Module: DATA STRUCTURES & ALGORITHMS

CA2 Pizza Runners Report

Lecturer: Ms Lim Ming Heng
Name: Samuel Tay Tze Ming (P2107404), Soh Hong Yu (P2100775)
Group: 9
Class: DAAA/FT/2B/01

Table of Contents

Introduction

This project is a practice on the various Data Structures, Algorithms and the concept of Object-Oriented Programming (OOP) by developing the Pizza Runners Program. The program will be able to generate a maze based on text and will allow the user to choose between different search algorithms such as the Left Hand Rule Search, Breadth First Search, Depth Search, etc, to find the route that connects the start to the end. The project is built on the Turtle library in Python and other Python-builtin libraries like os and random.

User Guideline

Switch to the project directory using the cd command:

```
cd "./DSA"
```

and start the program by using:

```
python main.py [file_name/random]
```

Note that [file_name/random] is an optional parameter. If the parameter is not given, the program will run the default example.txt. If file_name is given, file_name needs to be a valid txt file else the program will not be activated. If random is given, prompts will be shown to guide users how to use random maze generation.

Main Program

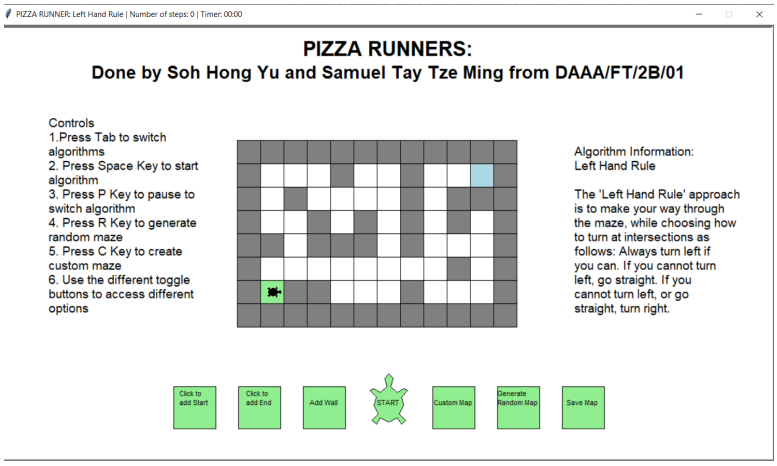


Figure 1: Main Page

The default map will automatically be created and you will be presented with the window [default parameters] in Fig. 1.

You will be given 6 different algorithm options to choose from. Left Hand Rule Search, Right Hand Rule Search, Breadth First Search, Depth Search, A* Search, Greedy Best Search. Choosing any one of the searches will tell the turtle to use that algorithm to solve the maze. The last feature, Free Roam, is an extra feature to allow you to roam freely around the maze. There is also a short

description of the algorithm that will change automatically based on the algorithm that has been selected. There are also a set of controls to tell the user what keys to press for them to be able to use the program.

The set of controls are:

1. Tab to switch algorithms
2. Space to start algorithm
3. P to pause during Free Roam
4. R to generate random maze
5. C to create custom maze
6. Click on the on screen buttons to use the different functions

Additionally there are buttons below the maze that allow you to:

1. Create a custom maze
2. Create start and end points
3. Add/Remove walls to custom maze
4. Save the custom maze
5. Generate a random maze
6. Start the selected algorithm

Algorithms

Option 1: Left Hand Rule Search

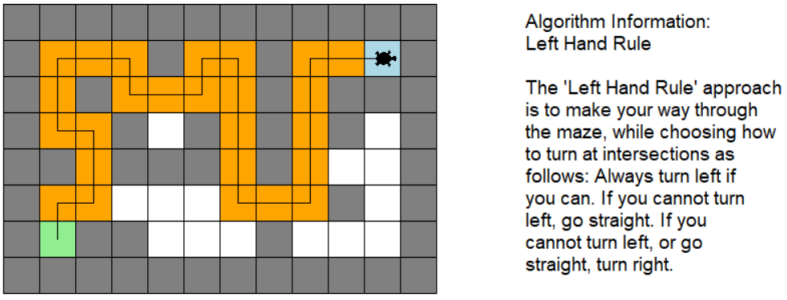


Figure 2: Left Hand Rule

The left-hand rule for solving a maze is a method of traversing a maze in which the solver keeps their left hand in contact with the wall of the maze as they move through it. The idea is that by always keeping one hand in contact with the wall, the solver will eventually come to the exit of the maze, as long as the maze is solvable. Note that if the solver is stuck in the middle with no walls to follow like the following figure, the solver will continuously move forward until it reaches the send or hits a wall.

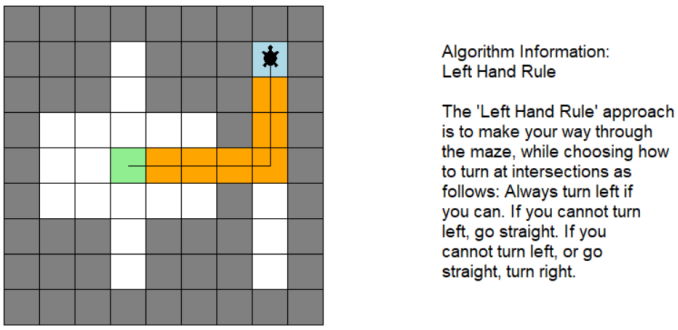


Figure 2.1: Left Hand Rule moving forward continuously due to no wall

Time Complexity: O(n), where n is number of cells in grid
This is because the algorithm will linearly search through the grid which means that the run time of the algorithm grows linearly with the size of the input.

Space Complexity: O(1)
This is because the algorithm uses a constant amount of memory and is independent of the size of the input.

Limitations:
However, we note that there are some limitations for the left hand rule algorithm. There are cases where the algorithm will constantly loop. This is due to the wall's position and the algorithm structure where an internal loop will be formed. *(Refer to Fig 2.2)*

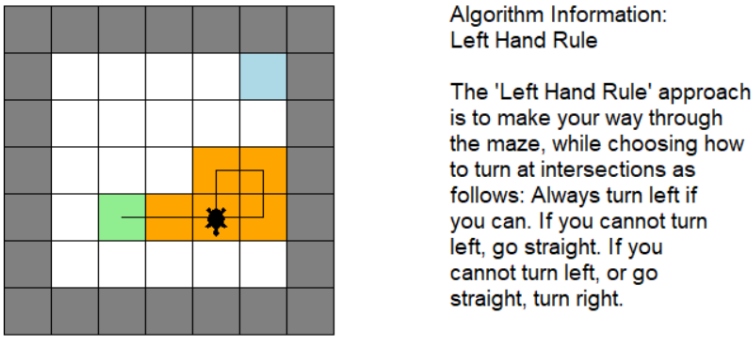


Figure 2.2: Limitation of Left hand rule algorithm

Option 2: Right Hand Rule Search

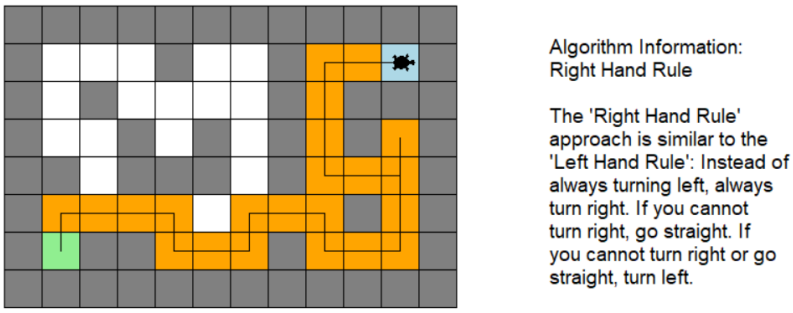


Figure 3: Right Hand Rule

The right-hand rule for solving a maze is the same as the left-hand rule except instead of keeping their left hand in contact with the wall of the maze as they move through it, they use their right. The same **time complexity, space complexity** and **limitations** are the same too.

