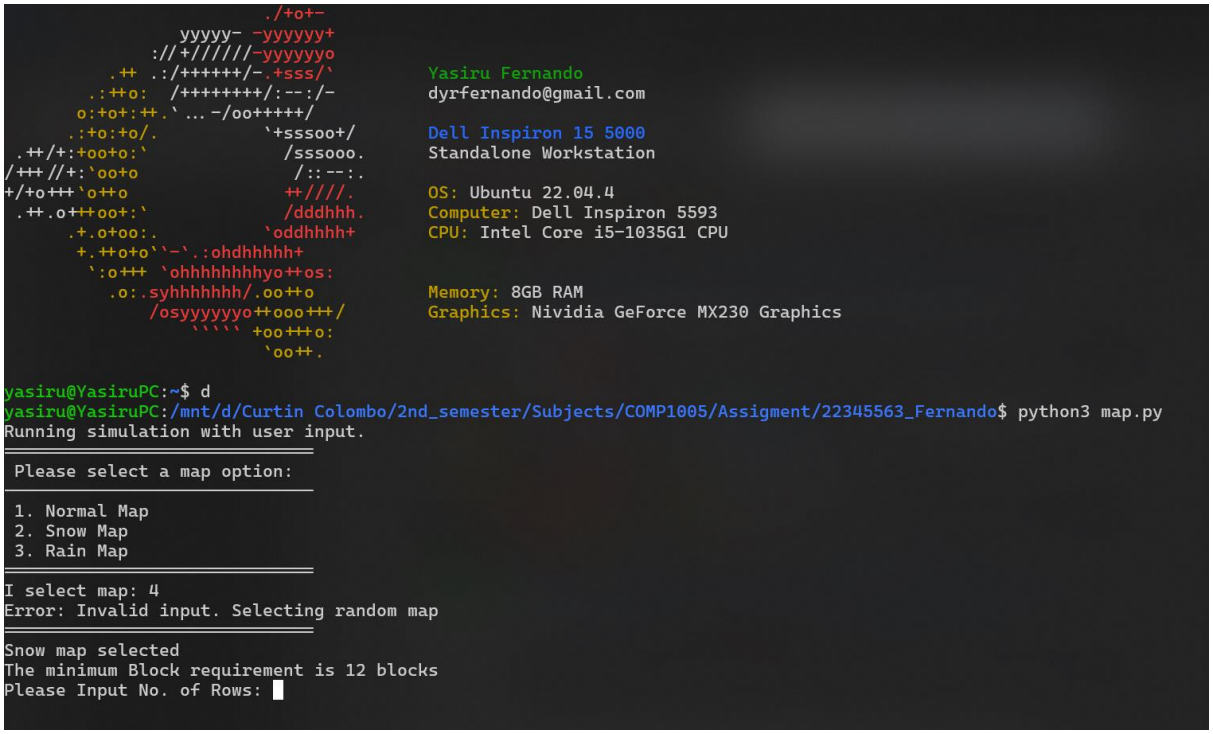


Figure 4: Handling invalid inputs

If the user intends to run the programme through command line arguments, type the file name followed by the scenario desired. The scenarios are named:

1. normal – the normal map with day and night RGB and thermal map simulation.
2. snow – the map that depicts a snowing scenario and the relevant temperature changes.
3. rain – the map that depicts a raining scenario and the relevant temperature changes.

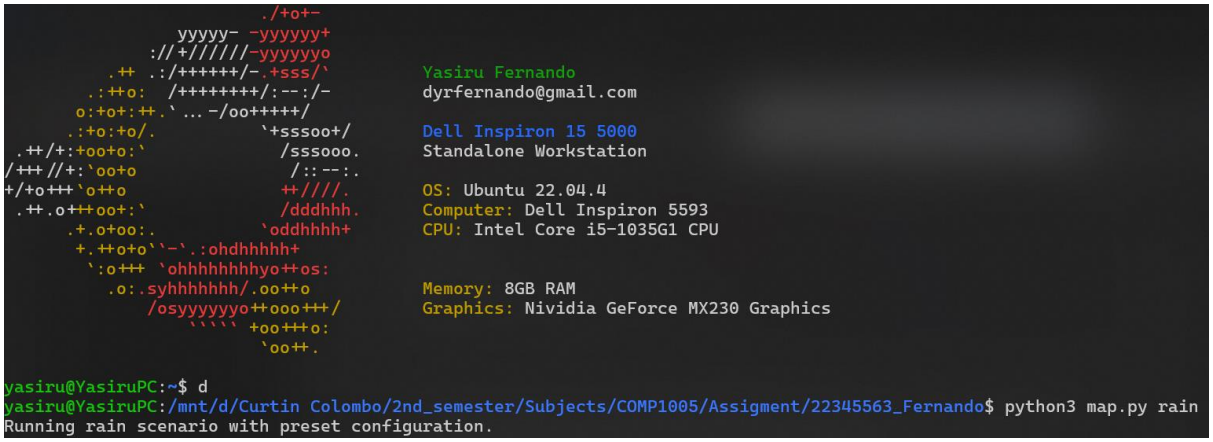


Figure 5: Running program with system arguments

3.0 Traceability Matrix

<table>
<tr><th>Features</th><th>Code Reference</th><th>Test Reference</th><th>Completion Date</th></tr>
<tr><td colspan="4">1. Classes</td></tr>
<tr><td>1.1 Object class</td><td>Line 7 – 63 in objects.py

(Acts as a super-class for all other objects)</td><td rowspan="10"><u>[Passed]</u>

Each object is successfully implemented to the map in ‘object_placement.py’.

Each object is displayed as desired in the main map.</td><td rowspan="10">2024/9/24</td></tr>
<tr><td>1.1.1 Tree</td><td>Line 65-67 in objects.py</td></tr>
<tr><td>1.1.2 House</td><td>Line 69-83 in objects.py</td></tr>
<tr><td>1.1.3 Apartment</td><td>Line 85-99 in objects.py</td></tr>
<tr><td>1.1.4 Street</td><td>Line 101-103 in objects.py</td></tr>
<tr><td>1.1.5 White_lines</td><td>Line 105-107 in objects.py</td></tr>
<tr><td>1.1.6 Pond</td><td>Line 109-111 in objects.py</td></tr>
<tr><td>1.1.7 Bushes</td><td>Line 113-115 in objects.py</td></tr>
<tr><td>1.1.9 MerryGo</td><td>Line 117-133 in objects.py</td></tr>
<tr><td>1.1.10 Slide</td><td>Line 138-140 in objects.py</td></tr>
<tr><td>1.1.8 Tanks</td><td><pre>class Tank(Object):
    def __init__(self, pos, colour, size):
        super().__init__(pos, colour, (size, size), heat_val = 70)

    def get_image(self):
        image = np.full((self.size[0], self.size[1], 3), self.colour)

        # Creating a shape.
        y, x = np.ogrid[:self.size[0], :self.size[1]]
        center = (self.size[0] / 2-0.5, self.size[1] / 2-0.5)
        radius = np.sqrt((x - center[0])**2 + (y - center[1])**2)
        mask = radius <= min(self.size[0], self.size[1]) // 2

        #applying to the square.
        image[~mask] = [153, 255, 204]

        return image</pre>
(Removed from Object.py)</td><td><u>[Failed]</u>

Several issues occurred after trying to implement it on the rooftops of houses. It was decided to remove this object from the code.

(test-reference not available)</td><td>Not-Implemented</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>1.2 Block class</td><td>Line 143-194 in objects.py
(Acts as a super-class for all other blocks)</td><td rowspan="4"><u>[Passed]</u>

Each block is successfully implemented to the map in ‘object_placement.py’.</td><td rowspan="4">2024/9/24</td></tr>
<tr><td>1.2.1 Ground</td><td>Line 197-199 in objects.py</td></tr>
<tr><td>1.2.2 Playground</td><td>Line 201-203 in objects.py</td></tr>
<tr><td>1.2.3 Forest</td><td>Line 205-207 in objects.py</td></tr>
</table>

1.2.4 Water	Line 209-211 in objects.py	Each block is displayed as desired in the main map.	
1.2.5 Road	Line 213-215 in objects.py		
1.3 Circles	(*Circles was created as a different shape type)*		
1.3.1 MerryGo	Line 117-133 in objects.py	**[Passed]** The MerryGo square was successfully converted into a circle and is implemented in the main map.	2024/10/6
1.4 Overlay handling	(*The issue of House and Apartment being overlayed by trees is handled)*		
1.4.1 House	Line 69-83 in objects.py *Implementation:* Line 86-91 in object_placement.py	**[Passed]** The Trees and Bushes no longer overlap the houses and apartments when their positioning is set to random.	2024/10/12
1.4.2 Apartment	Line 85-99 in objects.py *Implementation:* Line 107-112 in object_placement.py		

2. Canopy map generation			
2.1 RBG map generation			
2.1.1 3D-Array classes		**[Passed]**	2024/9/24
• Items	Line 7-63 in objects.py	Object classes with RGB colours are created and implemented in the main map.	
• Blocks	Line 143-194 in objects.py	Block classes that have RGB colours are created and implemented in the main map.	
2.1.2 Implementation	Line 8 – 64 in map.py (‘*generate_image’ function)*	**[Passed]**	

	Line 357-364 in map.py *(Subplots the RGB map)*	Generates the main map to consist of RGB colours.	
2.2 Block placement			2024/9/26
2.2.1 Ground	Line 35-37 in object_placement.py	**[Passed]** Each block class has been successfully placed in the desired positions of the main map.	
2.2.2 Playground	Line 135-143 in object_placement.py		
2.2.3 Forest	Line 50 – 55 in object_placement.py		
2.2.4 Water	Line 40-45 in object_placement.py		
2.2.5 Road	Line 116-123 in object_placement.py		
2.3 Item placement			
2.3.1 Tree	Line 59-63 in object_placement.py *(Adding trees to the forest)* Line 78-91 in object_placement.py *(Adding trees with the houses)* Line 155 in object_placement.py *(Adding trees to the playground)*	**[Passed]** Each item class has been successfully placed in the desired positions of the main map	2024/9/28
2.3.2 House	Line 71-77 in object_placement.py *(Adding houses to ground blocks)*		
2.3.3 Apartment	Line 94-98 in object_placement.py *(Adding apartments to ground blocks)*		
2.3.4 Street	Line 126 in object_placement.py *(Adding the street onto the road block)*		
2.3.5 White_lines	Line 127 in object_placement.py *(Adding white lines for the road)*		2024/9/29
2.3.6 Pond	Line 156 in object_placement.py *(Adding pond to park block)*		
2.3.7 Bushes	Line 107-112 in object_placement.py *(Adding bushes to blocks with houses)*		

