Escape Room Game: Propositional Logic Based Project

Team Glamour: Borzan Călina and Gozman-Pop Maria

08.12.2003

1 Introduction

The **Escape Room Game** project is a fun and interactive game where players solve puzzles, disarm traps, and collect keys to escape a virtual room. The game encourages logical thinking, and problem-solving, offering a challenging experience for players of all levels.

Built using Python and Tkinter, the game provides an easy-to-use graphical interface. Players can explore rooms, solve puzzles, and race against the clock to escape the traps. The game also features logic-based challenges like riddles and logic puzzles that keep players engaged.

The project uses automated reasoning tools to ensure all puzzles are solvable and appropriately challenging. The interface is designed to be visually appealing and intuitive, allowing users to navigate menus, view instructions, and start new games easily.

Overall, the Escape Room Game combines logic, fun, and creativity, offering an exciting and educational experience for all players.

2 Project Objectives

The primary objectives of the **Escape Room Game** project are:

- **Developing Logical Puzzles:** Create a series of challenging puzzles that require players to apply computer science knowledge or propositional logic to disarm traps, solve riddles, and progress through rooms.

The puzzles will incorporate logical reasoning and problem-solving skills, ensuring a mentally stimulating experience.

- **Automated Puzzle Generation and Validation:** Utilize propositional logic to automatically generate puzzles and validate solutions. This ensures that puzzles are logically consistent and solvable, providing a fair and challenging game environment.
- **Interactive Gameplay:** Design an intuitive and user-friendly interface where players can interact with the game. Players will navigate rooms, examine clues, and make logical decisions based on propositional statements and truth values.
- **Enhancing Decision-Making through Logic:** Integrate propositional logic into the game mechanics, allowing players to make decisions based on logical deductions, such as determining whether a trap is armed or if a puzzle solution is correct.
- **Educational Value:** Provide players with an opportunity to enhance their understanding of propositional logic while enjoying a gamified experience. The project aims to be both educational and entertaining, promoting logical thinking in an engaging environment.
- **Creating a Dynamic Escape Room Environment:** Develop a series of rooms, each containing unique logical challenges based on propositional logic principles. The game should allow for a flexible and dynamic gameplay experience.

By combining propositional logic with a captivating game environment, this project seeks to deliver both a challenging puzzle-solving experience and a deeper understanding of logical reasoning.

3 Technologies Used

The **Escape Room Game** integrates several key technologies that contribute to the game's logic-based puzzle generation, user interface design, and overall gameplay experience. The core technologies used in the development of this project are outlined below:

- **Prover9:** Prover9 is an automated theorem proving tool used in this project for solving logical problems based on propositional and first-order logic. It is utilized for generating the current state of the program and ensuring the interface reflects is through logical reasoning, being particularly effective in validating the game's logical consistency, solving puzzles, and generating valid scenarios based on logical constraints.
- **Tkinter:** Tkinter is a Python library that provides a simple and efficient way to create graphical user interfaces (GUIs). In this project, Tkinter is used to design the interactive interface through which players engage with the game. It allows for the easy creation of windows, buttons, labels, and other UI components. The library is chosen for its simplicity, flexibility, and compatibility with Python, making it an ideal choice for building the visual components of the Escape Room Game. Tkinter ensures that users can easily navigate between screens, interact with puzzles, and receive feedback on their actions.
- **Python:** Python controlls the flow of the game, and it interracts well with external tools like Prover9. Python's versatility and ease of use make it an excellent choice for this project, as it allows for quick development and easy integration of different components. Python handles the game's main logic, puzzle generation, and user input, ensuring that players have an intuitive and smooth experience. Additionally, Python's robust library ecosystem, including Tkinter and Prover9 integration, facilitates the seamless creation of both the game's logic and its interface.

Together, **Prover9**, **Tkinter**, and **Python** provide the technological backbone for the Escape Room Game. Prover9's powerful reasoning capabilities are used to generate and solve puzzles based on propositional logic, while Python coordinates the game's logic and user interface. Tkinter provides the user-friendly graphical interface that enhances the player experience, making the game both engaging and logical.

4 Methodology

The *Escape Room Game* employs propositional logic to model and solve the various game mechanics. The primary tool for reasoning about game states and solving puzzles is **Prover9**, an automated theorem prover. The game logic is encoded as a set of propositional formulas that define how rooms, doors, keys, puzzles, and traps interact within the game world.

4.1 Prover9 Logic

The game's rules are represented using Prover9's logical framework. Here are the core components modeled in Prover9:

- **Rooms and Doors**: The game world consists of multiple rooms connected by doors. Each door is associated with specific rooms and is only passable if the corresponding key has been obtained.
- **Keys**: Keys are required to open doors. Each key is linked to a specific door.
- **Puzzles**: Solving puzzles rewards the player with keys. Each puzzle's solution is linked to acquiring a key.
- **Traps**: Traps are present in certain rooms and must be disarmed by the player. If triggered, the player must solve a riddle to survive.
- **Movement Rules**: The player can move between rooms if they have the corresponding key and the door is open.