Option 3: Breadth First Search

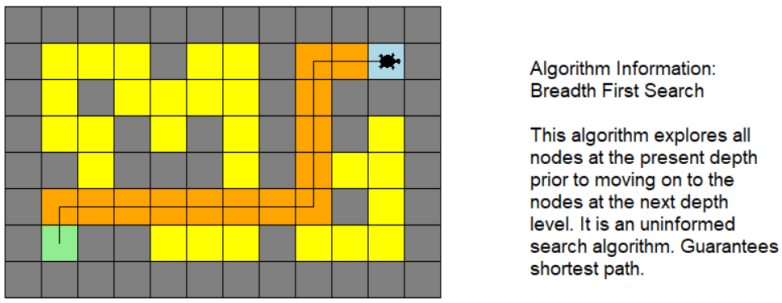


Figure 4: Breadth First Search

Breadth-first search (BFS) can be used to solve a maze by treating the maze as a graph and each cell or location in the maze as a vertex. The idea is to start at the starting point of the maze and explore all neighboring cells before moving on to the next level of cells. This algorithm will guarantee to find a solution if there is one and will also guarantee that the solution found is the shortest path.

Time Complexity: O(b ^ d), where b is number of children nodes and d is the minimum number of steps to reach goal [Worst Case]
This is because in the worst case, all the children nodes will be the value b. Note that this makes the BFS algorithm an exponential algorithm but in our case, it is much faster as the number of children nodes is small, the minimum number of steps to reach a goal is low and the end goal is found early.

Space Complexity: O(b ^ d) [Worst Case]

This is because for the BFS algorithm to work, the nodes are stored in a Queue data structure. In the worst case where every node is searched, the Queue structure will store all the nodes.
Limitations:

BFS consumes large memory space. High time complexity. It has long pathways, when all paths to a destination are on approximately the same search depth.

Data Structures

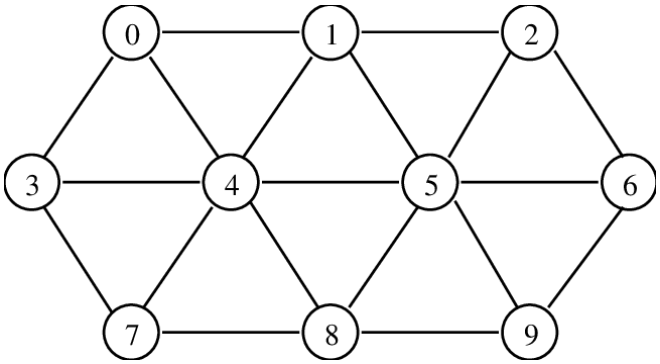
List & Dictionary - Storing information

As there is a lot of information in play for the maze like maze design, start and end points of the maze etc. We stored these information in lists and dictionaries so that they can be easily accessed and searched through without any fancy search or sorting algorithms.

Queue - Storing information for Algorithms

As mentioned in the algorithms page, different algorithms use different methods of keeping track of which node to check through next.
Queue is a first-in-first-out (FIFO) data structure. This means that the node added to the queue first will be explored first. This ensures that the BFS search is performed in a breadth-first manner as the shallowest nodes will be explored first before the deeper nodes. BFS is effective at solving problems with shallow depths and therefore a Queue data structure is used.

Block - Graph



Taken from: https://www.researchgate.net/figure/Example-10-node-graph-An-example-chromosome-for-the-10-node-graph-is-presented-in-Figure_fig1_2259848

For the implementation of the maze, each Block acts like a node in a graph based system. The Block data structure is connected to other Blocks to form a graph-like structure as shown above. The other blocks are connected through their adjacency of the Blocks and if they are adjacent to one other or not. They are all connected and stored in a list called neighbors. Each Block can also be assigned a different type like if it is a building, road or is the start or end point. This connected

graph structure allows the different algorithms like BFS, DFS and A* etc to traverse through the graph in the shortest distance traveled [All edges have the same weights].

Data Structures	**Summary**
List	Stores list of elements
Dictionary	Stores key value pairs for quick
Queue	Stores nodes for the algorithms (BFS)
Block	Node based graph structure.

Object-Oriented Programming

The project is implemented using Object-Oriented Programming (OOP) to make use of the concepts:

1) Encapsulation
2) Polymorphism
3) Abstraction
4) Inheritance

These concepts allow us to protect and restrict certain Class attributes and functions.

Encapsulation

Encapsulation is meant to provide a secure mechanism for hiding the internal implementation details of an object from the outside world, making it easier to change the implementation without affecting the users of the object. Encapsulation also provides a way to control access to the data, making it possible to limit or restrict the ways in which the object's data can be modified.

Implementation

Class	**Name**	**Type**	**Purpose**
Maze	self.__mapString self.__mapArr	Private Variables	To prevent other classes like algorithms and characters from accessing the array and modifying the maze [Data Hiding + Security].
	generate_mapString() draw_map()	Getter and Setter	
Character	self.__roam	Private Variables	As roaming is a variable in character and will not be used anywhere else in the code [Free Roam], this protects the character from external factors like code

			modification to the state etc [Data Hiding + Security].
Text	self.__text	Private Variables	To display the text modularly, encapsulation makes it easiest to maintain and update the code as changing this object does not affect the other object. This also allows the class to be more reusable to modify. We also want to prevent the text from randomly changing as it will need to be displayed on the graphical interface [Data Hiding + Security + Modularity + Reusability].
	__setText()	Private Functions	
	getText() changeText()	Getter and Setter	

