Contents

Chapter 1

Introduction

Welcome to the Logical Puzzle Game, a unique interactive experience that combines logic, storytelling, and problem-solving. In this game, players are presented with a story crafted from a set of logical statements and a goal. The challenge is to determine whether the goal, as explained through the narrative, can be achieved based on the given logic.

The objective of this project is to enhance our skills in Python and Prover9 by creating a game that integrates logical reasoning with storytelling. Through this hands-on experience, we aim to deepen our understanding of automated theorem proving and improve our ability to generate and solve complex logical problems.

Chapter 2

System Architecture

2.1 Overview

The Logical Puzzle Game is built using Python, leveraging the Tkinter library for the graphical user interface and Prover9 for automated theorem proving. The game generates logical propositions and evaluates whether a given goal can be derived from these propositions.

2.2 Components

- **Prover9:** An automated theorem prover used to verify logical statements. It checks if the logical goal can be derived from the provided statements.
- **Python:** The main programming language used for developing the game's logic and interface. Python's versatility allows seamless integration of various libraries.
- **Tkinter:** A Python library used for creating the graphical user interface. It provides the framework for user interaction.
- **Google Generative AI:** Used to generate stories based on logical statements. This adds a narrative layer to the logical puzzles.

Chapter 3

Implementation Details

3.1 Entry Point: `entry.py`

3.1.1 Main Logic and GUI Setup

The main application logic is contained in `entry.py`. This script initializes the GUI using Tkinter, sets up event handlers for user interactions, and manages the flow of the game.

Listing 3.1: Main Logic and GUI Setup

```
import tkinter as tk
from tkinter import messagebox
import subprocess
import random
from gen import generatePropositionalLogic

def run_prover9():
    """Executes Prover9 to evaluate the logical
        statements against the goal."""
    command = "./prover9 -f ./myProver9.in > ./myProver9.out"
    subprocess.run(command, shell=True, check=True)

def generate_story():
    """Generates a story based on user input and logical statements."""
    genre_text = genre_input.get()
    statements, goal = generate_logic_statements()
    prompt = f"Create a story in {genre_text} genre with: {statements}.
---------------Goal: {goal}"
    response = model.generate_content(prompt)
    result_text.delete(1.0, "end")
    result_text.insert("end", response.text)

root = tk.Tk()
root.title("AI Story Generator")

genre_label = tk.Label(root, text="Enter Genre:")
genre_label.pack()
genre_input = tk.Entry(root, width=150)
genre_input.pack()
```

```
generate_button = tk.Button(root, text="Generate Story",
                            command=generate_story)
generate_button.pack()

result_label = tk.Label(root, text="Generated Story:")
result_label.pack()
result_text = tk.Text(root, height=25, width=150)
result_text.pack()

root.mainloop()
```

3.1.2 Logic Handling

The logic for determining whether the goal is achievable is implemented via Prover9. The system writes logical propositions to a file, runs Prover9, and checks the output.

Listing 3.2: Prover9 Logic Handling

```
def possible_button_click():
    """Handles the logic for when the 'Possible' button is clicked."""
    file_path = "./myProver9.in"
    statements = clean_statements(generated_statements)
    content = f"""
formulas(assumptions).
{statements}
end_of_list.

formulas(goals).
{generated_goal}.
end_of_list.
    """
    with open(file_path, "w") as file:
        file.write(content)
    run_prover9()
    output_file_path = "./myProver9.out"
    if os.path.exists(output_file_path):
        with open(output_file_path, "r") as output_file:
            content = output_file.read()
            if "THEOREM PROVED" in content:
                messagebox.showinfo("Result", "You're right!")
            else:
                messagebox.showinfo("Result", "You're wrong!")
```

3.2 Logic Generation: `gen.py`

The logic generation is handled by `gen.py`. This script is responsible for creating propositional logic statements and determining a logical goal that can be either achievable or not.