For instance, the logic that governs moving between rooms is defined as:

$$\text{can\_move(player, A, B)} : -\text{door(Y)}, \text{connects(Y, A, B)}, \text{open(Y)}, \text{at(player, A)}.$$

This expression states that a player can move from room A to room B if:

- There is a door Y between rooms A and B (represented by `connects(Y, A, B)`).
- The door Y is open (represented by `open(Y)`).
- The player is currently in room A (represented by `at(player, A)`).

Explanation:

- `can_move(player, A, B)`: Defines the goal of whether the player can move from room A to room B.

- `door(Y)`: Indicates that Y is a door.
- `connects(Y, A, B)`: Specifies that the door Y connects room A with room B.
- `open(Y)`: Checks if the door is open, meaning the player can move through it.
- `at(player, A)`: Ensures the player is currently in room A to be able to move to room B.

This rule states that a player can move from room A to room B if there is an open door between them, and the player is currently in room A.

4.2 Dynamic State Generation

The game state is dynamically updated based on the player's actions. The method:

write_current_state()

generates a Prover9 input file, which includes both static rules (such as the definition of rooms, doors, and keys) and the current dynamic state. This dynamic state includes the player's current location, inventory, solved puzzles, and the status of doors and traps.

In the generated file, the dynamic state is represented by facts that describe the current state of the game. These facts are written as follows:

Stating that the player is currently located in room1:

at(player, room1).

The player having a specific item in their inventory:

has(player, item).

Indicating that the player has solved puzzle1:

solved(player, puzzle1).

Specifying that door1 is open, allowing the player to move through it:

open(door1).

Denoting that trap1 has been disarmed by the player:

disarmed(trap1).

Additionally, the state includes trap locations in rooms, written as:

trap_in_room(room2, trap1).

Finally, the method includes any actions taken by the player, such as opening a door or solving a puzzle, which are also written as facts in the Prover9 input file.

The dynamic state is followed by a section for goals, which define the desired outcomes or objectives for the player to achieve in the game.

Key elements of the dynamic state generation include:

- **Player Location**: The current room where the player is located is recorded.
- **Inventory**: The items the player has collected, such as keys, are tracked.
- **Doors and Traps**: The status of doors (whether they are open) and traps (whether they are disarmed) are updated based on the player's actions.

The dynamic game state is continuously written into the Prover9 input file, allowing the system to reason about the next possible actions and transitions.

For example, if a player solves a puzzle, the corresponding key is granted, and the state is updated to reflect that the player now possesses the key. Similarly, if a player disarms a trap, the room becomes safe, and the state is updated accordingly.

4.3 Solving Puzzles and Progression

The solver is tasked with determining whether the player can proceed based on the current game state. This involves reasoning about whether the player has the necessary keys to open doors, has solved the required puzzles, and has disarmed the traps in each room. The logical framework ensures that each action follows the rules of the game, and only valid states are considered for progression.

5 User Interface

The user interface (UI) of the game is designed using `Tkinter`, the standard Python GUI library. The interface consists of multiple screens, each dedicated to different parts of the game. These screens are designed to be intuitive, providing users with clear instructions, game controls, and real-time feedback. The key UI elements are the dialog windows, buttons, labels, and dynamic display areas that allow interaction with the game.

5.1 Key Components of the User Interface

5.1.1 TrapDialog

The `TrapDialog` is a custom dialog window that appears when a trap is triggered in the game. The player is presented with a riddle and multiple options. The player has a limited time to solve the riddle, and the countdown is displayed on the screen.

- **Riddle Display**: The riddle is displayed at the top, and the player can choose one of several options.
- **Option Selection**: Players select their answer using radio buttons. Once an option is selected, the "Submit" button becomes enabled.
- **Timer**: A countdown timer is shown, which decreases every second until time runs out. If the player fails to choose an answer in time, a message box will appear, and the game ends.
- **Submit**: After choosing an option, players can submit their choice, which will be checked against the correct answer.

Example:

```
self.submit_button = tk.Button(
    self, text="Submit", command=self.submit_choice, state=tk.DISABLED
)
```

5.1.2 WelcomeScreen

This is the initial screen shown to the player, providing them with options to start the game, view the rules, or exit. The screen includes buttons for navigation.

- **Start Game Button**: Navigates to the game menu to start the escape room.
- **View Rules Button**: Displays the game rules in a separate window.
- **Exit Button**: Exits the game.

Example:

```
start_button = tk.Button(
    self, text="Start Game", command=lambda: self.controller.show_frame("GameMenu'
)
```

5.1.3 RulesScreen

The rules screen provides instructions for the player on how to play the game. The player can read through the game rules and navigate back to the welcome screen.

- **Back Button**: Navigates the player back to the welcome screen.