Polymorphism

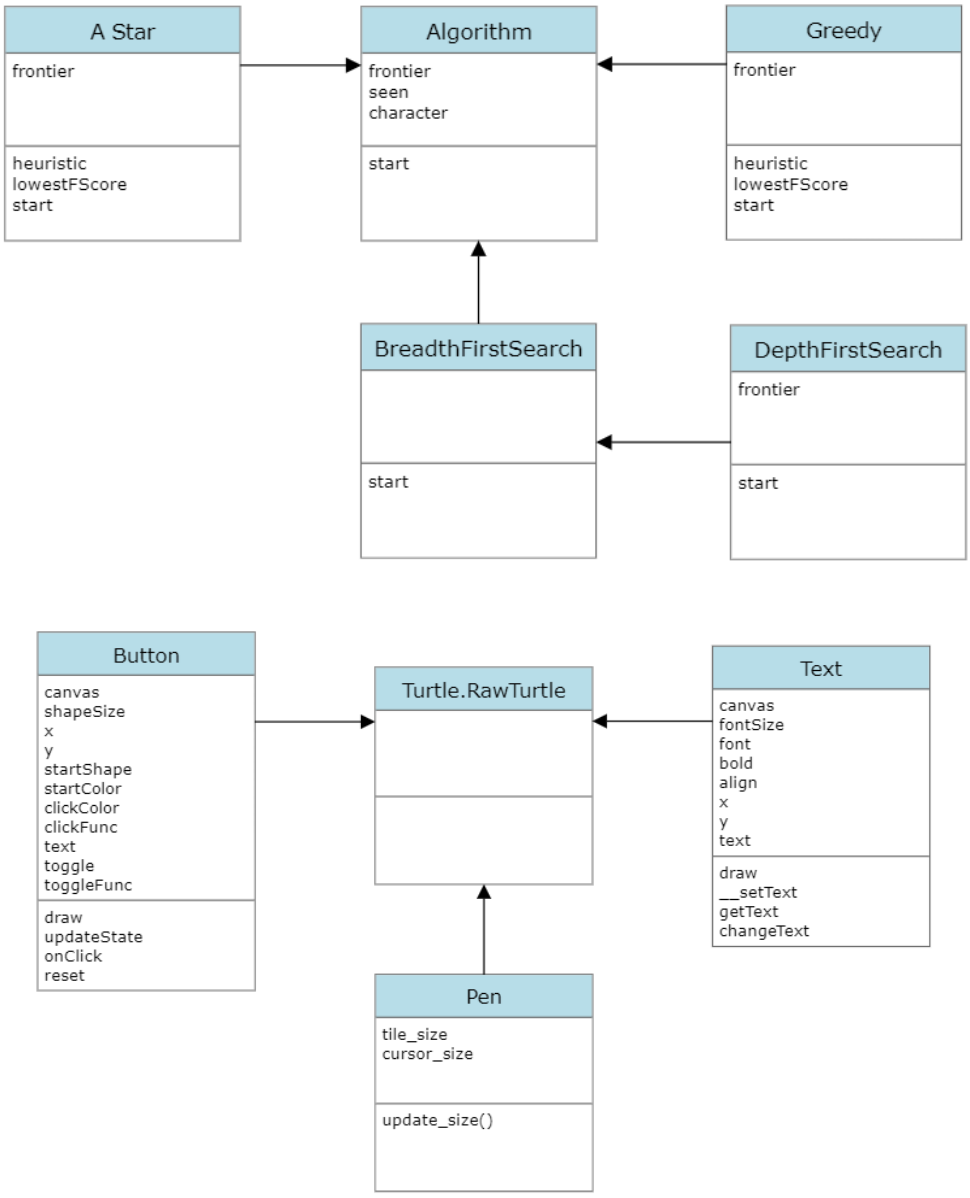
For polymorphism, both function and method overloading was done to overload and extend to fit class needs. For example, one case for overloading was overloading the remove function in Stack class. This function was inherited from the Queue class and as Stack if a LIFO while Queue is a FIFO, by overloading the Stack function instead of recreating classes, this save more computational space as well as make it flexible.

Abstraction

Algorithm class was create with the mind of using abstraction. As the algorithm will be using some of the same functions and variables, by setting Algorithm as an abstract class allows a more modular approach of programming where information on the technical implementation is not given. This allows us to tackle issues for specific algorithms much more easily.

Inheritance

Based on the class table below, we can see that some of the classes are connected through inheritance. Other classes like the UI elements also draw from the same concept of inheritance. This allows classes to use functions like goto and write to be able to make the UI elements better. Not only that but through the process of inheritance, classes can be edit more easily and extend the class to have many more functionality like updating state etc.



Summary of Challenges & Takeaways

Challenges faced by us

There are a few technical challenges we faced when working on this project. Creating the visual for the turtle was difficult as we were not allowed to use libraries such as Tkinter. Writing the algorithms such that the turtle would turn correctly and making sure the different algorithms would reach the end every single time was also quite challenging.

We struggled with time management as a group due to us having our own individual responsibilities outside this project. We had to find time to meet and do the project together and often found it hard to find a time suitable for both of us.

Key Takeaways

This project allowed us to learn how to manage our time better. It also encouraged us to divide the work such that it played to our strengths to have maximum efficiency when doing this project. Creating algorithms for this project also forced us to apply computational thinking - breaking down the problem into simpler problems to have a clearer view allowing us to weigh different solutions and selecting the most effective solution despite the project's restrictions.

Contributions

Name	Tasks	Contribution
Soh Hong Yu	1. GUI Features 2. Algorithm Switch 3. Right Hand Rule 4. Breadth First Search 5. Generate Maze 6. Report 7. Write code comments	50%
Samuel Tay Tze Ming	1. Maze file upload 2. Character Movement 3. Left Hand Rule 4. Breadth First Search 5. Report 6. Controls + Explanation of Models 7. Write code comments	50%

References (Harvard Citation)

1. Wikipedia (2022) Maze Generation algorithm, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Maze_generation_algorithm (Accessed: February 8, 2023).
2. Wikipedia (2022) Breadth-first search, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Breadth-first_search (Accessed: February 8, 2023).
3. Virtual Labs (2020) Depth first search, Virtual Labs. Available at: https://ds1-iiith.vlabs.ac.in/exp/depth-first-search/analysis/space-and-time-complexitiy-of-dfs.html (Accessed: February 8, 2023).
4. Johnson, M. (2008) *Informed search methods*, *Section 04*. Available at: https://www.massey.ac.nz/~mjjohnso/notes/59302/l04.html (Accessed: February 8, 2023).
5. Wikipedia (2023) A* search algorithm, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/A*_search_algorithm (Accessed: February 8, 2023).
6. Visvesvaraya Technological University (2022) A Star algorithm and explanation - A star (A*) search algorithm, advantages and disadvantages –, Studocu. Available at: https://www.studocu.com/in/document/visvesvaraya-technological-university/machine-learning/a-star-algorithm-and-explanation/37361609 (Accessed: February 8, 2023).
7. Team, G.L. (2022) Best first search algorithm in AI: Concept, implementation, advantages, disadvantages, Great Learning Blog: Free Resources what Matters to shape your Career! Available at: https://www.mygreatlearning.com/blog/best-first-search-bfs/#advantages-and-disadvantages-of-best-first-search (Accessed: February 8, 2023).

Appendix - Source Code Listings

main.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

# Import Libraries
import sys
import os
import turtle as t
from Maze.Maze import Maze
from Character.Character import Character
from Screen.Text import Text
from Screen.Button import Button
from datetime import datetime, timedelta

# Create Algorithm List ~ Hong Yu + Samuel
ALGO_LIST = ["Left Hand Rule", "Right Hand Rule", "Breadth First
Search",
             "Depth First Search", "A* Search", "Greedy Best First
Search", "Free Roam"]

# Create Algorithm Information to be displayed ~ Samuel
ALGO_INFO = {
```

```
    "Left Hand Rule": "The 'Left Hand Rule' approach is to make your
way through the maze, while choosing how to turn at intersections as
follows: Always turn left if you can. If you cannot turn left, go
straight. If you cannot turn left, or go straight, turn right.",
    "Right Hand Rule": "The 'Right Hand Rule' approach is similar to
the 'Left Hand Rule': Instead of always turning left, always turn
right. If you cannot turn right, go straight. If you cannot turn right
or go straight, turn left.",
    "Breadth First Search": "This algorithm explores all nodes at the
present depth prior to moving on to the nodes at the next depth level.
It is an uninformed search algorithm.\nGuarantees shortest path.",
    "Depth First Search": "This algorithm starts at the root node and
explores as far as possible along each branch before backtracking. It
is an uninformed search algorithm.\nDoes not guarantee shortest path",
    "A* Search": "This algorithm is the most optimal in terms of time
efficiency. It is an informed search algorithm that utilises 2
heuristics to find the shortest path.\nGuarantees shortest path.",
    "Greedy Best First Search": "This algorithm is an informed search
algorithm that only utilises one heuristic function that always
chooses the path which appear best at the moment.\nDoes not guarantee
shortest path.",
    "Free Roam": "Use Arrow keys to move around. Have fun!"
}