Listing 3.3: Logic Generation

```
from sympy import symbols, And, Or, Not, Implies, satisfiable
```

```
P, Q, R, S, T = symbols('P Q R S T')

def generate_random_statement():
    """Generates a random propositional logic statement."""
    var1 = random.choice([P, Q, R, S, T])
    var2 = random.choice([P, Q, R, S, T])
    expr1 = Not(var1) if random.choice([True, False]) else var1
    expr2 = Not(var2) if random.choice([True, False]) else var2
    return random.choice([And, Or, Implies])(expr1, expr2)

def generate_compatible_statements(num_statements=5):
    """Generates a list of logically compatible statements."""
    statements = []
    while len(statements) < num_statements:
        new_statement = generate_random_statement()
        if satisfiable(And(*(statements + [new_statement]))):
            statements.append(new_statement)
    return statements

def generate_goal(statements, achievable=True):
    """Generates a goal that is either achievable
        or not based on the statements."""
    while True:
        potential_goal = generate_random_statement()
        if achievable:
            if simplify(Implies(And(*statements), potential_goal)):
                return potential_goal
        else:
            if not simplify(Implies(And(*statements), potential_goal)):
                return potential_goal
```

3.3 Prover9 Integration

Prover9 is used to verify whether the generated goal is a logical consequence of the given statements. This integration is crucial for the game's logic validation.

Listing 3.4: Prover9 Integration

```
def write_prover9_input(statements, goal, filename="./myProver9.in"):
    """Writes the statements and goal to a Prover9 input file."""
    with open(filename, "w") as f:
        f.write("formulas(assumptions).\n")
        for statement in statements:
            f.write(f"    {convert_to_prover9(statement)}.\n")
        f.write("end_of_list.\n\n")
        f.write("formulas(goals).\n")
        f.write(f"    {convert_to_prover9(goal)}.\n")
        f.write("end_of_list.\n")

def run_prover9(input_file="./myProver9.in"):
```

```
    """Runs Prover9 with the given input file and
    checks if the goal is achievable."""
    prover9_path = "./prover9"  # Adjust this path as needed
    result = subprocess.run([prover9_path, "-f", input_file],
                capture_output=True, text=True)
    output = result.stdout
    return "THEOREM PROVED" in output
```

Chapter 4

Usage Instructions

4.1 Running the Game

To run the game, execute `entry.py` using Python. Ensure that Prover9 is installed and accessible on your system. This setup is necessary for the logical evaluation of the game.

4.2 User Interface

- **Genre Input:** Enter the genre of the story you wish to generate. This input customizes the narrative context.
- **Generate Story:** Click this button to generate a story based on the input genre and logical statements. The story will include logical challenges based on the generated propositions.
- **Possible/Not Possible:** Use these buttons to submit your answer regarding the achievability of the goal. The system will verify your answer using Prover9.

4.3 GUI Overview

The graphical user interface is designed to be intuitive and user-friendly, allowing players to easily interact with the game.

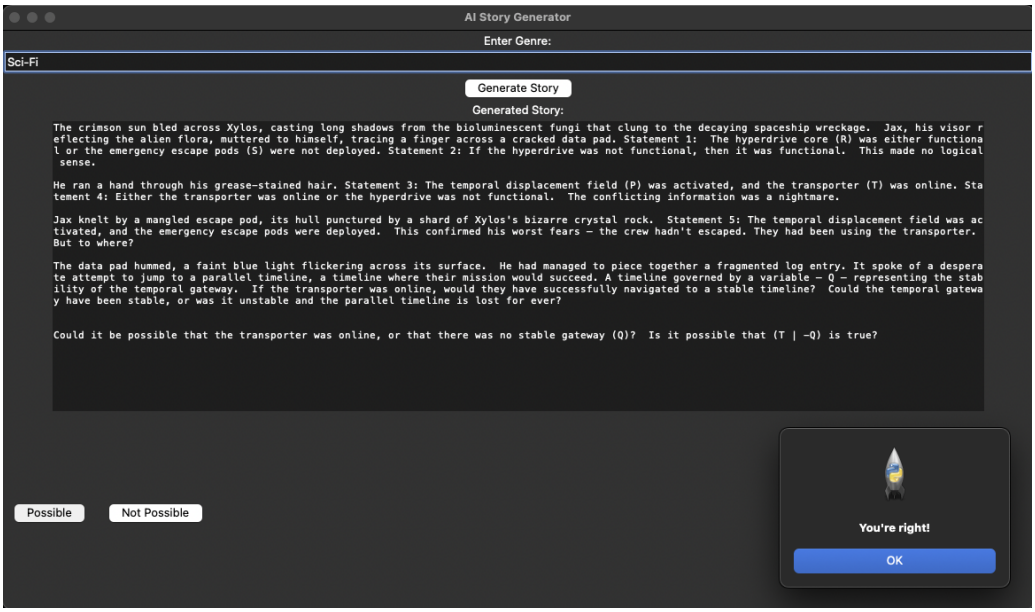

Figure 4.1: The window of the Logical Puzzle Game - Goal is possible.

