Integrating AI2THOR and Soar

Center for Integrated Cognition

Jim Beyer

Nathan Glenn

AI2THOR

- Allen Institute for AI
- Physics simulator, AI research environment written in Unity/C#
- Controlled via Python over HTTP
- Symbolic actions
- Object states

```
from ai2thor.controller import Controller
controller = Controller(width=800, height=600,
                        scene="FloorPlan1")
```

```
controller.step({"action": "RotateLeft"})
controller.step({"action": "MoveAhead"})
```

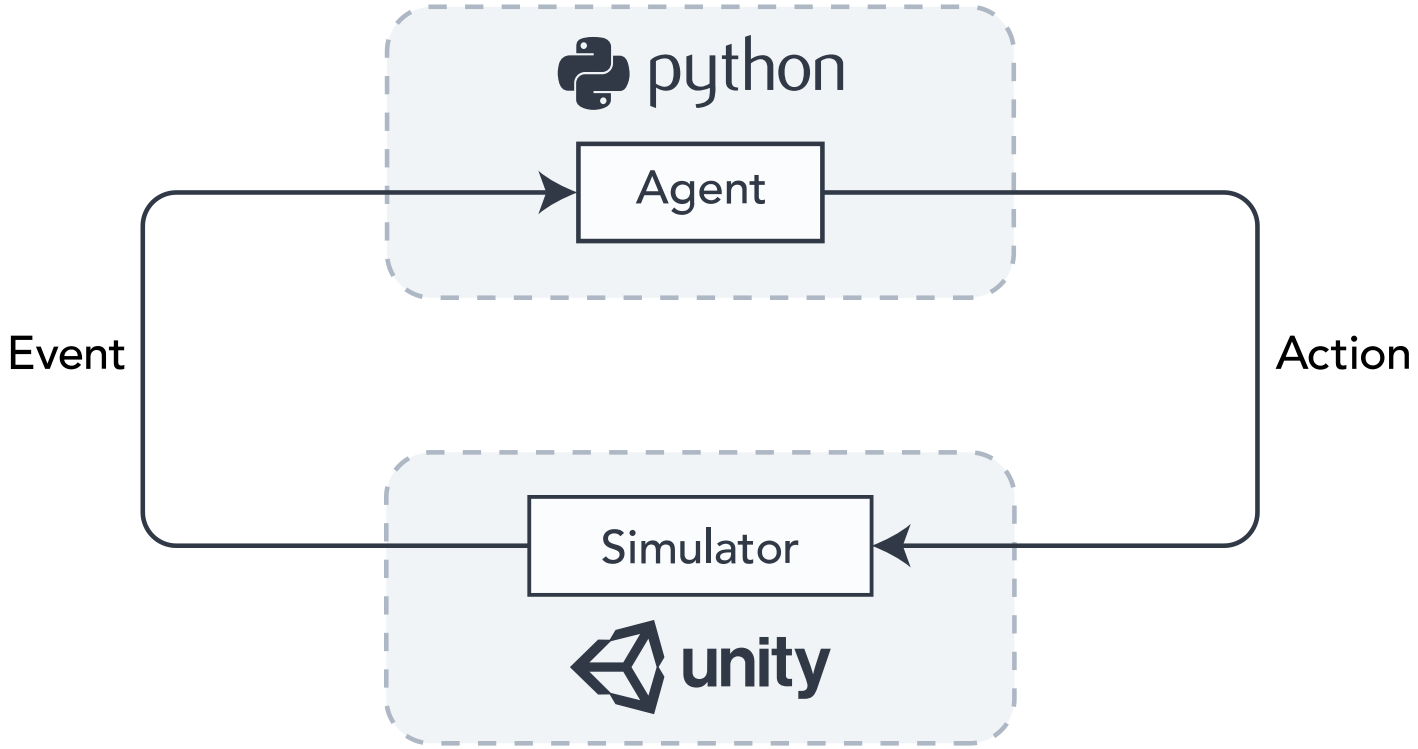
```
controller.step({"action": "PutObject",
                 "objectId": "Microwave|-00.24|+01.69|-02.53"})
controller.step({"action": "CloseObject",
                 "objectId": "Microwave|-00.24|+01.69|-02.53"})
```

```
event = controller.step({"action": "PickupObject",
                         "objectId": "Potato|-01.66|+00.93|-02.15"})
print(event.metadata["objects"][0]["name"])
```