# Zero based index to select algorithm
currentAlgo = 0

# Setup root screen ~ Hong Yu
root = t.Screen()
ogTitle = f"PIZZA RUNNER: {ALGO_LIST[currentAlgo]} | Number of steps:
0 | Timer: 00:00"
root.title(ogTitle)
title = ogTitle
```

```
root.setup(1200, 675)
root.cv._rootwindow.resizable(False, False)

# Setup global variables as None ~ Hong Yu + Samuel
character = None
heading = None
maze = None
isRunning = False
algoText = None
startBtn = None
firstTime = True
customMapBtn = None
wallBtn = None
addStartBtn = None
addEndBtn = None

# UI Improvements ~ Hong Yu
# Update timer
def updateTimer():
    global character
    global maze
    global isRunning
    # Check the state of the character
    if character.state:
        updateTitle()
    isRunning = True
    root.ontimer(updateTimer, 1000)
    return

# Update title ~ Hong Yu
def updateTitle():
    global title
    # Split to get and edit the different parts of the title
```

```
    titleArr = title.split(" | Timer: ")
    stepArr = titleArr[0].split("steps: ")
    stepArr[1] = str(character.step)
    titleArr[0] = "steps: ".join(stepArr)
    time = datetime.strptime(titleArr[1], "%M:%S")
    time += timedelta(seconds=1)
    titleArr[1] = time.strftime("%M:%S")
    title = " | Timer: ".join(titleArr)
    root.title(title)

# Reset title ~ Hong Yu
def resetTitle():
    global ogTitle
    global title
    ogTitle = f"PIZZA RUNNER: {ALGO_LIST[currentAlgo]} | Number of
steps: 0 | Timer: 00:00"
    title = ogTitle
    root.title(ogTitle)

# Break Text if string is too long ~ Hong Yu
def breakText(string, maxLength = 30):
    words = string.split()
    lines = []
    line = ""
    for word in words:
        if len(line) + len(word) + 1 <= maxLength:
            line += word + " "
        else:
            lines.append(line)
            line = word + " "
    lines.append(line)
    return "\n".join(lines)
```

```
# Main Functionalities
# Start algorithms ~ Hong Yu + Samuel
def start():
    # Access global variables
    global character
    global heading
    global maze
    global isRunning
    global startBtn
    global firstTime
    global customMapBtn
    # Check if custom is running
    if customMapBtn.toggleState:
        return
    # Reset title
    resetTitle()
    # Modify the heading
    headingText = heading.getText()
    if " Unsolvable" in headingText:
        headingText = headingText.split(" Unsolvable")[0]
    # Ensure character and maze is not on edit mode
    if not character.state and not maze.state:
        # Reset title for timer and step counter
        resetTitle()
        heading.changeText(headingText + " " + ALGO_LIST[currentAlgo])
        if not isRunning:
            updateTimer()
        solvable = character.start(ALGO_LIST[currentAlgo], firstTime)
        updateTitle()
        # Checks if solvable
        if solvable:
            heading.changeText(headingText)
        else:
```

```
            heading.changeText(headingText + " Unsolvable")
        # Reset startBtn
        startBtn.reset()
    elif not character.state:
        # Reset startBtn
        startBtn.reset()
    # Remove firstTime run
    firstTime = False

# Switch Algorithms ~ Hong Yu + Samuel
def switchAlgo():
    global character
    global currentAlgo
    global startBtn
    global algoText
    global customMapBtn
    if not customMapBtn.toggleState:
        if not character.state:
            currentAlgo += 1
            if currentAlgo > len(ALGO_LIST) - 1:
                currentAlgo = 0
            algoText.changeText(
                f"Algorithm
Information:\n{ALGO_LIST[currentAlgo]}\n\n{breakText(ALGO_INFO[ALGO_LI
ST[currentAlgo]])}")
            resetTitle()
            startBtn.reset()
        else:
            print("Algorithm cannot be switched when turtle is
running")

# Soh Hong Yu's Additional Features - Custom Maze
# Check x,y position of mouse click
```

```
def clickWalls(x, y):
    global maze
    global character
    global customMapBtn
    global wallBtn
    global addStartBtn
    global addEndBtn
    # Hide turtle
    if character is not None:
        character.hideturtle()
    if customMapBtn is not None:
        if customMapBtn.toggleState:
            gotRow = False
            gotCol = False
            # Loop through to check if the rows/cols are in range of
mouse click
            for row in range(maze.rows):
                if y < maze.get_mapArr()[row][0].y + maze.size / 2 and
y > maze.get_mapArr()[row][0].y - maze.size / 2:
                    gotRow = True
                    break
            for col in range(maze.columns):
                if x < maze.get_mapArr()[0][col].x + maze.size / 2 and
x > maze.get_mapArr()[0][col].x - maze.size / 2:
                    gotCol = True
                    break
            # Loop through and update the map according to the changes
            if gotCol and gotRow:
                if not wallBtn.toggleState:
                    # Check what kind of wall to add
                    if addStartBtn.toggleState:
                        # Ensure that only 1 start and end exist at 1
time
```

```
                    for r in range(maze.rows):
                        for c in range(maze.columns):
                            if maze.get_mapArr()[r][c].is_start():
                                maze.get_mapArr()[r][c].reset()
                                maze.get_mapArr()[r][c].draw()
                    maze.get_mapArr()[row][col].make_start()
                    addStartBtn.toggleState = False
                    addStartBtn.updateState()
                elif addEndBtn.toggleState:
                    for r in range(maze.rows):
                        for c in range(maze.columns):
                            if maze.get_mapArr()[r][c].is_end():
                                maze.get_mapArr()[r][c].reset()
                                maze.get_mapArr()[r][c].draw()
                    maze.get_mapArr()[row][col].make_end()
                    addEndBtn.toggleState = False
                    addEndBtn.updateState()
                else:
                    maze.get_mapArr()[row][col].make_wall()
            else:
                maze.get_mapArr()[row][col].reset()
            maze.get_mapArr()[row][col].draw()
            # Generate maze as string
            maze.generate_mapString()


# Add Wall
def addWall():
    global addStartBtn
    global addEndBtn
    global wallBtn
    if addStartBtn.toggleState:
        addStartBtn.toggleState = False
```

```
        addStartBtn.updateState()
    if addEndBtn.toggleState:
        addEndBtn.toggleState = False
        addEndBtn.updateState()

# Add End


def addEnd():
    global addStartBtn
    global addEndBtn
    global wallBtn
    if addStartBtn.toggleState:
        addStartBtn.toggleState = False
        addStartBtn.updateState()
    if wallBtn.toggleState:
        wallBtn.toggleState = False
        wallBtn.updateState()

# Add Start
def addStart():
    global addStartBtn
    global addEndBtn
    global wallBtn
    if addEndBtn.toggleState:
        addEndBtn.toggleState = False
        addEndBtn.updateState()
    if wallBtn.toggleState:
        wallBtn.toggleState = False
        wallBtn.updateState()

# Make maze
def make_maze():
```

```
    # Global maze
    global maze
    global character
    global customMapBtn
    checkStart = False
    checkEnd = False
    # Check if end and start exist to make sure maze is valid
    for r in range(maze.rows):
        for c in range(maze.columns):
            if maze.get_mapArr()[r][c].is_end():
                checkEnd = True
            if maze.get_mapArr()[r][c].is_start():
                checkStart = True
    if checkStart and checkEnd:
        # Draw map
        maze.draw_map(root)
        maze.generate_mapString()
        character = Character(
            canvas=root, x=maze.hashmap['start'][0].x,
y=maze.hashmap['start'][0].y, maze=maze, size=maze.size)
        character.reset_everything()
    else:
        # Reset custom map if cannot find start and end points
        print("No start point/end point found")
        customMapBtn.toggleState = True
        customMapBtn.updateState()
        custom_map()
    root.onscreenclick(None)
    root.mainloop()

