CIS 700 – Homework 4

[redacted]

March 4, 2022

Problem 1

I picked the wikiHow article **How to Survive in the Jungle** (https://www.wikihow.com/Survive-in-the-Jungle). One reason that I chose this article is that surviving in the jungle has similarities with text adventure games. For example, in text adventure games, players are expected to deal with dangerous situations and survive (like beating the enemy) in order to win; in the case of surviving in the jungle, people also need to protect themselves from dangers and survive (like predators, poisonous fruits, unhealthy water and so on). I think problems in surviving in the jungle and problems in text adventure games are very relatable. In addition, I love travel and adventure. And I enjoy watching TV shows about surviving in the wild. Therefore, I found this article interesting.

Problem 2

First, I selected the part about collecting water (method 1 step 1) and boiling water (method 1 step 2) because water is important for people to survive in the jungle. Since boiling water requires fire, I also translated parts about starting fire from the wikiHow article **How to Start a Fire with Sticks** (https://www.wikihow.com/Start-a-Fire-with-Sticks), including gathering tinder and firewood (part 1), making tools (part 2), and starting fire (part 4). Since food is also important for survival, next I translated the part about catching fish (method 2 step 3). I also translated the part about building the shelter for people to sleep and rest (method 3 step 1, 2, 3).

Problem 3

Give some example of the actions, types, and predicates you used in your domain.

a) Types

General types are **item** (objects that can be picked up and used), **player** (person), **direction** (directions for the person to move), and **location** (location names). I also created some specific types that are the subtypes of **item**: **water**, **wood**, **container**, **drill**, **spear**, **fish**, **foliage**. For example, **water** is an object that can be picked up, and there are some actions that can only be performed on **water**, like collecting water, filtering water, boiling water and drinking water. Similarly, **fish** can be picked up and there are some actions that can only be performed on **fish**, like catching fish, cooking fish and eating fish. The reason that I defined these subtypes is that I can easily write actions that only operate on these specific types instead of operating on all objects.

b) Predicates

Besides the predicates that are already defined, like *at*, *blocked*, *connected*, and *inventory*, I also created some other predicates.

(*has_water location*) indicates whether this location has water, which can be helpful for an action like collecting the water.

(*has_tree location*) indicates whether this location has trees where player can get wood and foliage.

(*has_fish location*) indicates whether this location has fish that player can catch and eat.

(*has_fire location*) indicates whether a fire has been created by the player at this location.

(*has_shelter location*) indicates whether a shelter has been created by the player at this location.

(*boiled water*) indicates whether the water has been boiled and thus is safe for drinking.

(*filtered water*) indicates whether the water has been filtered.

(*cooked fish*) indicates whether the fish has been cooked and thus is safe for eating.

(*thirsty player*) indicates whether the player is thirsty and thus need to drink water.

(*hungry player*) indicates whether the player is hungry and thus need to eat fish.

(*safe player location*) indicates whether the player is safely living in shelter and has weapon and fire on at this location.

c) Actions

Below are some examples of my action schema.

collect_water: use container to collect water from a location that has water.

The parameters are player, container, water and location. The preconditions are player is at this location that has water and player has a container in inventory and player currently do not have water in inventory. This action will add water to player inventory and set the properties of water to be not filtered and not boiled using predicates *boiled* and *filtered*.

make_fire: make fire at a location.

The parameters are player, drill and location. The preconditions are player is at this location that currently does not have fire and player has a drill in inventory. This action will set the predicate that this location now has a fire using predicate *has_fire*.

boil_water: boil water to kill any bacteria.
The parameters are player, water and location. The precondition are player is at this location that has fire and player has water in inventory and water is already filtered but not yet boiled (water can be filtered by another action **filter_water**). This action will mark the water as boiled using predicate *boiled*.

drink_water: drink boiled water.
The parameters are player and water. The preconditions are that player has water in inventory and water is boiled. This action will mark the player as not thirsty using predicate $thirsty$.

eat_fish: eat cooked fish.
The parameters are player and fish. The preconditions are that player has fish in inventory and fish is cooked (fish can be cooked by another action **cook_fish**). This action will mark the player as not hungry using predicate $hungry$.

build_shelter: build a shelter using wood and foliage.
The parameters are player, wood, foliage and location. The preconditions are that player has wood and foliage in inventory and player is at this location that currently does not have a shelter. This action will mark the location as having a shelter using the $has\_shelter$ predicate.

survive: live safely in the shelter with weapon and fire on.
The parameters are player, spear and location. The preconditions are that player has spear in inventory and player is at this location that has a shelter and fire and it is not safe currently. This action will mark the location as safe for player using the $safe$ predicate.

Problem 4

I created 4 problems. The initial states for each problem are the same, in order to be more consistent. Below are the initial states for the problems.

The objects are: npc (player); camp, path, riverside, rainforest, cave (location); in, out, north, south, east, west, up, down (direction); water; wood; container; drill; spear; fish; foliage.

