# CSE 291 – AI Agents
# 1/16 - How to Make a Simulation

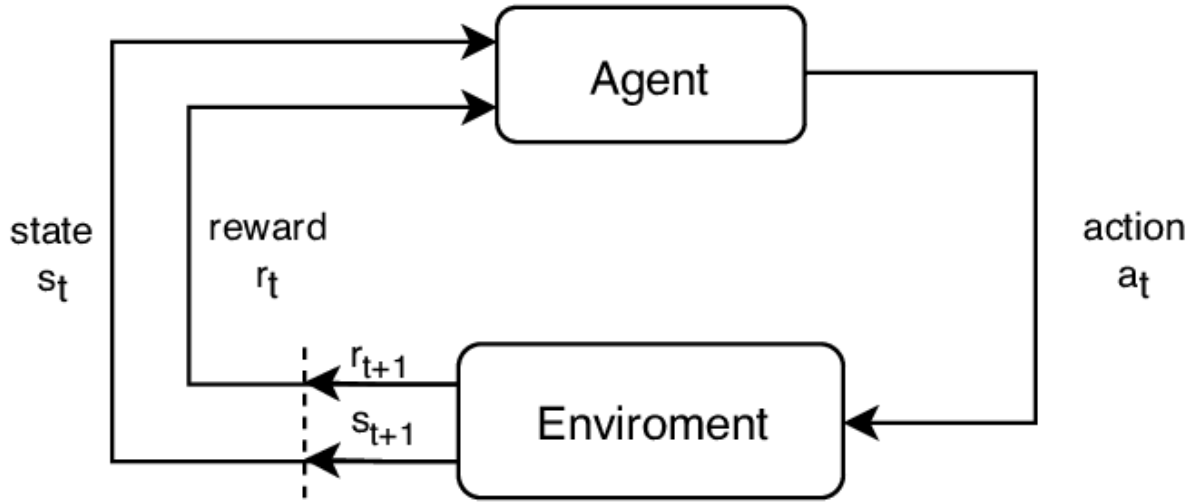
Prithviraj Ammanabrolu

# Recap: Why do we need simulations?

- Most tasks have many ways you can do them, e.g. “do the laundry” → how many clothes, which machine, what detergent, etc. etc.
- You usually do not know the “global” optimal solution ahead of time but usually know when you are done
- So you need to explore! Find many solutions and compare to see which is most efficient

# Basic components of a simulation

- From an MDP perspective, it contains at least <S, A, T>
  - S = set of all states
  - A = set of all actions
  - T = transition matrix T: (S, A)→S



# Basic components of a simulation

- S = set of all states
  - propositions that are true: you are in a house, door is open, knife in drawer
- A = set of all actions
  - take knife from drawer, walk through door
- T = transition matrix T: (S, A)→S
  - (you are in a house & door is open, walk through door) → you are outside

There are pre-conditions that need to be met to perform a certain action, and post-conditions that are true after

# Simulation making methods

- How changes based on why you're doing it
- If you're trying to make a fun video game
  - Is it simple? Just write the rules yourself
  - Is it complicated? Use a game engine like Unity (relatively simple) or Unreal (hard)
- Are you trying to make it easy to do AI stuff in it?

# Simulation making methods

- Are you trying to make it easy to do AI stuff in it?
- Classical planning eg Zork / Ai2 THOR / Mujoco etc. – you want to have ~PDDL enablement
- LLM based simulations eg AI Dungeon – some kind of basic guardrails

# PDDL – Planning Domain Definition Language

- Standard encoding for classic planning tasks
- Many specific languages for creating simulations have similarities with PDDL
- Syntax of the language isn’t as important as the core concepts (most good LLMs can take care of syntactic sugar)

# What's in a PDDL task?

- Objects: Things in the world that interest us.
- Predicates: Properties of objects that we are interested in; can be true or false.
- Initial state: The state of the world that we start in.
- Goal specification: Things that we want to be true.
- Actions/Operators: Ways of changing the state of the world.

2 .pddl files, domain and problem

Slide credits for most PDDL stuff: Malte Helmert and Sheila McIllraith UofT.

# Example

Gripper task with four balls: There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.

