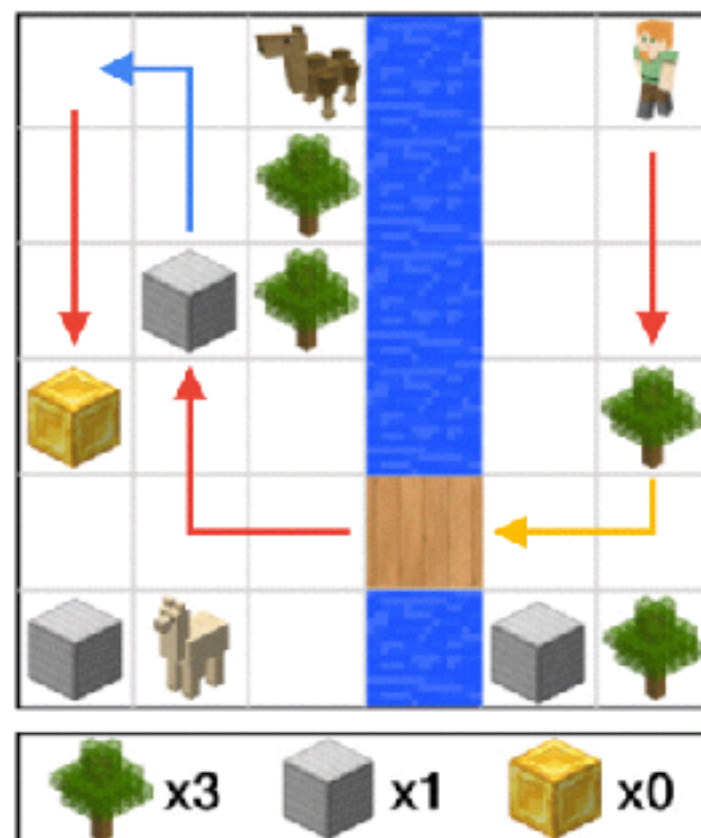


# Program Guided Agent

ICLR 2020 (Spotlight)

## Program

```
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
            place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```



# Shao-Hua Sun



Te-Lin Wu



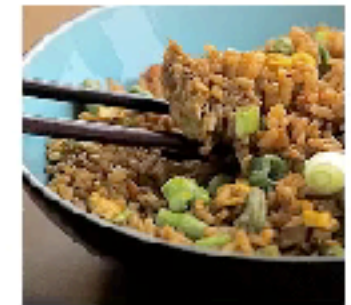
Joseph J. Lim

# Follow an Instruction to Solve a Complex Task

---

## Recipe: cooking fried rice

Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour  $\frac{2}{3}$  cups the whisked eggs into the stir-fried and scramble.



# Follow an Instruction to Solve a Complex Task

---

## Recipe: cooking fried rice

Stir-fry the onions until tender, and repeat this for garlic and carrots, if you have soy sauce, add some. Pour  $\frac{2}{3}$  cups the whisked eggs into the stir-fried and scramble.





# Natural Language Instruction

---

Recipe: cooking fried rice

Stir-fry the onions until tender, and repeat **this** for garlic and carrots, if you have soy sauce, add **some**. Pour **2/3** cups the whisked eggs into the stir-fried and scramble.

## Ambiguities in Language

- **Scoping**
- **Coreferences**
- **Entities**

Bandanau et al. in ICLR 2019

Misra et al. "Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction" in EMNLP 2018

Anderson et al. "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments" in CVPR 2018

Misra et al. "Mapping Instructions and Visual Observations to Actions with Reinforcement Learning" in EMNLP 2017

Hermann et al. "Grounded Language Learning in a Simulated 3D World" in arXiv 2017

# Program

---

Function: cooking fried rice

```
for item in [onions, garlic, carrots]:  
    if is_there("soy sauce"):  
        add("soy sauce", "pot")  
    while not tender(item):  
        stir_fry(item)  
pour(whisked("eggs"), "pot", 0.66)  
scramble("eggs")
```

## Advantages of Programs

- Explicit scoping
- Resolved Coreferences
- Resolved Entities

# Problem Formulation

---

## Program

```
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
            place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

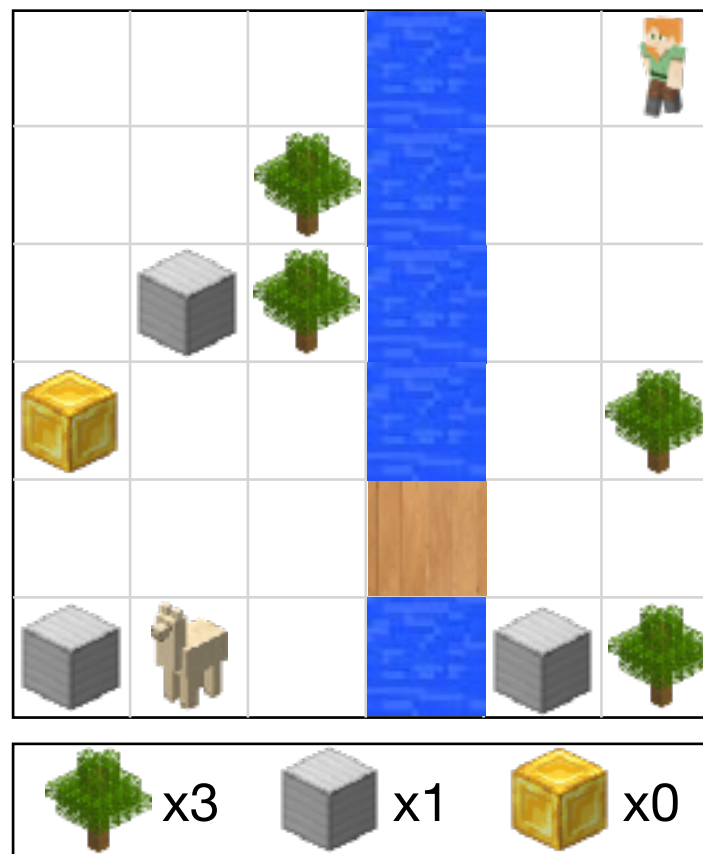
# Problem Formulation

---

## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron]<3:  
            mine(Iron)  
            place(Iron, 1, 1)  
        else:  
            goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

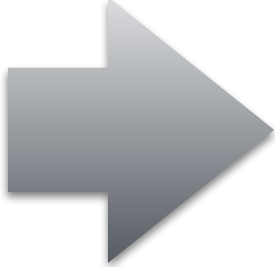
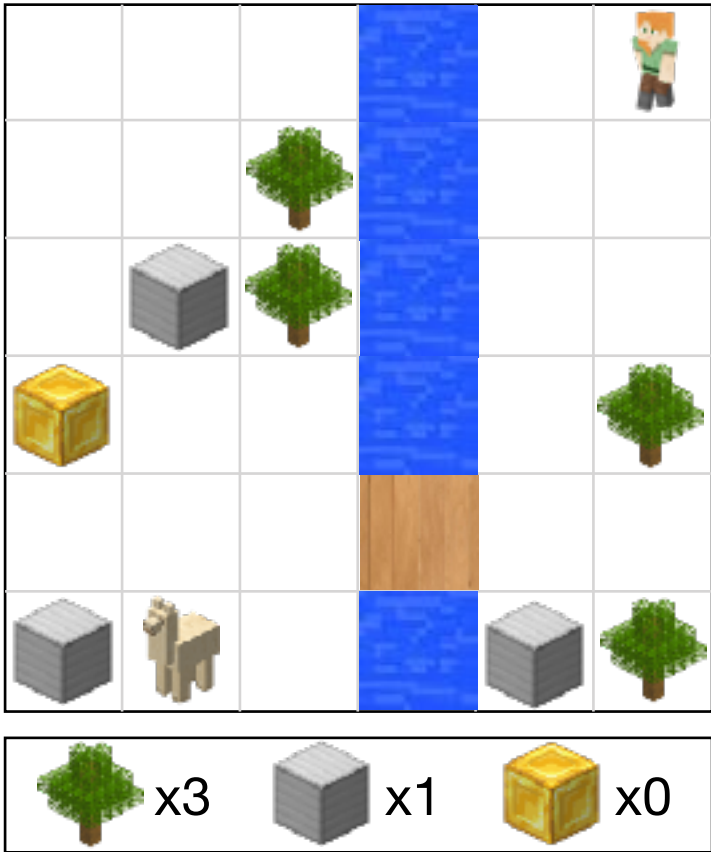
## State



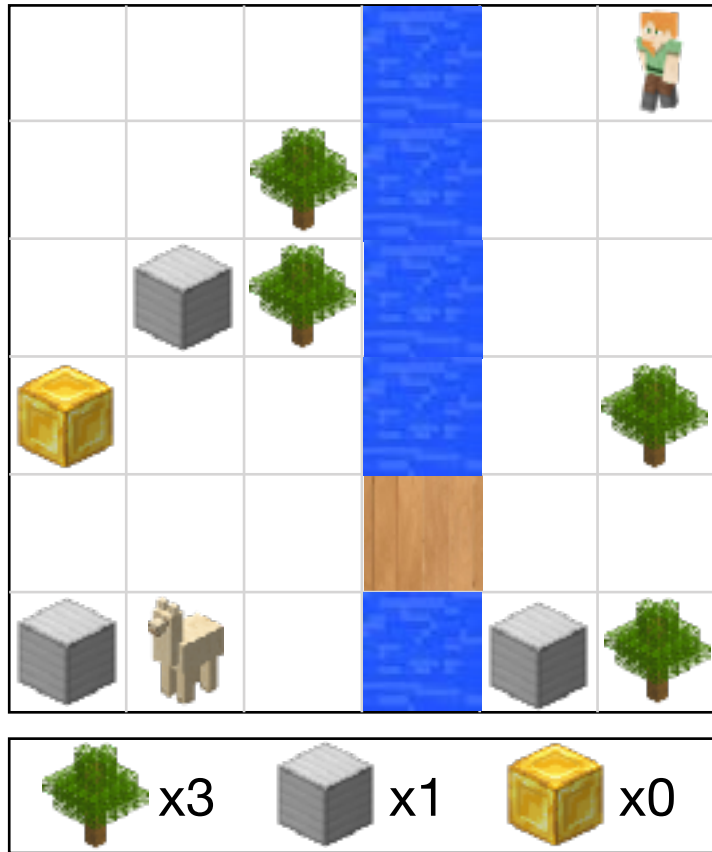
# Program

```
def run():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron]<3:
            mine(Iron)
            place(Iron, 1, 1)
    else:
        goto(4, 2)
    while env[Gold]>0:
        mine(Gold)
```

