Nile University

School of Computer science & Artificial intelligence

MATH211-Discrete Mathematics

Simulated Parking Service with Pathfinding

Team members:

- **Abdallah Ehab Emam - 211001541**
- **Omar Hesham Ghareeb – 211002041**
- **Yassin Saad Elasfar – 211002150**

Submitted to:

Dr. Yasser Abd El-latif

Contents

Abstract:

Nowadays finding a parking place in a busy day in the mall can be very frustrating, so we gathered ourselves as a team to come up with a solution. We have built a simulated program to help you find an empty parking slot that is closest to the gate you wish to enter.

Introduction:

Our program helps you determine the nearest unoccupied parking slot to the gate. By using python, which is an interpreted, object-oriented, high-level programming language, and Pygame which is a python module designed for writing video games. Though we have used it to create a graphical user interface (GUI) because we have used it before multiple times, and we are comfortable with using it. Finally, we have used the python pathfinding module which uses a set of algorithms to find the simplest path from point A to point Z.

Methods:-

- **Basic setup:**

The following code helps us generate a window to use it for displaying our content. We assigned 5 variables. Each variable has a different purpose. The "`clock`" variable creates an object to help track time. The "`FPS`" variable initializes the num of frames that our screen will update each second. The "`screen_width`" variable initializes how wide our window will be. The "`screen_height`" variable initializes how tall our window will be. The "`screen`" variable creates our screen with the specified height and width.

```
# To install pygame, type 'pip install pygame' in the
# windows powershell or the os terminal
# To create a blank screen as a setup for a game, use:
import pygame, sys
pygame.init()
clock = pygame.time.Clock()
FPS = 60
screen_width = 1280
screen_height = 704
screen = pygame.display.set_mode((screen_width, screen_height))
while True:
    clock.tick(FPS) # updates the screen, the amount of times it does so
depends on the FPS
    for event in pygame.event.get(): # Allows you to add various events
        if event.type == pygame.QUIT: # Allows the user to exit using the
X button
            pygame.quit()
            sys.exit()
```

- **Library:**

We have also imported these three libraries from the pathfinding module.

```
from pathfinding.core.diagonal_movement import DiagonalMovement

from pathfinding.core.grid import Grid
from pathfinding.finder.a_star import AStarFinder
```

The `pathfinding.core.diagonal_movement` library helps us include diagonal movement in our path. The `pathfinding.core.grid` converts our matrix into a grid.

The `pathfinding.finder.a_star` is the algorithm that our program uses to create the logic.

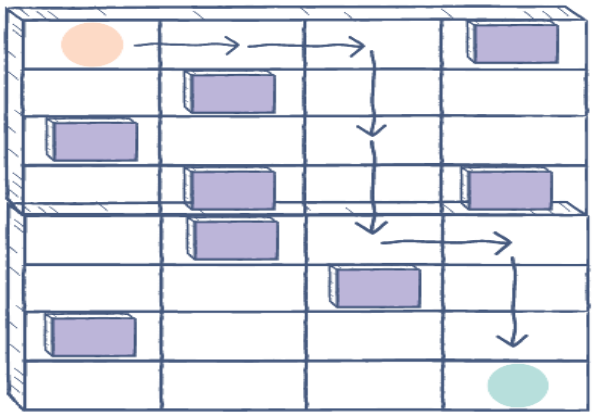
- **Algorithm:**

Our main algorithm is A * algorithm is a type of search algorithm that seeks the shortest path between the initial and final states. It is used in a variety of applications, including maps.

The A* algorithm is used in maps to calculate the shortest distance between the source (starting point) and the destination (final state).

Imagine a square grid which possesses many obstacles, scattered randomly. The initial and the final cell is provided. The aim is to reach the final cell in the shortest amount of time.

Here A* Search Algorithm comes to the rescue:



The A* algorithm has three parameters:

• g: the cost of transferring from the first cell to the current cell. Essentially, it is the sum of all the cells visited since leaving the first cell.