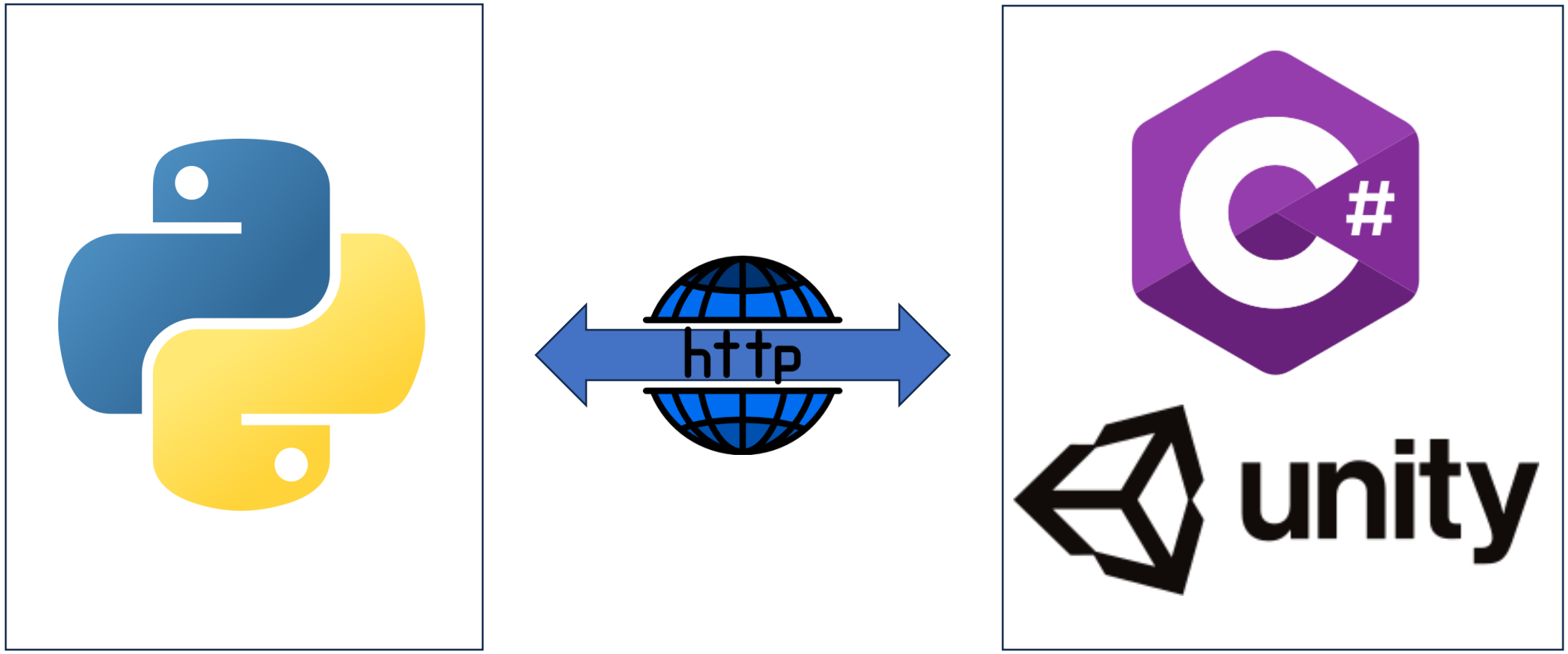

AI2-THOR
Frost Aid


Security Camera

AI2THOR Architecture



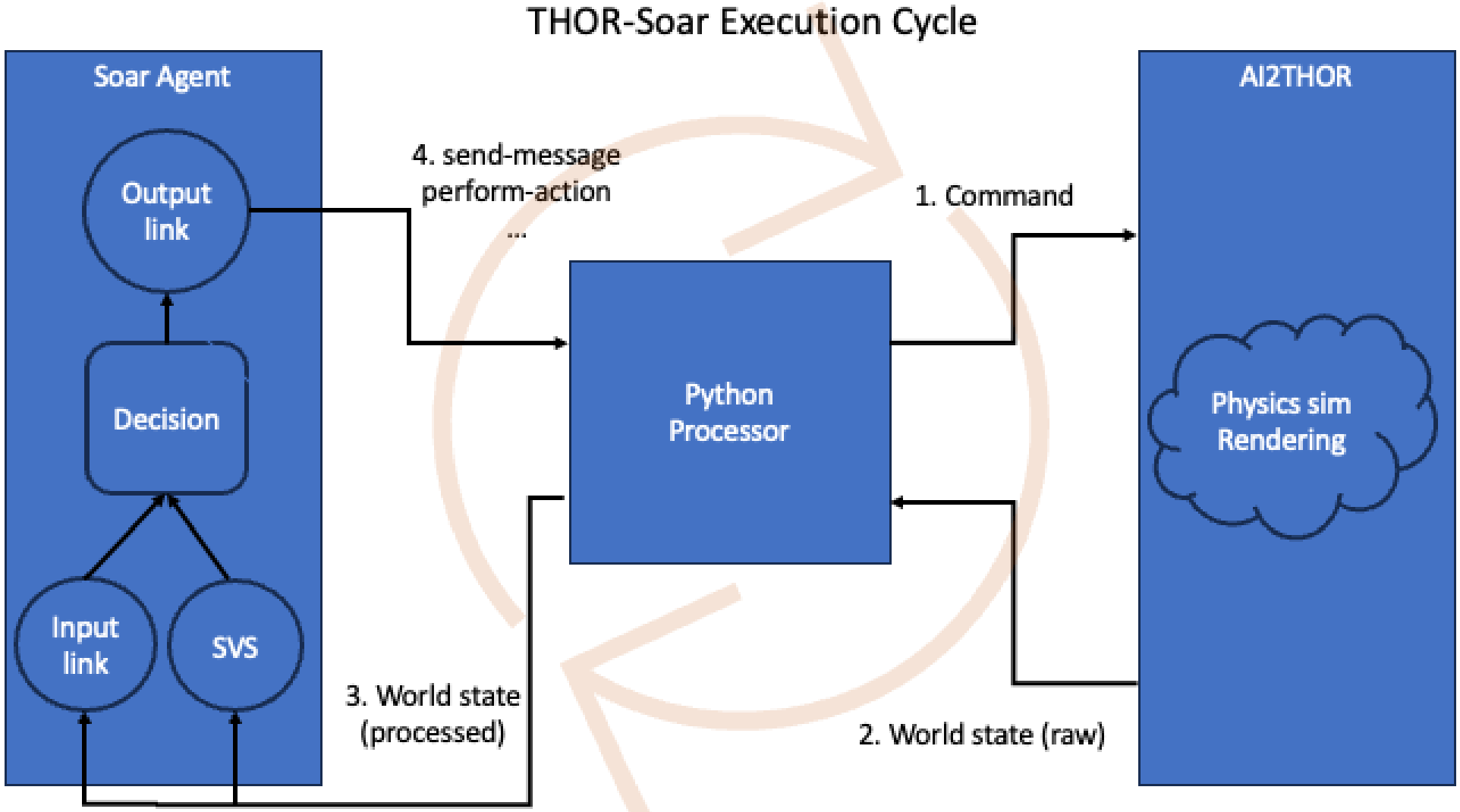
AI2THOR Architecture

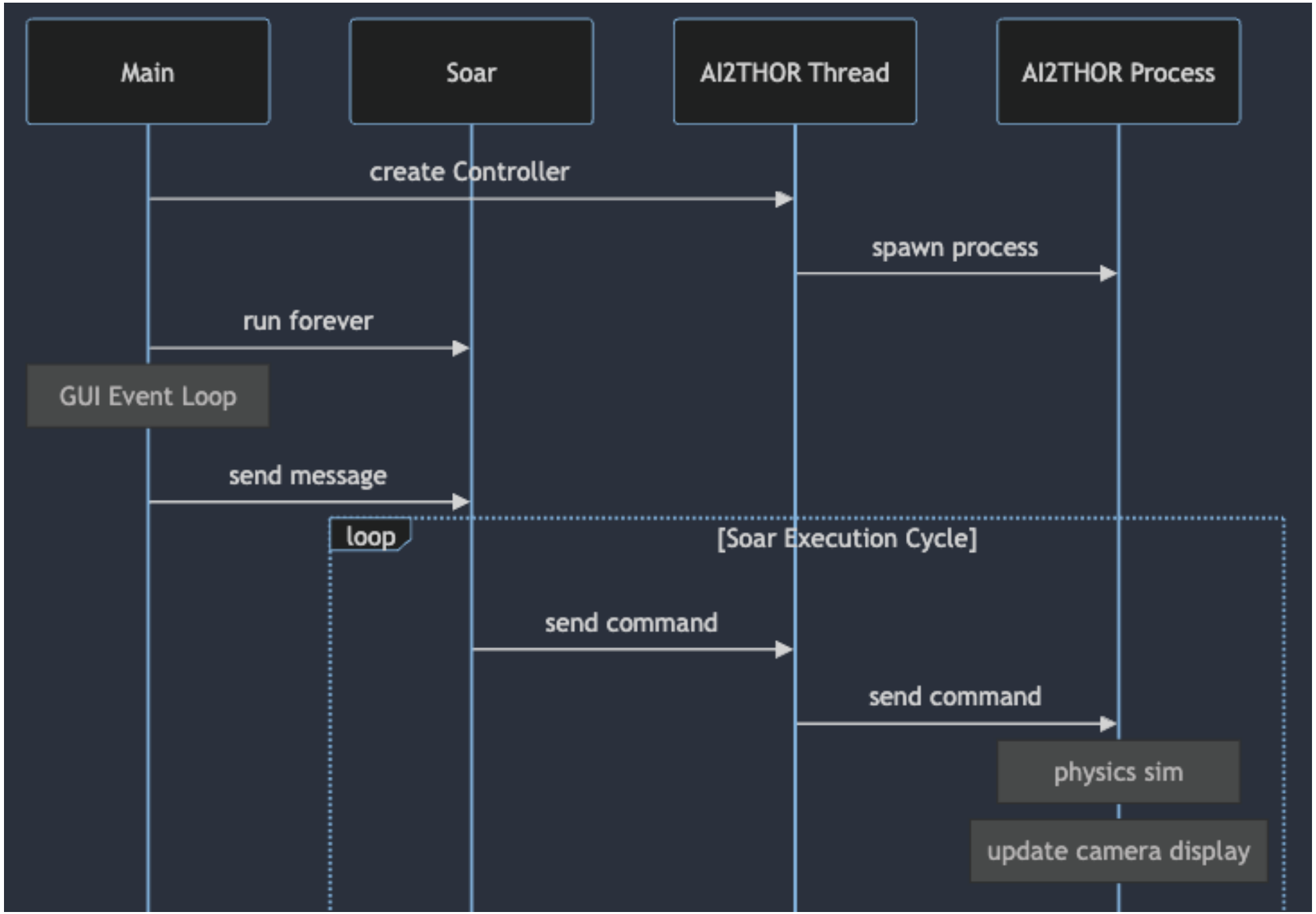


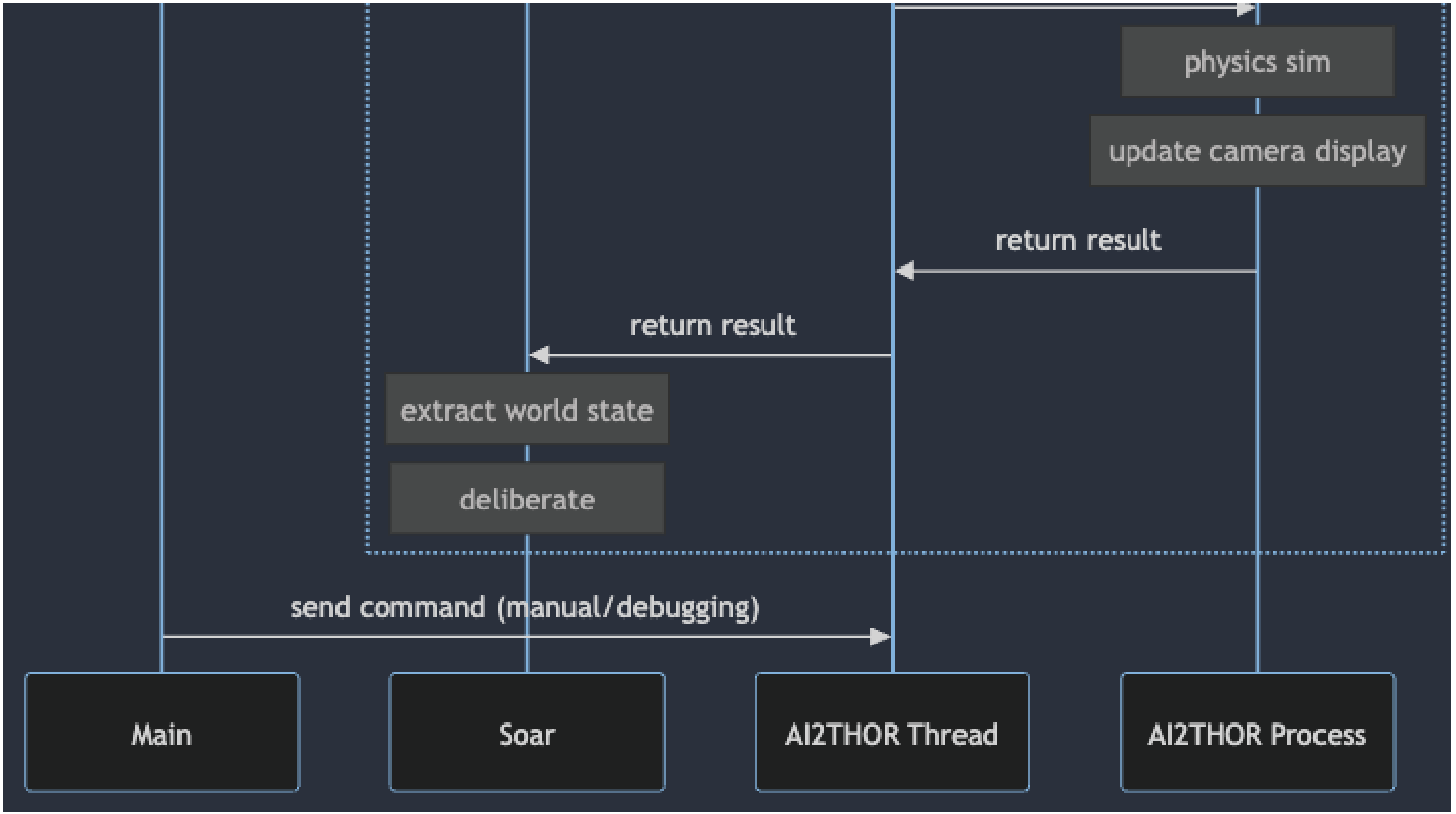
Client

Server

THOR-Soar Architecture




Main
Soar
AI2THOR Thread
AI2THOR Process
create Controller
spawn process
run forever
GUI Event Loop
send message
loop
[Soar Execution Cycle]
send command
send command
physics sim
update camera display


physics sim
update camera display
return result
return result
extract world state
deliberate
send command (manual/debugging)
Main
Soar
AI2THOR Thread
AI2THOR Process

AI2THOR APIs

- iTHOR – symbolic actions, rich scenes
- ManipulaTHOR – fine-grained motor control and sensing
- RoboTHOR – real remote environments
- ProcTHOR – JSON-based scenes

AI2THOR APIs

- **iTHOR – symbolic actions, rich scenes**
- ManipulaTHOR – fine-grained motor control and sensing
- RoboTHOR – real remote environments
- ProcTHOR – JSON-based scenes