# State



# Execution



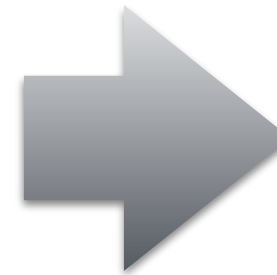
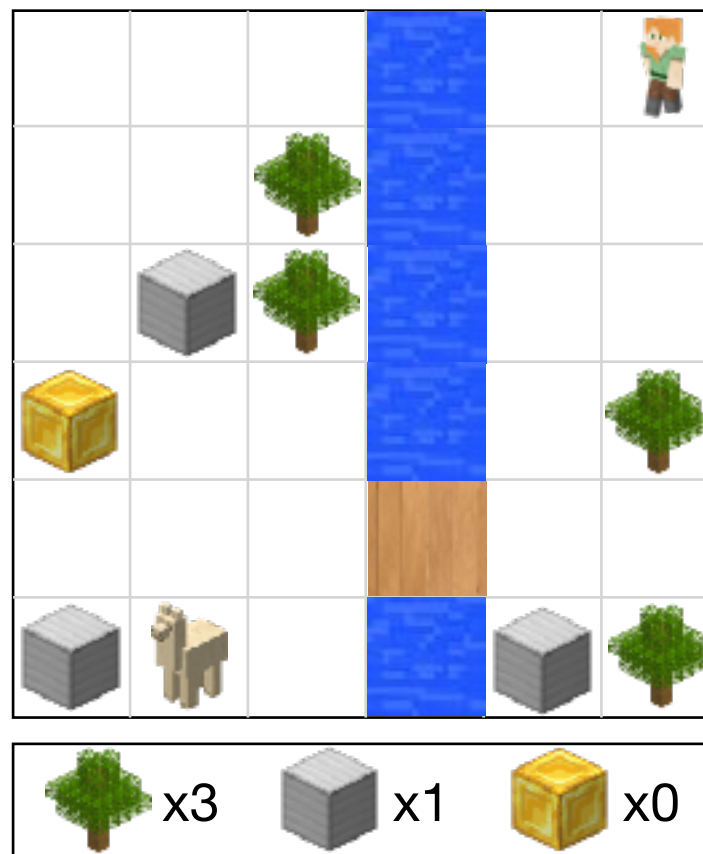