# Link to custom map button
def custom_map():
    global maze
```

```
    global character
    global customMapBtn
    # Hide character
    character.hideturtle()
    # Create custom map
    if not maze.state and not character.state:
        rows, cols = getRowsAndCols(True)
        maze.custom_map(rows, cols, root)
        if character is not None:
            character.reset_everything()
            character.hideturtle()
        # Allow click
        root.onscreenclick(clickWalls)
        root.mainloop()
    else:
        customMapBtn.toggleState = False
        customMapBtn.updateState()

# Save maze
def save_maze():
    global maze
    # Get file path and check if file path exist
    fileMsg = "Enter save filename/filepath"
    while True:
        filePath = root.textinput(fileMsg, fileMsg)
        if filePath[-4:] != ".txt":
            fileMsg = "Invalid filename/filepath! Enter save
filename/filepath"
            continue
        if ":\\" not in filePath:
            filePath = os.path.abspath(
                os.getcwd()) + "\\" + filePath
        break
```

```
    maze.saveFile(filePath) # Save file based on filepath
    return

# Soh Hong Yu's Additional Features - Randomized Maze
def generate_maze():
    global maze
    global character
    try:
        if not maze.state and not character.state:
            rows, cols = getRowsAndCols()
            maze.generate_maze(rows, cols, root)
            if character is not None:
                character.reset_everything()
    except AttributeError:
        rows, cols = getRowsAndCols()
        maze.generate_maze(rows, cols, root)

# get row and col for the maze
def getRowsAndCols(allowEven = False):
    rowsMsg = "Enter total number of rows"
    while True:
        try:
            rows = root.textinput(rowsMsg, rowsMsg)
            rows = int(rows)
            if rows >= 5 and rows <= 60:
                if allowEven or rows % 2 != 0:
                    break
                else:
                    rowsMsg = "Row number must be odd! Enter total
number of rows"
            else:
                rowsMsg = "Row number must be between 5 to 60! Enter
total number of rows"
```

```
        except ValueError:
            rowsMsg = "Invalid row number! Enter total number of rows"
    colsMsg = "Enter total number of cols"
    while True:
        try:
            cols = root.textinput(colsMsg, colsMsg)
            cols = int(cols)
            if cols >= 5 and cols <= 60:
                if allowEven or cols % 2 != 0:
                    break
                else:
                    colsMsg = "Col number must be odd! Enter total
number of cols"
            else:
                colsMsg = "Col number must be between 5 to 60! Enter
total number of cols"
        except ValueError:
            colsMsg = "Invalid col number! Enter total number of cols"
    return rows, cols

# Main Function ~ Hong Yu + Samuel
def main():
    # Global Variables
    global character
    global heading
    global maze
    global algoText
    global startBtn
    global customMapBtn
    global wallBtn
    global addStartBtn
    global addEndBtn
    filePath = None
```

```
    random = False
    # Get parameters from cmd
    if (len(sys.argv) == 1):
        print("Using default maze")
        filePath = "example.txt"
    if not filePath:
        txtFile = sys.argv[1]
        if txtFile == "random":
            print("Generating random map")
            random = True
        elif txtFile[-4:] != ".txt":
            print("Invalid file type")
            return
        filePath = txtFile
    # Check if not random and see if file exist
    if not random:
        if ":\\" not in filePath:
            filePath = os.path.abspath(
                os.getcwd()) + "\\" + filePath
        try:
            open(filePath, 'r', encoding="utf8")
        except FileNotFoundError:
            print("File does not exist! Please try again!")
            return
    # Create a maze instance
    maze = Maze(root)
    if not random:
        # Upload map
        error = maze.upload_map(filePath)
        if error:
            print(f"Map Upload Error: {error}")
            return
    elif random:
```

```
        # Generate maze
        generate_maze()
    # UI
    heading = Text("PIZZA RUNNERS:", root, x=0,
                   y=285, bold="bold", fontSize=24)
    heading.draw()
    doneBy = Text("Done by Soh Hong Yu and Samuel Tay Tze Ming from
DAAA/FT/2B/01",
                  root, x=0, y=250, bold="bold", fontSize=20)
    doneBy.draw()
    maze.draw_map(root)
    character = Character(
        canvas=root, x=maze.hashmap['start'][0].x,
y=maze.hashmap['start'][0].y, maze=maze, size=maze.size)
    instructions = Text("Controls\n" + breakText('1.Press Tab to
switch algorithms') + "\n"
                        + breakText("2. Press Space Key to start
algorithm") + "\n"
                        + breakText("3. Press P Key to pause to switch
algorithm") + "\n"
                        + breakText("4. Press R Key to generate random
maze") + "\n"
                        + breakText("5. Press C Key to create custom
maze") + "\n"
                        + breakText("6. Use the different toggle
buttons to access different options"),
                        root, x=maze.endX - maze.size * 2,
y=-maze.startY + maze.size, bold="normal", fontSize=14, align="right")
    instructions.draw()
    algoText = Text(f"Algorithm
Information:\n{ALGO_LIST[currentAlgo]}\n\n{breakText(ALGO_INFO[ALGO_LI
ST[currentAlgo]])}",
```

```
                    root, x=-maze.endX + maze.size * 2, y=-maze.startY
+ maze.size, bold="normal", fontSize=14, align="left")
    algoText.draw()
    addStartBtn = Button(root, x=-300, y=-250, startShape="square",
                    text="Click to\nadd Start", size=3.25,
clickText="Adding Start", clickFunc=addStart, toggle=True)
    addStartBtn.draw()
    addEndBtn = Button(root, x=-200, y=-250, startShape="square",
                    text="Click to\nadd End", size=3.25,
clickText="Adding End",clickFunc=addEnd, toggle=True)
    addEndBtn.draw()
    wallBtn = Button(root, x=-100, y=-250, startShape="square",
                    text="Add Wall", size=3.25, clickText="Remove
Wall", toggle=True, clickFunc=addWall)
    wallBtn.draw()
    customMapBtn = Button(root, x=100, y=-250, startShape="square",
                          text="Custom Map", size=3.25,
clickFunc=custom_map, clickText="Running\nCustom Map", toggle=True,
toggleFunc=make_maze)
    customMapBtn.draw()
    randomMapBtn = Button(root, x=200, y=-250, startShape="square",
                     text="Generate\nRandom Map", size=3.25,
clickFunc=generate_maze)
    randomMapBtn.draw()
    saveMapBtn = Button(root, x=300, y=-250, startShape="square",
                    text="Save Map", size=3.25, clickFunc=save_maze)
    saveMapBtn.draw()
    startBtn = Button(root, x=0, y=-250, startShape="turtle",
text="START",
                    size=3.25, clickFunc=start, clickText="RUNNING")
    startBtn.draw()
    root.delay(0)
    root.listen()
```

```
    # Key presses
    root.onkey(switchAlgo, 'Tab')
    root.onkey(start, 'space')
    root.onkey(generate_maze, 'r')
    root.onkey(custom_map, 'c')
    root.onkey(character.updateState, "p")
    root.ontimer(updateTimer, 1)
    root.onscreenclick(None)
    root.mainloop()
    return