		Line 129-130 in object_placement.py *(Adding bushes to the road)* Line 145-151 in object_placement.py *(Adding bushes to park block)*		
2.3.8	MerryGo	Line 152 in object_placement.py *(Adding MerryGo to park block)*		
2.3.9	Slide	Line 153 in object_placement.py *(Adding Slide to park block)*		2024/9/30
2.4	Day and night cycle	Line 54-57 in objects.py – day_night function (*Sets up brightness change for objects)* Line 180-184 in objects.py – day_night function (*Sets up brightness change for blocks and items within it)* Line 12 in map.py *(Calculates time factor based on the hour of the day)* Line 20-24 in map.py *(Adjusting block and item brightness based on time)*	**[Passed]** A clear difference between day and night can be identified on the main map.	2024/10/7

3. Thermal map generation				
3.1	Generating Thermal map	Line 118-139 in map.py- generate_thermal_image function	**[Passed]** A thermal view of the map is successfully generated.	2024/9/25

		(Generates the thermal image of the map) Line 367-371 in map.py *(Subplots the thermal view map)*		
3.2	Thermal equation	Line 108-115 in map.py – thermal_equation function *(Equation to calculate block and item heat change with time)*	**[Passed]** Items and blocks change their temperature values according to a sine curve with passing time.	2024/10/5
3.3	Heat diffusion	Line 67-103 in map.py – heat_diffusion function *(Simulates heat diffusion over time)* Line 300 in map.py *(Applies heat diffusion to blocks)*	**[Passed]** Heat diffusion is somewhat visible in each block	2024/10/10

4. CSV Reading.				
4.1	Reading a CSV file	Line 8- 28 in object_placement.py – csv_read function *(reads CSV data and appends them onto a list)* Line 162 in object_placement.py *(Reading data from CSV file)*	**[Passed]** Successfully read data from the CSV file. Read data is displayed in the main map.	2024/10/13
4.2	Applying CSV file data	Line 61 in object_placement.py Line 75 in object_placement.py Line 87 in object_placement.py Line 96 in object_placement.py Line 108 in object_placement.py	**[Passed]** CSV data, which essentially consists of colours for houses and trees, is successfully applied to the main map.	2024/10/13

		Line 128 in object_placement.py Line 148 in object_placement.py Line 154 in object_placement.py *(Applying CSV data)*		

5. Command Line Arguments				
5.1	Setting up scenario	Line 458-496 in map.py *(Setting up the 3 different scenarios)*	**[Passed]** When a command line argument is given, the scenario is successfully activated	2024/10/12
5.2	Calling scenario	Line 498-523 in map.py *(Calling scenarios)*		

6. Inputs				
6.1	User inputs	Line 144-235 in map.py – user_inputs function *(Setting up user inputs)*	**[Passed]** Successfully processes user inputs and applies them.	
6.1.1	Adding row & column number	Line 185-210 in map.py		2024/10/4
6.1.2	Map type	Line 152-182 in map.py		2024/10/5
6.1.3	Adding forest	Line 222-226 in map.py		2024/10/6
6.1.4	Adding parks	Line 229-233 in map.py		2024/10/6

7. Error Handling				
7.1	User input error handling			
7.1.1	Row and column input	Line 185-210 in map.py	**[Passed]**	2024/10/4

7.1.2	Map type error handling	Line 15x9-182 in map.py	Each error handling works as intended.	2024/10/5
7.1.3	Adding forest	Line 222-226 in map.py		2024/10/6
7.1.4	Adding parks	Line 229-233 in map.py		2024/10/6
7.1.5	Command line	Line 504 – 523 in map.py		2024/10/12
7.2	CSV error handling			
7.2.1	CSV	Line 16-26 in object_placement.py		2024/10/13

8. Printing Data				
8.1	Storing data	Line 123-124 in map.py *(Dictionaries to store data)* Line 396-401 in map.py *(Storing data)*	**[Passed]** Successfully stores data	2024/10/8
8.2	Calculations	Line 382-387 in map.py *(Calculating real temperature)* Line 390-393 in map.py *(Calculating average given temperature)*	**[Passed]** Successfully calculates data for storing	2024/10/8
8.3	Plotting	Line 409 – 417 in map.py *(Real vs Depicted temperature)* Line 420-428 in map.py *(Block temperature)*	**[Passed]** Successfully prints 4 plots and one table at the end of the simulation	2024/10/8

		Line 431-439 in map.py *(Item temperature)* Line 442-455 in map.py *(Average temperature bar chart)* Line 535-547 in map.py *(Summary table)*		

Scenarios				
9. Scenario - Rain				
9.1	Creating and applying rain	Line 238-239 in map.py *(Generating rain)* Line 242-244 in map.py *(Applying rain)*	**[Passed]** Rain successfully occurs in the map	2024/10/10
9.2	Flooding	Line 27-43 in map.py *(Handling flooding)* Line 287-289 in map.py *(Duration of flooding)* Line 332-353 in map.py (*Applying rain)*	[**Passed]** Flooding occurs successfully.	2024/10/10

10. Scenario - Snow				
10.1	Creating and applying snow	Line 247-248 in map.py *(Generating snow)* Line 251-253 in map.py *(Applying snow)*	**[Passed]** Snow successfully occurs on the map	2024/10/11
10.2	Snowing	Line 46-61 in map.py *(Handling snowing)*	[**Passed]**	2024/10/11

		Line 291-293 in map.py *(Duration of flooding)* Line 305-330 in map.py (*Applying rain)*	Snow successfully covers the desired areas of the map.	
10.3	Temperature handling	Line 256-271 in map.py *(Handles temperature to make it hotter inside the houses)*	[**Passed]** The temperature inside house and apartments are higher in comparison to the rest of the map.	2024/10/11

4.0 Discussion

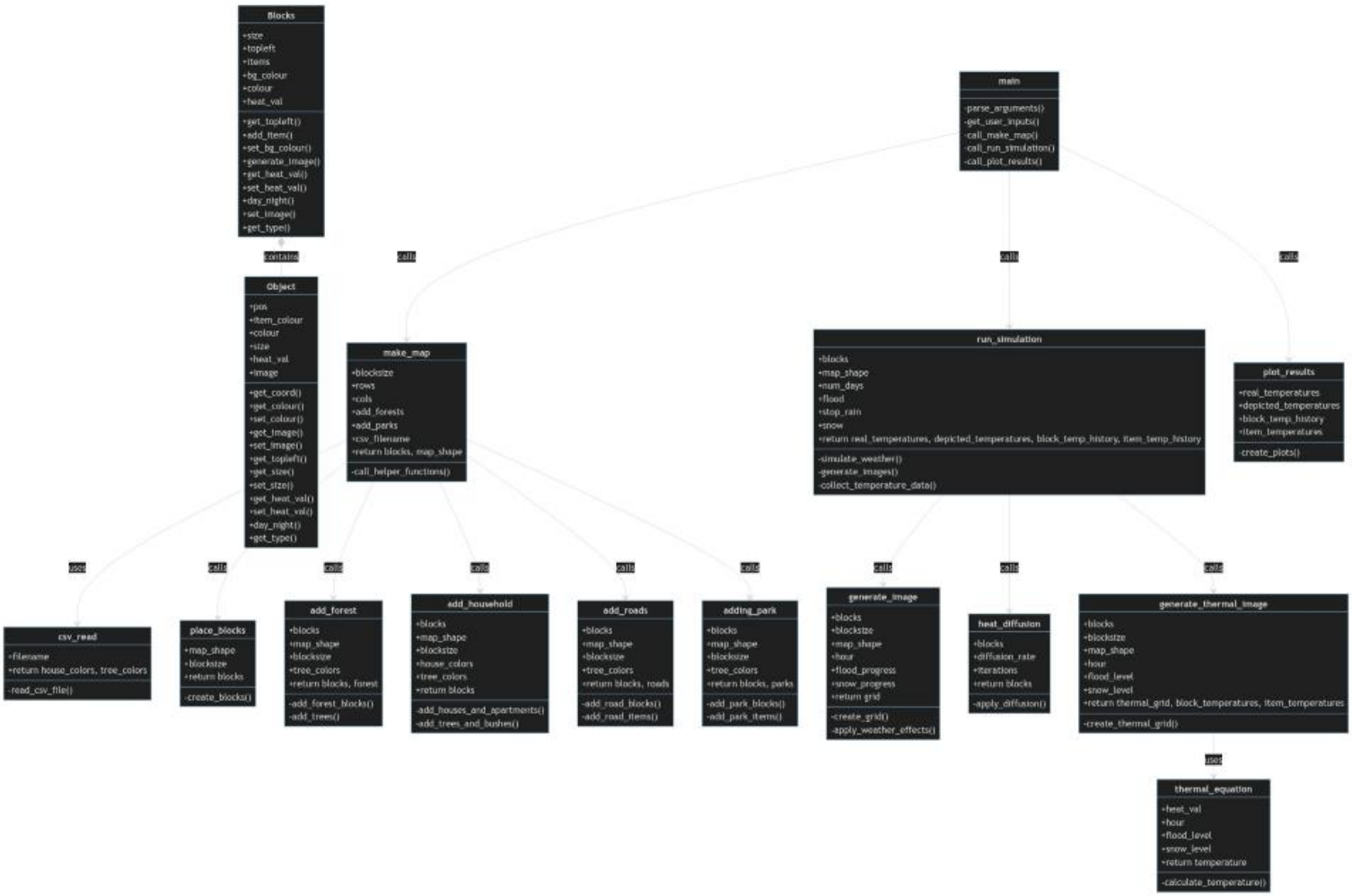


Figure 6: UML Class Diagram

4.1 Map Generation

The map generation is implemented in the 'make_map' (*Traceability matrix, 2.1)* function in the object_placement.py file. This function creates a grid comprising different types of blocks, each representing a different part of the map. The map size is customisable, with 12 blocks being the minimum number. Furthermore, users have the freedom to select whether they desire to add forests and parks to the map.