# Problem Formulation

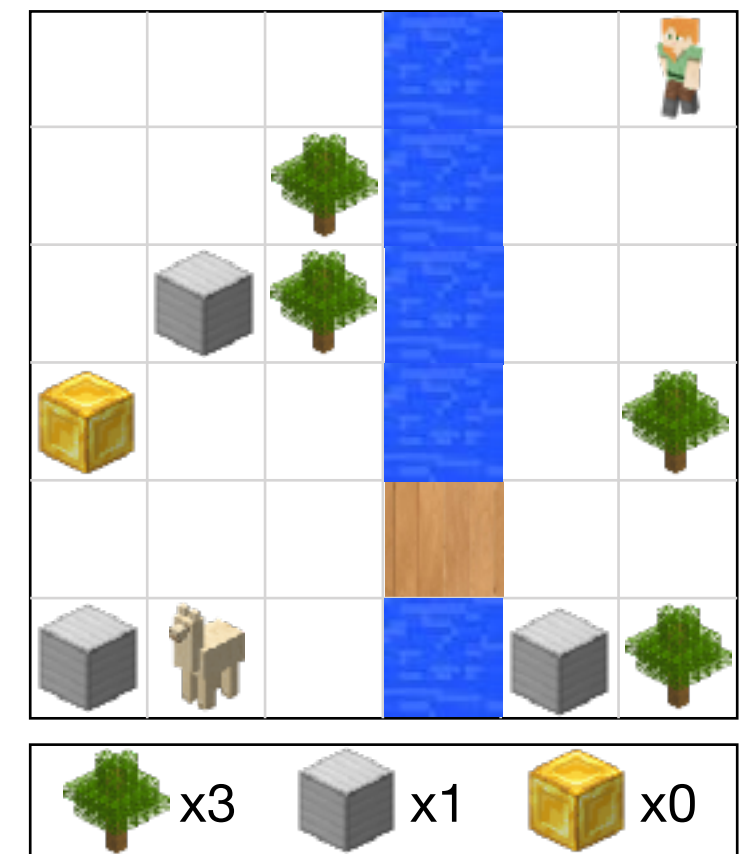
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron]<3:  
            mine(Iron)  
            place(Iron, 1, 1)  
        else:  
            goto(4, 2)  
            while env[Gold]>0:  
                mine(Gold)
```

## State



## Execution

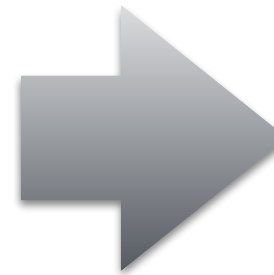
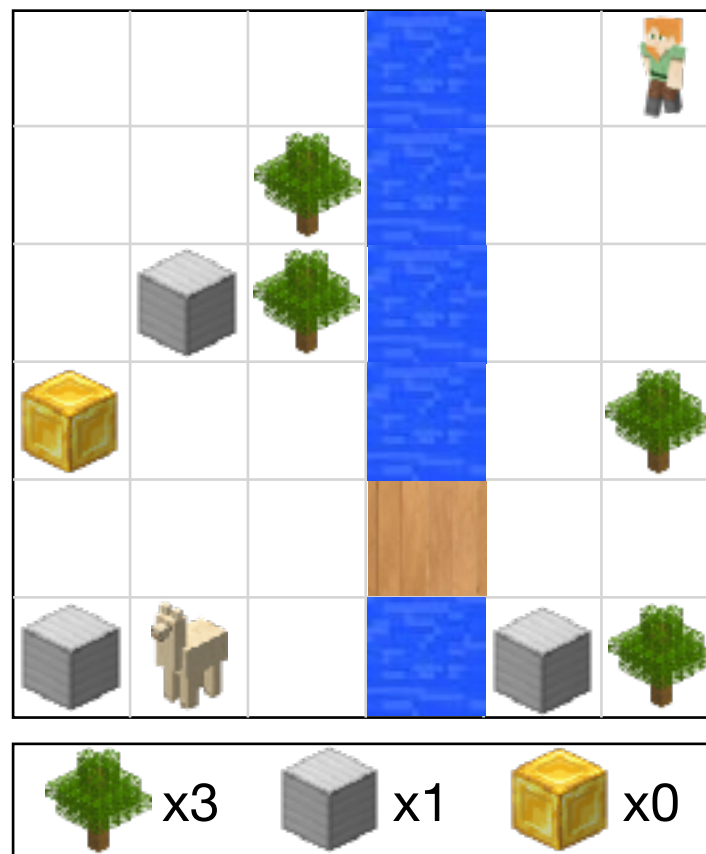


# Problem Formulation

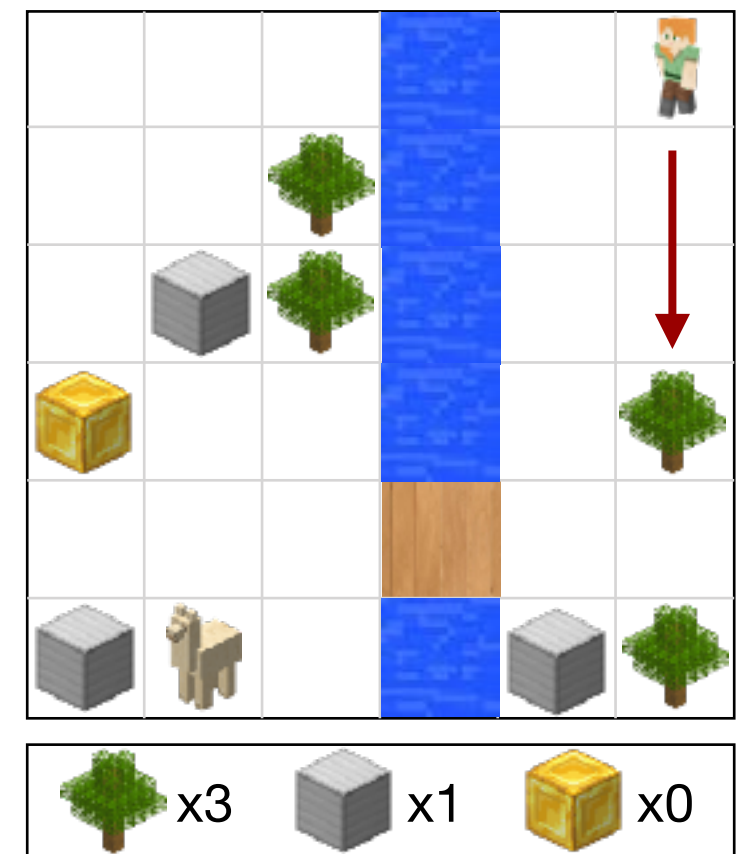
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
    if agent[Iron]<3:  
        mine(Iron)  
        place(Iron, 1, 1)  
    else:  
        goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

## State



## Execution

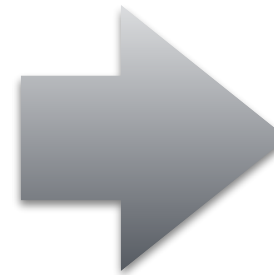
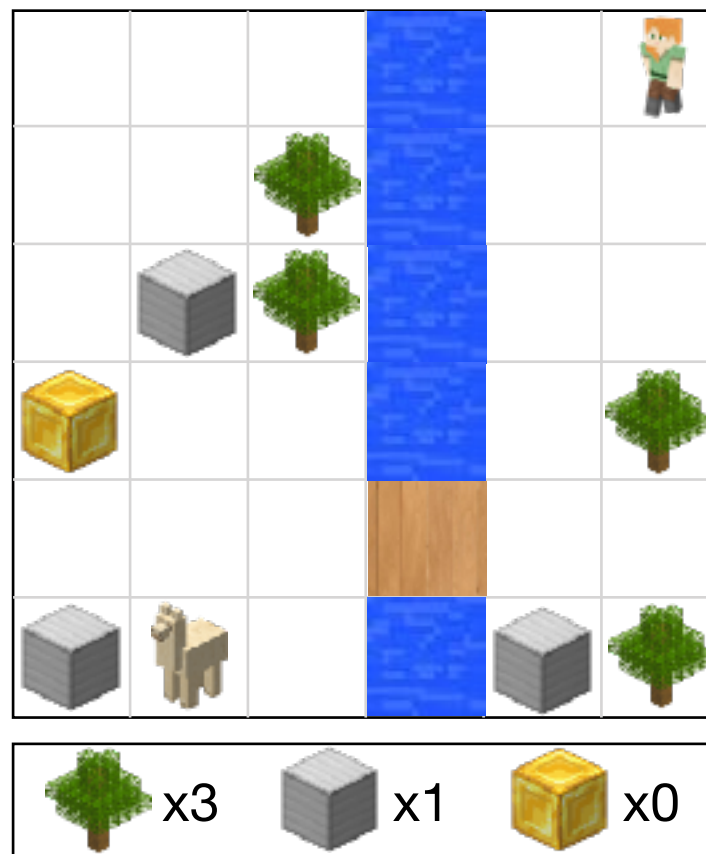


# Problem Formulation

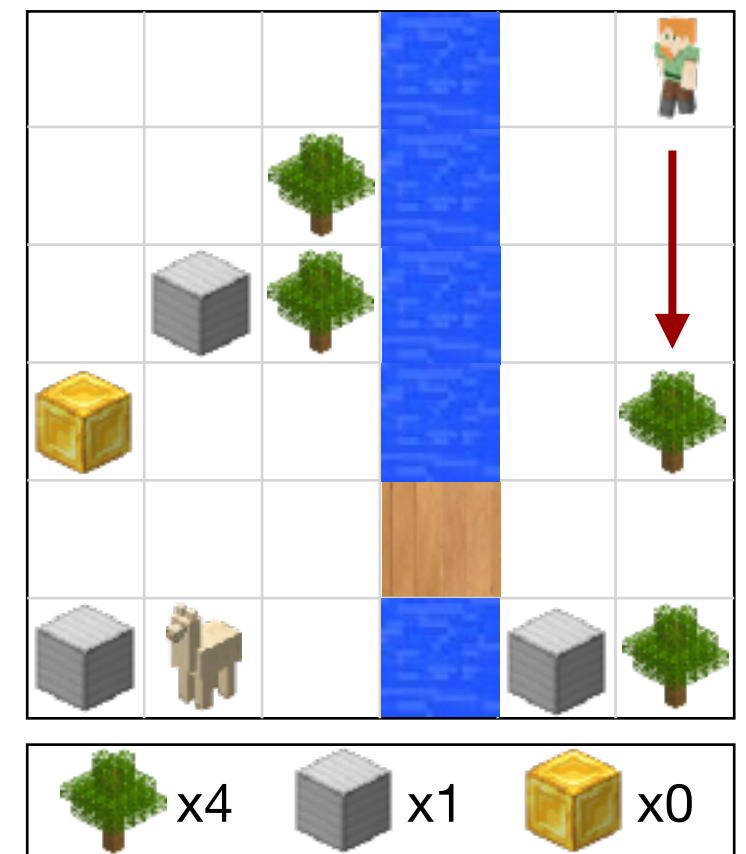
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
    if agent[Iron]<3:  
