Optimal Path Finder

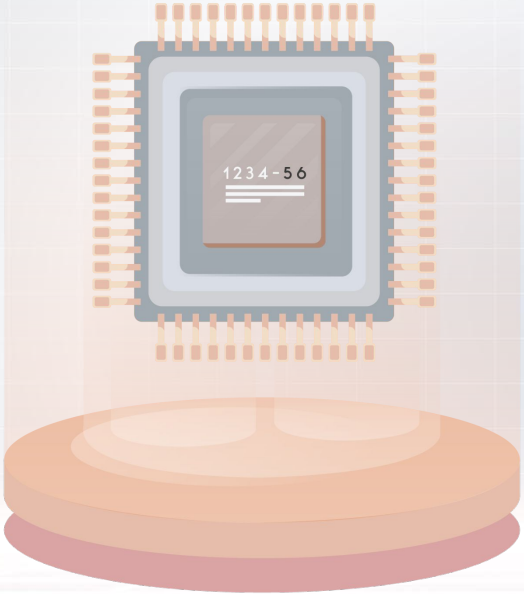
Presented by:
Gabriel Choong Ge Liang
Cao Rui
Chau Teng Cheng
Benjamin Chin Wei Hao

Have you taken a long way round by your silly navigation?

Don't worry

The good news is that we have a scheme (/ski:m/) to solve that problem.

*Not trying to compete with Google Maps



01

INTRODUCTION

You can enter a subtitle here if you need it

Introduction

This project uses the A* Algorithm to find the Optimal Path within the range of the campus

- This algorithm has two parameters:
  - Path cost
  - Heuristic cost
- Formula: f (n) = g (n) + h (n)
- Takes two arguments and returns the calculated path

What is A* -Algorithm ?

A*-Algorithm

- Most widely known form of best-first search: A* search.
- It evaluates nodes by combining **g(n),** the cost to reach the node, and **h(n)**, the cost to get from the node to the goal:

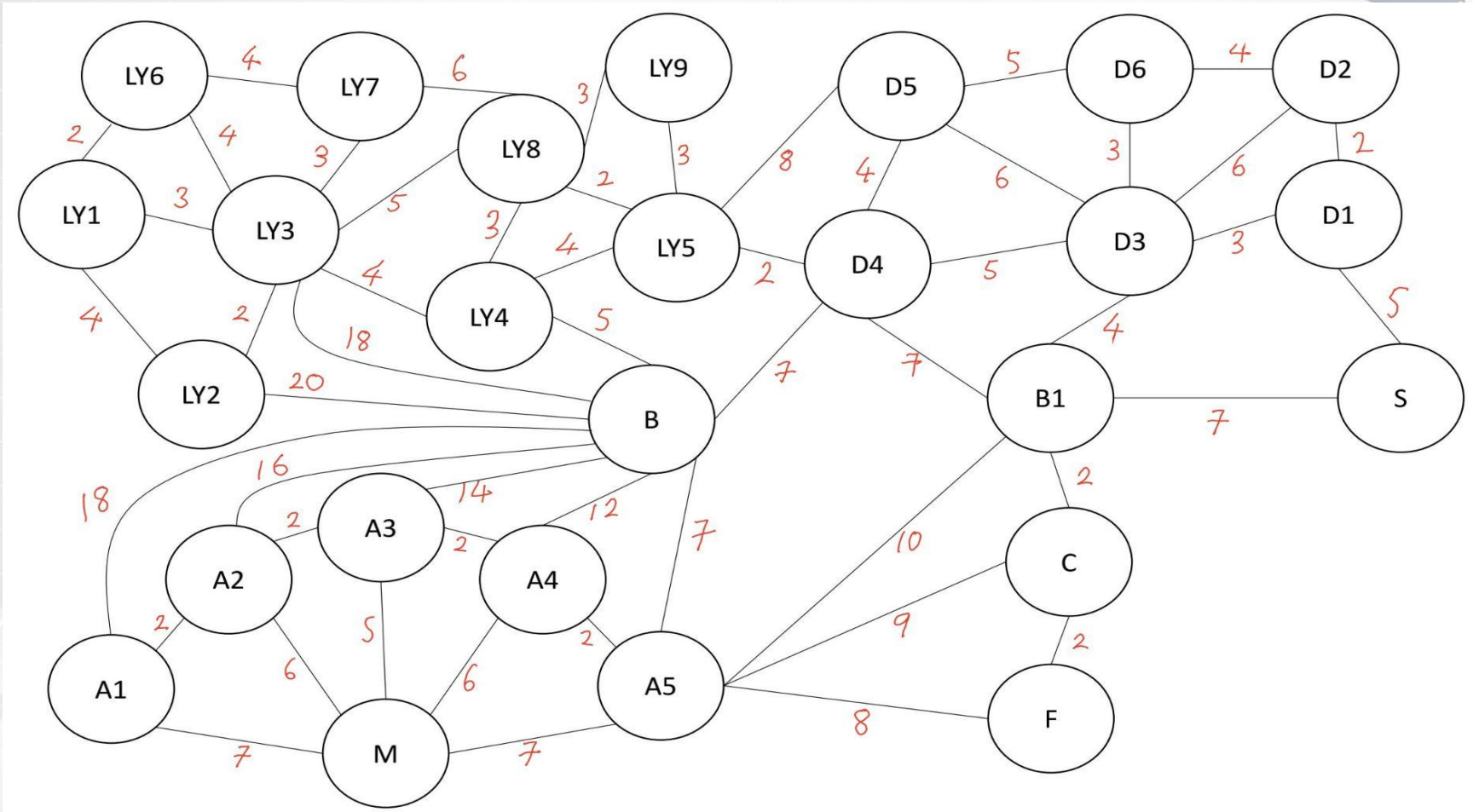
$$f(n) = g(n) + h(n)$$

g(n): path cost from the start node to node n. It depends on the state of the node.

h(n): estimated cost of the cheapest path from n to the goal

f(n): estimated cost of the cheapest solution through n.

The main data is come from the real path of our school's buildings.



Illustration

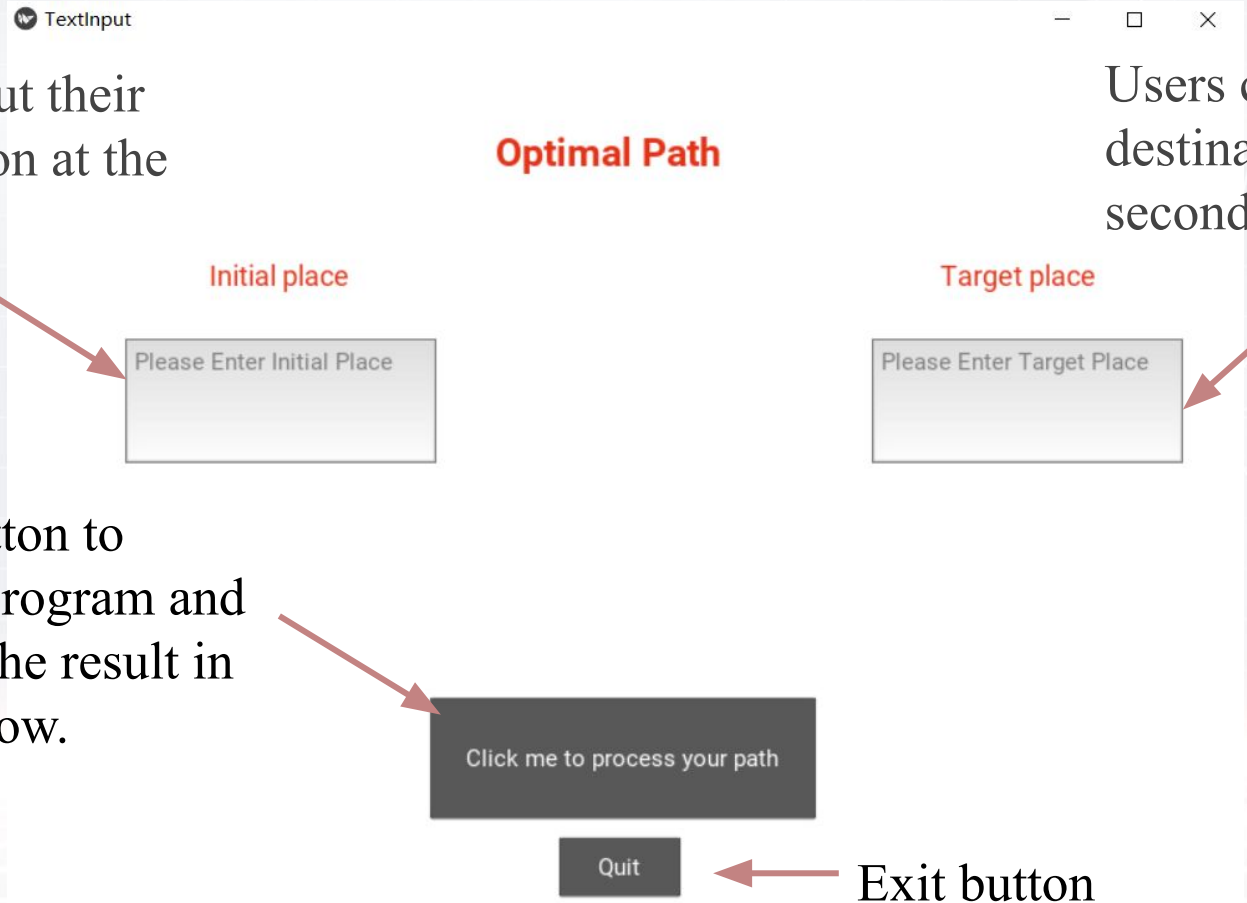
Since most users aren’t experienced with the terminal, we have included a Graphical User Interface (GUI)