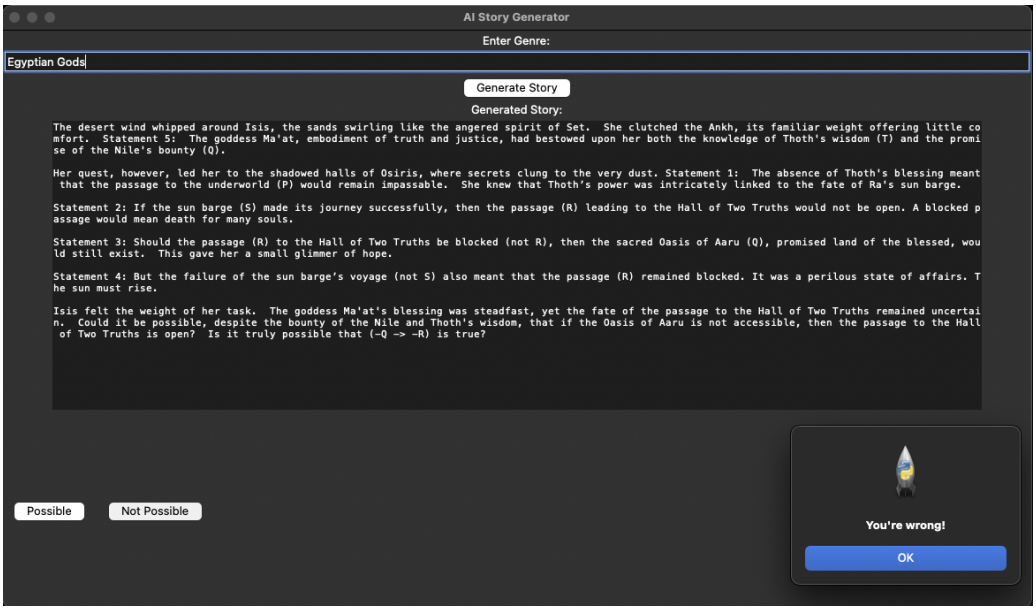

Figure 4.2: The window of the Logical Puzzle Game - Goal is not possible.