        mine(Iron)  
        place(Iron, 1, 1)  
    else:  
        goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

## State



## Execution

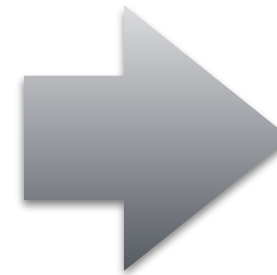
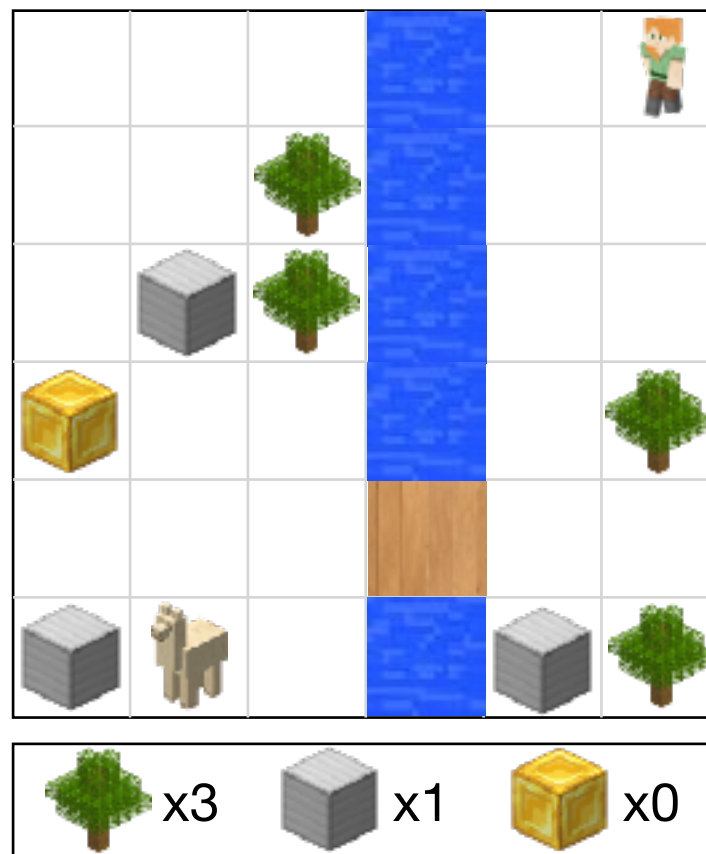


# Problem Formulation

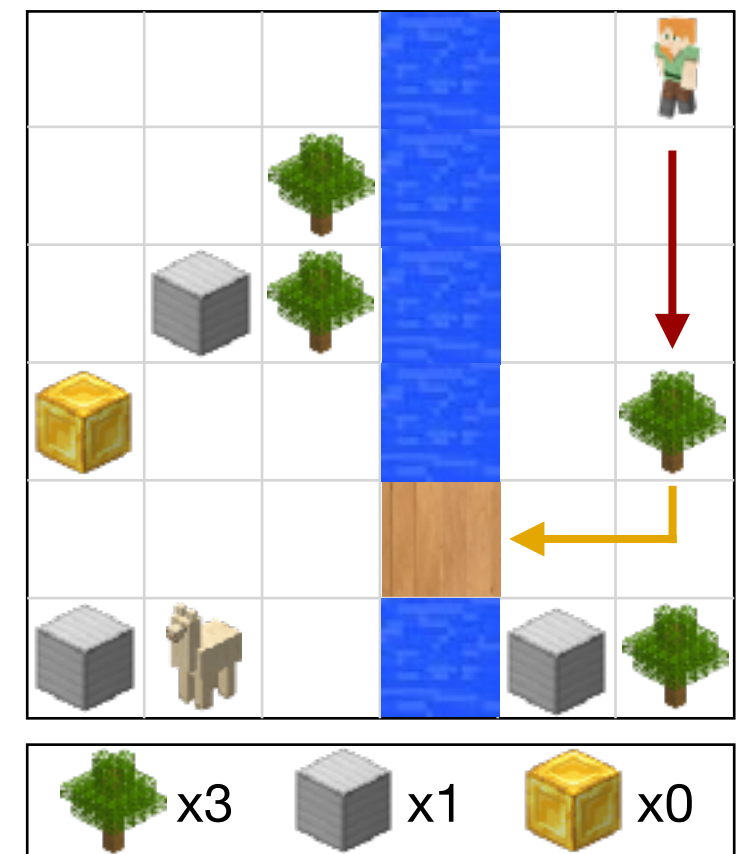
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
    if agent[Iron]<3:  
        mine(Iron)  
        place(Iron, 1, 1)  
    else:  
        goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

## State



## Execution

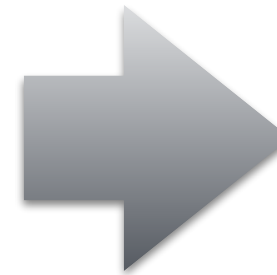
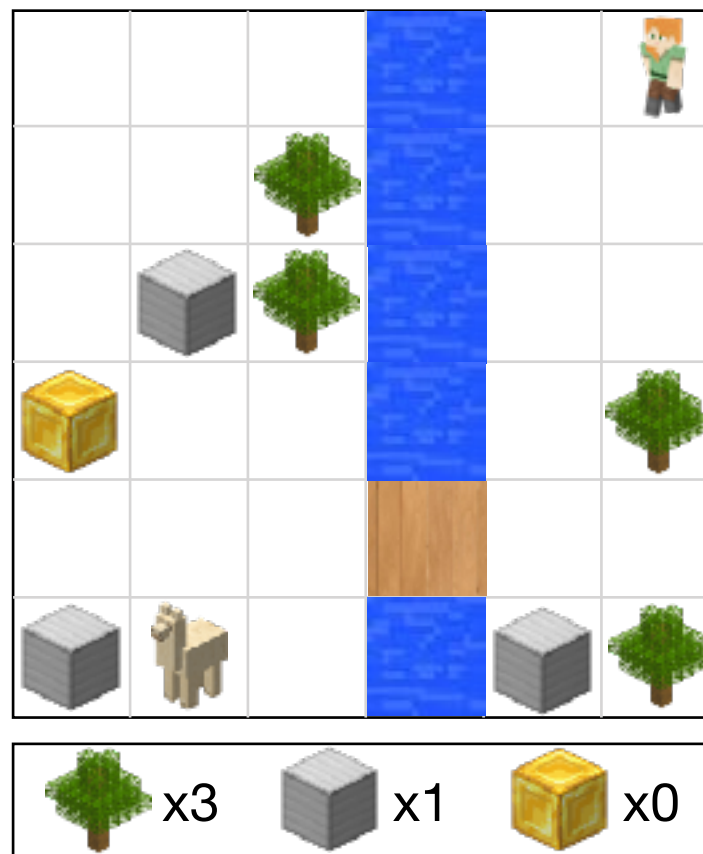


# Problem Formulation

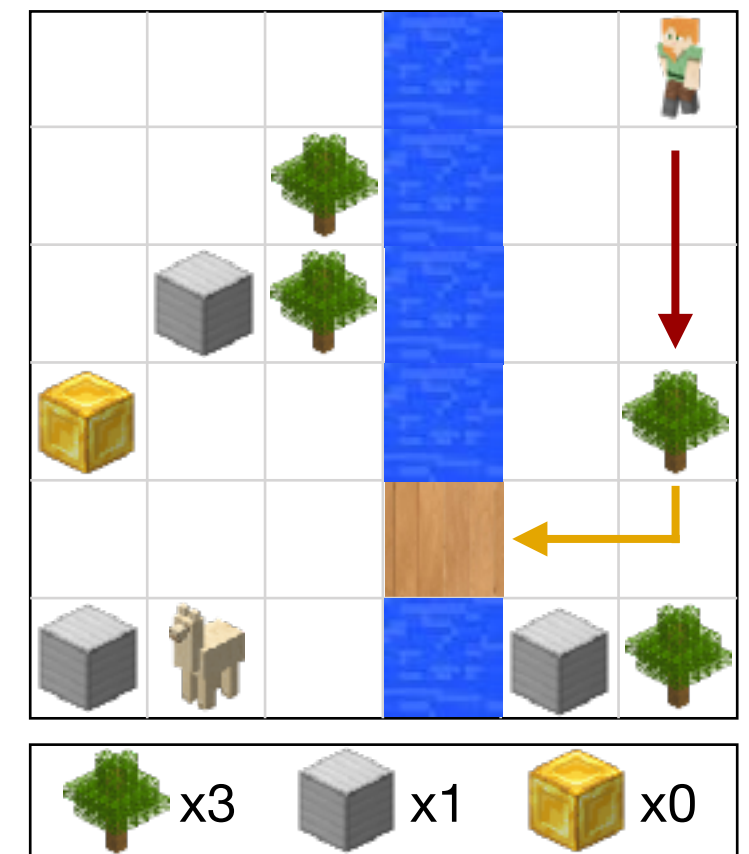
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron]<3:  
            mine(Iron)  
            place(Iron, 1, 1)  
        else:  
            goto(4, 2)  
            while env[Gold]>0:  
                mine(Gold)
```