# Check and ensure it is the main python file run
if __name__ == "__main__":
    main()
```

Algorithm.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

# Import Queue
from Algorithm.Queue import Queue

# Create Abstract  ~ Hong Yu
class Algorithm:
```

```
  def __init__(self, character):
    self.character = character
    self.frontier = Queue()
    self.seen = []

  # Abstract Functions
  def start(self, start, end):
    raise NotImplementedError("Subclass must implement abstract
method")
```

AStar.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from Algorithm.Algorithm import Algorithm
import time

# Inherit Algorithm ~ Samuel
class AStar(Algorithm):
  def __init__(self, character):
    # Super parent
    super().__init__(character)
    #Priority Queue
```

```
  self.frontier = []

# Calculate heuristic [Manhattan Distance]
def heuristic(self, node, end):
  return abs(node.row - end.row) + abs(node.col - end.col)

# Returns node from the open list with the lowest f score
def lowestFScore(self, openset):
  winner = openset[0]
  for nodes in openset:
    if nodes.f_score < winner.f_score:
      winner = nodes
  return winner

# Overloading Abstraction Function
def start(self, start, end):
  # Start priority queue
  self.frontier = [start]
  start.g_score = 0
  start.f_score = self.heuristic(start, end)
  # Check and ensure frontier > 0
  while len(self.frontier) > 0:
    currentGrid = self.lowestFScore(self.frontier)
    # Update neighbors
    currentGrid.update_neighbors(self.character.maze.get_mapArr())
    # Stop if state changes
    if not self.character.state:
        return
    # Check if it is the end
    if currentGrid.is_end():
      # Backtrack back to start position
      path = [currentGrid]
```

```
        while not (currentGrid.row == start.row and currentGrid.col ==
start.col):
          path.append(currentGrid.parent)
          currentGrid = currentGrid.parent
        return path
      # Color cell if not start
      if not currentGrid.is_start():
        currentGrid.make_open()
        currentGrid.draw()
      # Append frontier and seen
      self.frontier.remove(currentGrid)
      self.seen.append(currentGrid)
      # Loop through all neighbor
      for neighbor in currentGrid.neighbors:
        # Ignore if already checked
        if neighbor in self.seen:
          continue
        # Update g_score and f_score
        temp_g = currentGrid.g_score + 1
        if neighbor not in self.frontier:
          neighbor.g_score = temp_g
          neighbor.f_score = self.heuristic(neighbor, end) +
neighbor.g_score
          self.frontier.append(neighbor)
          neighbor.parent = currentGrid
        elif temp_g < neighbor.g_score:
          neighbor.g_score = temp_g
          neighbor.f_score = self.heuristic(neighbor, end) +
neighbor.g_score
          neighbor.parent = currentGrid
      time.sleep(0.01)
    # Invalid Maze
    if len(self.frontier) == 0:
```

```
      print("Invalid Maze")
    return
```

BreadthFirstSearch.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from Algorithm.Algorithm import Algorithm
import time

# Inherit Algorithm ~ Samuel
class AStar(Algorithm):
  def __init__(self, character):
    # Super parent
    super().__init__(character)
    #Priority Queue
    self.frontier = []

  # Calculate heuristic [Manhattan Distance]
  def heuristic(self, node, end):
    return abs(node.row - end.row) + abs(node.col - end.col)
```

```
  # Returns node from the open list with the lowest f score
  def lowestFScore(self, openset):
    winner = openset[0]
    for nodes in openset:
      if nodes.f_score < winner.f_score:
        winner = nodes
    return winner

  # Overloading Abstraction Function
  def start(self, start, end):
    # Start priority queue
    self.frontier = [start]
    start.g_score = 0
    start.f_score = self.heuristic(start, end)
    # Check and ensure frontier > 0
    while len(self.frontier) > 0:
      currentGrid = self.lowestFScore(self.frontier)
      # Update neighbors
      currentGrid.update_neighbors(self.character.maze.get_mapArr())
      # Stop if state changes
      if not self.character.state:
          return
      # Check if it is the end
      if currentGrid.is_end():
        # Backtrack back to start position
        path = [currentGrid]
        while not (currentGrid.row == start.row and currentGrid.col ==
start.col):
          path.append(currentGrid.parent)
          currentGrid = currentGrid.parent
        return path
      # Color cell if not start
      if not currentGrid.is_start():
```

```
        currentGrid.make_open()
        currentGrid.draw()
      # Append frontier and seen
      self.frontier.remove(currentGrid)
      self.seen.append(currentGrid)
      # Loop through all neighbor
      for neighbor in currentGrid.neighbors:
        # Ignore if already checked
        if neighbor in self.seen:
          continue
        # Update g_score and f_score
        temp_g = currentGrid.g_score + 1
        if neighbor not in self.frontier:
          neighbor.g_score = temp_g
          neighbor.f_score = self.heuristic(neighbor, end) +
neighbor.g_score
          self.frontier.append(neighbor)
          neighbor.parent = currentGrid
        elif temp_g < neighbor.g_score:
          neighbor.g_score = temp_g
          neighbor.f_score = self.heuristic(neighbor, end) +
neighbor.g_score
          neighbor.parent = currentGrid
      time.sleep(0.01)
    # Invalid Maze
    if len(self.frontier) == 0:
      print("Invalid Maze")
    return
```

DepthFirstSearch.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from turtle import RawTurtle

# Inherit RawTurtle ~ Hong Yu
class Text(RawTurtle):
    def __init__(self, text, canvas = None, font=None, bold="normal",
fontSize=15,align="center",x=0,y=0):
        # Super parent
        super().__init__(canvas)
        # Setup
        self.canvas = canvas
        self.fontSize = fontSize
        self.font = font
        self.bold = bold
        self.align = align
        self.x = x
        self.y = y
        # Set text to private to prevent text alteration
        self.__text = text
        self.hideturtle()

    # Draw text for visualisation
```

```
    def draw(self):
        self.hideturtle()
        self.penup()
        self.speed('fastest')
        self.setpos(self.x,self.y)
        self.write(self.__text,
font=(self.font,self.fontSize,self.bold),align=self.align)

    # Private Setter function
    def __setText(self, text):
        self.__text = text

    # Getter Function
    def getText(self):
        return self.__text

    # Public setter function to modify text and re render
    def changeText(self, text):
        self.__setText(text)
        self.clear()
        self.draw()
```

Greedy.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
```

```
"""

from Algorithm.Algorithm import Algorithm
import time

# Inherit Algorithm ~ Samuel
class Greedy(Algorithm):
  def __init__(self, character):
    # Super parent
    super().__init__(character)
    #Priority Queue
    self.frontier = []

  # Calculate heuristic [Manhattan Distance]
  def heuristic(self, node, end):
    return abs(node.row - end.row) + abs(node.col - end.col)

  # Returns node from the open list with the lowest f score
  def lowestFScore(self, openset):
    winner = openset[0]
    for nodes in openset:
      if nodes.f_score < winner.f_score:
        winner = nodes
    return winner

  # Overloading Abstraction Function
  def start(self, start, end):
    # Start priority queue
    self.frontier = [start]
    start.f_score = self.heuristic(start, end)
    # Check and ensure frontier > 0
    while len(self.frontier) > 0:
      currentGrid = self.lowestFScore(self.frontier)
```

```
      # Update neighbors
      currentGrid.update_neighbors(self.character.maze.get_mapArr())
      # Stop if state changes
      if not self.character.state:
            return
      # Check if it is the end
      if currentGrid.is_end():
        # Backtrack back to start position
        path = [currentGrid]
        while not (currentGrid.row == start.row and currentGrid.col ==
start.col):
          path.append(currentGrid.parent)
          currentGrid = currentGrid.parent
        return path
      # Color cell if not start
      if not currentGrid.is_start():
        currentGrid.make_open()
        currentGrid.draw()
      # Append frontier and seen
      self.frontier.remove(currentGrid)
      self.seen.append(currentGrid)
      # Loop through all neighbor
      for neighbor in currentGrid.neighbors:
        # Ignore if already checked
        if neighbor in self.seen:
          continue
        # Update g_score and f_score
        neighbor.f_score = self.heuristic(neighbor, end)
        self.frontier.append(neighbor)
        neighbor.parent = currentGrid
      time.sleep(0.01)
    # Invalid Maze
    if len(self.frontier) == 0:
```

```
      print("Invalid Maze")
    return
```

