SOKOBAN GAME USING BFS

A MINI PROJECT REPORT

Submitted by

NANDHINI A 9517202209034

DIVYA MAKI S 9517202209012

MALATHI G 9517202209028

in partial fulfillment for the award of the

degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(AN AUTONOMOUS INSTITUTION)

MAY 2024

MEPCO SCHLENK ENGINEERING COLLEGE SIVAKASI

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this mini project report "**SOKOBAN GAME USING BFS"** is the bonafide work of **NANDHINI A(9517202209034) , DIVYA MAKI S(9517202209012) , MALATHI G(9517202209028)**who carried out the mini project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.A.Shenbagarajan B.E., M.E.,	**Dr.J.Angela Jennifa Sujana, M.Tech.,Ph.D**
Ph.D Assistant Professor	**Professor & Head**
Artificial Intelligence and Data Science	Artificial Intelligence and Data Science
Mepco Schlenk Engineering College	Mepco Schlenk Engineering College
Sivakasi – 626 005	Sivakasi – 626 005
Virudhunagar District.	Virudhunagar District**.**

Submitted for the project viva-voce examination to be held on______________.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

MEPCO SCHLENK ENGINEERING COLLEGE SIVAKASI

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this mini project report "**SOKOBAN GAME USING BFS"** is the bonafide work of **NANDHINI A(9517202209034) , DIVYA MAKI S(9517202209012) , MALATHI G(9517202209028)**who carried out the mini project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.S.Shiny B.E., M.E., Ph.D,	**Dr.J.Angela Jennifa Sujana, M.Tech.,Ph.D**
Assistant Professor	**Professor & Head**
Artificial Intelligence and Data Science	Artificial Intelligence and Data Science
Mepco Schlenk Engineering College	Mepco Schlenk Engineering College
Sivakasi – 626 005	Sivakasi – 626 005
Virudhunagar District.	Virudhunagar District**.**

Submitted for the project viva-voce examination to be held on______________.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we praise and thank **“The Almighty”,** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **, Dr.A.Shenbagarajan B.E., M.E., Ph.D** Assistant Professor**,Dr.S.Shiny B.E., M.E., Ph.D,** Assistant Professor Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Dr.A.Shenbagarajan B.E., M.E., Ph.D** Assistant Professor, **Dr.S.Shiny B.E., M.E., Ph.D,** Assistant Professor Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

ABSTRACT

Sokoban is a classic puzzle game where the player takes on the role of a warehouse keeper. The objective is to push boxes to designated storage locations within a confined space. The game is typically played on a grid where the player can move horizontally or vertically, but cannot pull boxes, only push them. Each level is a puzzle that requires strategic planning and problem-solving skills to complete, as the player must avoid getting boxes stuck in corners or against walls in a way that prevents further movement. The challenge lies in finding the correct sequence of moves to arrange all boxes in their target locations. Sokoban is known for its simplicity in rules but complexity in solving, making it a timeless and engaging game for puzzle enthusiasts.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Sokoban is a classic puzzle game where the player pushes boxes onto designated spots within a confined warehouse grid. Created by Hiroyuki Imabayashi in 1981 and released by Thinking Rabbit in 1982, the game challenges players with strategic planning since the character can only push boxes, not pull them. Each level presents a unique puzzle, progressively increasing in complexity, requiring precise moves to avoid getting stuck. Its simple top-down 2D design belies the depth of its strategic gameplay. Praised for enhancing problem-solving and spatial reasoning skills, Sokoban has maintained its appeal across decades and platforms, from early computers to modern smartphones, continuing to engage puzzle enthusiasts worldwide.

1.2 TECHNOLOGY

1.2.1 TKinter

Tkinter is a standard Python library for creating graphical user interfaces (GUIs). The name "Tkinter" comes from "Tk interface," as it is a Python binding to the Tk GUI toolkit. Tkinter provides a set of tools and widgets for building desktop applications with graphical interfaces.

Key features of Tkinter include:

- **Cross-Platform:** Tkinter is included with most Python installations, making it cross-platform and allowing your Tkinter-based applications to run on Windows, macOS, and Linux without modification.

- **Widgets:** Tkinter provides a variety of standard GUI elements, or widgets, such as buttons, labels, entry fields, text boxes, and more. These widgets can be arranged and customized to create complex GUIs.

- **Event-Driven Programming:** Tkinter follows an event-driven programming paradigm. You define functions (event handlers) that get executed in response to specific events, such as button clicks or mouse movements.

- **Geometry Management:** Tkinter includes several geometry managers, such as pack(), grid(), and place(), which help in organizing and arranging widgets within the GUI window.

- **Canvas Widget:** Tkinter includes a canvas widget that allows you to draw shapes, images, and other graphical elements.

- **Theming:** Tkinter allows you to customize the appearance of your GUI using different themes and styles.