4.2 Object Placement

Each object is placed within its desired block using the 'add_item' method implemented on the Block class. Each item follows different methods of being implemented in the map, with some being hard-coded and the others being random. The logic of these item placements is set in different functions of the 'object_placement.py' file. *(Traceability matrix, 2.2 & 2.3)*

4.3 Day-Night Cycle

The Day-Night cycle is simulated within the 'run_simulation' function in map.py. 'time_factor' is calculated through a sine equation and implemented to blocks and items through the 'day_night' method, thus changing the brightness of the blocks depending on the time of the day. *(Traceability matrix 2.4)*

4.4 Weather Effects.

Both snow and rain follow the same logic and are implemented within the 'run_simulation' function. In the 'generate_image' function, based on the scenario's progress, a blue or white overlay will be set to certain blocks and items. Moreover, the 'apply_rain' and 'apply_snow' functions create visual effects of rain and snow that become intensive as the scenario progresses. *(Traceability matrix, 9 & 10)*

4.5 Temperature Simulation

Each block and item has a 'heat_val', a base thermal value. This value is sent through the 'thermal_equation' function that possesses a sine equation to show how the temperature changes according to the time of the day and weather conditions. Moreover, the 'heat_diffusion' function simulates heat transfer between neighbouring blocks and items. *(Traceability matrix, 3)*

4.6 Visualisation

The simulation is visualised using Matplotlib. The main RGB map is generated using the 'generate_image' function, and 'generate_thermal_image' is used to create a thermal map based on block and item temperatures. The 'plot_results" function creates various plots to show temperature changes over time.

4.7 User Input and Configuration

The user inputs are handled in the 'user_inputs' functions, allowing customization of map size, simulation duration, and weather conditions. The program also supports command-line arguments for predefined scenarios (normal, rain, snow). *(Traceability matrix, 6)*

5.0 Showcase

The canopy simulation consists of three scenarios, each different from each other. The scenarios are called by either user input selections or through command line arguments.

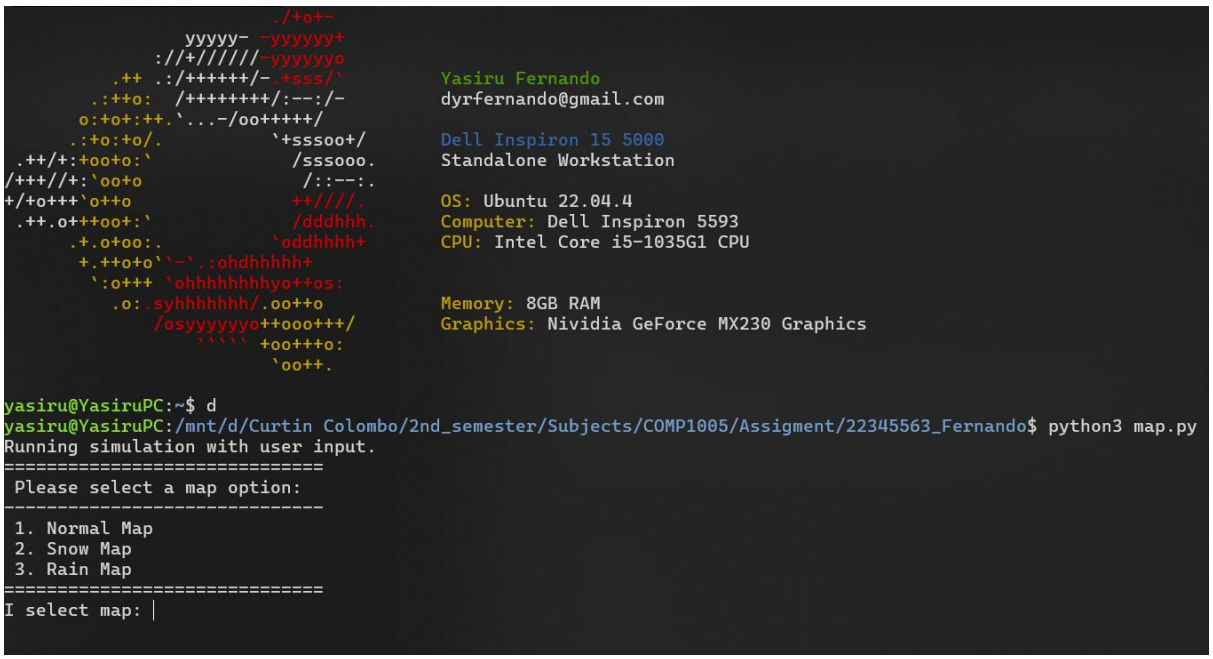


Figure 7: Selecting Scenario

As shown in Figure 7 the user can select the desired scenario from the 3 given scenarios. If the user is to use the command line to initiate the programme, they must type the scenario they desire, 'normal', 'snow', and 'rain', for the program to execute the scenarios.

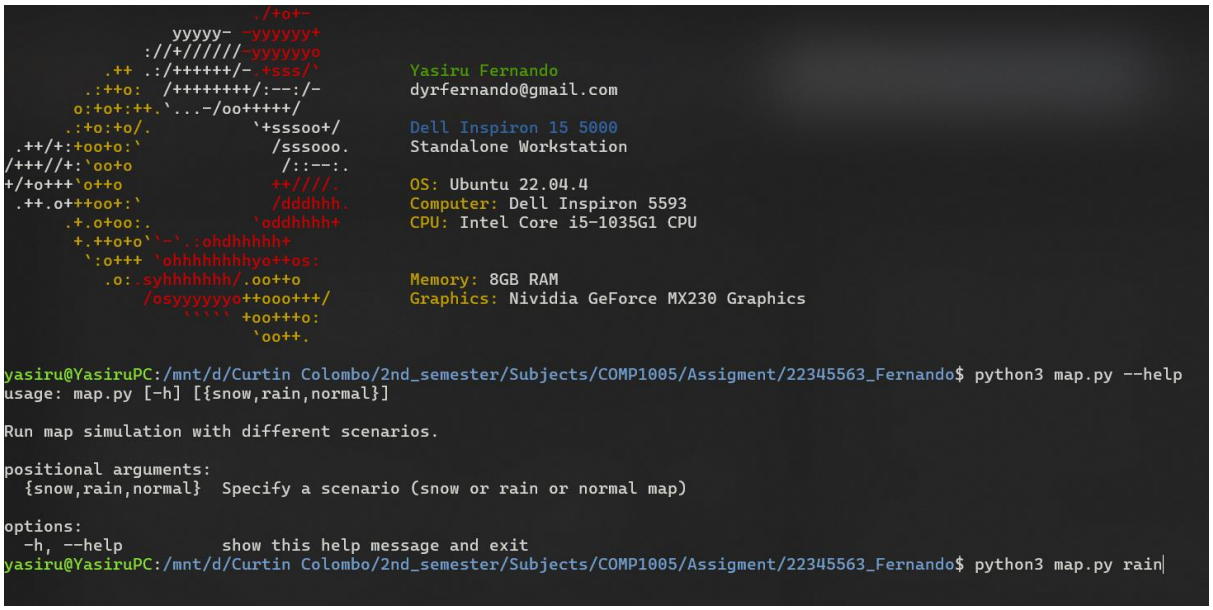


Figure 8: Selecting scenario through user inputs

5.1 Scenario 1 – Normal map

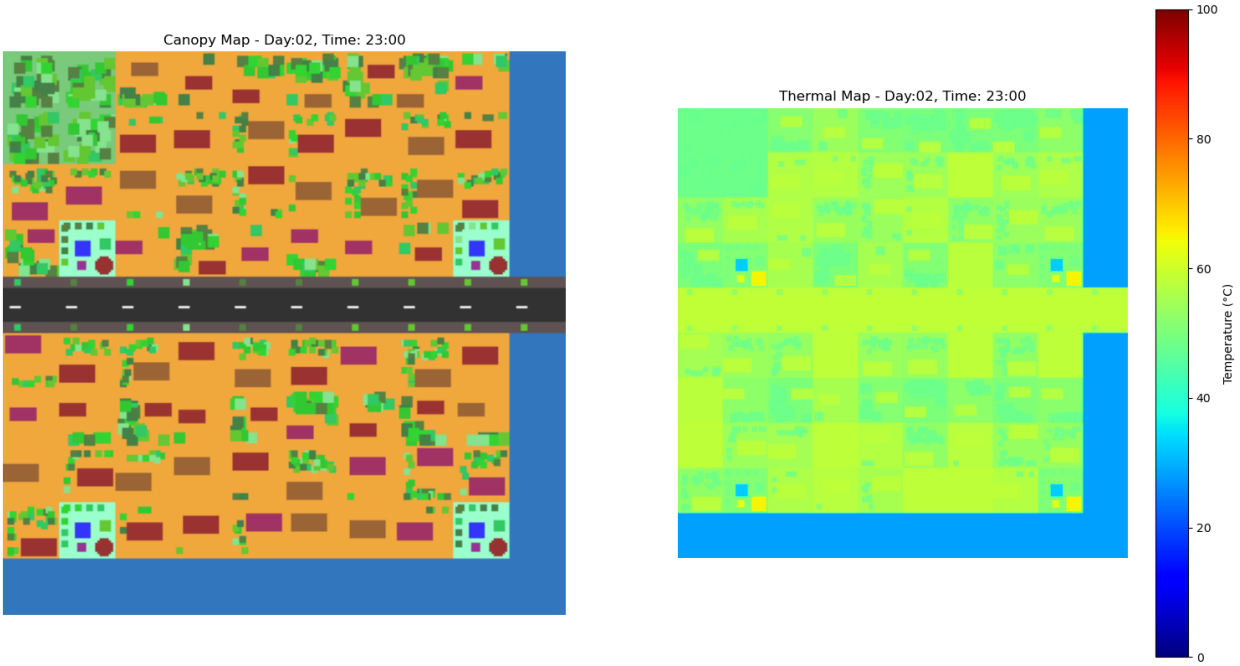


Figure 9: Canopy map - Normal

Displayed in Figure 9 is the normal scenario for canopy simulation. The simulation depicts a normal day and how the thermal values of each item and object change according to the hour of the day. Figure 9 image displays the temperature values of 11 pm at night when items and objects are generally colder. Figure 10 shows how the temperature values gradually rise, peak around 12 pm, and then gradually decrease (Depicted Temperature). The figure also depicts how the day's actual temperature changes, giving the user a clear idea about the temperature changes and the actual temperature values depicted by the colour bar values.