Queue.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

# Create Queue class ~ Hong Yu + Samuel
# As Queue is FIFO, add and remove is based on append() and pop(0)
class Queue:
  def __init__(self):
    self.frontier = []

  # Check empty
  def isEmpty(self):
    return len(self.frontier) == 0

  # Append to frontier
  def add(self, grid):
    self.frontier.append(grid)

  # Remove from frontier
```

```
  def remove(self):
    return self.frontier.pop(0)

  # Check if same
  def contains(self, grid):
    return any(element.row == grid.row and element.col == grid.col for
element in self.frontier)

  # Method overloading
  def __str__(self):
    queueString = "["
    for i in self.frontier:
      queueString += str(i) + ","
    queueString = queueString[:-1]
    queueString += "]"
    return queueString
```

Stack.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from Algorithm.Queue import Queue

# Inherit Queue ~ Hong Yu + Samuel
```

```
class Stack(Queue):
  def __init__(self):
    # Super parent
    super().__init__()

  # Method overloading
  def remove(self):
    return self.frontier.pop()
```

Button.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from turtle import RawTurtle

# Inherit RawTurtle ~ Hong Yu
class Button(RawTurtle):
    def __init__(self, canvas=None, x=0, y=0, text="",
startShape="square", size=2, startColor="lightgreen",
clickColor="green", clickFunc=None, clickText=None, toggle=False,
toggleFunc=None):
        super().__init__(canvas)
        # Setup button
        self.canvas = canvas
```

```
        self.shapeSize = size
        self.x = x
        self.y = y
        self.startShape = startShape
        self.startColor = startColor
        self.clickColor = clickColor
        self.clickFunc = clickFunc
        self.text = text
        self.toggle = toggle
        # Make if button is toggleable
        if self.toggle:
            self.toggleState = False
            self.toggleFunc = toggleFunc
        if clickText:
            self.clickText = clickText
        else:
            self.clickText = self.text
        self.hideturtle()
        return

    # Draw and create a new instance of button
    def draw(self):
        # Setup
        self.hideturtle()
        self.speed('fastest')
        self.shapesize(self.shapeSize)
        self.shape(self.startShape)
        self.fillcolor(self.startColor)
        self.penup()
        self.goto(self.x, self.y)
        self.onclick(self.onClick)
        self.setheading(90)
        self.showturtle()
```

```
        self.clear()
        self.write(self.text, align='center')
        # Modify state
        if self.toggle:
            if self.toggleState:
                self.fillcolor(self.clickColor)
                self.clear()
                self.write(self.clickText, align='center')
            else:
                self.fillcolor(self.startColor)
                self.clear()
                self.write(self.text, align='center')

    # Update state of button
    def updateState(self):
        if self.toggleState:
            self.fillcolor(self.clickColor)
            self.clear()
            self.write(self.clickText, align='center')
        else:
            self.fillcolor(self.startColor)
            self.clear()
            self.write(self.text, align='center')

    # Handle onclick of button
    def onClick(self, x, y):
        self.fillcolor(self.clickColor)
        self.clear()
        # Check if toggleable
        if self.toggle:
            self.toggleState = not self.toggleState
            if self.toggleState:
                self.fillcolor(self.clickColor)
```

```
                self.clear()
                self.write(self.clickText, align='center')
                if self.clickFunc != None:
                    self.clickFunc()
            else:
                self.fillcolor(self.startColor)
                self.clear()
                self.write(self.text, align='center')
                if self.toggleFunc != None:
                    self.toggleFunc()
        else:
            self.write(self.clickText, align='center')
            if self.clickFunc != None:
                self.clickFunc()
            self.reset()
        return

    # Reset button
    def reset(self):
        self.fillcolor(self.startColor)
        self.clear()
        self.write(self.text, align='center')
        return
```

Character.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
```

```
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from turtle import RawTurtle

# Inherit RawTurtle ~ Hong Yu
class Button(RawTurtle):
    def __init__(self, canvas=None, x=0, y=0, text="",
startShape="square", size=2, startColor="lightgreen",
clickColor="green", clickFunc=None, clickText=None, toggle=False,
toggleFunc=None):
        super().__init__(canvas)
        # Setup button
        self.canvas = canvas
        self.shapeSize = size
        self.x = x
        self.y = y
        self.startShape = startShape
        self.startColor = startColor
        self.clickColor = clickColor
        self.clickFunc = clickFunc
        self.text = text
        self.toggle = toggle
        # Make if button is toggleable
        if self.toggle:
            self.toggleState = False
            self.toggleFunc = toggleFunc
        if clickText:
            self.clickText = clickText
        else:
            self.clickText = self.text
        self.hideturtle()
```

```
        return

    # Draw and create a new instance of button
    def draw(self):
        # Setup
        self.hideturtle()
        self.speed('fastest')
        self.shapesize(self.shapeSize)
        self.shape(self.startShape)
        self.fillcolor(self.startColor)
        self.penup()
        self.goto(self.x, self.y)
        self.onclick(self.onClick)
        self.setheading(90)
        self.showturtle()
        self.clear()
        self.write(self.text, align='center')
        # Modify state
        if self.toggle:
            if self.toggleState:
                self.fillcolor(self.clickColor)
                self.clear()
                self.write(self.clickText, align='center')
            else:
                self.fillcolor(self.startColor)
                self.clear()
                self.write(self.text, align='center')

    # Update state of button
    def updateState(self):
        if self.toggleState:
            self.fillcolor(self.clickColor)
            self.clear()
```

```
            self.write(self.clickText, align='center')
        else:
            self.fillcolor(self.startColor)
            self.clear()
            self.write(self.text, align='center')

    # Handle onclick of button
    def onClick(self, x, y):
        self.fillcolor(self.clickColor)
        self.clear()
        # Check if toggleable
        if self.toggle:
            self.toggleState = not self.toggleState
            if self.toggleState:
                self.fillcolor(self.clickColor)
                self.clear()
                self.write(self.clickText, align='center')
                if self.clickFunc != None:
                    self.clickFunc()
            else:
                self.fillcolor(self.startColor)
                self.clear()
                self.write(self.text, align='center')
                if self.toggleFunc != None:
                    self.toggleFunc()
        else:
            self.write(self.clickText, align='center')
            if self.clickFunc != None:
                self.clickFunc()
            self.reset()
        return

    # Reset button
```

```
    def reset(self):
        self.fillcolor(self.startColor)
        self.clear()
        self.write(self.text, align='center')
        return
```

Block.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from Maze.Pen import Pen

# Create node which acts like a graph node ~ Hong Yu
class Block:
    def __init__(self, row, col, size, total_rows, total_cols, canvas,
x=None, y=None):
        # Preset details
        self.row = row
        self.col = col
        self.x = x if x is not None else row * size
        self.y = y if y is not None else col * size
        self.color = "white"
        self.neighbors = []
        self.size = size
```