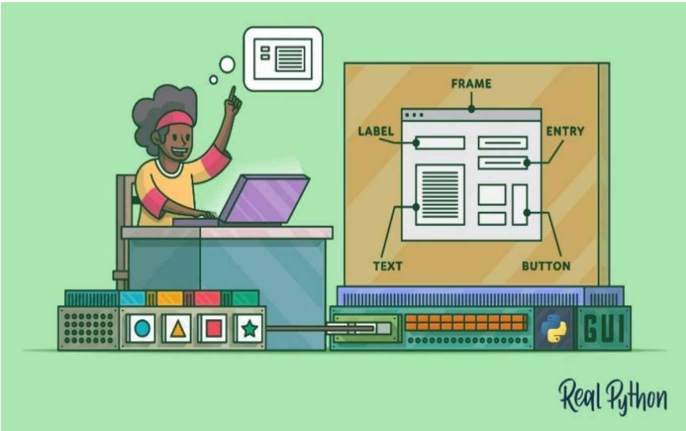

Fig 1.1 Tkinter

CHAPTER 2

PROPOSED

SYSTEM

2.1 DIAGRAM:

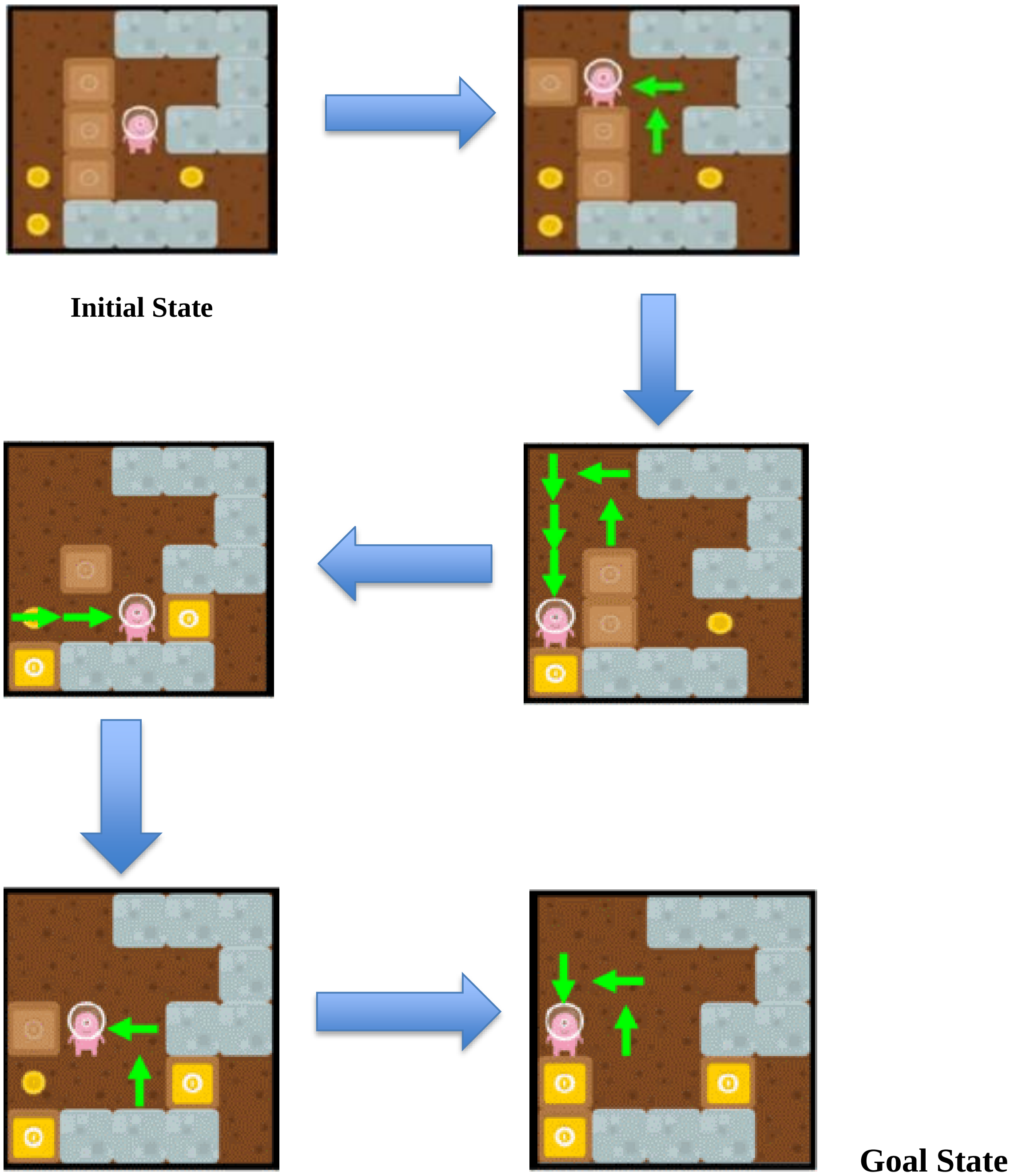

Fig 2.1 Diagram

2.2 PROPOSED SOLUTION

Imported Packages:

1) **Tkinter:** This package is a standard GUI (Graphical User Interface) toolkit for Python and is used for creating the graphical interface.

```
pip install tk
```

Fig 2.2. Installing Tkinter

Libraries Used:

1) **Tkinter Library:**

tkinter: This is the main module of the tkinter library, which provides classes and functions to create and manage GUI elements like windows, buttons, labels, and canvases.

- **Usage**: Creating the main game window, level selection window, buttons, labels, and handling user interactions.
- **Example**: root = tk.Tk(), canvas = tk.Canvas(master, width=...).

```
import tkinter as tk
```

Fig 2.3 Importing Tkinter

tkinter.messagebox: This module is part of the tkinter library and provides predefined functions to create message boxes.

- **Usage**: Displaying alerts or messages to the user.
- **Example**: messagebox.showinfo("Congratulations!", "You Win!").

```
import tkinter.messagebox as messagebox
```

Fig 2.4 Importing Tkinter.messagebox

2)Standard PythonLibraries:

time: This library provides time-related functions, such as delaying execution or getting the current time.

- **Usage**: Adding delays between player moves to simulate movement.
- **Example**: time.sleep(0.5).

```
import time
```

Fig 2.5 Importing time

copy: This library provides functions to create shallow or deep copies of objects.

- **Usage**: Creating deep copies of the game state to explore different moves without modifying the original state.
- **Example**: curPositionCopy = copy.deepcopy(curPosition).

```
import copy
```

Fig 2.6 Importing copy

collections: This library provides specialized container datatypes.

- **Usage**: Using deque to implement a queue for the breadth-first search algorithm to solve the game.
- **Example**: queue = collections.deque([]).

```
import collections
```

Fig 2.7 Importing collections

2.3 Features:

- **Tkinter GUI**: The code uses Tkinter, a standard Python GUI library, to create a graphical user interface for the Sokoban game. Tkinter provides widgets like Canvas, Button, and Label for creating the game interface.
- **Grid Representation**: The Sokoban game is represented using a grid of characters, where each character represents a specific element of the game (e.g., walls, player, boxes, goals).
- **Levels**: The game has multiple levels, each represented by a different grid configuration. The user can choose a level to play from a selection window.

- **Player Movement**: The player can move in four directions: up, down, left, and right. The movement is restricted by walls, and the player can push boxes around the grid.
- **Box Pushing**: When the player moves towards a box, if there is empty space or a goal behind the box, the box can be pushed in that direction.
- **Goal States**: Certain grid cells represent goal states. The objective of the game is to move all boxes onto goal states.
- **Win Condition**: The game checks for a win condition after each player move. If all boxes are placed on goal states, the player wins the game.
- **Reset Game**: There's an option to reset the game to its initial state if the player wants to start over.
- **Solution Solver**: There's a solver function (solve_sokoban) implemented using a breadth-first search algorithm. This function finds a solution path for a given Sokoban level grid. The solver considers the positions of boxes and the player and navigates through possible moves until it finds a solution or determines that there's no solution.
- **Auto-Solve**: The player can click a "Start" button to automatically solve the level using the solver function and then play the solved path step by step.
- **File Paths for Images**: Image files (wall, player, box, goal) are loaded from specific file paths on the system.
- **Object-Oriented Approach**: The code utilizes a class (`SokobanGUI`) to encapsulate the game logic and GUI elements. This makes the code modular and easier to manage.
- **Global Variables**: The current level of the game is stored in a global variable (`current_level`). This variable is updated when the user selects a level to play.

