Open-World Adventure Game

Liqian Ma, Qingwei Lan, Wentao Yao

Project Introduction and Goal

We built a maze game with a knowledge base and inference engine called "open-world adventure", where a role-played hero can navigate, explore, and fight. The goal of this project is to make use of Prolog and its inference engine to create a knowledge base with our own rules.

This world is a rule-based world with complex systems such as weather, monster fighting, exploration, day-night cycles, etc. The hero travels through the world by choosing its next move and the world's reasoning engine will determine whether the move is a valid one or not. The reasoning engine will also use existing rules and facts to determine events that may happen after the hero makes the move. For example, if the hero enters a location with thunderstorms, the hero will lose some health and stamina.

Knowledge Representation & Reasoning

Map System

We have a map with a predefined size like 10×10. At each coordinate, we have 6 different objects:

- empty: walkable spot
- wall: not walkable
- start: hero starts adventure at this location
- gem: if found, the game ends
- rock: initially not walkable, but can be broken
- peril: a negative number indicates a peril (monster) and the hero needs to defeat the peril in order to advance

At the start of the game, we will define categories for each position and these positions are static and non-changeable during the session.

Combat System

The combat system defines how the hero fights a monster. The damage dealt by the hero and the monster is dependent on the hero and monster's attack values and the results of rolling a dice. Again, we introduced randomness (rolling a dice) into the system to make the game more fun.

Knowledge Encoding: The abilities of the monster and hero are encoded as facts in the KB.

```
monster_health(100).
hero_health(100).
hero_stamina(100).
hero_attack(20).
```

Reasoning: The combat process is defined by rules that depend on the abilities of the monster, the abilities of the hero, and the outcome of rolling a dice.

The combat code is too complicated to be listed here but can be found here.

Day-Night Cycle System

The world holds a clock that counts time. Each move made by the hero takes 1 "tick" and a single day consists of 24 "ticks". The first 12 "ticks" are daytime and the following 12 "ticks" are nighttime. This system would have an impact on the hero's abilities and the abilities of monsters in the world. The latter part would be implemented as a follow-up because it is too complicated for our initial version.

Knowledge Encoding: The global time is kept through a global counter. This counter is asserted into the KB as a fact.

```
time(0).
```

Reasoning: The day/night cycle is inferred from the global time counter by dividing the time by 24 and taking the remainder. If the remainder is within range $[0, 12)$, then it is daytime. If it is within $[12, 24)$, then it is nighttime. The rule is shown below:

```
is_day() :- time(T), T mod 24 #< 12.
is_night() :- time(T), T mod 24 #>= 12, T mod 24 #< 24.
```

Weather System

There are 4 types of weather conditions:

- **Clear**: the weather is clear and nothing will happen; this is the default.
- **Rainy**: the hero will lose 1 stamina point for each clock cycle standing on the cell with rainy weather.
- **Thunder**: bad weather; the hero will lose 1 stamina point and 1 health point for each clock cycle standing on the cell with thunder weather.
- **Foggy**: the hero cannot see anything on a cell with foggy weather. If the cell has a monster, the hero won't see it and may risk walking right into it, causing an unnecessary fight.

During each clock tick, the world will randomly modify the weather for 25% of the cells in the map. Randomness is included to ensure the game contains a sense of uncertainty.

Knowledge Encoding: The weather for each cell is a fact (knowledge) that is asserted into the KB.

```
weather(C, R, rainy).
weather(C, R, thunder).
weather(C, R, foggy).
```

Reasoning: We built rules that implement the events listed above for each weather condition. For example: the hero cannot see anything on the cell with foggy weather, and if the hero tries to inspect the cell, it will print "Foggy weather, cannot see". This is accomplished through a rule that checks the weather for a cell location to see if the weather there is foggy.

The code for the weather system is also complicated and can be found here.