The initial states are: path is at the east of camp; riverside is at the east of path; rainforest is at the north of riverside; cave is at the east of rainforest; rainforest has trees; riverside has water and fish; there is container at camp; npc is at camp and is thirsty and hungry.

a) The first problem is how to make fire at riverside.
The goal is that the predicate ($has\_fire\ riverside$) is true, which means that the player has successfully started the fire at riverside. The solution is: 1. go east from camp to path. 2. go east from path to riverside. 3. go north from riverside to rainforest. 4. get wood at rainforest. 5. go south from rainforest to riverside. 6. make a firing drill using wood. 7. make fire using drill at riverside. Below is the computed solution.

```
Time: 0.012757301330566406s
plan:
go east npc camp path
go east npc path riverside
go north npc riverside rainforest
get_wood npc wood rainforest
go south npc rainforest riverside
make_drill npc wood drill
make_fire npc drill riverside
```

b) The second problem is how to make the player no longer thirsty (by drinking boiled water). The goal is that the predicate (*thirsty npc*) is not true, which means that the player is no longer thirsty. The solution is: 1. get container at camp. 2. go east from camp to path. 3. go east from path to riverside. 4. collect water using container at riverside. 5. go north from riverside to rainforest. 6. filter water. 7. get wood at rainforest. 8. make a firing drill using wood. 9. make fire using drill at rainforest. 10. boil water using fire at rainforest. 11. drink water. Below is the computed solution.

```
Time: 0.6982676982879639s
plan:
get container npc camp
go east npc camp path
go east npc path riverside
collect_water npc container water riverside
go north npc riverside rainforest
filter_water npc water
get_wood npc wood rainforest
make_drill npc wood drill
make_fire npc drill rainforest
boil_water npc water rainforest
drink_water npc water
```

c) The third problem is how to make the player no longer hungry (by eating cooked fish). The goal is that the predicate (*hungry npc*) is not true, which means that the player is no longer hungry. The solution is: 1. go east from camp to path. 2. go east from path to riverside. 3. go north from riverside to rainforest. 4. get wood at rainforest. 5. go south from rainforest to riverside. 6. make a firing drill using wood. 7. make fire using drill at riverside. 8. make a spear using wood. 9. catch fish using spear at riverside. 10. cook fish using fire at riverside. 11. eat fish. Below is the computed solution.

```
Time: 0.29029083251953125s
plan:
go east npc camp path
go east npc path riverside
go north npc riverside rainforest
get_wood npc wood rainforest
go south npc rainforest riverside
make_drill npc wood drill
make_fire npc drill riverside
make_spear npc spear wood
catch_fish npc spear fish riverside
cook_fish npc fish riverside
eat_fish npc fish
```

d) The last problem is how to make the player safe in cave (living safely in a shelter with weapon and fire on).
The goal is that the predicate ($safe\ npc\ cave$) is true, which means that the player is safe at cave. The solution is: 1. go east from camp to path. 2. go east from path to riverside. 3. go north from riverside to rainforest. 4. get wood at rainforest. 5. make a firing drill using wood. 6. make a spear using wood. 7. get foliage at rainforest. 8. go east from rainforest to cave. 9. make fire using drill at cave. 10. build a shelter using wood and foliage at cave. 11. survive in the cave, which has shelter and fire, and player has a spear. Below is the computed solution.

```
Time: 0.41378164291381836s
plan:
go east npc camp path
go east npc path riverside
go north npc riverside rainforest
get_wood npc wood rainforest
make_drill npc wood drill
make_spear npc spear wood
get_foliage npc foliage rainforest
go east npc rainforest cave
make_fire npc drill cave
build_shelter npc wood foliage cave
survive npc spear cave
```

Problem 5

One limitation I encountered is that for my action schema, I often need to define many types and predicates for it to work. wikiHow article has very detailed descriptions and mentions lots of objects and actions. For example, in the article, it mentions too many different objects, like bamboo stalks, coconut shell, stick, branch and so on. To translate precisely, I need to create

way too many types, predicates and actions to cover the entire description. And our domain will be very complex and the computational requirement will be large as well. Therefore, I have to narrow things down by only translating key parts of the description.

Another limitation I noticed is that as the goal becomes more complex and thus needs more steps to finish, the computing time takes much longer. This is a practical concern. If we want to fully translate the wikiHow article, the solution will probably have many steps. In this way, it is going to take much more time to compute a plan.

Problem 6

I think my PDDL can be potentially used as an interesting challenge for text-adventure-style game. The domain contains corresponding actions that are similar to the actions in the text adventure game. The problem definitions and goals in PDDL can be considered as a series of player's actions that lead to some consequences in text adventure game (like catch fish, beat troll, and so on). To cover all possible consequences that can occur in text adventure game, the PDDL has to create the same number of problem definitions, one for each consequence. In this way, the PDDL is pretty much the same as the text adventure game.

Problem 7

I can build the training data as follows. Each prompt corresponds to one description of one step in the article. And each completion contains a description of types and predicates that are used and a description of the corresponding actions (with parameter, precondition, and effect) for that description. Then I use GPT-3 to train. And after training, given a description in the article, it could generate information about expected types, predicates and actions. And I can use that information to construct PDDL. In this way, GPT-3 can semi-automatically convert a wikiHow article.