- Objects: The two rooms, four balls and two robot arms.
- Predicates: Is x a room? Is x a ball? Is ball x inside room y? Is robot arm x empty?
- Initial state: All balls and the robot are in the first room. All robot arms are empty.
- Goal specification: All balls must be in the second room.
- Actions/Operators: The robot can move between rooms, pick up a ball or drop a ball.

# Objects

- Rooms: rooma, roomb
- Balls: ball1, ball2, ball3, ball4
- Robot arms: left, right

In PDDL:

```
(:objects rooma roomb
          ball1 ball2 ball3 ball4
          left right)
```

# Predicates

ROOM(x) – true iff x is a room
BALL(x) – true iff x is a ball
GRIPPER(x) – true iff x is a gripper (robot arm)
at-robby(x) – true iff x is a room and the robot is in x
at-ball(x, y) – true iff x is a ball, y is a room, and x is in y
free(x) – true iff x is a gripper and x does not hold a ball
carry(x, y) – true iff x is a gripper, y is a ball, and x holds y
In PDDL:

```
(:predicates (ROOM ?x) (BALL ?x) (GRIPPER ?x)
             (at-robby ?x) (at-ball ?x ?y)
             (free ?x) (carry ?x ?y))
```

# Initial State

ROOM(rooma) and ROOM(roomb) are true.
BALL(ball1), …, BALL(ball4) are true.
GRIPPER(left), GRIPPER(right), free(left) and free(right) are true.
at-robby(rooma), at-ball(ball1, rooma), …, at-ball(ball4, rooma) are true.
Everything else is false.
In PDDL:

```
(:init (ROOM rooma) (ROOM roomb)
        (BALL ball1) (BALL ball2)  (BALL ball3) (BALL ball4)
        (GRIPPER left) (GRIPPER right) (free left) (free right)
        (at-robby rooma)
        (at-ball ball1 rooma) (at-ball ball2 rooma)
        (at-ball ball3 rooma) (at-ball ball4 rooma))
```

# Goal Specification

at-ball(ball1, roomb), …, at-ball(ball4, roomb) must be true.

Everything else we don’t care about.

In PDDL:

```
(:goal (and (at-ball ball1 roomb)
            (at-ball ball2 roomb)
            (at-ball ball3 roomb)
            (at-ball ball4 roomb)))
```

# (Movement) Action/Operator

Description: The robot can move from x to y.

Precondition: ROOM(x), ROOM(y) and at-robby(x) are true.

Effect: at-robby(y) becomes true. at-robby(x) becomes false. Everything else doesn't change.

In PDDL:

```
(:action move :parameters (?x ?y)
        :precondition (and (ROOM ?x) (ROOM ?y) (at-robby ?x))
        :effect (and (at-robby ?y) (not (at-robby ?x))))
```

# How is this used for planning?

- Classic symbolic planners can read in PDDLs and give you solutions.
- There are many planners https://planning.wiki/ref/planners/atoz - will cover more later

# Other simulator creating languages

- Complex text games often use Inform7
- Syntax is annoying so this was partially an attempt to make a more “natural language” way to make sims

# Initial State

ROOM(rooma) and ROOM(roomb) are true.
BALL(ball1), …, BALL(ball4) are true.
GRIPPER(left), GRIPPER(right), free(left) and free(right) are true.
at-robby(rooma), at-ball(ball1, rooma), …, at-ball(ball4, rooma) are true.
Everything else is false.
In PDDL:

```
(:init (ROOM rooma) (ROOM roomb)
        (BALL ball1) (BALL ball2)  (BALL ball3) (BALL ball4)
        (GRIPPER left) (GRIPPER right) (free left) (free right)
        (at-robby rooma)
        (at-ball ball1 rooma) (at-ball ball2 rooma)
        (at-ball ball3 rooma) (at-ball ball4 rooma))
```

# Initial State

ROOM(rooma) and ROOM(roomb) are true.

BALL(ball1), …, BALL(ball4) are true.

GRIPPER(left), GRIPPER(right), free(left) and free(right) are true.

at-robby(rooma), at-ball(ball1, rooma), …, at-ball(ball4, rooma) are true.

Everything else is false.

In Inform7:

Ball1 and Ball2 and Ball3 and Ball4 are in RoomA.

GripperLeft and GripperRight are in RoomA.

# (Movement) Action/Operator

Description: The robot can move from x to y.

Precondition: ROOM(x), ROOM(y) and at-robby(x) are true.