```
        self.total_rows = total_rows
        self.total_cols = total_cols
        self.canvas = canvas
        self.parent = None
        self.g_score = float('inf')
        self.f_score = float('inf')

    # Get position of block
    def get_pos(self):
        return self.row, self.col

    # Check if is reset
    def is_reset(self):
        return self.color == "white"

    # Check if is open
    def is_open(self):
        return self.color == "yellow"

    # Check if is path
    def is_path(self):
        return self.color == "orange"

    # Check if is wall
    def is_wall(self):
        return self.color == "grey"

    # Check if is start
    def is_start(self):
        return self.color == "lightgreen"

    # Check if is end
    def is_end(self):
```

```
        return self.color == "lightblue"

    # Reset color
    def reset(self):
        self.color = "white"

    # Set start color
    def make_start(self):
        self.color = "lightgreen"

    # Set open color
    def make_open(self):
        self.color = "yellow"

    # Set wall color
    def make_wall(self):
        self.color = "grey"

    # Set end color
    def make_end(self):
        self.color = "lightblue"

    # Set path color
    def make_path(self):
        self.color = "orange"

    # Draw/Color block
    def draw(self):
        try:
            self.pen.hideturtle()
        except AttributeError:
            self.pen = Pen(canvas=self.canvas, tile_size=self.size)
        self.pen.speed("fastest")
```

```
        self.pen.hideturtle()
        self.pen.fillcolor(self.color)
        self.pen.setpos(self.x, self.y)
        self.pen.stamp()
        return

    # Clear drawings
    def clear_stamps(self):
        try:
            self.pen.clearstamps()
        except AttributeError:
            return

    # Update neighbors [Connect them using edges and vertex]
    def update_neighbors(self, grid):
        self.neighbors = []
        # DOWN
        if self.row < self.total_rows - 1 and not grid[self.row +
1][self.col].is_wall():
            self.neighbors.append(grid[self.row + 1][self.col])

        # UP
        if self.row > 0 and not grid[self.row -
1][self.col].is_wall():
            self.neighbors.append(grid[self.row - 1][self.col])

        # RIGHT
        if self.col < self.total_cols - 1 and not
grid[self.row][self.col + 1].is_wall():
            self.neighbors.append(grid[self.row][self.col + 1])

        # LEFT
```

```
        if self.col > 0 and not grid[self.row][self.col -
1].is_wall():
            self.neighbors.append(grid[self.row][self.col - 1])

    # Method overloading
    def __lt__(self, other):
        return False

    # Method overloading
    def __str__(self):
        return str([self.row, self.col])
```

Maze.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from Maze.Pen import Pen

# Create node which acts like a graph node ~ Hong Yu
class Block:
    def __init__(self, row, col, size, total_rows, total_cols, canvas,
x=None, y=None):
        # Preset details
        self.row = row
```

```
        self.col = col
        self.x = x if x is not None else row * size
        self.y = y if y is not None else col * size
        self.color = "white"
        self.neighbors = []
        self.size = size
        self.total_rows = total_rows
        self.total_cols = total_cols
        self.canvas = canvas
        self.parent = None
        self.g_score = float('inf')
        self.f_score = float('inf')

    # Get position of block
    def get_pos(self):
        return self.row, self.col

    # Check if is reset
    def is_reset(self):
        return self.color == "white"

    # Check if is open
    def is_open(self):
        return self.color == "yellow"

    # Check if is path
    def is_path(self):
        return self.color == "orange"

    # Check if is wall
    def is_wall(self):
        return self.color == "grey"
```

```
    # Check if is start
    def is_start(self):
        return self.color == "lightgreen"

    # Check if is end
    def is_end(self):
        return self.color == "lightblue"

    # Reset color
    def reset(self):
        self.color = "white"

    # Set start color
    def make_start(self):
        self.color = "lightgreen"

    # Set open color
    def make_open(self):
        self.color = "yellow"

    # Set wall color
    def make_wall(self):
        self.color = "grey"

    # Set end color
    def make_end(self):
        self.color = "lightblue"

    # Set path color
    def make_path(self):
        self.color = "orange"

    # Draw/Color block
```

```
    def draw(self):
        try:
            self.pen.hideturtle()
        except AttributeError:
            self.pen = Pen(canvas=self.canvas, tile_size=self.size)
        self.pen.speed("fastest")
        self.pen.hideturtle()
        self.pen.fillcolor(self.color)
        self.pen.setpos(self.x, self.y)
        self.pen.stamp()
        return

    # Clear drawings
    def clear_stamps(self):
        try:
            self.pen.clearstamps()
        except AttributeError:
            return

    # Update neighbors [Connect them using edges and vertex]
    def update_neighbors(self, grid):
        self.neighbors = []
        # DOWN
        if self.row < self.total_rows - 1 and not grid[self.row +
1][self.col].is_wall():
            self.neighbors.append(grid[self.row + 1][self.col])

        # UP
        if self.row > 0 and not grid[self.row -
1][self.col].is_wall():
            self.neighbors.append(grid[self.row - 1][self.col])

        # RIGHT
```

```
        if self.col < self.total_cols - 1 and not
grid[self.row][self.col + 1].is_wall():
            self.neighbors.append(grid[self.row][self.col + 1])

        # LEFT
        if self.col > 0 and not grid[self.row][self.col -
1].is_wall():
            self.neighbors.append(grid[self.row][self.col - 1])

    # Method overloading
    def __lt__(self, other):
        return False

    # Method overloading
    def __str__(self):
        return str([self.row, self.col])
```

Text.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from turtle import RawTurtle

# Inherit RawTurtle ~ Hong Yu
```

```
class Text(RawTurtle):
    def __init__(self, text, canvas = None, font=None, bold="normal",
fontSize=15,align="center",x=0,y=0):
        # Super parent
        super().__init__(canvas)
        # Setup
        self.canvas = canvas
        self.fontSize = fontSize
        self.font = font
        self.bold = bold
        self.align = align
        self.x = x
        self.y = y
        # Set text to private to prevent text alteration
        self.__text = text
        self.hideturtle()

    # Draw text for visualisation
    def draw(self):
        self.hideturtle()
        self.penup()
        self.speed('fastest')
        self.setpos(self.x,self.y)
        self.write(self.__text,
font=(self.font,self.fontSize,self.bold),align=self.align)

    # Private Setter function
    def __setText(self, text):
        self.__text = text

    # Getter Function
    def getText(self):
        return self.__text
```

```
    # Public setter function to modify text and re render
    def changeText(self, text):
        self.__setText(text)
        self.clear()
        self.draw()
```

Pen.py

```
"""
Member 1:
Name: Soh Hong Yu
Class: DAAA/FT/2B/01
Admin No.: P2100775
Member 2:
Name: Samuel Tay Tze Ming
Class: DAAA/FT/2B/01
Admin No.: P2107404
"""

from turtle import RawTurtle

# Inherit Raw Turtle ~ Hong Yu
class Pen(RawTurtle):
    def __init__(self, tile_size=25, cursor_size = 20, canvas = None):
        super().__init__(canvas)
        # Setup Pen
        self.shape('square')
        self.hideturtle()
        self.tile_size = tile_size
        self.cursor_size = cursor_size
        self.shapesize(tile_size / cursor_size)
        self.pencolor('black')
```

```
        self.penup()
        self.speed('fastest')

    # Due to custom sizing, we need to update the size of the pen
    def update_size(self, tile_size):
        self.tile_size = tile_size
        self.shapesize(self.tile_size / self.cursor_size)
```