Working Assumptions

- Object identification
- Motor control
- Navigation/environment mapping

iTHOR Object Types

Object Type	Scenes	Actionable Properties
AlarmClock	Bedrooms (All)	Pickupable
AluminumFoil	Kitchens (1/30)	Pickupable
Apple	Kitchens (All)	Pickupable, Sliceable

Pysoarlib

- Wraps Python SML bindings
- Manages:
  - Registering handlers
  - I/O
  - Agent/kernel lifetime
  - Configuration
- https://github.com/Center-for-Integrated-Cognition/pysoarlib

Agent Config

- Agent source, smem source, open debugger, remote/local
- Frustum culling
- AI2THOR params like viewing distance

Input Link

- Robot's position and pose
- Held object ID
- Absolute time

```
(I5 ^arm A1 ^moving-status stopped ^pose P1)
  (A1 ^holding-object none ^moving-status wait)
  (P1 ^camera-pitch 0.000000 ^pitch 0.000000 ^roll 0.000000 ^x -1.000000
        ^y 1.000000 ^yaw 3.141593 ^z 0.900999)
(T1 ^ms 200)
```

Input Link

- Objects:
  - Location/position
  - States: open/closed, cooked/raw, on/off
  - Material name, mass, temperature
  - Handle

```
(I2 ^language L2 ^objects O1 ^room-name kitchen ^scene-name FloorPlan1_physics
      ^self I5 ^time T1)
  (O1 ^object O2 ^object O77 ^object O76 ^object O75 ^object O74 ^object O73
    (O2 ^category apple ^grabbed false ^in-view false ^mass 0.200000
        ^material food ^object-handle Apple1 ^sliced false
        ^temperature room-temp)
    (O77 ^broken false ^category window ^in-view false ^object-handle Window76
        ^temperature room-temp)
    (O76 ^broken false ^category vase ^grabbed false ^in-view false
        ^mass 1.000000 ^material glass ^object-handle Vase75
        ^temperature room-temp)
```

Object Attributes

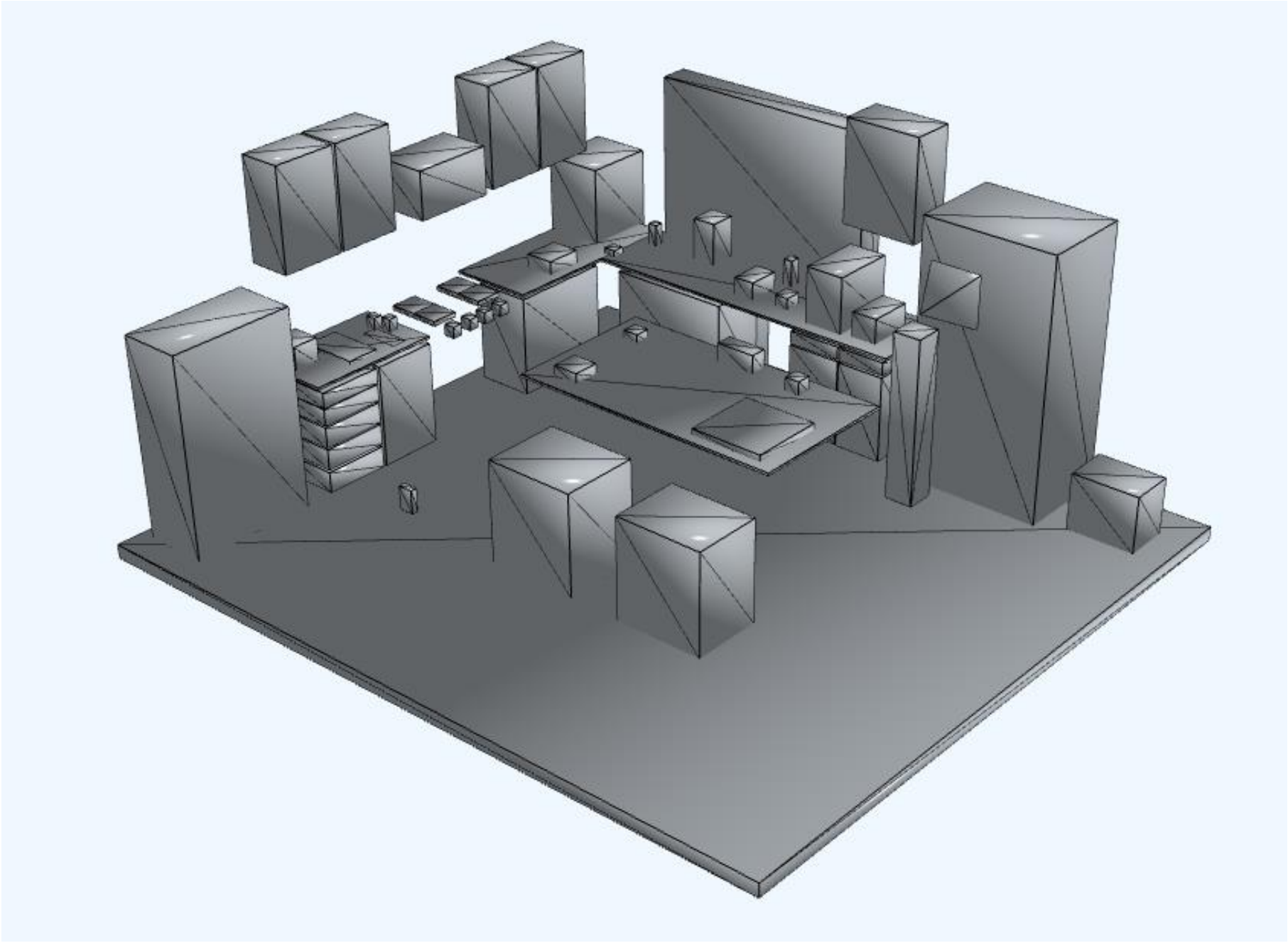
Name	Values
category	pot, coffee machine, microwave, …
material	wood, paper, glass, metal, …
mass	[float]
temperature	cold, room-temp, hot
liquid-contents	none, water, coffee, wine
fill-amount	empty, full
ms-remaining	[int]
grabbed	[bool]

Name	Values
activated	[bool]
open	[bool]
broken	[bool]
dirty	[bool]
sliced	[bool]
used-up	[bool]
cook-state	raw, cooked, burnt
in-view	[bool]

