CIS 700 HW4: Convert WikiHow to PDDL

March 3rd, 2022

1. What wikiHow article did you pick and why?

Instead of taking one of the thoughtfully curated options from the homework writeup, I ended up converting the article “How to Throw an Anime Party”. This did cause some problems down the line, so I somewhat regret it (it lacks a cohesive story unlike other options such as “How to Survive on a Desert Island’ and “How to Survive a Comet Hitting Earth”, but at the time, I just found the concept of the article itself funny. I’ve done a lot of things in my life because they were funny and this is one of them.

2. What portions of the article did you select to translate to PDDL?

From the section “Preparing Food”, I translated the steps “Make or order main and side dishes” and “Serve beverages”. From the section “Decorating for the Party”, I combined the steps “Put out anime balloons” and “Hang paper lanterns” into one decoration-related step.

3. Examples of the actions, types, and predicates used.

I modeled different actions mentioned in the article such as cooking food (a cook action) and serving drinks (an open and a serve action). For types, I subclassed the item type to create a food and a drink type - because the actions that can be taken with food (such as being cooked) aren’t necessarily transferable to items in general. I generally used predicates to model state, from modeling if a person has money or not to monitoring the state of decorations (like whether a balloon is inflated or not). Some code examples are given below:

Actions:

```
(:action cook
    :parameters (?p - player ?l - location ?f - food)
    :precondition (and (has_kitchen ?l) (at ?p ?l))
    :effect (inventory ?p ?f)
  )
```

```
  (:action serve
    :parameters (?p - player ?d - drink ?l - location)
    :precondition (and (inventory ?p ?d) (opened ?d) (at ?p ?l))
    :effect (and (not (inventory ?p ?d)) (at ?d ?l))
  )
```

Types:

```
    food drink - item
```

Predicates:

```
    (has_kitchen ?l - location) ; a location has a kitchen
    (has_fridge ?l - location) ; a location has a fridge
    (inflated ?balloon - item) ; the balloon is blown up
```

4. Goals, states, and solutions

My goal was to have the player's apartment/house to be in a state that allows one to host an anime themed party. To that end, the initial state models the person's house as well as typical locations of different required objects. For each of the three problems:

Cooking Food (Have food prepared for party guests):

```
  (:init
     (connected kitchen east livingroom)
     (connected livingroom west kitchen)
     (connected bedroom west livingroom)
     (connected livingroom east bedroom)
     (connected bedroom south closet)
     (connected closet north bedroom)
     (connected cupboard south kitchen)
     (connected kitchen north cupboard)
     (has_kitchen kitchen)
     (at npc bedroom)
     (at phone bedroom)
     (inventory npc money)
     (orderable sushi)
  )
```

```
  (:goal (and
     (at stirfry livingroom)
     (at sushi livingroom)
     (at soup livingroom)
     (at clams livingroom)
     (at teriyaki livingroom)
   ))
```

```
go west npc bedroom livingroom
go west npc livingroom kitchen
cook npc kitchen stirfry
cook npc kitchen sushi
cook npc kitchen soup
cook npc kitchen clams
cook npc kitchen teriyaki
go east npc kitchen livingroom
drop stirfry npc livingroom
drop sushi npc livingroom
drop soup npc livingroom
drop clams npc livingroom
drop teriyaki npc livingroom
```

Setting up Drinks (Have drinks prepared for party guests):

```
  (:init
     (connected kitchen east livingroom)
     (connected livingroom west kitchen)
     (connected bedroom west livingroom)
     (connected livingroom east bedroom)
     (connected bedroom south closet)
     (connected closet north bedroom)
     (connected cupboard south kitchen)
     (connected kitchen north cupboard)
     (has_kitchen kitchen)
     (has_fridge bedroom)
     (at npc bedroom)
     (at phone bedroom)
     (inventory npc money)
     (at soymilk kitchen)
     (opened soymilk)
     (at soda cupboard)
     (at wine kitchen)
     (at teabag cupboard)
  )
```

```
  (:goal (and
     (at soymilk livingroom)
     (at soda livingroom)
     (at sake livingroom)
     (at tea livingroom)
     (at beer livingroom)
   ))
```

```
get phone npc bedroom
buy_sake npc
get_beer npc bedroom
go west npc bedroom livingroom
go west npc livingroom kitchen
go north npc kitchen cupboard
get teabag npc cupboard
get soda npc cupboard
go south npc cupboard kitchen
get soymilk npc kitchen
```

```
brew npc kitchen
go east npc kitchen livingroom
drop soymilk npc livingroom
drop soda npc livingroom
drop sake npc livingroom
drop beer npc livingroom
drop tea npc livingroom
```

Decorations (Have the venue decorated):

```
   (:init
      (connected kitchen east livingroom)
      (connected livingroom west kitchen)
      (connected bedroom west livingroom)
      (connected livingroom east bedroom)
      (connected bedroom south closet)
      (connected closet north bedroom)
      (connected cupboard south kitchen)
      (connected kitchen north cupboard)
      (at npc bedroom)
      (at tv livingroom)
      (at balloon closet)
      (at lantern closet)
   )
```

```
   (:goal (and
    (on tv)
    (inflated balloon)
    (hung lantern)
   ))
```

```
plan:
go south npc bedroom closet
get balloon npc closet
get lantern npc closet
go north npc closet bedroom
go west npc bedroom livingroom
hang_lantern npc
blow_balloon npc
turn_on_tv npc livingroom
```

5. Limitations of PDDL encountered

One limitation of PDDL that I was annoyed by, though not a fault of the language itself, is the fact that I think the solver runs with performance that is exponential with regards to how many clauses are inputted, so with some of my goals having many clauses, it took a long time to test PDDL iterations before I got one with correct syntax that behaved how I expected. Another issue I ran into is the lack of number support - I think I could've modeled, for example, a player's budget by manually defining every integer I wanted to support myself as a predicate, but this would be way too labor consuming. Some extensions of PDDL I came across on the web do allow numbers, but I wasn't sure if I could use these in the class. Additionally, I would've liked some higher level way to encode crafting - like specific ingredient lists required in the inventory to make specific foods. The only way I could think to do this was to create a specific cook action for each food, which I again found very labor intensive.

6. Viability as a text-based game

I think my PDDL is about halfway to being interesting as a challenge. The introduction of some of the mechanics I wish I could've implemented/described in the prior section, creating more tradeoffs/decisions for the player to make, would increase the player's engagement and enjoyment. For example, creating the budget/inventory and crafting system which forces players to decide how to allocate money and resources such as ingredients and craft materials into food and decorations to maximize partygoer enjoyment would be the exact kind of constrained optimization scenario that engineers enjoy.

7. GPT-3 auto-conversion

I can think of two approaches. The first is to provide a natural language description of the wikihow article steps that are converted and the corresponding PDDL as the prompt, as well as the natural language description of the article you wish to convert, and hope that GPT-3 generates something close to valid PDDL. This could be further improved by fine-tuning GPT-3 to produce such PDDL pairs, with hopes that it learns the syntax of PDDL. A more out-there approach I can think of is to try to "teach" GPT-3 what different syntaxes mean by describing them using natural language in the prompt, though I think this is less likely to work. However, if it does work, it'll be more generalizable to PDDL syntax extensions.