## State



## Execution



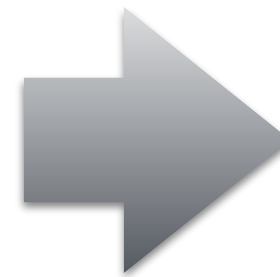
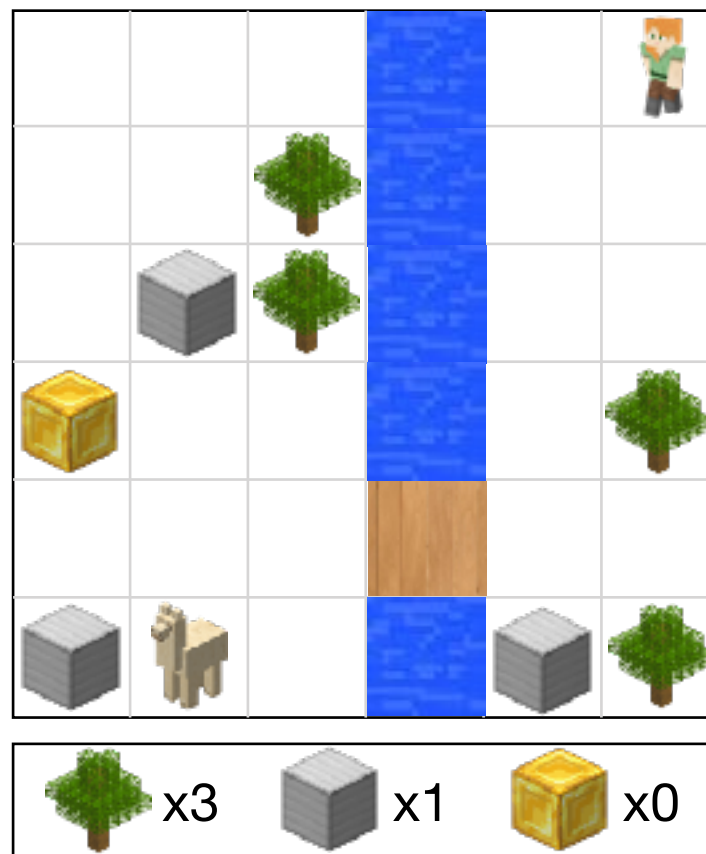