Moreover, in Figure 9, we can see that several blocks are generally colder on the thermal map. This is due to heat diffusion effect. The blocks with a higher number of trees tend to be colder than those with fewer trees. This is a natural phenomenon in the real world, and the program allows users to experience it easily.

The block placements are hard coded using loops, but the item placement is mostly randomised. An environment closer to a water supply, a canal, is picked for the map to include the temperature changes of the water. Moreover, the map is flexible, allowing users to include the many rows and columns desired. The user also gets to choose whether to include forests and parks. In short, the program is made so that the user gets to experience heat diffusion and thermal changes in the environment according to the time of the day with ease.

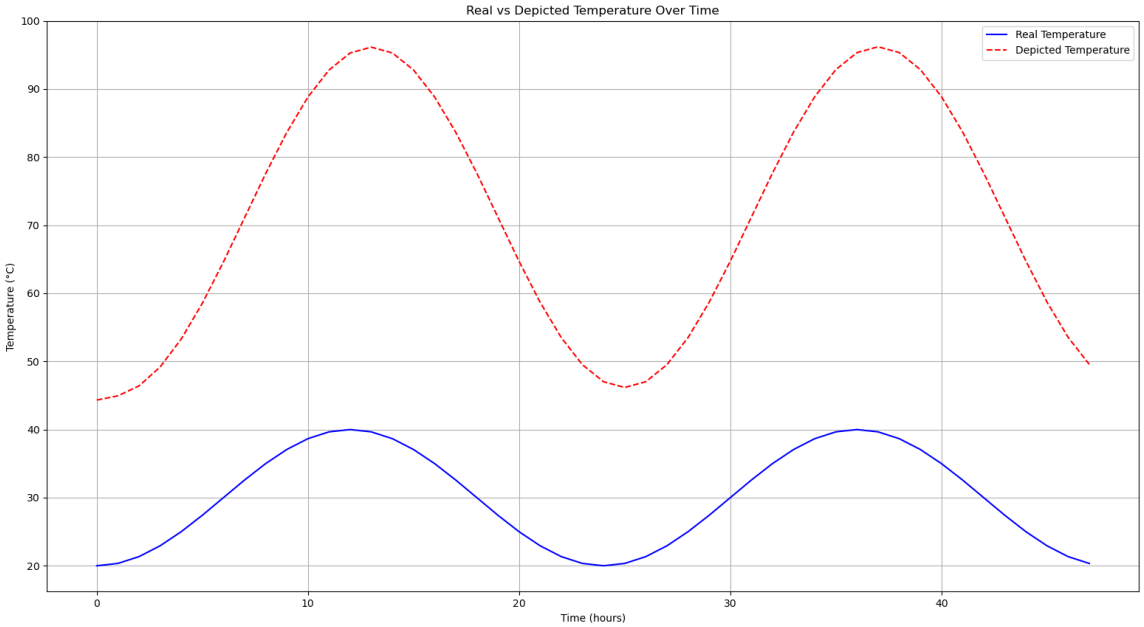


Figure 10: Normal map - Average temperature change.

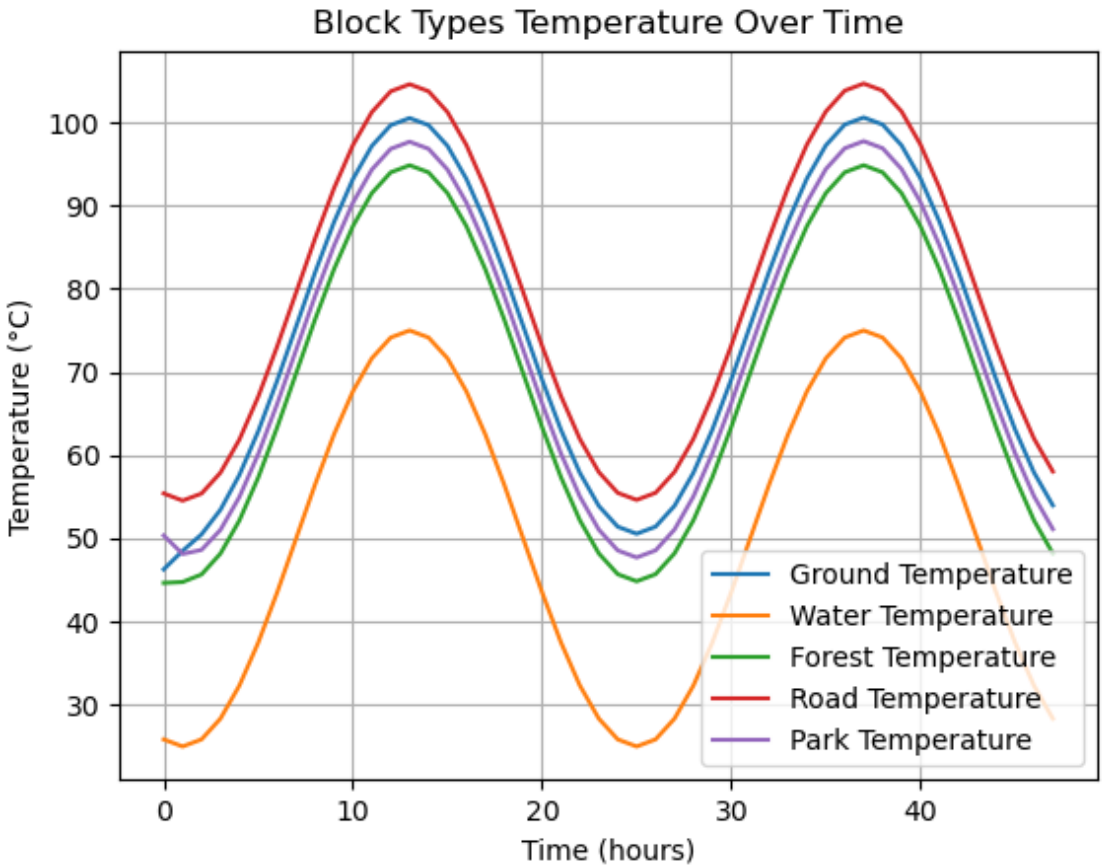


Figure 11: Normal map - Temperature change of blocks

Figure 11 and Figure 12 depict the temperature change of each item and each block. As true in real life, the temperature values of each item and object increase, reaching a peak near 12 pm and 2 pm before gradually decreasing. A slight increase in minimum temperature can be noticed in items.

However, the change is not significant. This is mostly due to heat diffusion affecting each item, resulting in a slightly higher minimum temperature than the previous night.