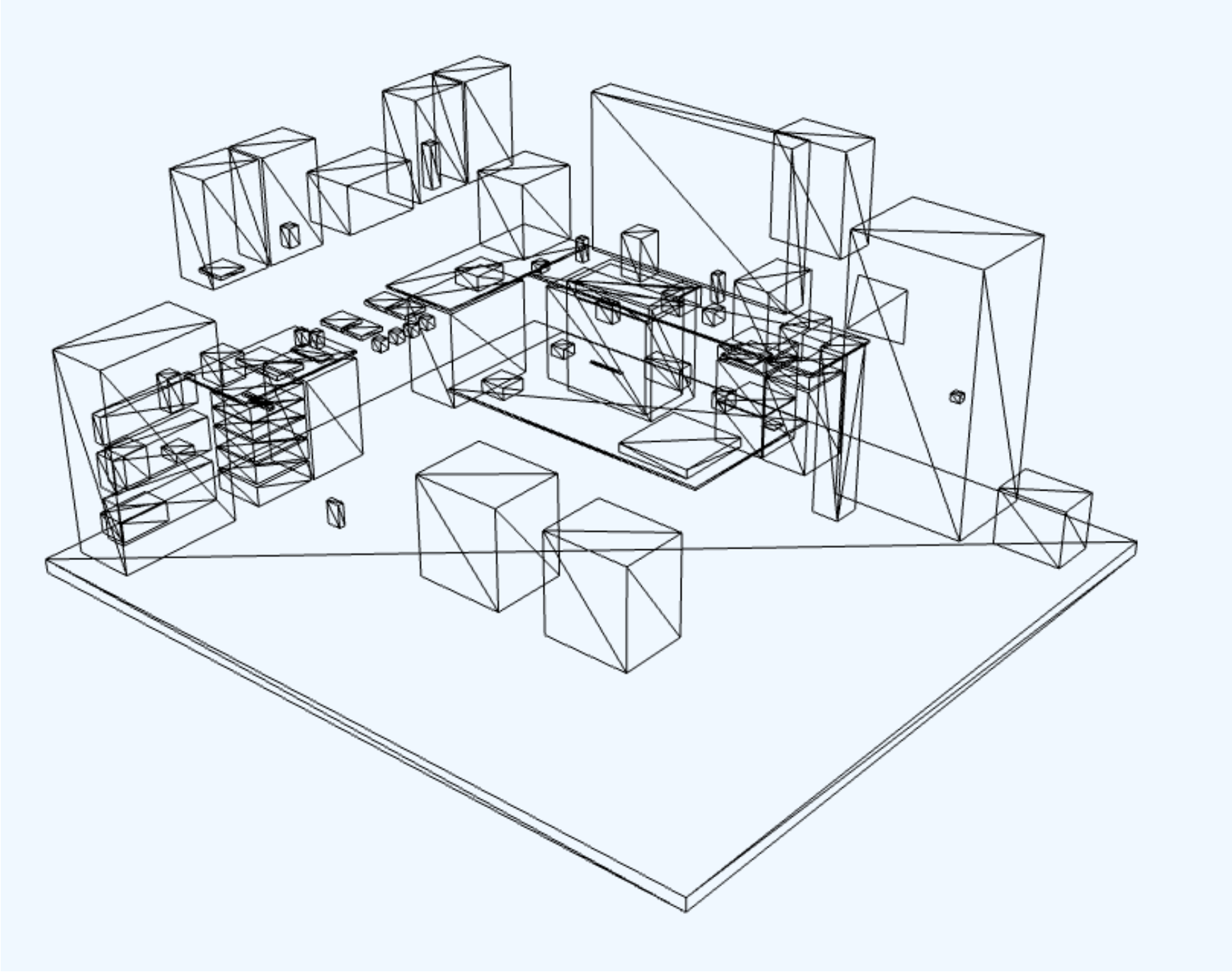
SVS

```
svs S1.scene.world.Apple1
id:     Apple1
parent: world

Local transform:
pos:   -0.465155 0.475745 1.15673
rot:   0         0        0
scale: 0.106014  0.106025 0.11345

World transform:
pos:                -0.465155 0.475745 1.15673
rot (quaternion): 0          0        0       1
scale:              0.106014  0.106025 0.11345

Tags:
object-source perception

vertices
0.5  0.5  0.5
0.5  0.5  -0.5
0.5  -0.5 0.5
0.5  -0.5 -0.5
-0.5 0.5  0.5
-0.5 0.5  -0.5
-0.5 -0.5 0.5
-0.5 -0.5 -0.5
```

SVS



SVS



Modeling Time

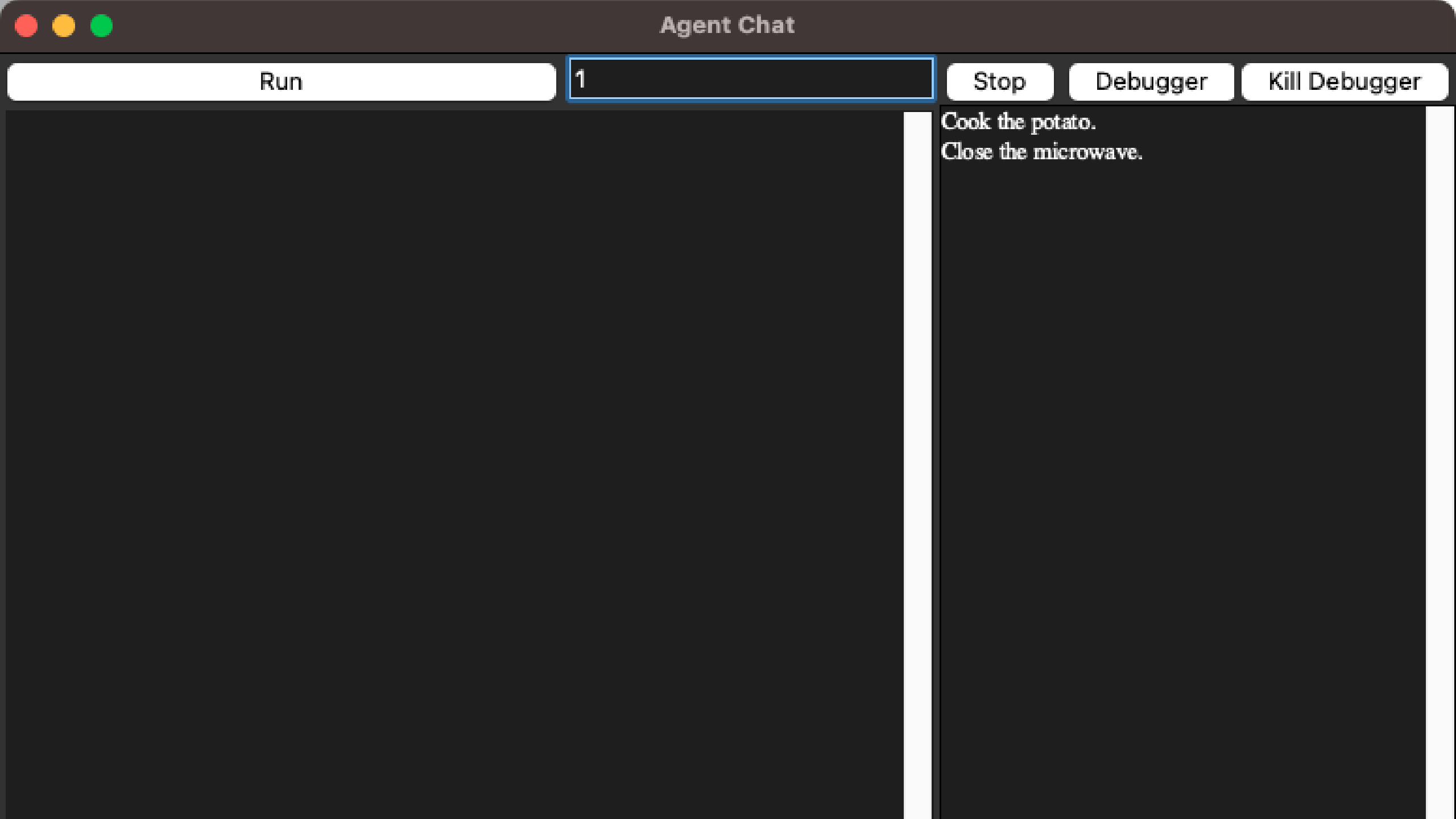
- Missing in iTHOR API 😢
- Simulated in Python wrapper
- Simulated time passes per action
- Simulated time passes per Soar decision cycle