# Problem Formulation

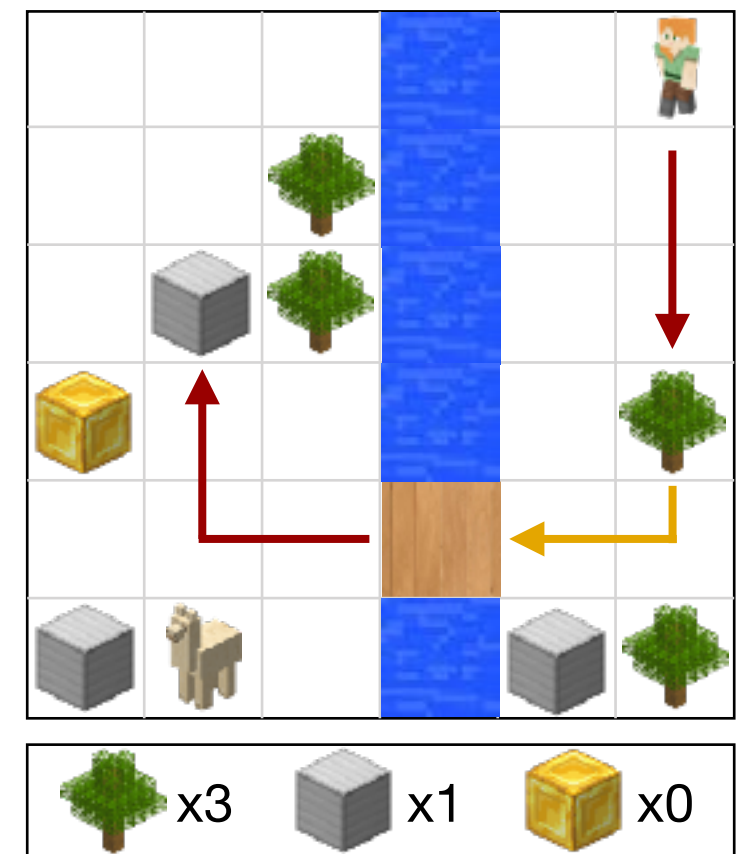
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron]<3:  
            mine(Iron)  
            place(Iron, 1, 1)  
        else:  
            goto(4, 2)  
            while env[Gold]>0:  
                mine(Gold)
```

## State



## Execution

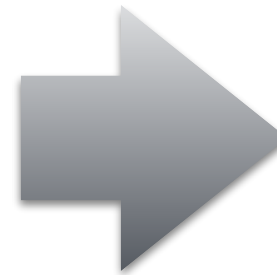
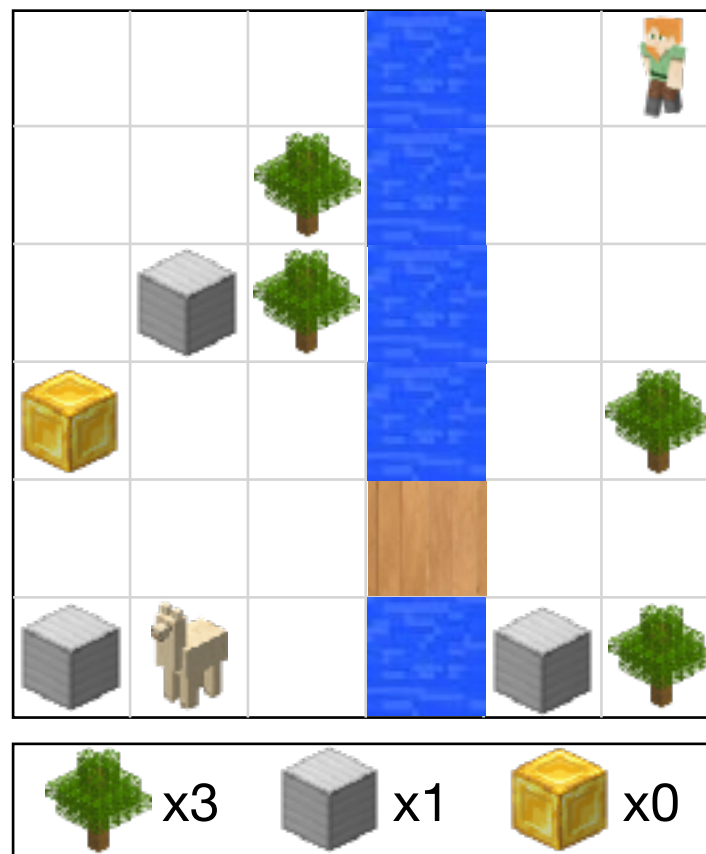


# Problem Formulation

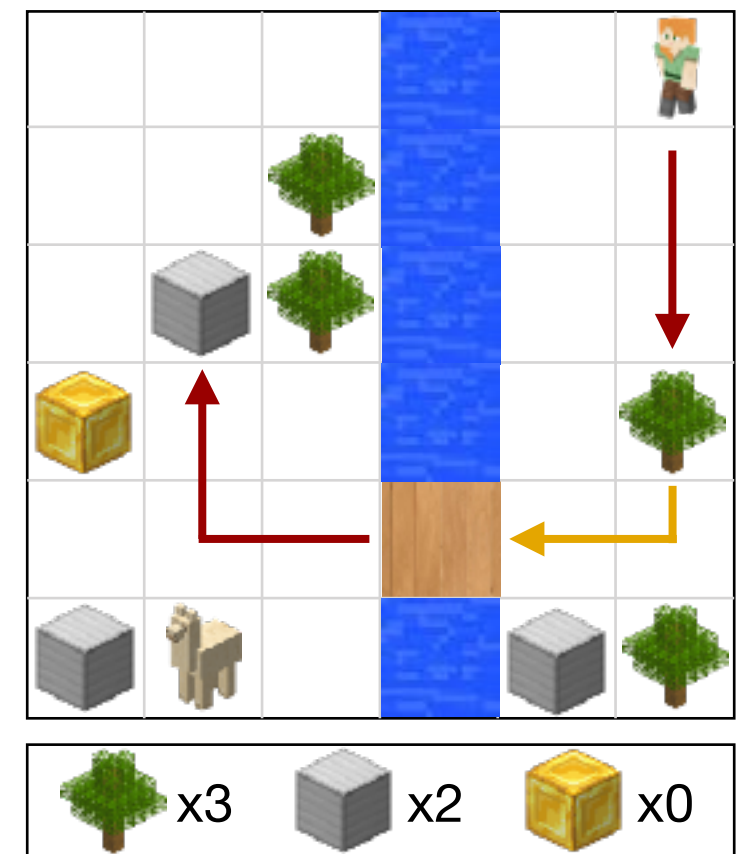
Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron]<3:  
            mine(Iron)  
            place(Iron, 1, 1)  
        else:  
            goto(4, 2)  
            while env[Gold]>0:  
                mine(Gold)
```

State



Execution

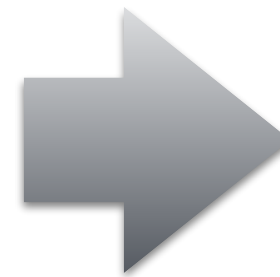
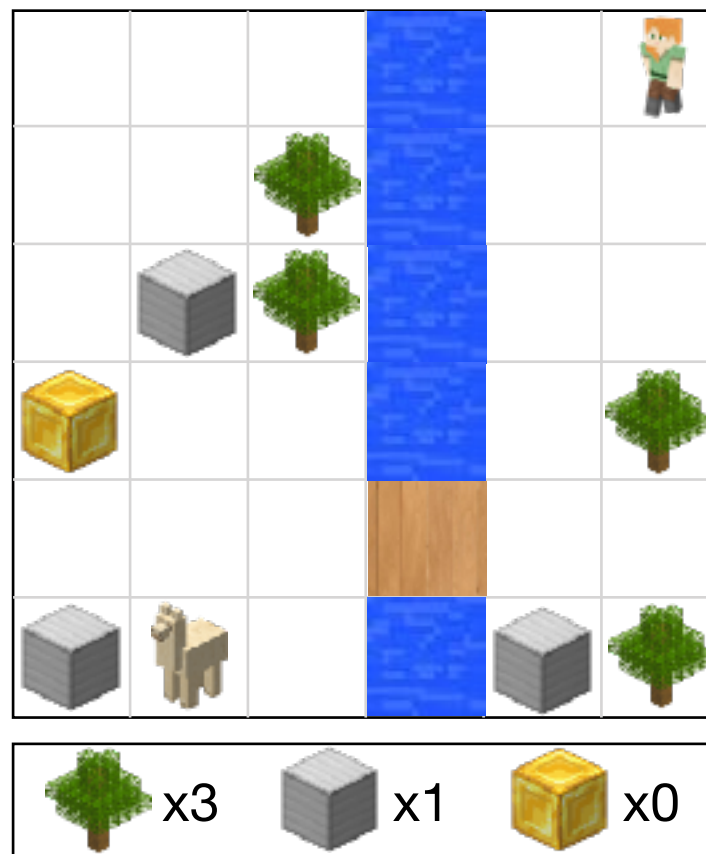