Example:

```
back_button = tk.Button(
    self, text="Back to Welcome", command=lambda: self.controller.show_frame("Welc
)
```

5.1.4 GameMenu

The game menu screen provides several options for the player to start a game or navigate back to the welcome screen. It is shown after the player chooses to start the game from the welcome screen.

- **Start Escape Room Button**: Starts the actual game in the escape room environment.

- **Back Button**: Navigates back to the welcome screen.
- **Exit Button**: Exits the game.

Example:

```
start_game_button = tk.Button(
    self, text="Start Escape Room", command=lambda: self.controller.show_frame("Es
)
```

5.1.5 EscapeRoomGameScreen

This screen represents the main gameplay area. It displays the player's current actions and provides buttons for various in-game actions such as looking around, solving puzzles, opening doors, and checking the safety of rooms.

- **Look Around**: Allows the player to explore the current room.
- **Solve Puzzle**: Enables the player to solve a puzzle in the room, provided that they found it.
- **Open Door**: Allows the player to open a door and move to another room, if they have obtained the key.
- **Move to Another Room**: Allows the player to navigate between rooms.
- **Disarm Trap**: Disarms a trap in the room (if one exists).
- **Check Safety**: Lets the player check if the room is safe.
- **Exit Game**: Exits the game.

Example:

```
self.look_button = tk.Button(
    button_frame, text="Look Around", command=self.look_around
)
```

5.2 Layout and Appearance

- **Color Scheme**: The interface uses a color scheme with beige, brown, and gold tones, suggestive to the name of the theme. For example, the background color of many frames and buttons is a soft beige (#D4C1A0), with accents in darker brown tones for text and buttons.
- **Button Styling**: The buttons are styled with consistent backgrounds and text colors to ensure a cohesive and easy-to-navigate experience. The interactive buttons, like "Submit" and "Look Around", change based on the player's actions (e.g., enabling or disabling).

5.3 Dynamic Updates

Throughout the game, certain UI elements update dynamically:

- **Timer**: In the `TrapDialog`, a countdown timer updates every second, reflecting the remaining time for the player to answer the riddle.
- **Display Area**: The main gameplay screen dynamically updates the text area to show the current game state, such as what the player is doing or any game events that occur.
- **Button States**: Buttons such as "Solve Puzzle" or "Disarm Trap" are initially disabled and are only enabled when the corresponding action becomes available to the player.

Example of dynamically updating a label:

```
self.timer_label.config(text=f"Time Remaining: {self.remaining_time} seconds")
```

5.4 Event Handling and Navigation

- **Event Handlers**: Button presses are associated with specific methods that handle the game's logic, such as starting the game, solving puzzles, or interacting with traps.
- **Navigation**: The `controller.show_frame()` method is used to switch between different screens (Welcome, Game Menu, Rules, etc.). Each screen is a separate Tkinter frame, and the `show_frame` method manages which frame is visible at any given time.

Example of switching frames:

```
self.controller.show_frame("GameMenu")
```

The user interface is designed to be engaging, user-friendly, and functional, providing a seamless and immersive experience for players throughout their escape room adventure.

6 Future Enhancements

While the current version of the Escape Room Game provides an engaging experience using propositional logic to define game rules and player interactions, several enhancements could improve the game. These enhancements could include:

- **Introducing More Interconnected Rooms:** Introduce more rooms that could potentially lead back to one another, making the game more difficult, more interesting, and more engaging.
- **Expanded Puzzle Complexity:** Introduce more complex puzzles and riddles that require multi-step reasoning and deeper logical deduction, utilizing propositional logic to create dynamic and challenging scenarios.
- **AI-Based Opponent:** Incorporate an AI opponent that competes with the player by solving puzzles and attempting to escape, using propositional logic to model the opponent's actions and strategies.
- **Multi-Player Support:** Develop a multi-player version of the game, where multiple players collaborate or compete to escape. The game's logic system can be extended to account for multiple participants and their interactions.
- **Better User Interface:** Improve the game's graphical user interface (GUI) by adding animations and sound effects to make the experience more immersive.
- **Extended Logical Framework:** Expand the use of propositional logic in defining game mechanics, such as adding more complex conditions and logical relations for room connections, trap triggers, and key unlocking mechanisms.

7 Conclusion

In conclusion, the Escape Room Game is an application of propositional logic in game development. By using propositional logic, we have been able to create a dynamic environment where the player's actions, puzzles, and interactions are governed by logical rules. These rules not only define the game's structure but also influence how the player interacts with the environment, solving puzzles, and navigating through rooms.

The use of propositional logic allows for a flexible and expandable game design that can easily be enhanced with more complex scenarios and logical structures. The integration of propositional logic in this project showcases how formal logic can be applied to interactive systems, providing a rich and engaging player experience. The game's design offers a unique approach to problem-solving and game mechanics, which can be further refined and expanded in future versions.