Timers and Cooking

- Cooking accumulates energy in cooked items
- Food is cooked or burnt when energy is accumulated
- Coffee maker, toaster, etc. turn off when timer expires

Timers/Cooking Configuration

- ms_per_decision_cycle = 50
- ms_per_action = 2000
- cook_energy_joules = {potato: 15000, bread: 1000, default: 1000}
- cook_power_watts = {microwave: 1000, toaster: 250, default: 1000}
- burn_tolerance = 0.9


Agent Chat
Run
1
Stop
Debugger
Kill Debugger
Cook the potato.
Close the microwave.


Agent Controller
L
Λ
R
U
<
V
>
D
Open
Close
Grab
Put
Clean
Dirty
Fill Coffee
Empty
Toggle
Cook
Break
Slice
Use Up
Approach
Lookat
Load Scene
Reset
Toggle Timed
Record
Replay
Next Timer
Scale

Unity Basics

- AI2Thor built on Unity and C# scripts (Some Python as well).
- (Most) all games objects based on novel defined type:
  SimObjectPhysics
- All objects have unique object IDs.
- New objects have been introduced to AI2Thor:
- Carrot (and slices)
- Soda Can
- Dishwasher
- Oven
- Fully understanding the system has been a work in progress.

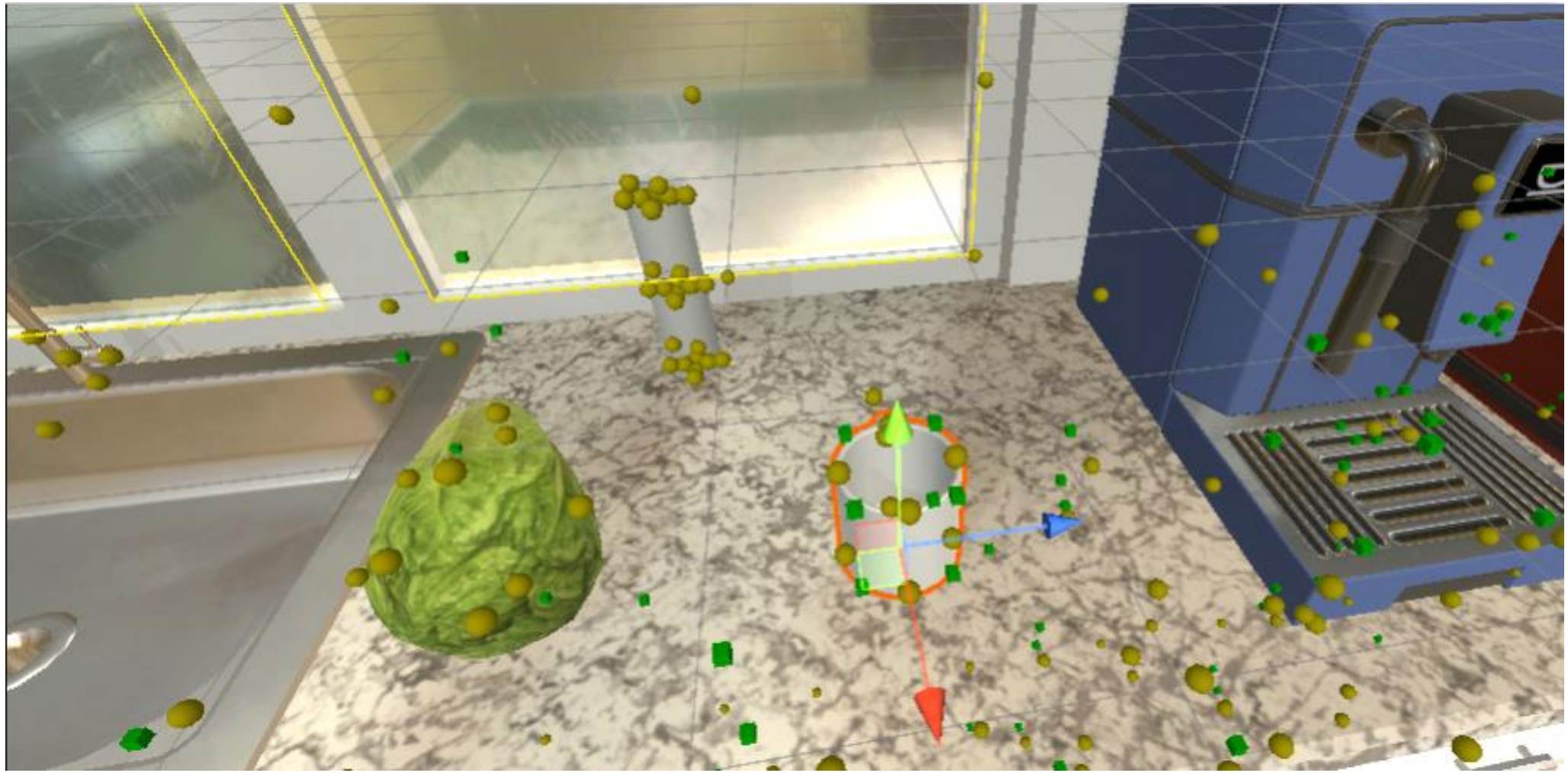
Robot View


Drawer|-01.56|+00.84|-00.20
ButterKnife|-00.41|+01.11|-00.46
Drawer|-01.56|+00.66|-00.20
CounterTop|-00.08|+01.15|00.00
offeeMachine|-01.98|+00.90|-00.1
Tomato|-00.39|+01.14|-00.81
Mug|-01.76|+00.90|-00.62
Lettuce|-01.81|+00.97|-00.94
aperTowelRoll|-02.06|+01.01|-00.8
Dishwasher|-01.63|-00.02|-00.40
CounterTop|-01.87|+00.95|-01.21
Floor|+00.00|+00.00|+00.00
Window|-02.23|+01.43|-01.20
Cabinet|-01.55|+00.50|-01.97
MoveAhead