Effect: at-robby(y) becomes true. at-robby(x) becomes false. Everything else doesn't change.

In PDDL:

```
(:action move :parameters (?x ?y)
        :precondition (and (ROOM ?x) (ROOM ?y) (at-robby ?x))
        :effect (and (at-robby ?y) (not (at-robby ?x))))
```

# (Movement) Action/Operator

Description: The robot can move from x to y.

Precondition: ROOM(x), ROOM(y) and at-robby(x) are true.

Effect: at-robby(y) becomes true. at-robby(x) becomes false. Everything else doesn’t change.

In Inform7:

Move to RoomB

Before moving to RoomB:

    try silently opening the door

    continue the action

# In Class Activity!

- Go to https://blog.zarfhome.com/2025/01/the-visible-zorker
- Read the blog (if you haven't already)
- Play Zork1 until you get to a score of 35, save your transcript (cheat sheet of actions on next slide and in Gradescope assignment)
- Take a sequence of 5 actions and write the pseudo PDDL version Objects, Predicates, Initial State, Goal Spec, Actions – you can use https://fareskalaboud.github.io/LearnPDDL/ as a cheat sheet
- Submit both on Gradescope under In Class Activity 1/16

You just started up a game
and now you're staring at
*text* and a *blinking cursor*
and you *don't know what to do*! (>|)

**Don't panic kids—**
**Crazy Uncle Zarf is here to help you get started…**

These commands are very common:

| | |
|---|---|
| **EXAMINE** *it* | **PUSH** *it* |
| **TAKE** *it* | **PULL** *it* |
| **DROP** *it* | **TURN** *it* |
| **OPEN** *it* | **FEEL** *it* |

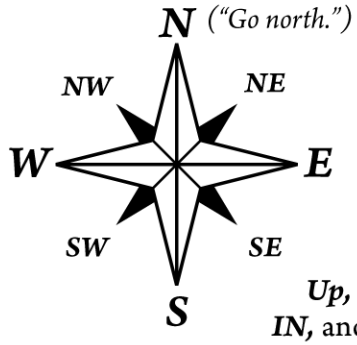
**PUT** *it* **IN** *something*
**PUT** *it* **ON** *something*

***When in doubt, examine more.***

*Does the game intro suggest* **ABOUT, INFO, HELP**? *Try them first!*

You are standing in an open field west of a white house,
with a boarded front door. There is a small mailbox* here.

*Try opening!




Also: ***Up, Down, IN,*** and ***OUT***

You can try all sorts of commands
on the things you see.
Try the commands that make sense!
Doors are for opening; buttons are for pushing;
pie is for eating. *(Mmm, pie.)*

◇◇◇◇

If you meet a person, these should work:

**TALK TO** *name*
**ASK** *name* **ABOUT** *something*
**TELL** *name* **ABOUT** *something*
**GIVE** *something* **TO** *name*
**SHOW** *something* **TO** *name*

*Each game has slightly different commands, but they all look* ***pretty much like these.***

You could also try:

| | |
|---|---|
| **EAT** *it* | **CLIMB** *it* |
| **DRINK** *it* | **WAVE** *it* |
| **FILL** *it* | **WEAR** *it* |
| **SMELL** *it* | **TAKE** *it* **OFF** |
| **LISTEN TO** *it* | **TURN** *it* **ON** |
| **BREAK** *it* | **DIG IN** *it* |
| **BURN** *it* | **ENTER** *it* |
| **LOOK UNDER** *it* | **SEARCH** *it* |

**UNLOCK** *it* **WITH** *something*

Or even:

| | |
|---|---|
| **LISTEN** | **JUMP** |
| **SLEEP** | **PRAY** |
| **WAKE UP** | **CURSE** |
| **UNDO**† | **SING** |

†*Take back one move — handy!*

"What if I only want to type one or two letters?"

◇◇◇◇

**N/E/S/W/NE/SE/NW/SW: GO**
*in the indicated compass direction.*
**L: LOOK**
*around to see what is nearby.*
**X: EXAMINE**
*a thing in more detail.*
**I:** *take* **INVENTORY**
*of what you possess.*
**Z: WAIT**
*a turn without doing anything.*
**G:** *do the same thing* **AGAIN**

◇◇◇◇

*A service of the*
*People's Republic of Interactive Fiction:*
***http://pr-if.org***