Several dependencies are included within this functionality.

We chose the kivy framework for the Graphical User Interface (GUI)

It’s a modern UI framework that supports the Python Programming Language

GUI of project



Users can input their current location at the textbox.

Users can input their destination at the second textbox.

Click this button to proceed the program and you will get the result in another window.

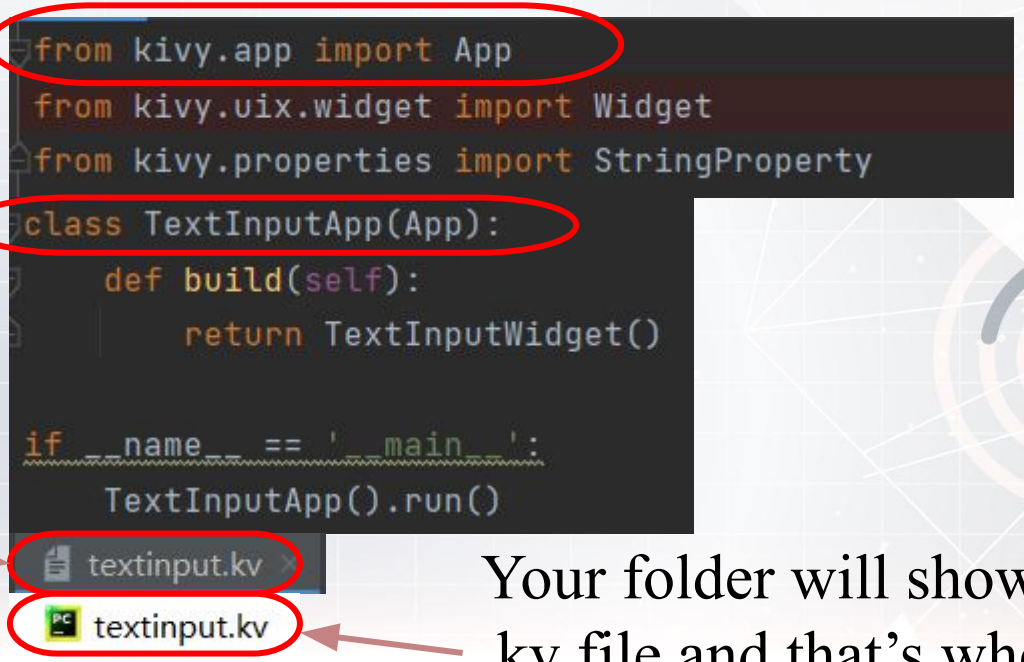
Exit button

How can it realize?

Use “App” method

Cite the App method

Create a new file in Pycharm whose name need to be the same as the lowercase-words of the App function name.