CHAPTER 3

PROJECT REQUIREMENT

3.1 HARDWARE REQUIREMENTS:

- Operation System WINDOWS 7 AND ABOVE
- Any Processor Corei3 and above
- RAM of 512 MB+
- Monitor of 14.1 or 15 -17 inch
- Keyboard and Mouse.

3.2 SOFTWARE REQUIREMENTS:

- Windows OS
- Python version 3.11.5
- Tkinter
- Image Files:
    - ‘wall.png’
    - ‘playerD.png’
    - ‘box.png’
    - ‘target.png’
    - ‘valid_box.png’

CHAPTER 4

IMPLEMENTATION

4.1 PROGRAM CODE

```
import tkinter as tk
import tkinter.messagebox as messagebox
import time
import copy
import collections

# Grid configurations for levels
grid_level1 = [
    list("OOOOOOOO"),
    list("O   OP O"),
    list("O    B O"),
    list("O  O   O"),
    list("OOOOOBGO"),
    list("O G    O"),
    list("OOOOOOOO")
]

grid_level2 = [
    list("OOOOO"),
    list("OGPOO"),
    list("OBBGO"),
    list("O B O"),
    list("O G O"),
    list("OO  O"),
    list(" OOOO")
]

grid_level3=[
```

```
        list("OOOOOO"),
        list("OOOPGO"),
        list("OG B O"),
        list("OO B O"),
        list("OGB OO"),
        list(" OOOOO")
    ]

    class SokobanGUI:
        def __init__(self, master, grid):
            self.master = master
            self.grid = grid
            self.initial_grid = [row[:] for row in grid]  # Make a deep copy of the initial grid
            self.player_position = self.find_player_position()
            self.box_positions = self.find_box_positions()
            self.goal_positions = self.find_goal_positions()

            self.canvas = tk.Canvas(master, width=len(grid[0]) * 40, height=len(grid) * 40, bg='seashell')
            self.canvas.pack(pady=20)

            self.load_images()
            self.draw_grid()

        def load_images(self):
            self.wall_image = tk.PhotoImage(file="C:\\Users\\MALATHI GANESAN\\Desktop\\wall.png")
            self.player_image = tk.PhotoImage(file="C:\\Users\\MALATHI
GANESAN\\Desktop\\playerD.png")
            self.box_image = tk.PhotoImage(file="C:\\Users\\MALATHI GANESAN\\Desktop\\box.png")
            self.goal_image = tk.PhotoImage(file="C:\\Users\\MALATHI GANESAN\\Desktop\\target.png")
            self.box_on_goal_image = tk.PhotoImage(file="C:\\Users\\MALATHI
GANESAN\\Desktop\\valid_box.png")

        def reset_game(self):
            self.grid = [row[:] for row in self.initial_grid]  # Reset grid to its initial state
```

```
        self.player_position = self.find_player_position()
        self.box_positions = self.find_box_positions()
        self.goal_positions = self.find_goal_positions()
        self.draw_grid()

    def find_player_position(self):
        for y in range(len(self.grid)):
            for x in range(len(self.grid[y])):
                if self.grid[y][x] == 'P':
                    return x, y
        return None

    def find_box_positions(self):
        boxes = []
        for y in range(len(self.grid)):
            for x in range(len(self.grid[y])):
                if self.grid[y][x] == 'B':
                    boxes.append((x, y))
        return boxes

    def find_goal_positions(self):
        goals = []
        for y in range(len(self.grid)):
            for x in range(len(self.grid[y])):
                if self.grid[y][x] == 'G':
                    goals.append((x, y))
        return goals

    def draw_grid(self):
        self.canvas.delete("all")
        for y, row in enumerate(self.grid):
            for x, cell in enumerate(row):
                x0, y0 = x * 40, y * 40
                if cell == 'O':
```

```
                self.canvas.create_image(x0, y0, anchor='nw', image=self.wall_image)
            elif cell == 'P':
                self.canvas.create_image(x0, y0, anchor='nw', image=self.player_image)
            elif cell == 'B':
                self.canvas.create_image(x0, y0, anchor='nw', image=self.box_image)
            elif cell == 'G':
                self.canvas.create_image(x0, y0, anchor='nw', image=self.goal_image)
            elif cell == 'GB':
                self.canvas.create_image(x0, y0, anchor='nw', image=self.box_on_goal_image)

    # Draw boxes over goal states separately
    for x, y in self.box_positions:
        if self.grid[y][x] == 'GB':
            x0, y0 = x * 40, y * 40
            self.canvas.create_image(x0, y0, anchor='nw', image=self.box_on_goal_image)

def move_player(self, direction):
    x, y = self.player_position
    dx, dy = direction
    new_x, new_y = x + dx, y + dy

    if self.grid[new_y][new_x] == 'O':
        return False  # Cannot move into walls

    elif self.grid[new_y][new_x] == 'B' or self.grid[new_y][new_x] == 'GB':
        # Check if the box can be pushed
        new_box_x, new_box_y = new_x + dx, new_y + dy
        if self.grid[new_box_y][new_box_x] in ['O', 'B', 'GB']:
            return False  # Cannot push the box into walls or other boxes
        # Move the box
        if self.grid[new_box_y][new_box_x] == 'G':
            self.grid[new_box_y][new_box_x] = 'GB'  # Box placed on goal state
        else:
            self.grid[new_box_y][new_box_x] = 'B'
```

```
            if self.grid[new_y][new_x] == 'GB':
                self.grid[new_y][new_x] = 'G'  # Restore goal state
            else:
                self.grid[new_y][new_x] = ' '
            self.box_positions.remove((new_x, new_y))
            self.box_positions.append((new_box_x, new_box_y))

        elif self.grid[new_y][new_x] == 'G':
            # Moving onto a goal state
            pass  # No additional logic needed, just proceed to move player

        elif self.grid[new_y][new_x] == ' ':
            # Moving to an empty space
            pass  # No additional logic needed, just proceed to move player

        # Move the player
        if self.grid[y][x] == 'P':
            if (x, y) in self.goal_positions:
                self.grid[y][x] = 'G'  # Restore goal state
            else:
                self.grid[y][x] = ' '
        elif self.grid[y][x] == 'GP':
            self.grid[y][x] = 'G'  # Restore goal state

        self.grid[new_y][new_x] = 'P'
        self.player_position = (new_x, new_y)

        self.draw_grid()
        self.check_win()
        return True

    def check_win(self):
        for box_x, box_y in self.box_positions:
```

```
            if (box_x, box_y) not in self.goal_positions:
                return
        self.display_win_message()

    def display_win_message(self):
        messagebox.showinfo("Congratulations!", "You Win!")
        self.master.destroy()
        choose_level()

    def follow_path(self, path):
        for move in path:
            direction = {
                'U': (0, -1),