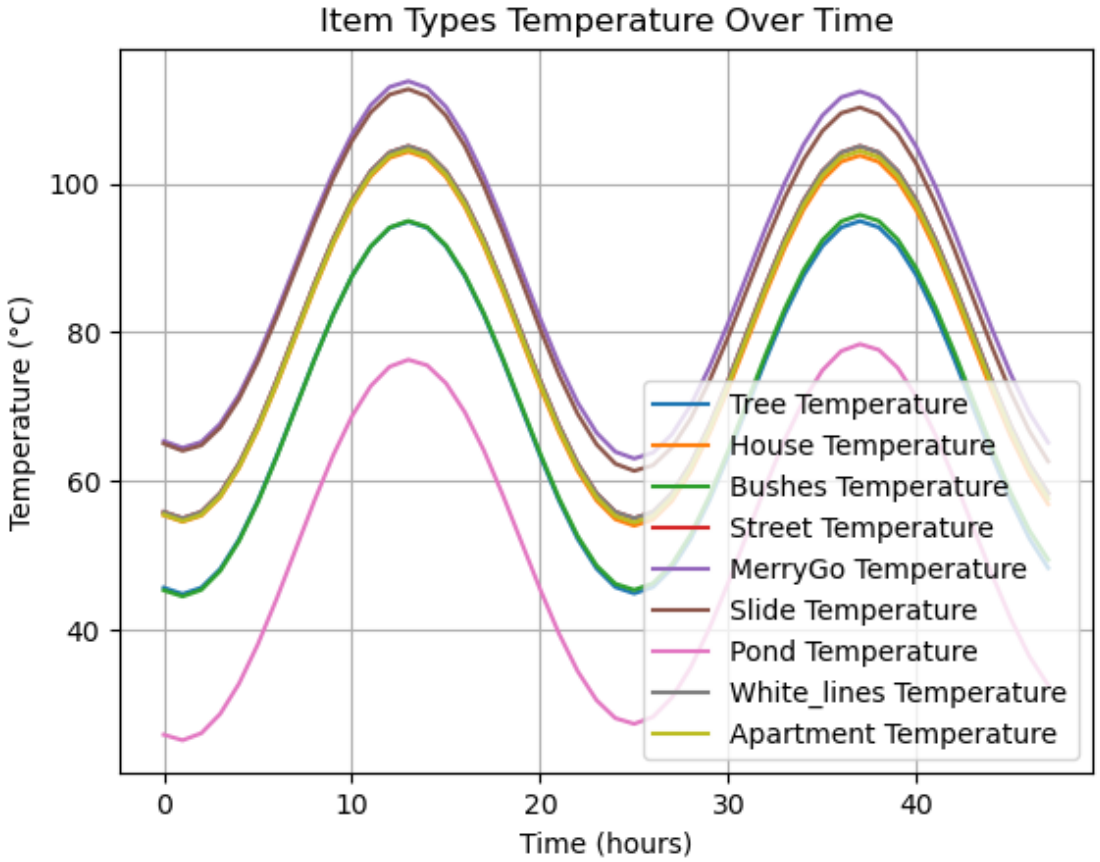


Figure 12: Normal map - Temperature changes of items

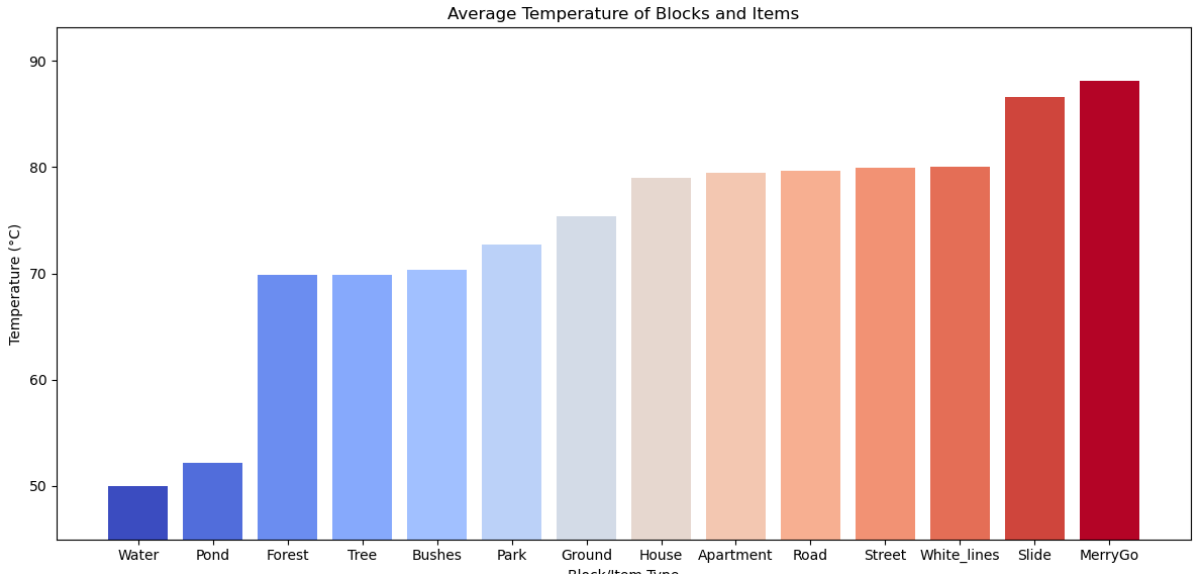
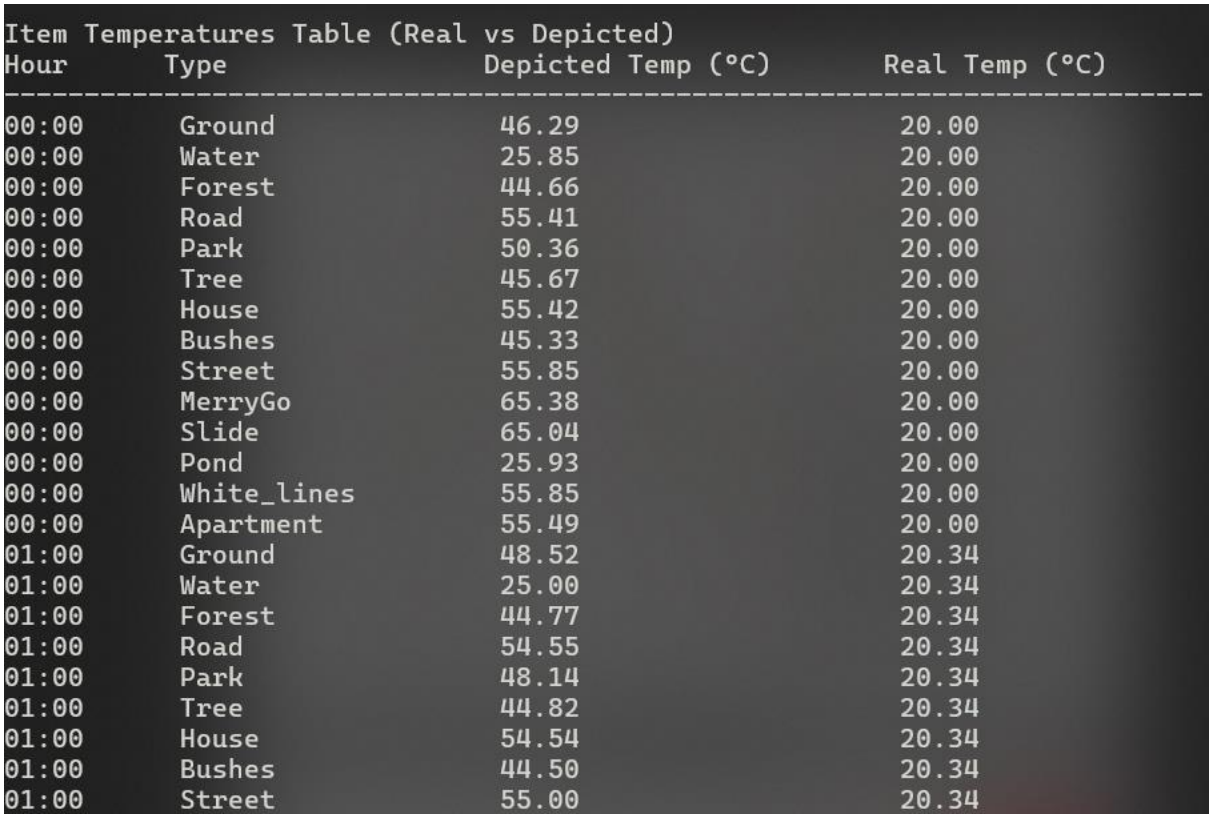


Figure 13: Normal map - Average Temperature



Item Temperatures Table (Real vs Depicted)

Hour	Type	Depicted Temp (°C)	Real Temp (°C)
00:00	Ground	46.29	20.00
00:00	Water	25.85	20.00
00:00	Forest	44.66	20.00
00:00	Road	55.41	20.00
00:00	Park	50.36	20.00
00:00	Tree	45.67	20.00
00:00	House	55.42	20.00
00:00	Bushes	45.33	20.00
00:00	Street	55.85	20.00
00:00	MerryGo	65.38	20.00
00:00	Slide	65.04	20.00
00:00	Pond	25.93	20.00
00:00	White_lines	55.85	20.00
00:00	Apartment	55.49	20.00
01:00	Ground	48.52	20.34
01:00	Water	25.00	20.34
01:00	Forest	44.77	20.34
01:00	Road	54.55	20.34
01:00	Park	48.14	20.34
01:00	Tree	44.82	20.34
01:00	House	54.54	20.34
01:00	Bushes	44.50	20.34
01:00	Street	55.00	20.34

Figure 14: Normal map - Real temperature vs Depicted temperature

Figure 12 shows the average temperature of each item and block for the timeline the simulation is run. Slide and MerryGo, the two playground items, have the highest temperature values. This is true to real life as they are generally constructed from metal, which absorbs more heat. The water temperature levels are lower than the pond due to the water depicting a canal, which has a larger water body and generally heats slower. This again confirms that through the programme, the user can experience an overview of a real-life temperature change of objects according to the time of the day.

Figure 14 is a part of the table being printed at the end of the programme. The table depicts each item and object and their dedicated values and actual values for each hour of the day.

5.2 Scenario 2 – Rain map

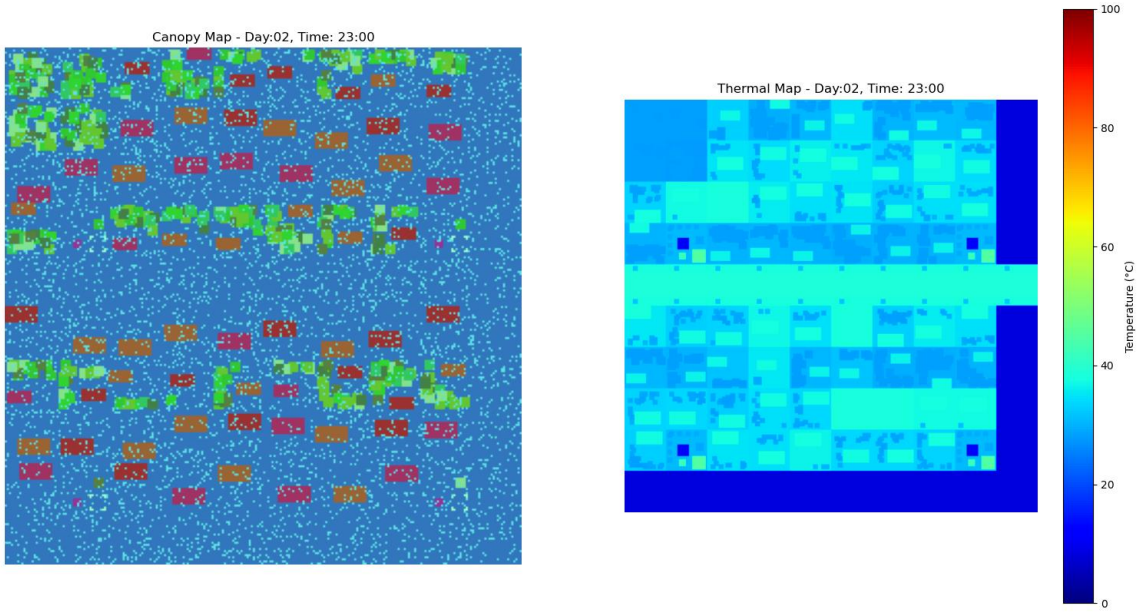


Figure 15: Canopy map – Rain

The Rain map depicts a simulation of a sudden rain occurring. Due to the intensity of the rain, the nearby canals get flooded, thus drowning most of the landscape. Figure 15 shows the landscape with most of the items and blocks flooded. Rain is depicted as random-size particles that are visible throughout the map. The road, the park, and the bushes have all drowned, leaving the taller structures, such as trees, apartments and houses, safe from the flood.