# Problem Formulation

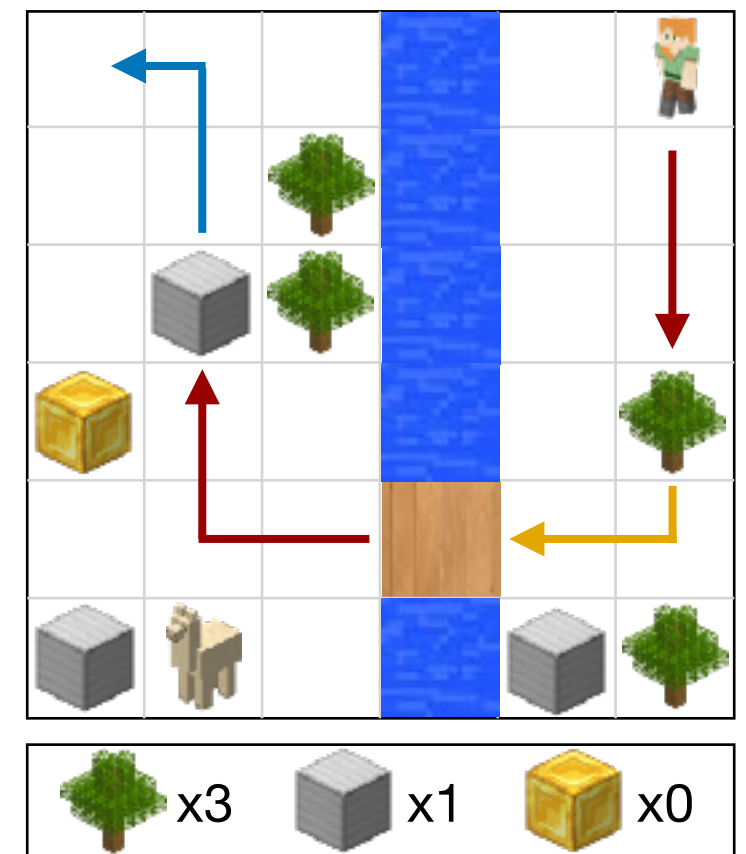
## Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron]<3:  
            mine(Iron)  
            place(Iron, 1, 1)  
        else:  
            goto(4, 2)  
            while env[Gold]>0:  
                mine(Gold)
```

## State



## Execution

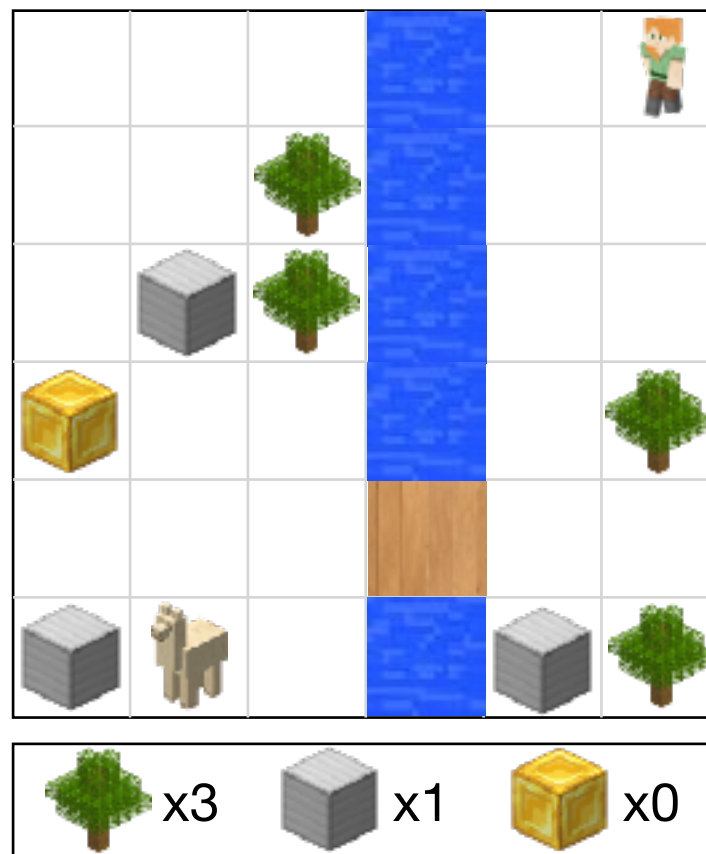


# Problem Formulation

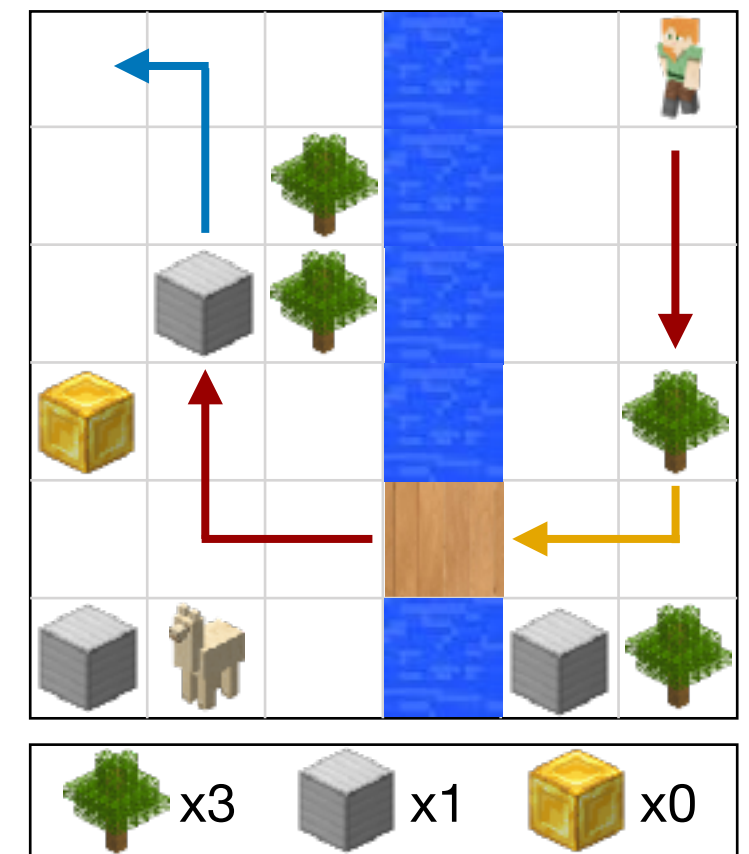
Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
    if agent[Iron]<3:  
        mine(Iron)  
        place(Iron, 1, 1)  
    else:  
        goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

State



Execution

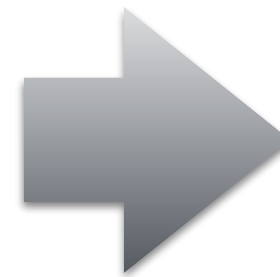
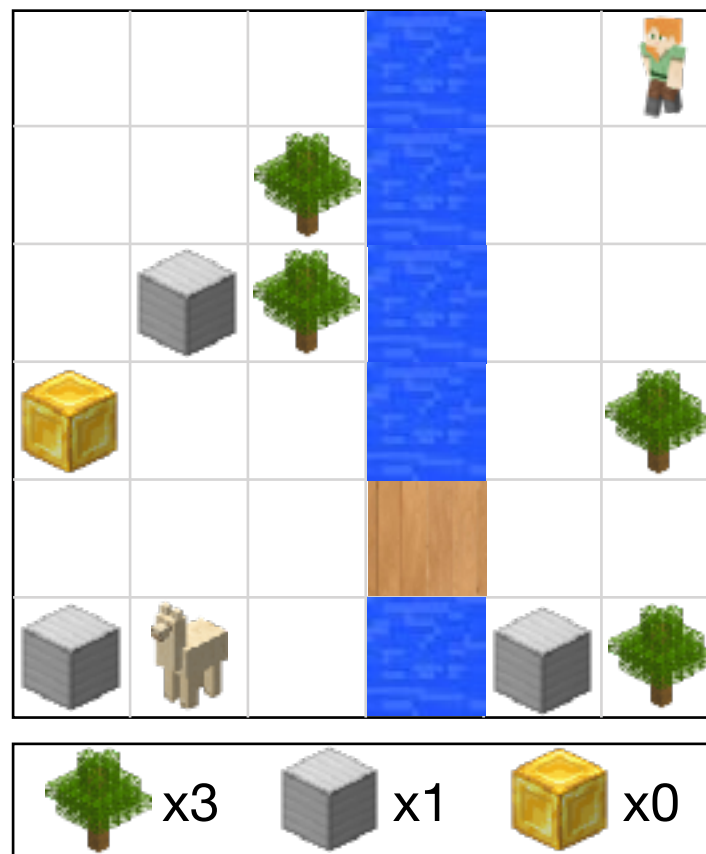


# Problem Formulation

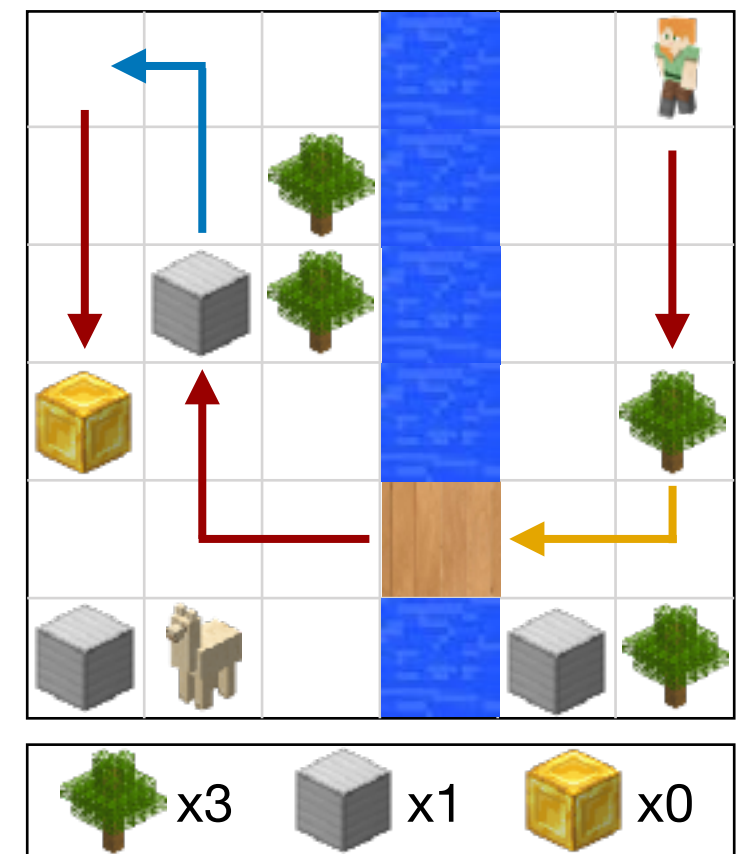
Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
    if agent[Iron]<3:  
        mine(Iron)  
        place(Iron, 1, 1)  
    else:  
        goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

State



Execution



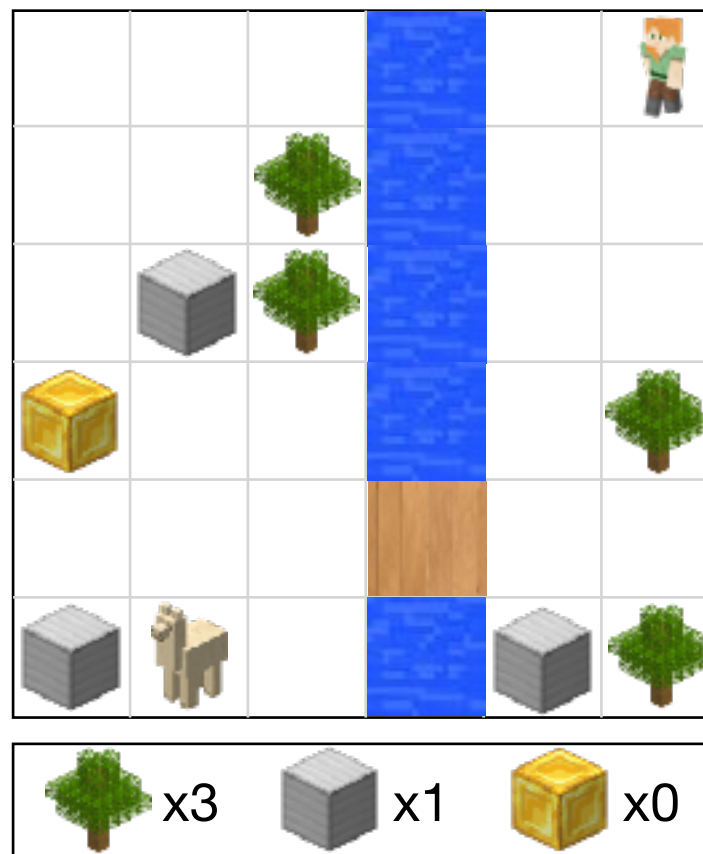


# Problem Formulation

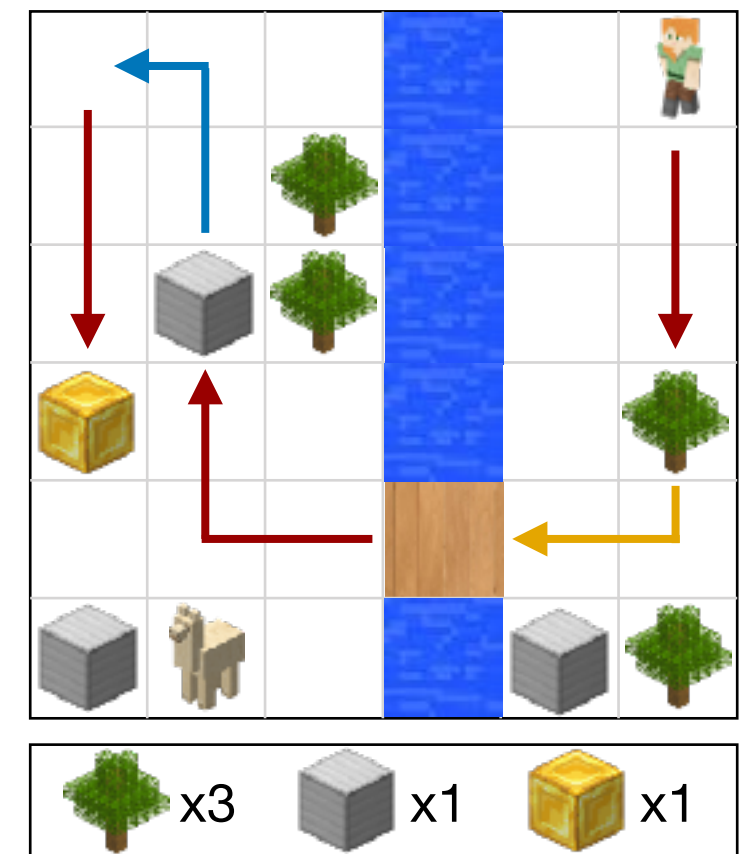
Program