                'D': (0, 1),
                'L': (-1, 0),
                'R': (1, 0)
            }[move]
            self.move_player(direction)
            self.master.update()
            time.sleep(0.5)  # Adjust speed of automatic movement

def solve_sokoban(grid):
    maxRowLength = len(grid[0])
    lines = len(grid)

    boxRobot = []
    wallsStorageSpaces = []
    possibleMoves = {'U': [-1, 0], 'R': [0, 1], 'D': [1, 0], 'L': [0, -1]}

    for i in range(lines):
        boxRobot.append(['-'] * maxRowLength)
        wallsStorageSpaces.append(['-'] * maxRowLength)

    for i in range(len(grid)):
```

```
    for j in range(maxRowLength):
        if grid[i][j] == 'B' or grid[i][j] == 'P':
            boxRobot[i][j] = grid[i][j]
            wallsStorageSpaces[i][j] = ' '
        elif grid[i][j] == 'G' or grid[i][j] == 'O':
            wallsStorageSpaces[i][j] = grid[i][j]
            boxRobot[i][j] = ' '
        elif grid[i][j] == ' ':
            boxRobot[i][j] = ' '
            wallsStorageSpaces[i][j] = ' '
        elif grid[i][j] == '*':
            boxRobot[i][j] = 'B'
            wallsStorageSpaces[i][j] = 'G'
        elif grid[i][j] == '.':
            boxRobot[i][j] = 'P'
            wallsStorageSpaces[i][j] = 'G'

movesList = []
visitedMoves = []

queue = collections.deque([])
source = [boxRobot, movesList]
if boxRobot not in visitedMoves:
    visitedMoves.append(boxRobot)
queue.append(source)
robot_x = -1
robot_y = -1
completed = 0
solution_path = []

while len(queue) != 0 and completed == 0:
    temp = queue.popleft()
    curPosition = temp[0]
    movesTillNow = temp[1]
```

```
    for i in range(lines):
        for j in range(maxRowLength):
            if curPosition[i][j] == 'P':
                robot_y = j
                robot_x = i
                break
        else:
            continue
        break

    for key in possibleMoves:
        robotNew_x = robot_x + possibleMoves[key][0]
        robotNew_y = robot_y + possibleMoves[key][1]
        curPositionCopy = copy.deepcopy(curPosition)
        movesTillNowCopy = copy.deepcopy(movesTillNow)
        if curPositionCopy[robotNew_x][robotNew_y] == 'B':
            boxNew_x = robotNew_x + possibleMoves[key][0]
            boxNew_y = robotNew_y + possibleMoves[key][1]
            if curPositionCopy[boxNew_x][boxNew_y] == 'B' or
wallsStorageSpaces[boxNew_x][boxNew_y] == 'O':
                continue
            else:
                curPositionCopy[boxNew_x][boxNew_y] = 'B'
                curPositionCopy[robotNew_x][robotNew_y] = 'P'
                curPositionCopy[robot_x][robot_y] = ' '
                if curPositionCopy not in visitedMoves:
                    matches = 0
                    for k in range(lines):
                        for l in range(maxRowLength):
                            if wallsStorageSpaces[k][l] == 'G':
                                if curPositionCopy[k][l] != 'B':
                                    matches = 1
                    movesTillNowCopy.append(key)
                    if matches == 0:
```

```
                        completed = 1
                        solution_path = movesTillNowCopy
                        break
                    else:
                        queue.appendleft([curPositionCopy, movesTillNowCopy])
                        visitedMoves.append(curPositionCopy)
            else:
                if wallsStorageSpaces[robotNew_x][robotNew_y] == 'O' or
curPositionCopy[robotNew_x][robotNew_y] != ' ':
                    continue
                else:
                    curPositionCopy[robotNew_x][robotNew_y] = 'P'
                    curPositionCopy[robot_x][robot_y] = ' '

                    if curPositionCopy not in visitedMoves:
                        movesTillNowCopy.append(key)
                        queue.appendleft([curPositionCopy, movesTillNowCopy])
                        visitedMoves.append(curPositionCopy)

        if completed == 0:
            print("Can't make it")
        return solution_path

    def choose_level():
        selection_window = tk.Tk()
        selection_window.title("Choose Level")

        label = tk.Label(selection_window, text="Choose a level to start", font=("Helvetica", 16, "bold"),
bg='lightblue')
        label.pack(pady=20)

        button_level1 = tk.Button(selection_window, text="Level 1", command=lambda:
[selection_window.destroy(), start_level(1)],
                                  font=("Helvetica", 14), bg='lightgreen', activebackground='darkgreen')
```

```
        button_level1.pack(pady=10)

        button_level2 = tk.Button(selection_window, text="Level 2", command=lambda:
[selection_window.destroy(), start_level(2)],
                                  font=("Helvetica", 14), bg='lightgreen', activebackground='darkgreen')
        button_level2.pack(pady=10)

        button_level3 = tk.Button(selection_window, text="Level 3", command=lambda:
[selection_window.destroy(), start_level(3)],
                                  font=("Helvetica", 14), bg='lightgreen', activebackground='darkgreen')
        button_level3.pack(pady=10)

        button_close = tk.Button(selection_window, text="Close", command=selection_window.destroy,
                                 font=("Helvetica", 14), bg='lightcoral', activebackground='darkred')
        button_close.pack(pady=10)

        selection_window.configure(bg='lightblue')
        selection_window.mainloop()

    def start_level(level):
        global current_level  # Declare current_level as global so it can be modified inside the function
        current_level = level  # Set the current level to the selected level

        def solve_and_play():
            nonlocal sokoban_gui, grid
            sokoban_gui.reset_game()  # Reset the game state before starting a new level
            solution_path = solve_sokoban(grid)
            if solution_path:
                print("Solution path:", ''.join(solution_path))
                sokoban_gui.follow_path(solution_path)
            else:
                print("No solution found.")

        root = tk.Tk()
```

```
    root.title(f"Sokoban - Level {level}")

    if level == 1:
        grid = grid_level1
    elif level == 2:
        grid = grid_level2
    elif level == 3:
        grid = grid_level3

    sokoban_gui = SokobanGUI(root, grid)

    start_button = tk.Button(root, text="Start", command=solve_and_play, font=("Helvetica", 14), bg='seashell', activebackground='sienna')
    start_button.pack(pady=20)

    root.configure(bg='sienna')
    root.mainloop()

current_level = 1  # Initialize the current level to 1

choose_level()
```