Chapter 5

Appendix

5.1 entry.py

```
import os
import re
import subprocess
import time
import random
from tkinter import messagebox

import google.generativeai as genai
import tkinter as tk
from gen import generatePropositionalLogic

genai.configure(api_key="AIzaSyDs8chSG4auDXILuVbGNZsLI0vssqSYsAg")

model = genai.GenerativeModel("gemini-1.5-flash")

import random

def return_true_with_probability():
    return random.random() < 0.8

def run_prover9():
    original_directory = os.getcwd()

    prover9_directory = "./"

    os.chdir(prover9_directory)

    command = "./prover9 -f ./myProver9.in > ./myProver9.out"
    try:
        subprocess.run(command, shell=True, check=True)
        print("Command executed successfully")
    except subprocess.CalledProcessError as e:
        print(f"Error executing command: {e}")
    finally:
        os.chdir(original_directory)
        print(f"Returned to original directory: {original_directory}")

def clean_statements(statements):
    all_statements = "\n".join(statements)

    cleaned_statements = re.sub(r"Statement \d+: ", "", all_statements)
```

```
    statement_list = cleaned_statements.split("\n")

    statement_list = [statement.strip() + "." if not statement.strip().
       endswith('.') else statement.strip() for statement in statement_list]

    return "\n".join(statement_list)

def generate_logic_statements():
    odds = return_true_with_probability()
    prover9_statement, prover9_goal = generatePropositionalLogic(odds)
    return prover9_statement, prover9_goal

def generate_story():
    global generated_statements
    global generated_goal

    genre_text = genre_input.get()
    generated_statements, generated_goal = generate_logic_statements()
    print(generated_statements)
    statements_text = generated_statements

    prompt = (f"if i give you some propositional logic statements, can you
       create a story with following genre: {genre_text}, {statements_text}.
        Please format with Statement: text_of_statement but do not include
       the propositional logic. Please include only the story and their
       statement number, nothing else."
              f"This is the conclusion that should be reached, and you
                  should make it as a final question in such a manner that
                  someone is in doubt whether it is possible or not: Goal of
                  {generated_goal} is true")
    response = model.generate_content(prompt)

    result_text.delete(1.0, "end")
    result_text.insert("end", response.text)

def possible_button_click():
    file_path = "./myProver9.in"

    statements = clean_statements(generated_statements)

    content = f"""
formulas(assumptions).
{statements}
end_of_list.

formulas(goals).
{generated_goal}.
end_of_list.
    """

    try:
        with open(file_path, "w") as file:
            file.write(content)
        print(f"Content written to {file_path}")
    except Exception as e:
        print(f"Error writing to file: {e}")

    time.sleep(1)
    run_prover9()
```

```
    time.sleep(2)

    output_file_path = "./myProver9.out"

    if os.path.exists(output_file_path):
        with open(output_file_path, "r") as output_file:
            content = output_file.read()

            if "THEOREM PROVED" in content:
                root = tk.Tk()
                root.withdraw()
                messagebox.showinfo("Result", "You're right!")
                root.destroy()
            else:
                messagebox.showinfo("Result", "You're wrong!")
                print("Theorem not proved.")
    else:
        print(f"Output file {output_file_path} not found.")

def not_possible_button_click():
    file_path = "./myProver9.in"

    statements = clean_statements(generated_statements)

    content = f"""
formulas(assumptions).
{statements}
end_of_list.

formulas(goals).
{generated_goal}.
end_of_list.
        """

    try:
        with open(file_path, "w") as file:
            file.write(content)
        print(f"Content written to {file_path}")
    except Exception as e:
        print(f"Error writing to file: {e}")

    time.sleep(1)
    run_prover9()

    time.sleep(2)

    output_file_path = "./myProver9.out"

    if os.path.exists(output_file_path):
        with open(output_file_path, "r") as output_file:
            content = output_file.read()

            if "THEOREM PROVED" in content:
                root = tk.Tk()
                root.withdraw()
                messagebox.showinfo("Result", "You're wrong!")
                root.destroy()
            else:
                messagebox.showinfo("Result", "You're right!")
                print("Theorem not proved.")
```

```
    else:
        print(f"Output file {output_file_path} not found.")

root = tk.Tk()
root.title("AI Story Generator")

genre_label = tk.Label(root, text="Enter Genre:")
genre_label.pack()

genre_input = tk.Entry(root, width=150)
genre_input.pack()

generate_button = tk.Button(root, text="Generate Story", command=
    generate_story)
generate_button.pack()

result_label = tk.Label(root, text="Generated Story:")
result_label.pack()

result_text = tk.Text(root, height=25, width=150)
result_text.pack()

possible_button = tk.Button(root, text="Possible", command=
    possible_button_click)
possible_button.pack(side="left", padx=10)

not_possible_button = tk.Button(root, text="Not Possible", command=
    not_possible_button_click)
not_possible_button.pack(side="left", padx=10)

root.mainloop()
```