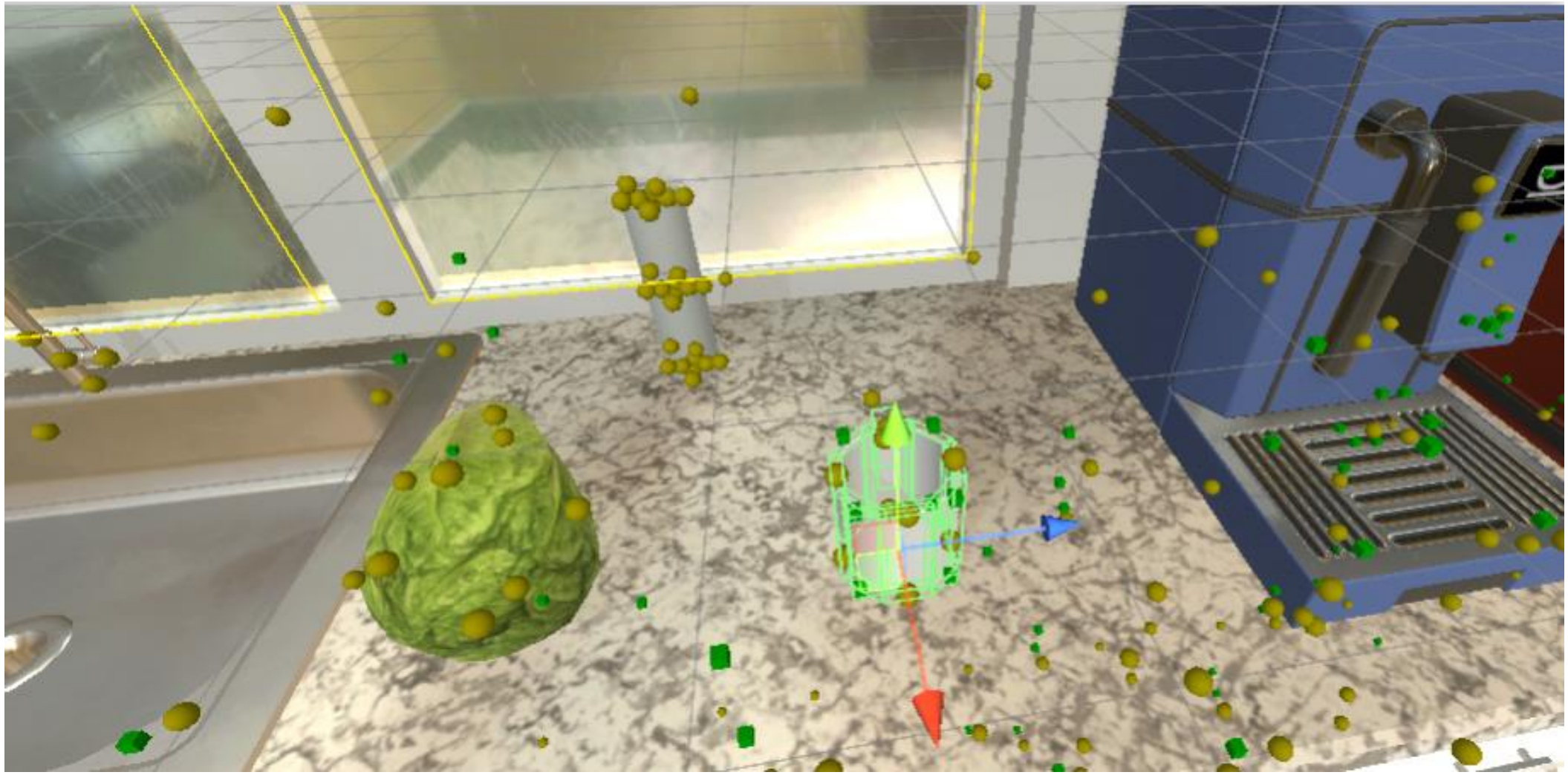
Adding Objects

1. Artwork ($$)
2. Boundaries (boxes, cylinders, spheres, capsules)
3. Triggers (more boundary structures)
4. Receptacle (for objects that can hold other things) (again, more boundary structures)
5. Visible Points (points that surround the surface of the object)
6. Action-Specific components that can reference existing (C#) scripts:
    1. Openable
    2. Cookable
    3. Cleanable