4.2 OUTPUT

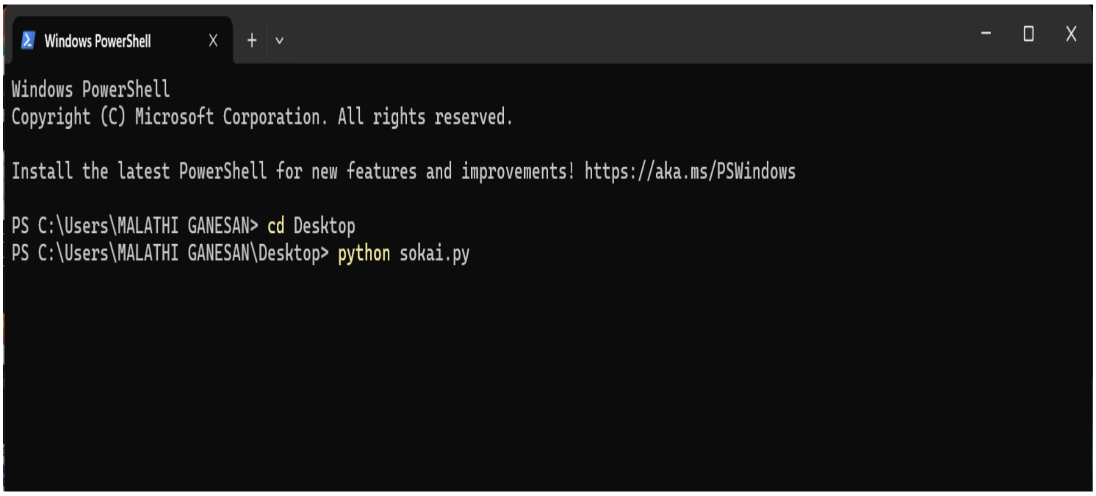

Fig 4.2.1. Compilation

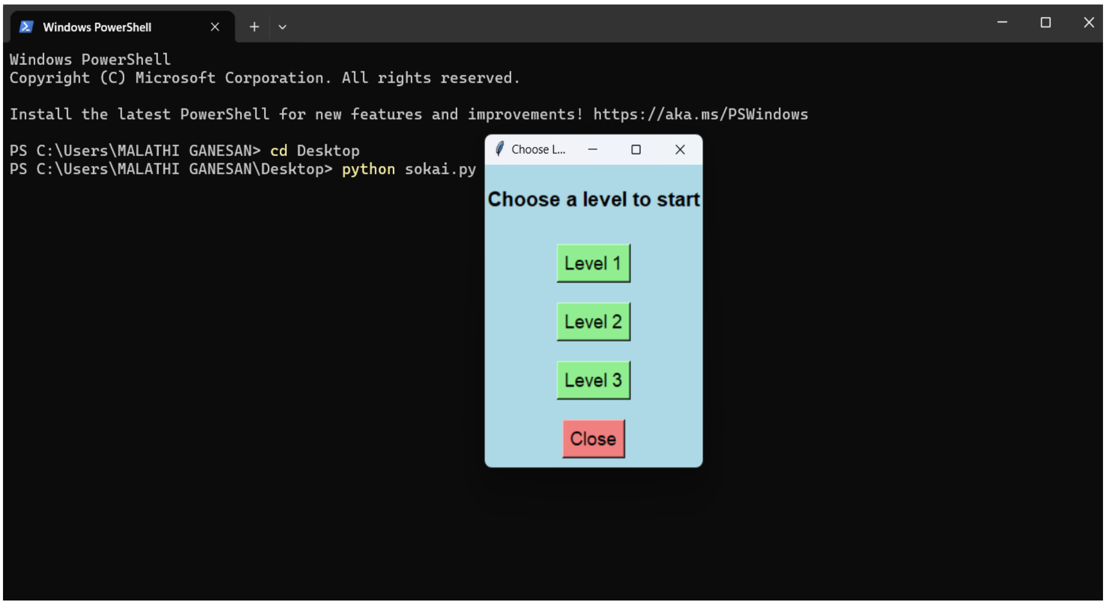

Fig.4.2.2 Level Selection Page

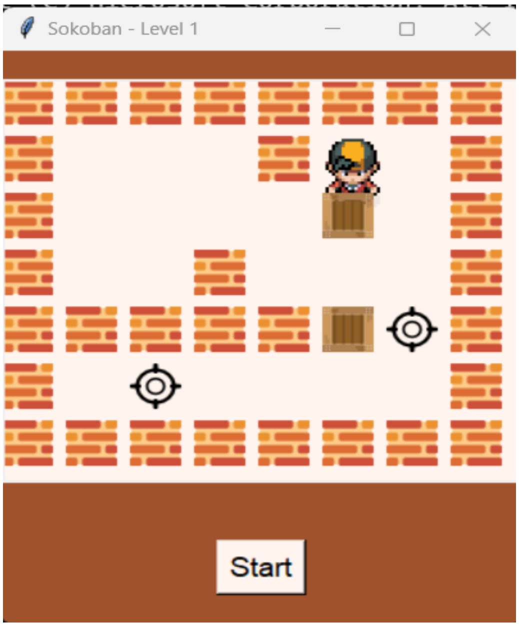

Fig.4.2.3. Level1

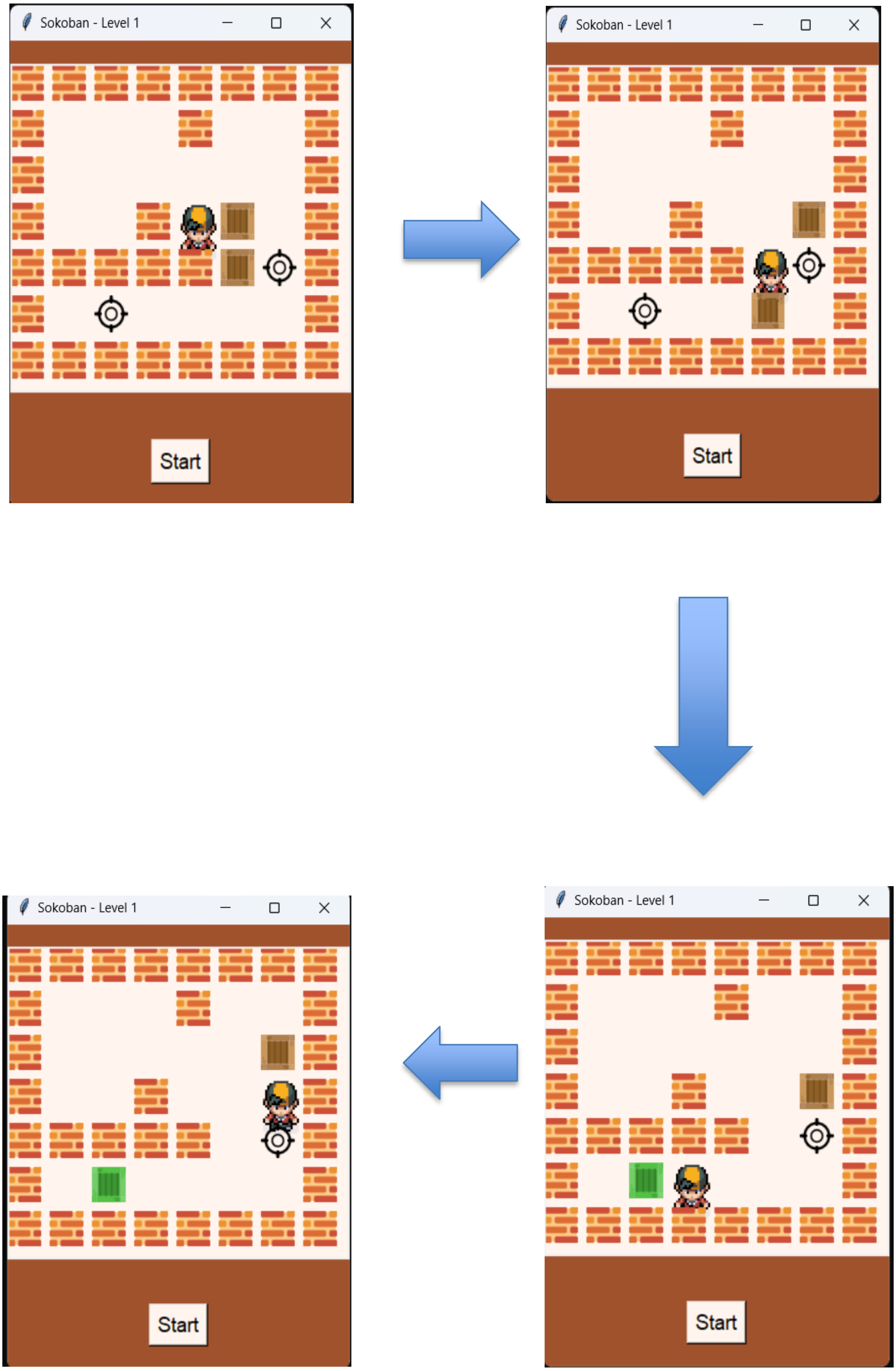

Fig 4.2.4.Level 1 -Moves

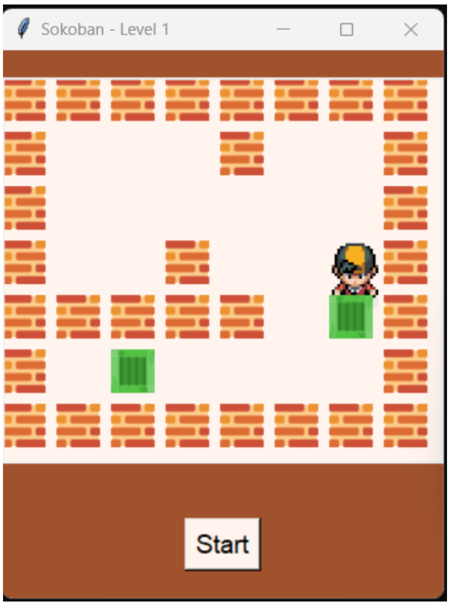

Fig 4.2.5.Level 1 Finished

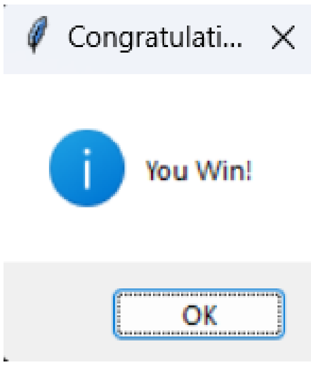

Fig 4.2.6.Winning Message for Level 1

Fig.4.2.7.Solution Path for Level 1

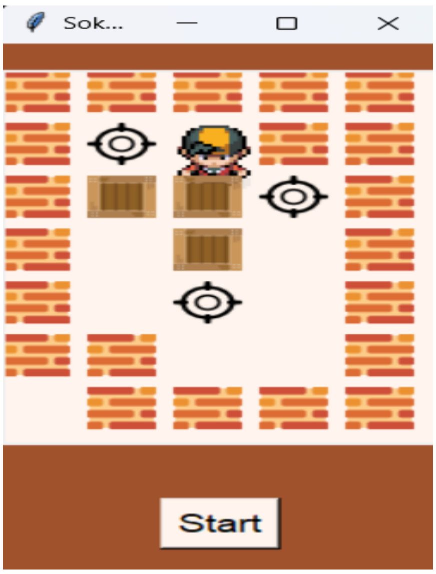

Fig.4.2.8.Level2

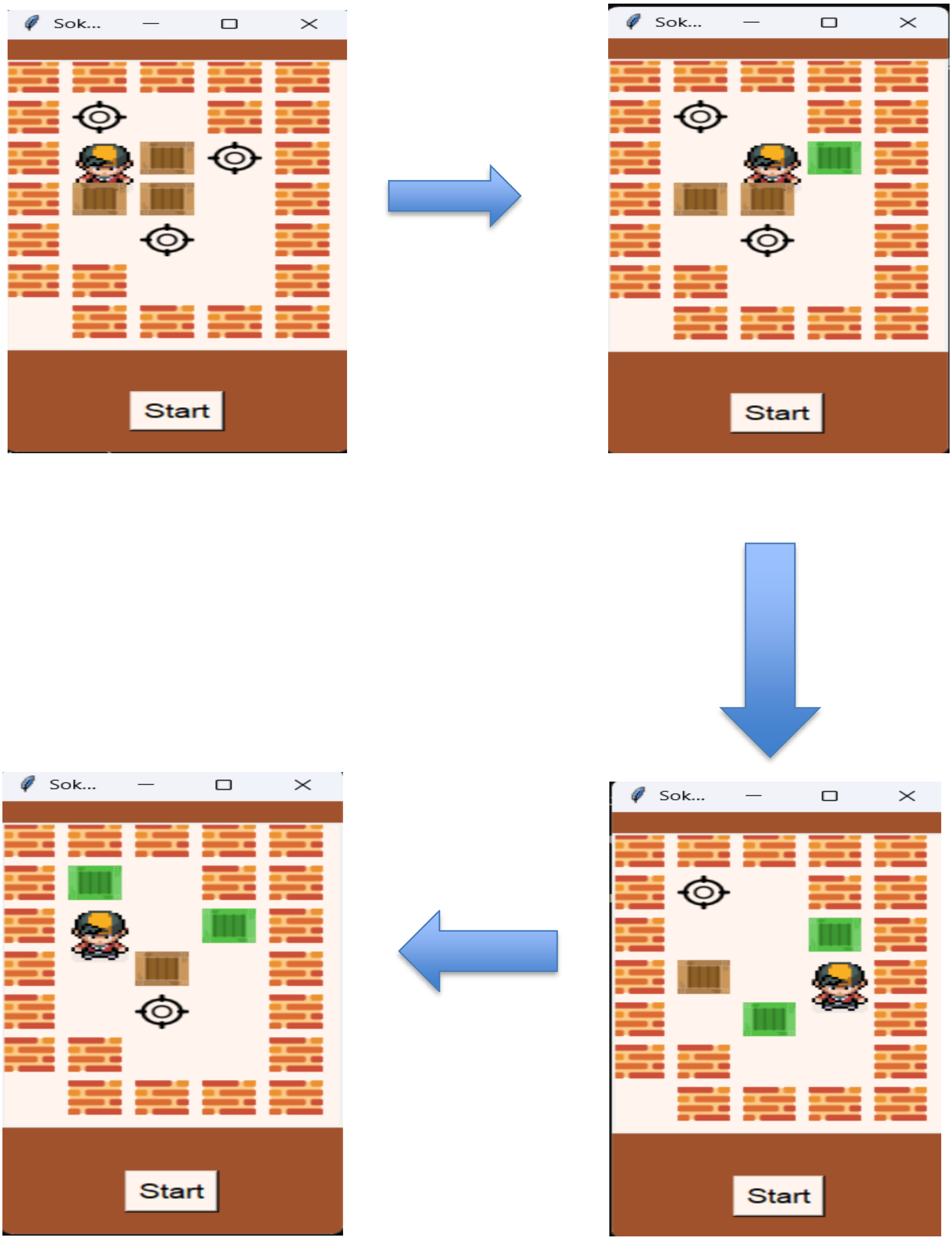

Fig.4.2.9.Level 2 -Moves

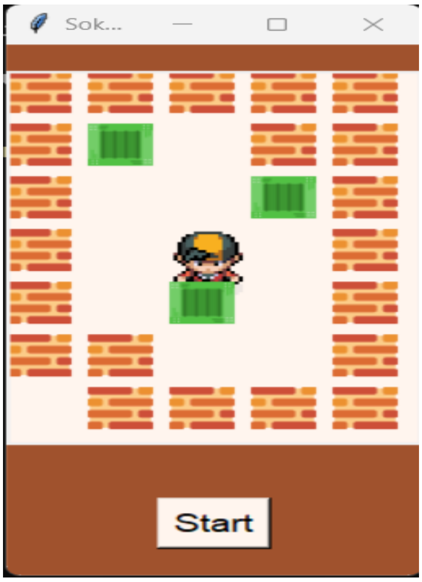

Fig.4.2.10.Level 2 Finished

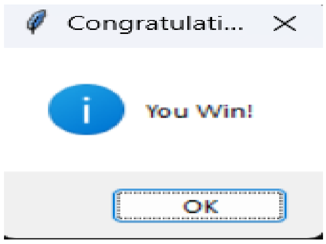

Fig.4.2.11.Winning Message for Level 2

Fig.4.2.12.Solution Path for Level 2

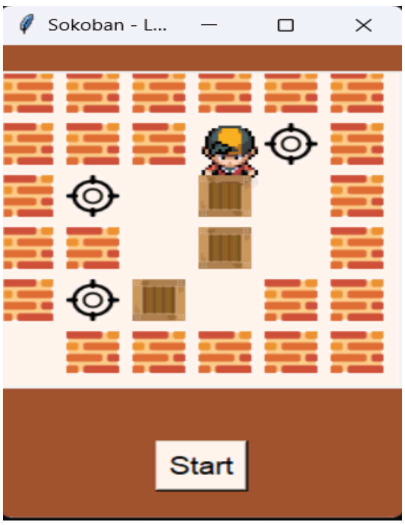