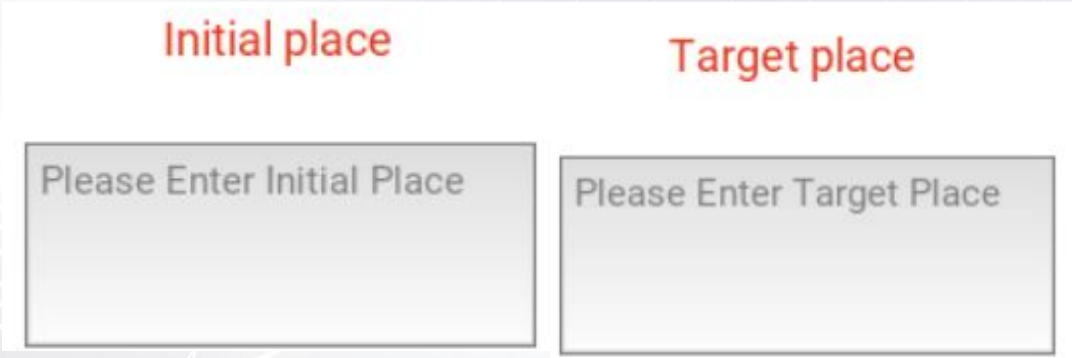
Your folder will show a .kv file and that’s where we design our widgets mainly.

Define the textbox where user can input his two place- initial place, target place.

With position, hint_text, size of the box, so on.

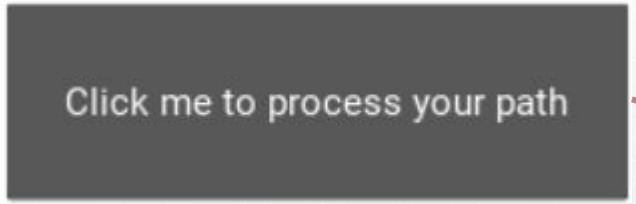
And also the function works with the keyboard for example: Press Tab to switch first textbox to second textbox.

For the second textbox, when you press Enter, you can call the function in .py file and call out the Resultscreen.




```
TextInput:
    id: IP
    multiline:False
    pos:80,330
    allow_copy:True
    auto_indent:True
    hint_text:'Please Enter Initial Place'
    size:200,80
    write_tab:False

TextInput:
    id: TP
    multiline:False
    pos:560,330
    allow_copy:True
    auto_indent:True
    hint_text:'Please Enter Target Place'
    size:200,80
    write_tab:True
    on_text_validate:
        root.press_button(self)
        root.btn_touch_up()
```




```
Button:
    text:'Click me to process your path'
    size:250,80
    pos:275,100
    on_press:root.press_button(self)
    on_release:root.btn_touch_up()
```

```
Button:
    text:'Quit'
    size:80,40
    pos:358,50
    on_press:root.release_button(self)
    on_press: app.stop()
```

To design the background for example: colors, shapes, so on.

```
<TextInputWidget>:
    canvas:
        Color:
            rgba:[1,1,1,1]
        Rectangle:
            pos:self.pos
            size:self.size

    Label:
        text:'Optimal Path'
        font_size:25
        bold:True
        color:.9,.2,.1,1
        pos: 130,480
        text_size:cm(15),mm(30)
        halign:'right'
        valign:'middle'
        shorten:True
        shorten_from:'right'
        markup:True

    Label:
        text:'Initial place'
        font_size:18
        bold:False
        color:.9,.2,.1,1
        pos: 80,400
        text_size:cm(5),mm(10)
        halign:'right'
        valign:'middle'
        shorten:True
        shorten_from:'right'
        markup:True

    Label:
        text:'Target place'
        font_size:18
        bold:False
        color:.9,.2,.1,1
        pos: 560, 400
        text_size:cm(5),mm(10)
        halign:'right'
        valign:'middle'
        shorten:True
        shorten_from:'right'
        markup:True
```

Optimal Path

Initial place

Target place

```
    def btn_touch_up(self):
        result = StringProperty(path)
        from subprocess import Popen, PIPE
        process = Popen(['python', 'resultscreen.py'], stdout=PIPE, stderr=PIPE)


    def release_button(self,arg):
        print('Thanks for your using. Looking forward to your next process')


class TextInputApp(App):
    def build(self):
        return TextInputWidget()
```

Purpose of pressing button to show another window which is result.