5.2 gen.py

```
import random
import argparse
import subprocess
from sympy import symbols, And, Or, Not, Implies, satisfiable, simplify

P, Q, R, S, T = symbols('P Q R S T')
variables = [P, Q, R, S, T]

def random_variable():
    """Return a random propositional variable."""
    return random.choice(variables)

def random_operator(expr1, expr2):
    """Return a random logical operator applied to two expressions."""
    operator = random.choice([And, Or, Implies])
    return operator(expr1, expr2)

def random_not(expr):
    """Randomly decide whether to apply a NOT operator."""
    return Not(expr) if random.choice([True, False]) else expr

def generate_random_statement():
    """Generate a random propositional logic statement."""
    var1 = random_variable()
```

```
    var2 = random_variable()
    expr1 = random_not(var1)
    expr2 = random_not(var2)
    return random_operator(expr1, expr2)

def generate_compatible_statements(num_statements=5):
    """Generate a list of logically compatible propositional logic
       statements."""
    statements = []

    while len(statements) < num_statements:
        new_statement = generate_random_statement()
        if satisfiable(And(*(statements + [new_statement]))):
            statements.append(new_statement)

    return statements

def generate_goal(statements, achievable=True):
    """Generate a goal that is a logical consequence of the statements if
       achievable is True, otherwise make it unachievable."""
    while True:
        potential_goal = generate_random_statement()
        implication = Implies(And(*statements), potential_goal)
        if achievable:
            if simplify(implication) == True:
                return potential_goal
        else:
            if not simplify(implication) == True:
                return potential_goal

def convert_to_prover9(expr):
    """Convert a sympy expression to Prover9 syntax."""
    if expr.func == And:
        return f"({convert_to_prover9(expr.args[0])} & {convert_to_prover9(
           expr.args[1])})"
    elif expr.func == Or:
        return f"({convert_to_prover9(expr.args[0])} | {convert_to_prover9(
           expr.args[1])})"
    elif expr.func == Implies:
        return f"({convert_to_prover9(expr.args[0])} -> {convert_to_prover9(
           expr.args[1])})"
    elif expr.func == Not:
        return f"-{convert_to_prover9(expr.args[0])}"
    else:
        return str(expr)

def write_prover9_input(statements, goal, filename="./myProver9.in"):
    """Write the statements and goal to a Prover9 input file."""
    with open(filename, "w") as f:
        f.write("formulas(assumptions).\n")
        for statement in statements:
            f.write(f"    {convert_to_prover9(statement)}.\n")
        f.write("end_of_list.\n\n")
        f.write("formulas(goals).\n")
        f.write(f"    {convert_to_prover9(goal)}.\n")
        f.write("end_of_list.\n")

def run_prover9(input_file="./myProver9.in"):
    """Run Prover9 with the given input file and return whether the goal is
       achievable."""
```

```
    prover9_path = "./prover9"
    result = subprocess.run([prover9_path, "-f", input_file], capture_output
        =True, text=True)
    output = result.stdout
    return "THEOREM PROVED" in output

def generatePropositionalLogic(achievable_flag):
    while True:
        statements = generate_compatible_statements(5)

        goal = generate_goal(statements, achievable_flag)

        write_prover9_input(statements, goal)

        goal_achieved = run_prover9()

        if goal_achieved == achievable_flag:
            prover9_statements = []
            for i, statement in enumerate(statements, 1):
                prover9_statement = convert_to_prover9(statement)
                prover9_statements.append(f"Statement {i}: {
                    prover9_statement}")

            prover9_goal = convert_to_prover9(goal)

            return prover9_statements, prover9_goal

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Generate a propositional
        logic problem with an achievable or unachievable goal.")
    parser.add_argument('--achievable', action='store_true', help="Set this
        flag if the goal should be achievable.")
    args = parser.parse_args()

    while True:
        statements = generate_compatible_statements(5)
        goal = generate_goal(statements, args.achievable)

        write_prover9_input(statements, goal)

        goal_achieved = run_prover9()

        if goal_achieved == args.achievable:
            for i, statement in enumerate(statements, 1):
                prover9_statement = convert_to_prover9(statement)
                print(f"Statement {i}: {prover9_statement}")
            prover9_goal = convert_to_prover9(goal)
            print(f"Goal: {prover9_goal}")
            break
```

Chapter 6

Conclusion

This project successfully combines logical reasoning and storytelling to create an engaging puzzle game. By integrating Python and Prover9, we have developed a tool that challenges players' problem-solving skills and enhances our understanding of automated theorem proving. The project demonstrates the potential of combining AI-driven storytelling with logical problem-solving frameworks.

Bibliography

[1] University of New Mexico - Prover9 and Mace4.
https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.cs.unm.edu/~mccune/mace4/&ved=2ahUKEwig4ZXds5iKAxV3hP0HHVFDM9oQFnoECAwQAQ&usg=AOvVaw2cygJXZM4UC0U6R9Pwl_WU

[2] Wikipedia - Prover9.
https://en.wikipedia.org/wiki/Prover9