Fig4.2.13.Level 3

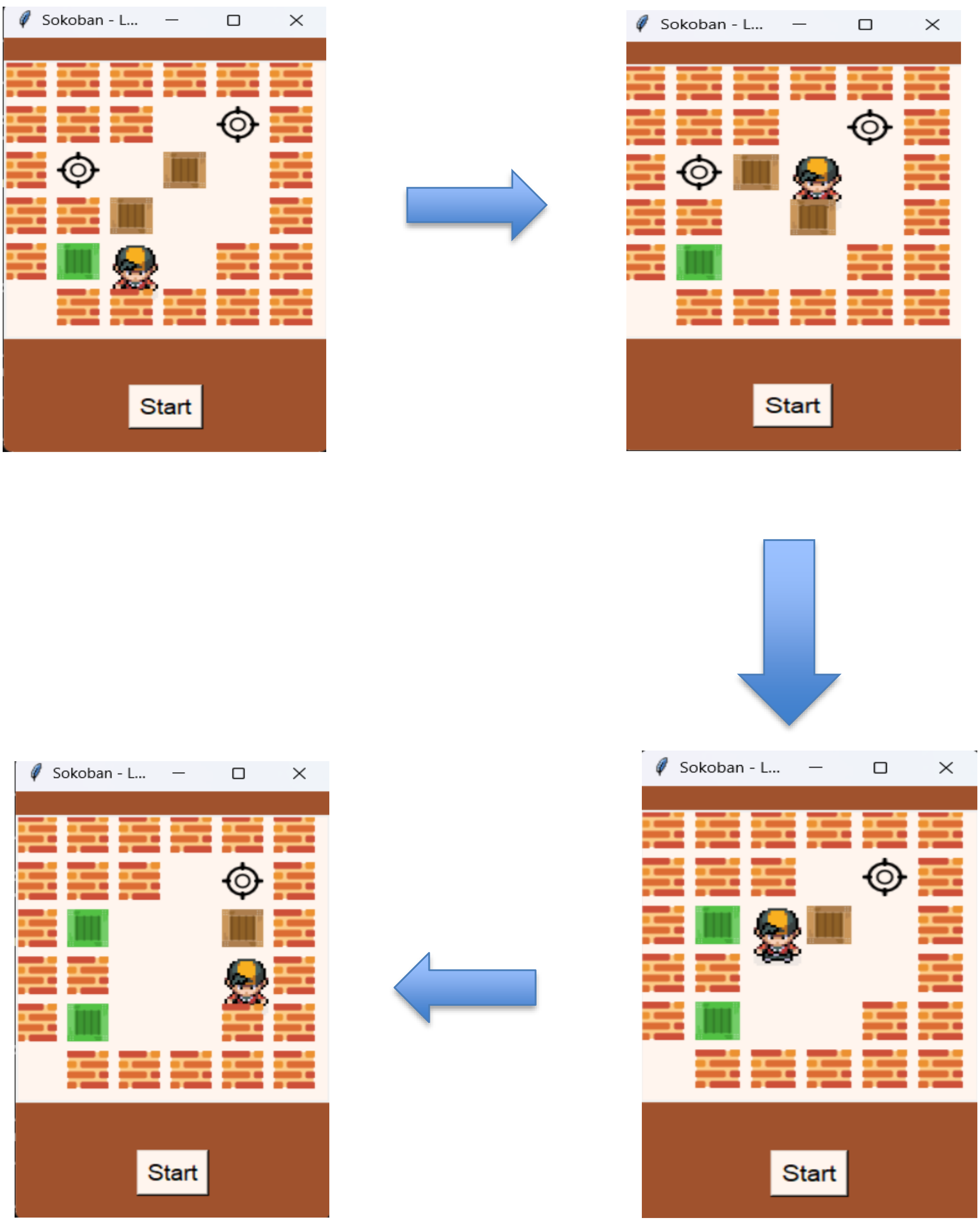

Fig.4.2.14.Level 3 -Moves

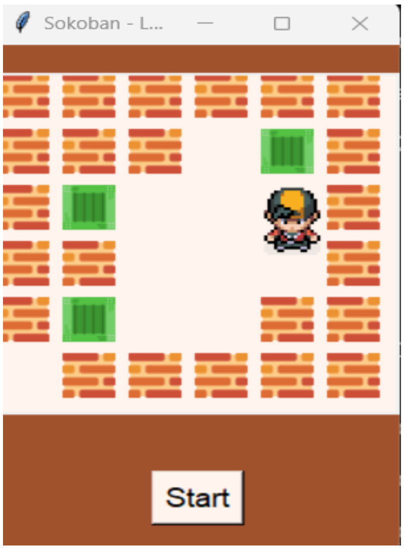

Fig.4.2.15.Level3 Finished

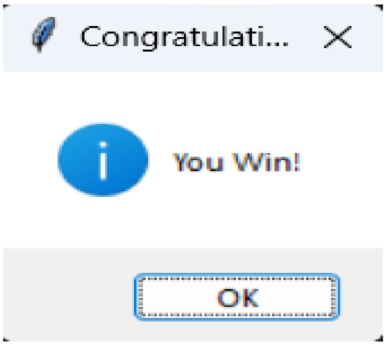

Fig.4.2.16.Winning Message for Level 3

Fig.4.2.17.Solution Path for Level 3

CHAPTER 6

CONCLUSION

The Sokoban solver game presents an engaging fusion of traditional puzzle mechanics and modern computational prowess, culminating in a compelling gaming experience. With its intuitive Tkinter interface, players are immersed in a world of strategic box pushing and goal reaching across multiple meticulously designed levels. The game's elegance lies in its simplicity, yet it offers a depth of challenge that keeps players captivated and eager to conquer each puzzle. Moreover, the inclusion of a solver function adds a layer of