Purpose of pressing ‘Quit’ button to show thanks message.

These are the code for second window whose aim is to show the result when the user click the button so call Resultscreen.

Why we design this?

If don’t, the user may only see the result in Pycharm.

So to avoid this problem and make the user get the result clearly and exactly, we create another window to show the result.

```python
from kivy.app import App
from kivy.uix.widget import Widget
import UI_kivy

final_result = UI_kivy.result


class Result(Widget):

    def __init__(self,**kwargs):
        super().__init__(**kwargs)

    def press_button1(self,arg):

        print(final_result)

class Resultscreen(App):          #App class is inherited
    r = final_result

    def build(self):
        return Result()

if __name__ == '__main__':
    Resultscreen().run()
```

Resultscreen

#will be replaced by the real variable of the main part

Exit

Routing and Mapping of the campus

- Use of dictionary for quick lookup times

```
campus= { "B" : "LY4", "B" : "D4", "B" : "A5", "B" : "A4", "B" : "A3", "B" : "A2", "B" : "A1", "B" : "LY2",  "B" : "LY3",
        "A1" : "M", "A2" : "M","A3" : "M", "A4" : "M", "A5" : "M" , "LY2" : "LY1", "LY3" : "LY1", "LY4" : "LY3",
       "LY2" : "LY3", "LY1" : "LY6", "LY6" : "LY7", "LY6" : "LY3", "LY7" : "LY8", "LY7" : "LY3", "LY3" : "LY8",
        "LY8" : "LY4", "LY8" : "LY9", "LY8" : "LY5", "LY4" : "LY5", "LY5" : "LY9", "LY5" : "D4","LY5" : "D5",
        "D5" : "D4", "D5" : "D6", "D5" : "D3", "D6" : "D2", "D6" : "D3", "D4" : "D3", "D3" : "D2", "D2" : "D1",
        "D3" : "D1", "D4" : "B1", "D3" : "B1", "B1" : "C", "B1" : "A5", "B1" : "S", "D3" : "S", "D1" : "S",
        "C" : "A5", "C" : "F", "F" : "A5" }
```

Routing and Mapping of the campus

- (x, y) coordinates for the campus

```
location = { "LY1" : (0, 0), "LY2" : (0, 0), "LY3" : (0, 0), "LY4" : (0, 0), "LY5" : (0, 0), "LY6" : (0, 0), "LY7" : (0, 0),
             "LY8" : (0, 0), "LY9" : (0, 0), "D1" : (0, 0), "D2" : (0, 0), "D3" : (0, 0), "D4" : (0, 0), "D5" : (0, 0), "D6" : (0, 0),
             "A1" : (0, 0), "A2" : (0, 0), "A3" : (0, 0), "A4" : (0, 0), "A5" : (0, 0), "B" : (0, 0), "B1" : (0, 0), "S" : (0, 0),
             "C" : (0, 0), "F" : (0, 0), "M" : (0, 0)}
```

A* algorithm

- Process in a nutshell

  For every iterative step do:
  min (f (n))

- f (n) = g (n) + h (n)

Implementation of A* search

```
from model import astar
from functions import UserInput, answer
from route import location


map  = location

# Reference the values to the dictionary
current_location = UserInput(" ", " ")
current_location.set_start(input())
current_location.set_end(input())

start = current_location.get_start()
end = current_location.get_end()

path = astar(map, start, end)
answer(path)
```

Implementing A* search

- Get user inputs
- astar(map, start, end)
- return path

Impl

```python
def astar(route, start, end):

    # Create start and end node
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0  # zero because it's the start node
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0  # zero because it's the end node

    # Initialize both open and closed list
    open_list = []
    closed_list = []

    # Add the start node
    open_list.append(start_node)

    # Loop until you find the end
    while len(open_list) > 0:

        # Get the current node
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index
```

Implementing A* search

```
# Found the goal
if current_node == end_node:
    path = []
    current = current_node
    while current is not None:
        path.append(current.position)
        current = current.parent
    return path[::-1] # Return reversed path
```