• h: the estimated cost of moving from the current cell to the final cell, also known as the heuristic value. The true cost cannot be determined until the last cell is reached. As a result, h is the estimated cost. We must make certain that the cost is never overestimated.

• f: the product of g and h. So, **F** = **G** + **H**

The algorithm makes decisions by taking the f-value into consideration. The algorithm moves to the cell with the smallest f-value. This process is repeated until the algorithm reaches its target cell.

- **Classes**:

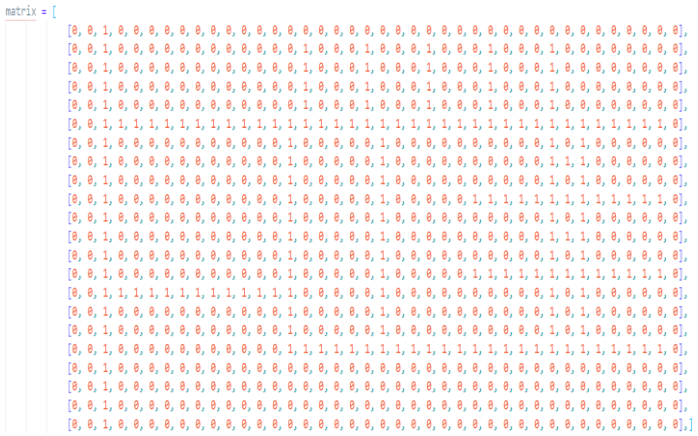
We created two classes ("`class Pathfinder`" and "`class Car`"). The `Pathfinder` which finds the closest path from point A to point Z. The `Car` class which controls the car movement.

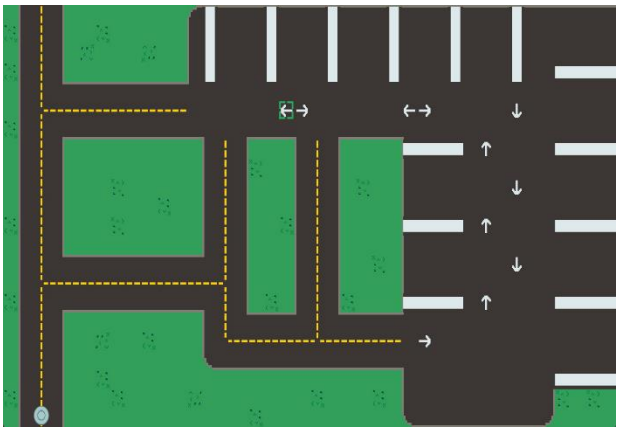
```
class Pathfinder:
    def __init__(self, matrix):
        # setup
        self.matrix = matrix
        self.grid = Grid(matrix=matrix)
        self.select_surf =
pygame.image.load('selection.png').convert_alpha()
        # pathfinding
        self.path = []
        # Car
        self.car = pygame.sprite.GroupSingle(Car(self.empty_path))


class Car(pygame.sprite.Sprite):
    def __init__(self, empty_path):
        # basic
        super().__init__()
        self.image = pygame.image.load('roomba.png').convert_alpha()
        self.rect = self.image.get_rect(center=(80, 684))
```

```
        # movement
        self.pos = self.rect.center
        self.speed = 3.5
        self.direction = pygame.math.Vector2(0, 0)
        # path
        self.path = []
        self.collision_rects = []
        self.empty_path = empty_path
```

- **Map & Matrix:**





Our map is reflected on the matrix. Every drivable place on the map is made out of 1s on the matrix, and the opposite is true for the 0s.

Result:

Lastly our program is very reliable and helps you find the closest parking slot and the shortest amount of time possible.

Future work:

This program is unfinished yet. We still have a lot to work on and improve. Our next objective is to make the program calculate the closest distance from the gate to the nearest empty parking slot, which it does not do currently. We also hope to add a tool that calculates the amount of money you should pay for the parking place.