versatility, providing assistance to players when faced with particularly daunting levels. As players navigate through the game's intricacies, they not only hone their problem-solving skills but also indulge in moments of triumph as they overcome each obstacle. Whether seeking a cerebral challenge or a leisurely diversion, the Sokoban solver game stands as a testament to the enduring appeal of puzzle-solving gameplay, offering an enriching experience that resonates with players of all backgrounds and skill levels.

REFERENCES

- **Wikipedia - Sokoban**: Provides a comprehensive overview of the Sokoban game, its rules, and its history.

  https://en.wikipedia.org/wiki/Sokoban

- **Python Tkinter Documentation**: Official documentation for Tkinter, useful for understanding how to create GUI applications in Python.

  https://docs.python.org/3/library/tkinter.html

- **Tkinter PhotoImage Documentation**: Details on how to work with images in Tkinter, which is essential for rendering the game elements.

  https://tkdocs.com/shipman/photoimage.html

- **Breadth-First Search Algorithm**: Explanation of the BFS algorithm, which is used in the Sokoban solver to find the optimal path.

  https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

- **Sokoban Solver in Python**: A tutorial that walks through creating a Sokoban solver in Python.

  https://blog.rachum.com/posts/sokoban-solver/

- **Python Sokoban Game Example**: A detailed example of implementing a Sokoban game in Python, which can provide additional insights and code snippets.

  https://inventwithpython.com/blog/2011/10/17/sokoban-invent-your-own-computer-games-with-python/