In the thermal view, we can see how the temperature changes and gradually decreases due to the cooling of the flood water. The thermal view taken of the Canopy Rain map at 23:00 (Figure 15) has significantly colder thermal properties compared to the Canopy normal map in Figure 9. This is once again true in life situations where the atmosphere gets colder due to rain, thus affecting the surrounding areas' objects to have thermal properties that are slightly or significantly lower than their original.

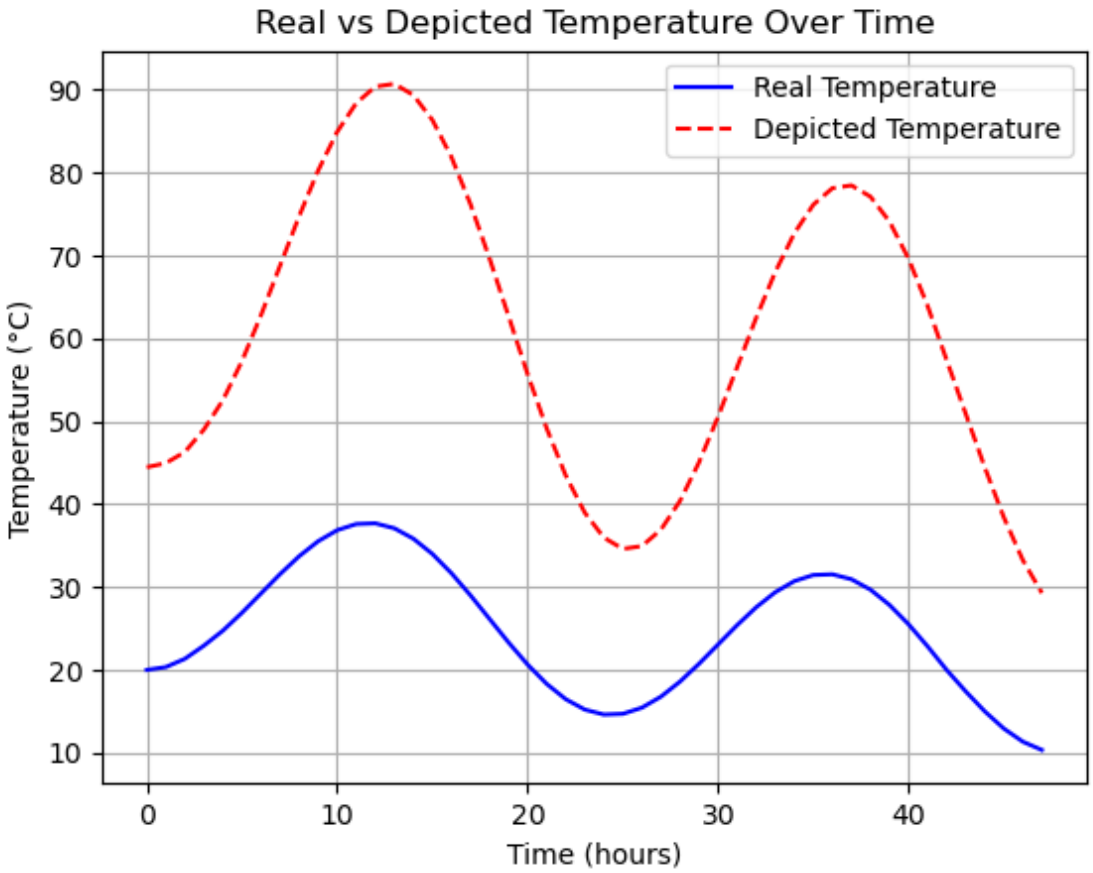


Figure 16: Rain map - Average temperature change

Figure 16 shows how the average temperature of the map, both real and depicted, decreases over time. This is due to the atmospheric cooling caused by the rain and flood. With each passing hour, the cooling effect takes place, thus resulting in the decrease of item and block temperatures, as shown in Figure 17 and Figure 18.

