AGI: A Garry Oddyssey
A Text-Based Adventure Game

Candidate Number: 291085

December 5, 2024

1 Overview

For this coursework, I developed a text-based adventure game set in an abandoned OpenAI facility. Starting from a basic codebase that provided simple room navigation and a backpack system, I expanded it into an engaging narrative where players must navigate through the facility, solve puzzles, and prevent an AGI system from converting the universe into paperclips (inspired by the famous paperclip maximizer thought experiment).

You play Gary, a new intern at OpenAI who's first day on the job is not ideal, as no-one is awating him in the lobby. Gary however, decides he will figure out what happened in the AI lab.

The game combines elements of corporate satire with AI safety themes, featuring characters like Sam Altman and various AI-related puzzles. Players need to think strategically about using items, managing access levels, and interacting with NPCs to progress through the story. I've also implemented a save game system to allow players to preserve their progress.

2 Running the Game

The game requires Python 3.7+ but has no additional dependencies. The game was written and tested in VS Code. To play:

```
python game.py
```

3 Tutorial

When you start the game, you'll find yourself outside the OpenAI facility. Here's a quick guide to get started:

1. Basic commands:
    - `go [direction]` - Move between rooms
    - `search` - Look for items and puzzles

- `take [item]` - Pick up an item
- `use [item]` - Use an item
- `speak` - Talk to NPCs
- `solve [answer]` - Solve puzzles
- `save` - Save your progress
- `load` - Load a saved game

2. First steps:
   - Find the basic keycard in the lobby
   - Use it to access the corridor
   - Explore Roon's den to find the scientific keycard puzzle

4 Map

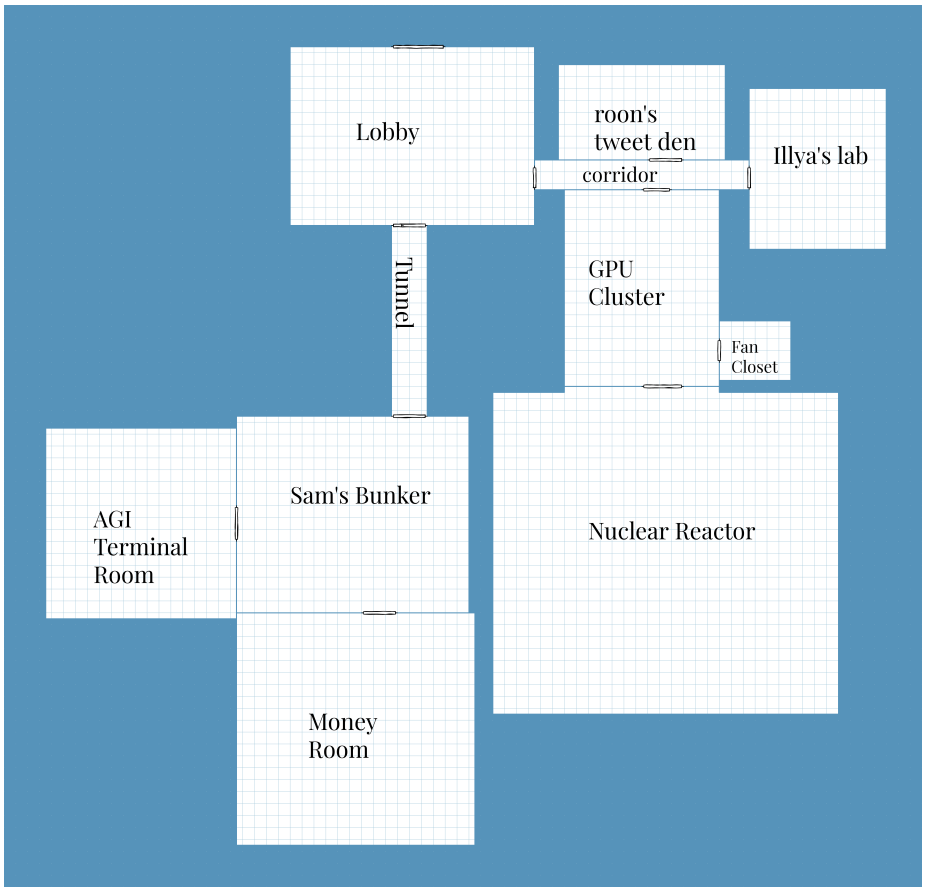

Figure 1: Game Map

5 Game Features

- 12 unique rooms with varying security clearance requirements
- Three-tiered keycard security system
- Interactive NPCs with dynamic dialogue progression
- Multiple puzzle types (password-based and item-based)
- Inventory management system
- Save/Load game functionality
- Win condition requiring strategic item use

6 Save Game System

I implemented a JSON-based save game system that preserves:

- Player location and inventory
- Room lock states
- Puzzle completion status
- NPC dialogue progress
- Overall game progress

The save system uses JSON for data persistence, making it easy to debug and modify if needed. Players can save/load at any time using the `save` and `load` commands.

7 Modifications to Original Code

The original codebase provided basic room navigation and a simple backpack class. I made the following significant modifications:

7.1 Enhanced Room Class

Added support for:

- Security levels and locked states
- Item and NPC storage
- Puzzle integration
- Detailed room descriptions with dynamic content
- Exit management with keycard requirements

7.2 New Classes

1. **Item**: Implements game items with properties like:
   - Portability flags
   - Usage capabilities
   - Keycard functionality with security levels
2. **NPC**: Manages character interactions with:

- Progressive dialogue system
- Item interaction capabilities
- State-based responses

3. **Puzzle**: Implements puzzle mechanics:
   - Password-based solutions
   - Item-based solutions
   - Room unlocking rewards
   - Item rewards
4. **Player**: Represents the main character in the game who can:
   - Move between rooms
   - Pick up and carry items in a backpack
   - Interact with the game environment

8 Interesting Design Features

8.1 Security System

Implemented a keycard-based security system with three access levels:

- Basic (Level 1): General access
- Scientific (Level 2): Laboratory access
- Executive (Level 3): Restricted areas

8.2 Puzzle System

Created two types of puzzles:

- Password puzzles (e.g., roon's phone puzzle requiring "xitter" as the solution)
- Item-based puzzles (e.g., cooling the GPU cluster with a fan)

8.3 Save Game Mechanism

Implemented a robust save system that:

- Uses JSON for data persistence
- Saves complete game state
- Handles complex object relationships
- Provides error handling for corrupt saves

9 Testing

Implemented comprehensive unit tests covering:

- Room navigation and exit validation

- Item interaction and inventory management
- Puzzle mechanics and solutions
- Security system functionality
- Save/Load functionality
- Win condition verification

Also conducted extensive play-testing, which was more enjoyable.

10 Challenges and Solutions

The main challenges I encountered were:

1. **State Management**: Tracking room states, puzzle completion, and NPC interactions required careful design of class relationships.
2. **Save System**: Implementing save/load functionality required careful consideration of object serialization and state restoration.
3. **Security System**: Implementing the keycard system required modifications to both Room and Item classes to handle access levels and unlocking mechanics.
4. **Puzzle Flexibility**: Creating a system that could handle both password and item-based puzzles while maintaining clean code structure.

These were addressed through careful class design and extensive testing to ensure robust gameplay mechanics.