Mesh Layer



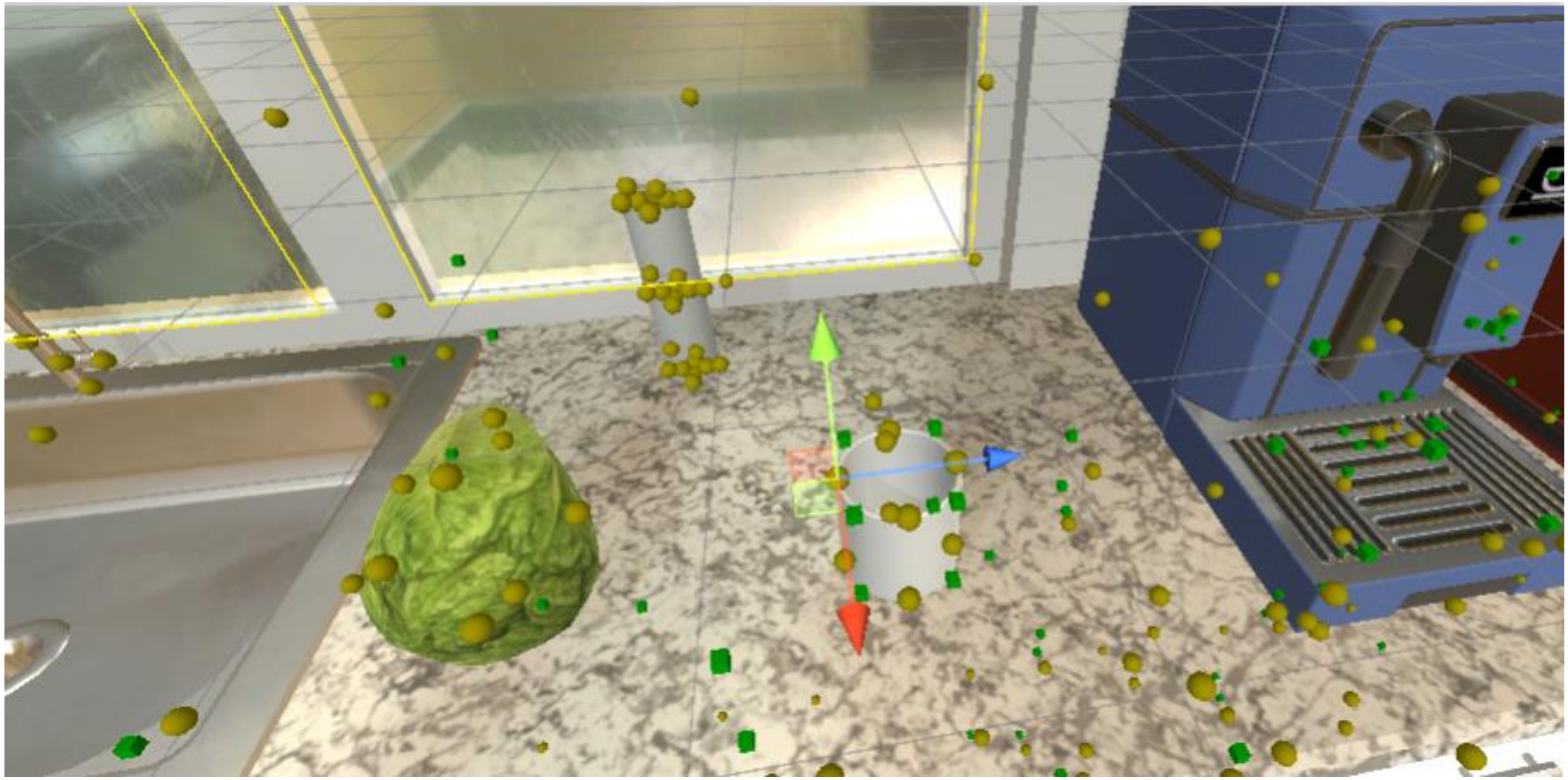
Collisions Layer



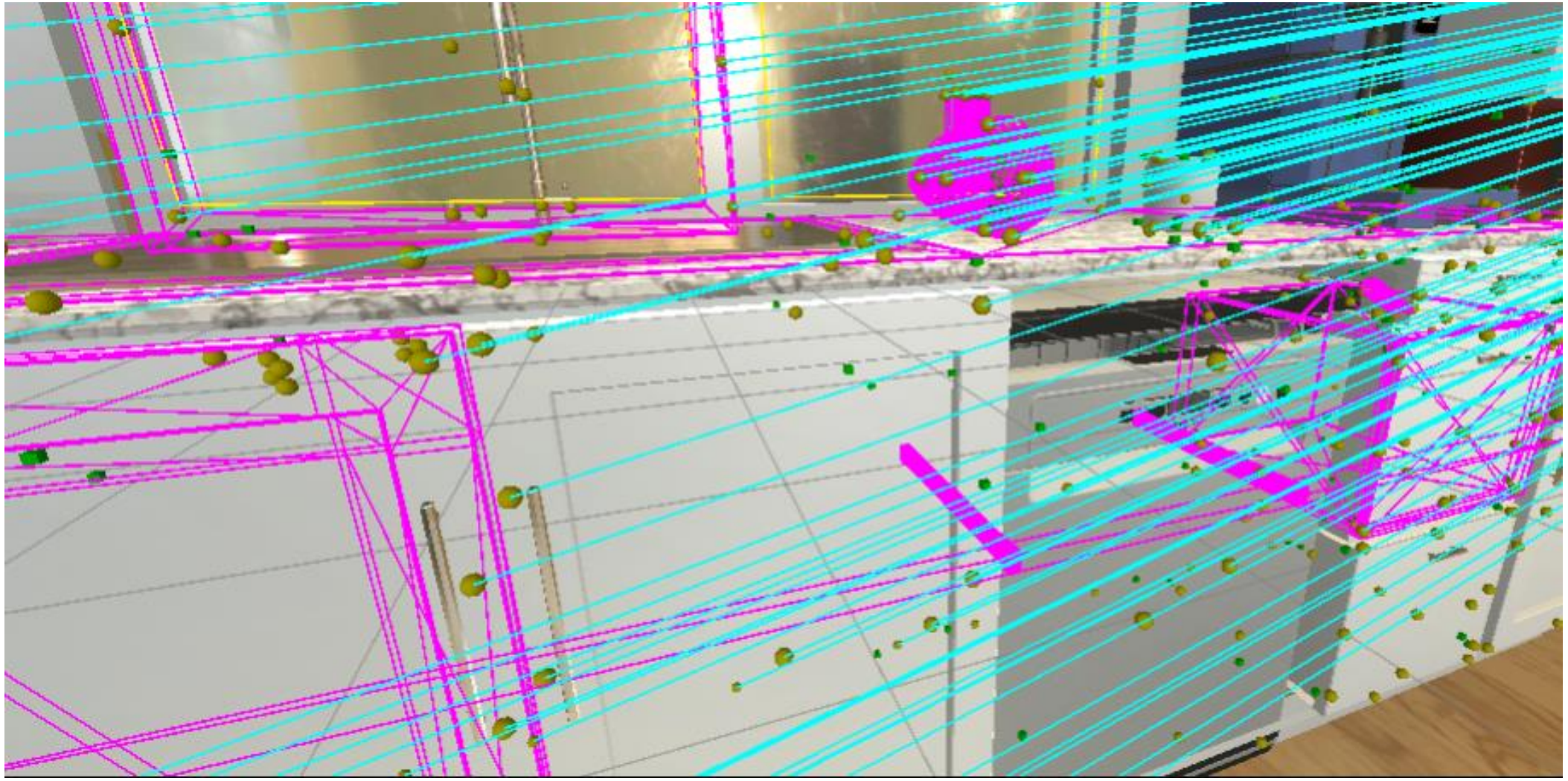
Visibility

- Objects are considered 'visible' if they are close enough to interact with and faced by robot.
- For each visible point, a raycast is created from the robot camera through the point
- If ray intersects boundary box, object is considered 'visible’
- Collision boxes and list of visible points needed to show 'visiblity' for an object.

Visibility Point



Robot Visibility



Acting on Objects

- Action on an Object
- To act on an object, a ServerAction class instance is created; assigned with the ObjectID.
- Can then be processed:
- CleanObject(mySrvrAct);
- CookObject(mySrvrAct);

Nuggets and Coal

- Looks nice
- Interesting planning/interaction domain
- World model granularity fits research goals
- “Faked” world editing is easy

- Difficult build process for forks
- Difficult to cross granularities
- World editing is complex

Thank you!

Questions and comments are welcome