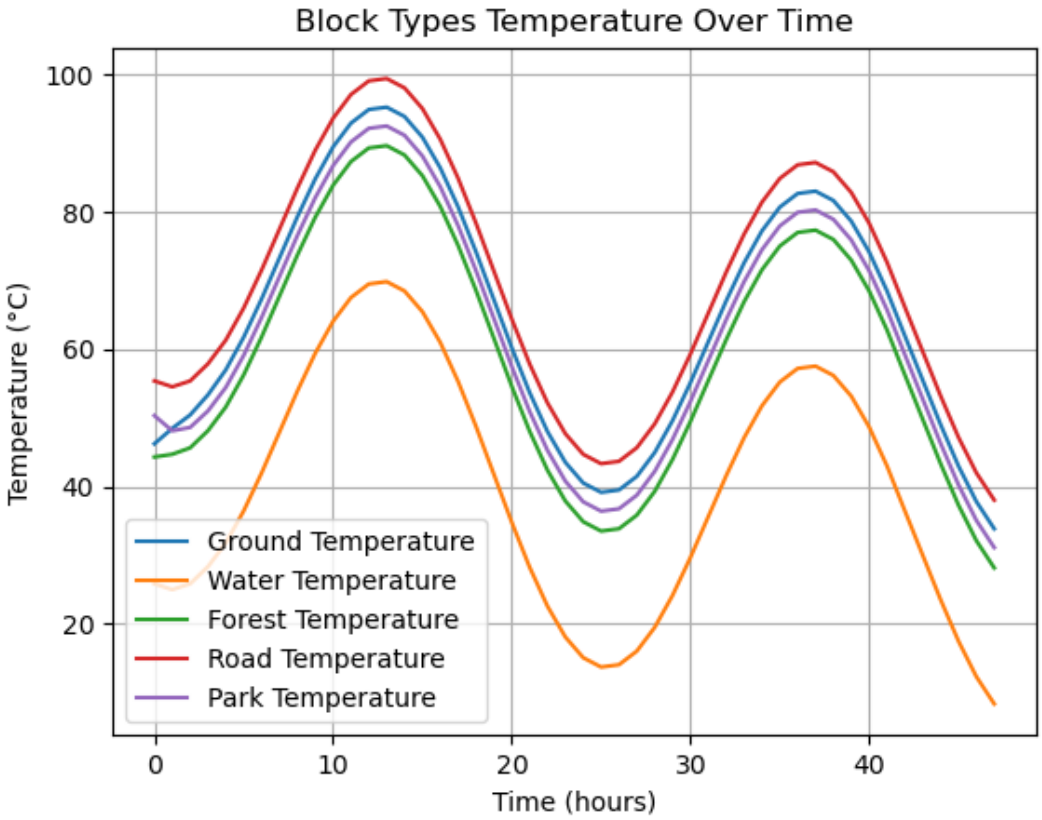


Figure 17: Rain map - Block temperature

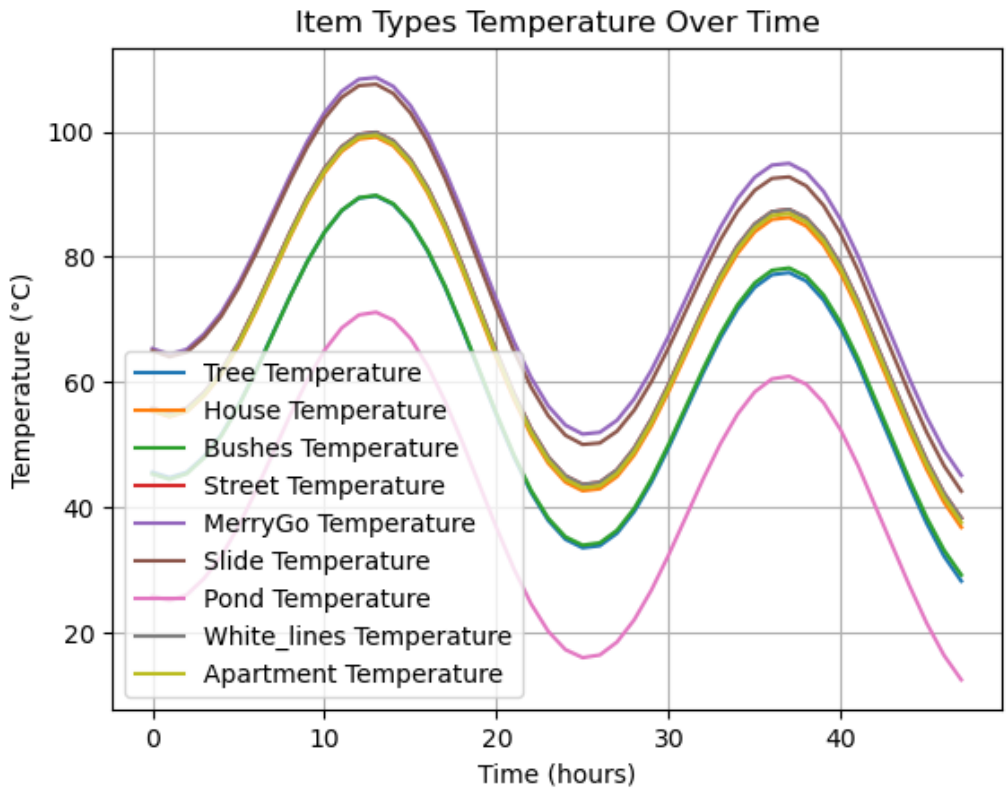


Figure 18: Rain map - Item temperature

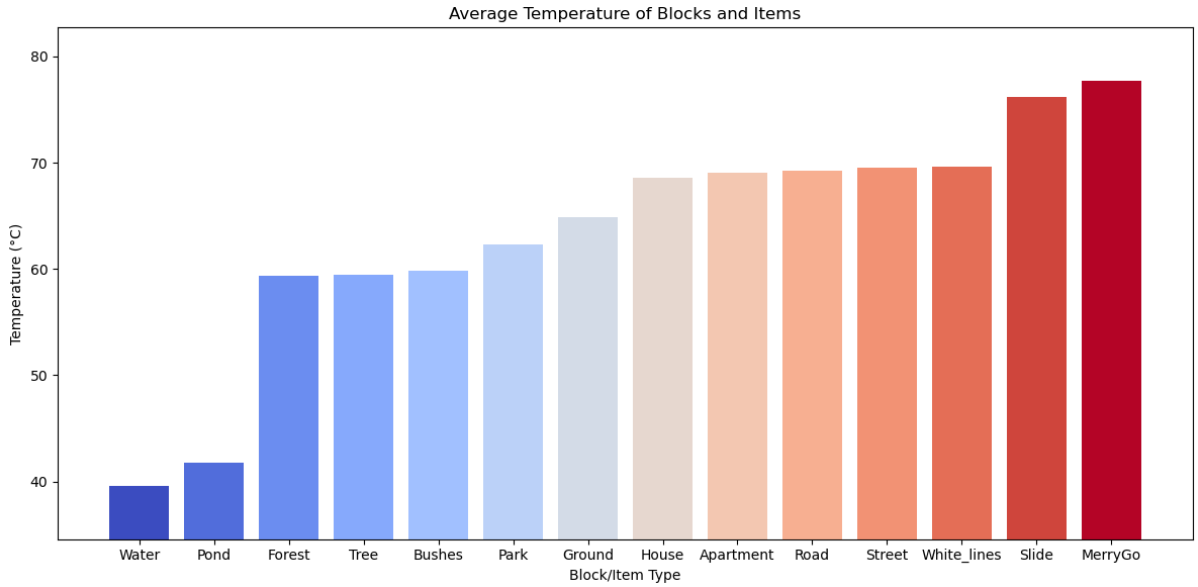


Figure 19: Rain map - Average temperature

Figure 19 depicts how the average temperatures of each item and block have decreased compared to Figure 13. In estimation, the temperature depicted dropped by 10 C on the rain map compared to the normal map.

Item Temperatures Table (Real vs Depicted)

Hour	Type	Depicted Temp (°C)	Real Temp (°C)
00:00	Ground	46.25	20.00
00:00	Water	25.85	20.00
00:00	Forest	44.32	20.00
00:00	Road	55.41	20.00
00:00	Park	50.36	20.00
00:00	Tree	45.66	20.00
00:00	House	55.40	20.00
00:00	Bushes	45.37	20.00
00:00	Street	55.85	20.00
00:00	MerryGo	65.38	20.00
00:00	Slide	65.04	20.00
00:00	Pond	25.93	20.00
00:00	White_lines	55.85	20.00
00:00	Apartment	55.51	20.00
01:00	Ground	48.49	20.34
01:00	Water	25.00	20.34
01:00	Forest	44.72	20.34
01:00	Road	54.55	20.34
01:00	Park	48.14	20.34
01:00	Tree	44.80	20.34
01:00	House	54.52	20.34
01:00	Bushes	44.54	20.34
01:00	Street	55.00	20.34
01:00	MerryGo	64.47	20.34
01:00	Slide	64.07	20.34
01:00	Pond	25.18	20.34

Figure 20: Rain map - Real map vs Depicted map

5.3 Scenario 3 – Snow map

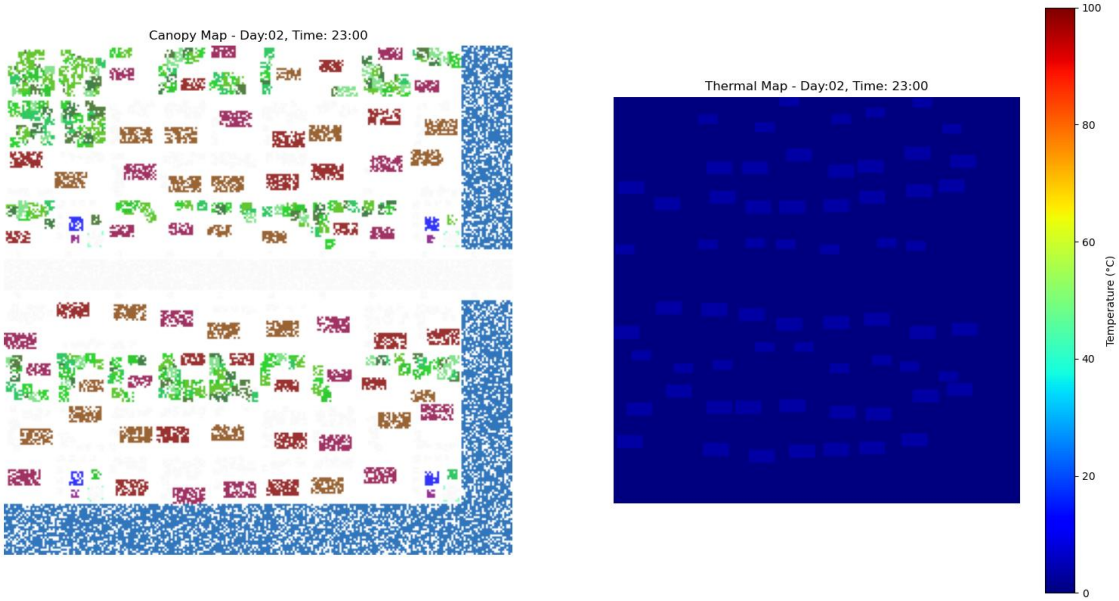


Figure 21: Canopy map – Snow

The third scenario depicts a sudden snowstorm, and the entire landscape is drowned by snow, similar to the Flooding. Here, the user can identify that the temperature values of each item and object drop significantly, with the temperature values depicted in Figure 21 being considerably lower than those displayed in Figure 15.

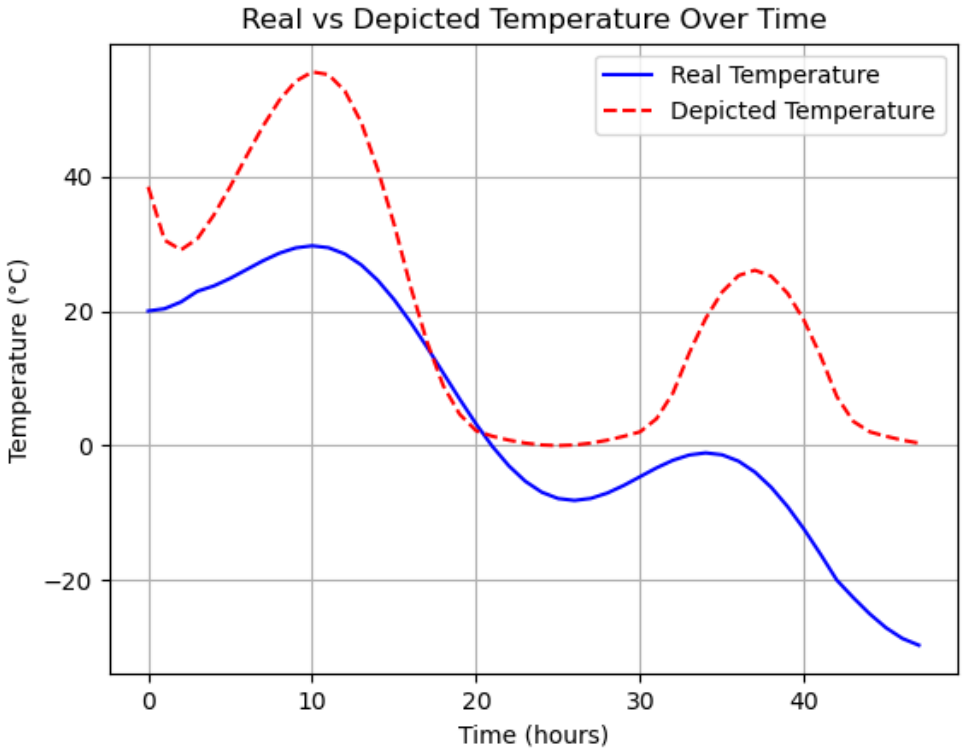


Figure 22: Snow map - Average temperature change

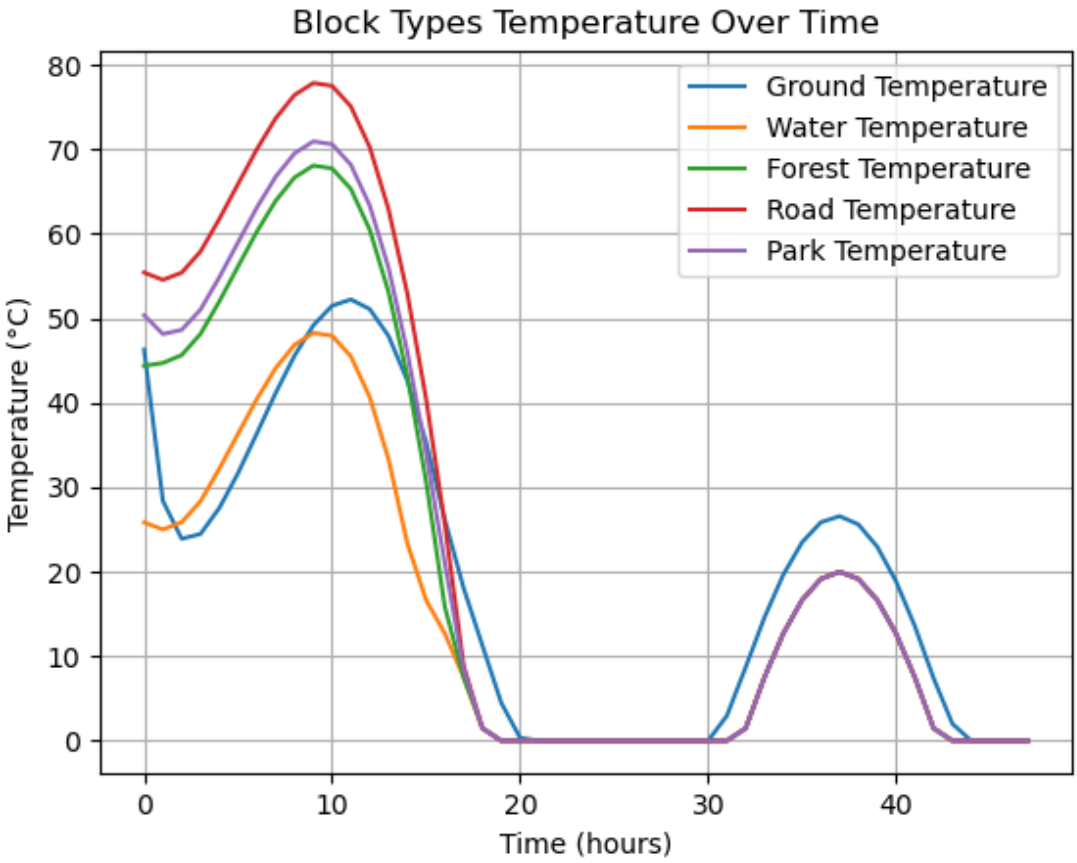


Figure 23: Snow map - Block temperature

Figure 22, Figure 23 and Figure 24 depict the temperature changes in each block and item. At one point, the temperature has reached zero, displaying an extremely cold environment. However, while observing the simulation and the graphs, it is noticeable that the houses and apartments have maintained a higher temperature despite the surrounding low-temperature zones. This is due to people inside the said building using various sources of heat generation to warm themselves. Therefore, the house and apartment items contain a high thermal average for the simulation duration.

It should be noted that the Slide and Merrygo are depicted as having the highest temperature values due to an error in the programme. This is due to an error in the thermal calculations of the items.

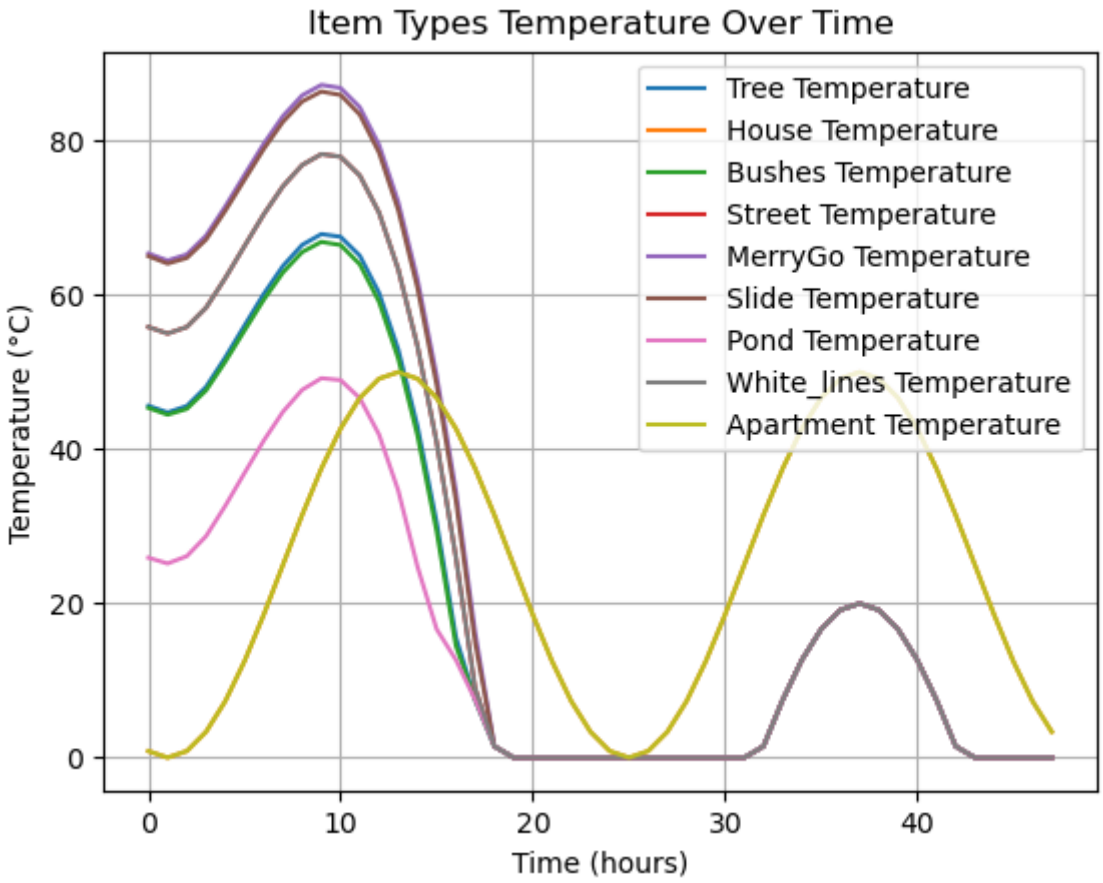


Figure 24: Snow map - Item Temperature

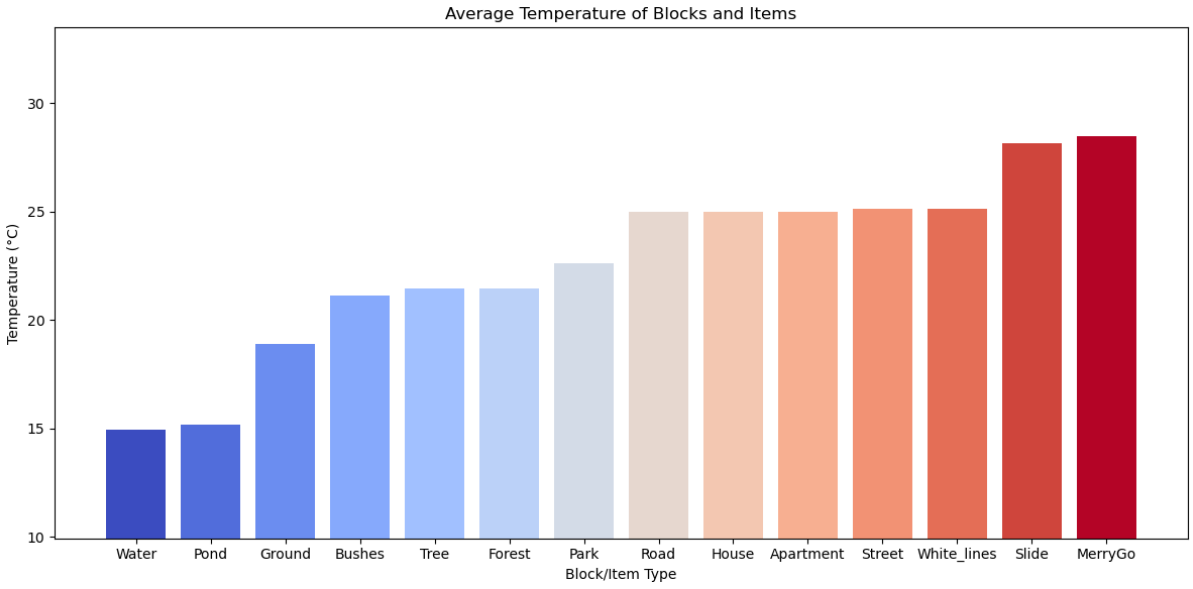


Figure 25: Snow map - Average Temperature

Item Temperatures Table (Real vs Depicted)

Hour	Type	Depicted Temp (°C)	Real Temp (°C)
00:00	Ground	46.29	20.00
00:00	Water	25.85	20.00
00:00	Forest	44.37	20.00
00:00	Road	55.41	20.00
00:00	Park	50.36	20.00
00:00	Tree	45.62	20.00
00:00	House	0.85	20.00
00:00	Bushes	45.36	20.00
00:00	Street	55.85	20.00
00:00	MerryGo	65.38	20.00
00:00	Slide	65.04	20.00
00:00	Pond	25.93	20.00
00:00	White_lines	55.85	20.00
00:00	Apartment	0.85	20.00
01:00	Ground	28.38	20.34
01:00	Water	25.00	20.34
01:00	Forest	44.71	20.34
01:00	Road	54.55	20.34
01:00	Park	48.14	20.34
01:00	Tree	44.77	20.34
01:00	House	0.00	20.34
01:00	Bushes	44.48	20.34
01:00	Street	55.00	20.34
01:00	MerryGo	64.47	20.34
01:00	Slide	64.07	20.34
01:00	Pond	25.18	20.34
01:00	White_lines	55.00	20.34
01:00	Apartment	0.00	20.34
02:00	Ground	23.90	21.34
02:00	Water	25.85	21.34
02:00	Forest	45.65	21.34
02:00	Road	55.40	21.34
02:00	Park	48.63	21.34

Figure 26: Snow map - Real temperature vs Depicted temperature

6.0 Conclusion

The Canopy simulation is a very user-friendly programme that allows the user to learn about the thermal behaviour of different objects for each hour of the day. It provides true-to-life simulations of how the thermal properties of objects work and how environmental occurrences such as flooding, snowing and heat diffusion affect the thermal properties of objects. While it does have a few issues regarding its outputs, especially concerning the snow simulation, the programme provides a very educational output for the user, with graphs and data printed out at the end of the programme.

7.0 Future Work

This programme requires several updates for it to achieve its maximum capability. As such, the following are the future work to be done

1. Fixing the thermal simulation issues regarding the Snow map. Specifically correcting the current incorrect data output by the graphs and optimising the equations to provide more true-to-life data
2. Perfecting heat diffusion with timestep carryover. Ensuring the data of a specific hour is carried over to the next hour, thus creating a more true-to-life simulation.
3. Adding several more scenarios would allow the users to grasp the aspects of thermal behaviours of objects more widely.

8.0 References

W3Schools. 2024. “W3Schools Online Web Tutorials.” W3schools.com. W3Schools. 2024. https://www.w3schools.com/.

Matplotlib. 2012. “Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation.” Matplotlib.org. 2012. https://matplotlib.org/.

“Fundamentals of Programming” 2024. Curtin.edu.au. 2024. https://lms.curtin.edu.au/ultra/courses/_139867_1/cl/outline.

Python. 2020. “The Python Standard Library — Python 3.8.1 Documentation.” Python.org. 2020. https://docs.python.org/3/library/index.html.

“How to Use Python Numpy.where() Method | DigitalOcean.” n.d. Www.digitalocean.com. https://www.digitalocean.com/community/tutorials/python-numpy-where.

“Overview — NumPy V1.21 Manual.” n.d. Numpy.org. https://numpy.org/doc/stable/index.html.