```
def run():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
    if agent[Iron]<3:  
        mine(Iron)  
        place(Iron, 1, 1)  
    else:  
        goto(4, 2)  
    while env[Gold]>0:  
        mine(Gold)
```

State



Execution



# Exemplar Instructions

---

## Programs

```
def Task():
    if is_there[River]:
        mine(Wood)
        build_bridge()
        if agent[Iron] < 3:
            mine(Iron)
            place(Iron, 2, 3)
        else:
            goto(4, 2)
        while env[Gold] > 0 :
            mine(Gold)
```

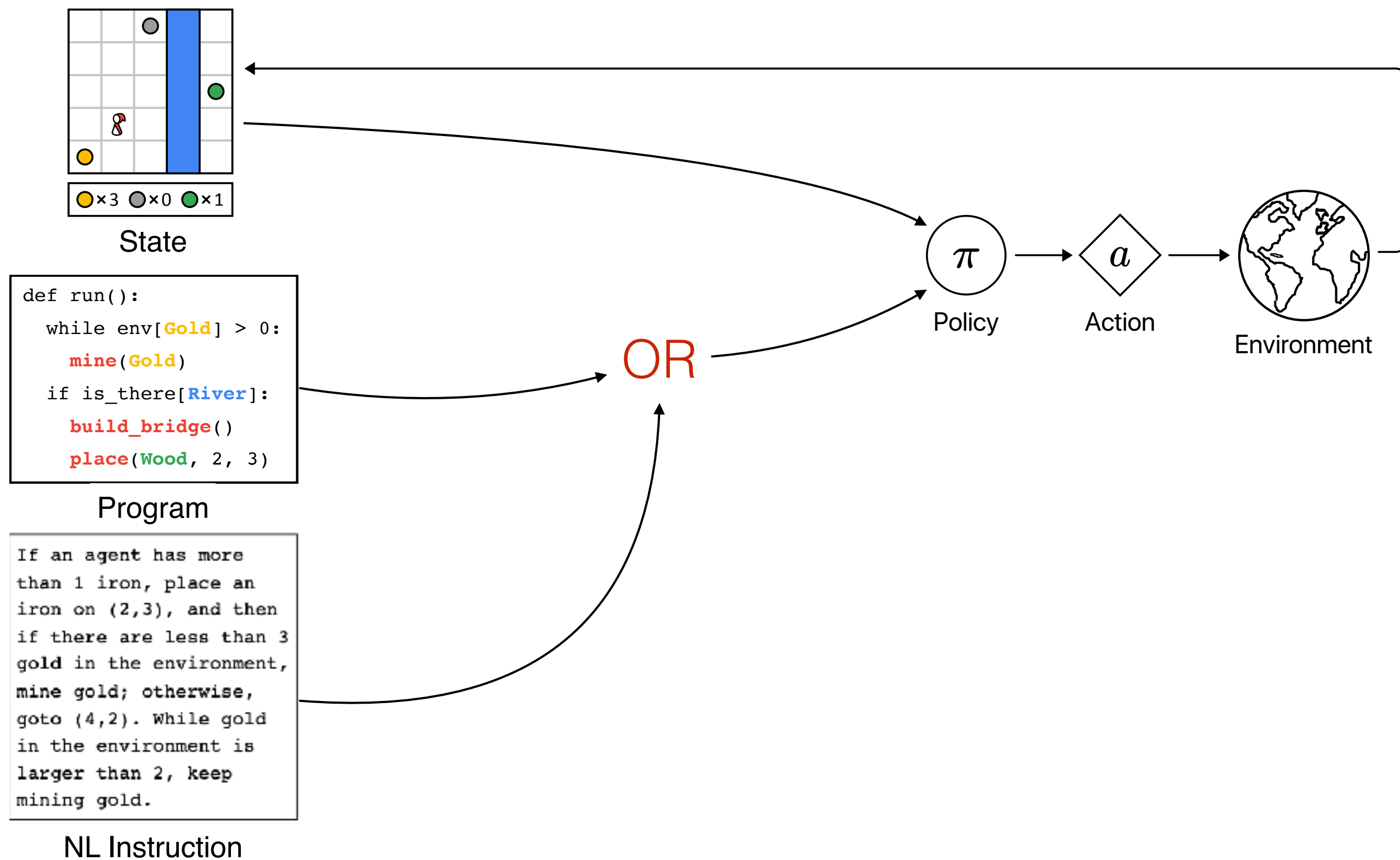
```
def Task():
    if is_there[River]:
        build_bridge()
        place(Gold, 3, 4)
    if agent[Gold] == 13:
        while agent[Gold] <= 12:
            place(Gold, 8, 3)
        if agent[Iron] >= 8:
            place(Wood, 2, 4)
        elif env[Gold] <= 10:
            sell(Iron)
```

## Natural Language Instructions

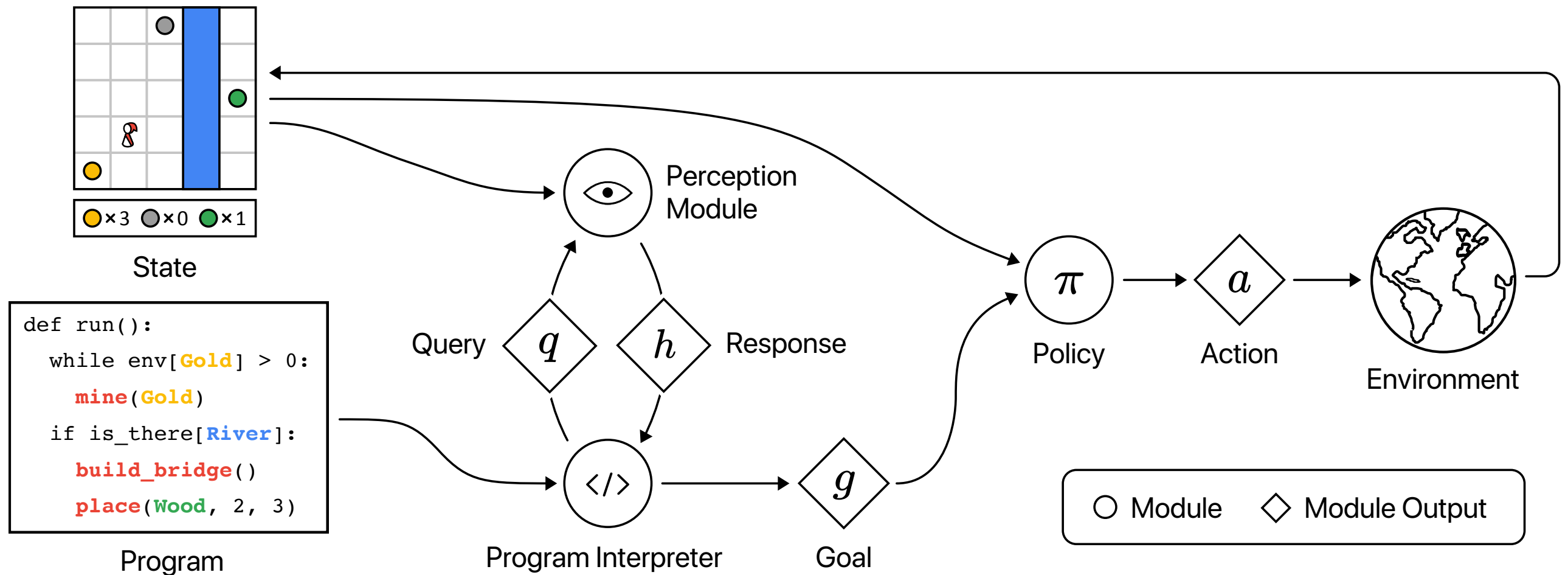
If a river is in the environment, mine a wood and then use it to build a bridge. And then if agent has less than there iron, place an iron at (2,3). Otherwise if no river, goto location (4,2). Finally, whenever there's still gold in the environment, mine a gold.

While agent has no more than 11 wood, place wood at (2,4) and iron at (1,1), then place iron at (8,5) and mine gold twice, then mine gold. After the preceding procedure, sell gold and sell iron 4 times.

# End-to-end Learning Baseline

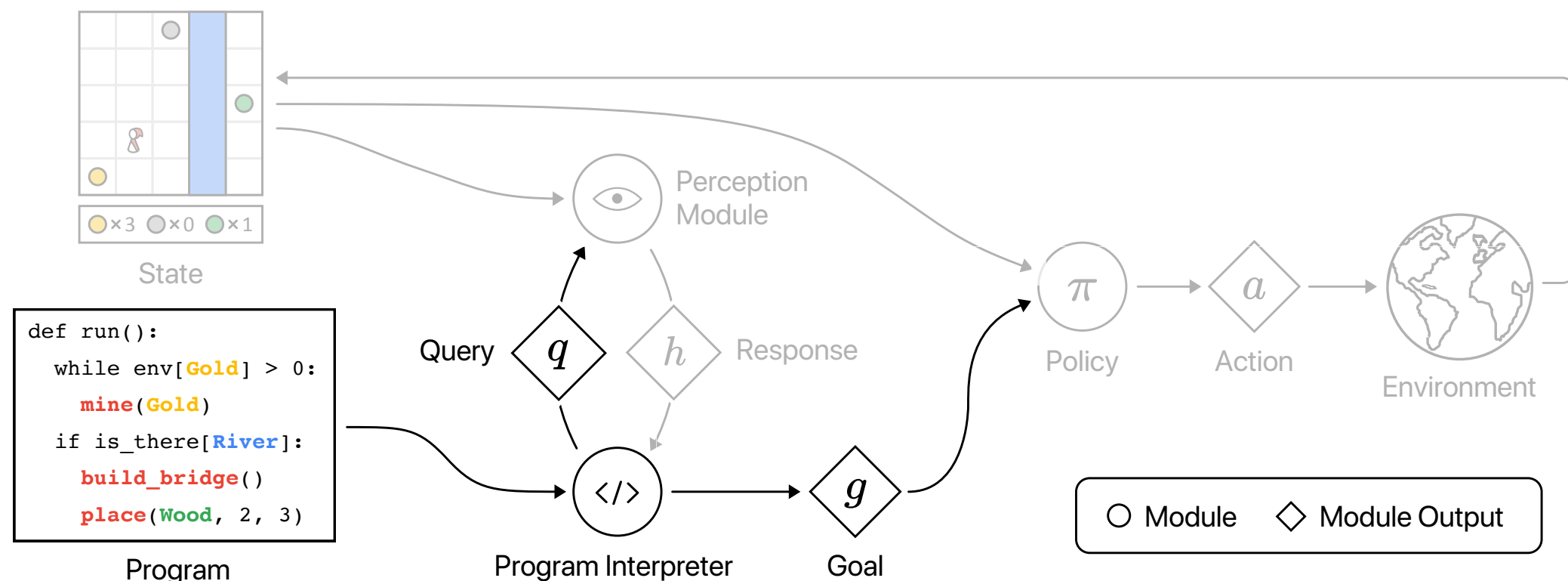


# Program Guided Agent



# Program Interpreter

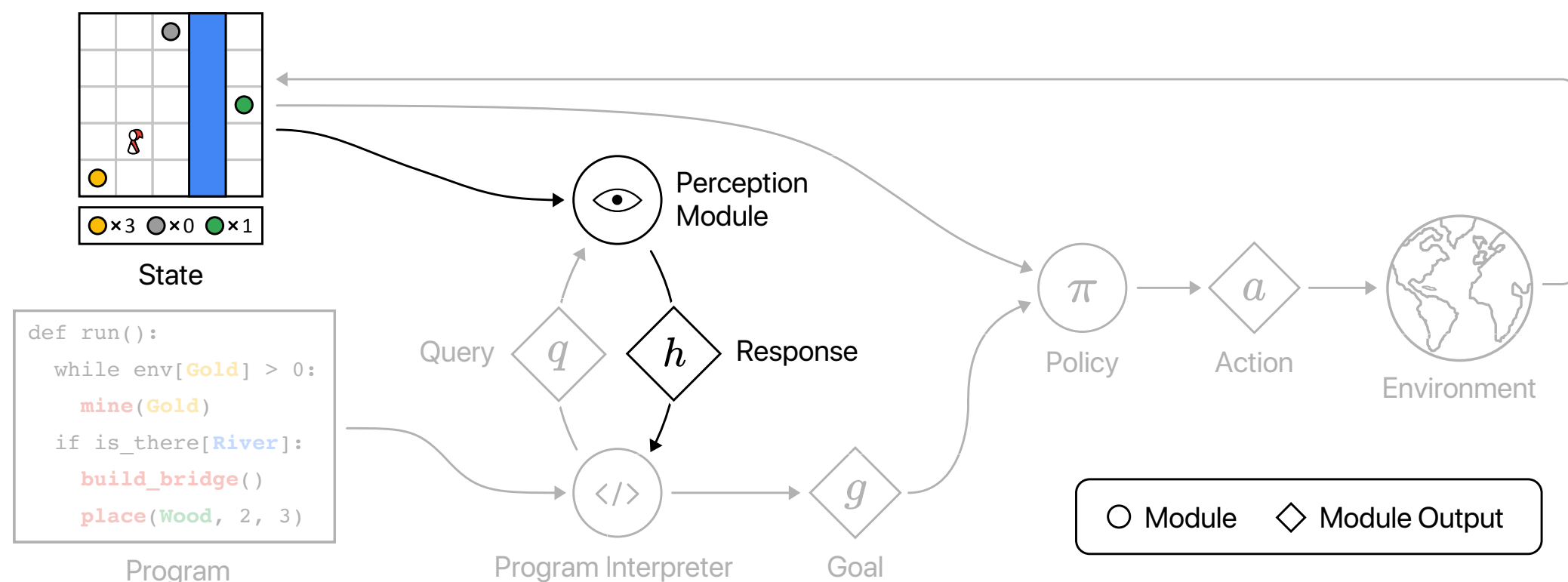
- Comprehend a given program to 3 categories:
  - **Subtasks (actions):** what agent should perform
  - **Perception:** information from the environment
  - **Control flow:** decide to call different subtasks according to perceived information





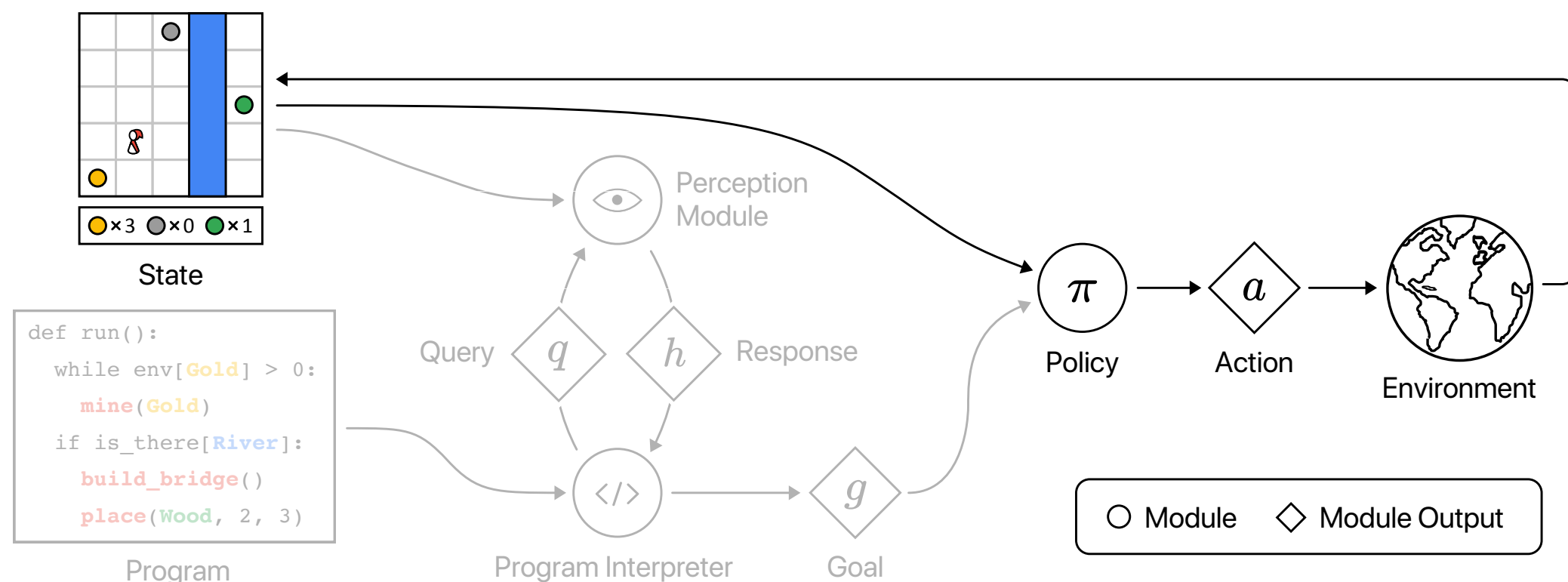
# Perception Module

- Extract environmental information for choosing a path in a program
- **Input**
  - **Query**: a symbolically represented query (e.g. `is_there[River]`)
  - **State s**: environment map and agent inventory status
- **Output**
  - Predicted **answer** to the query (e.g. True/False)



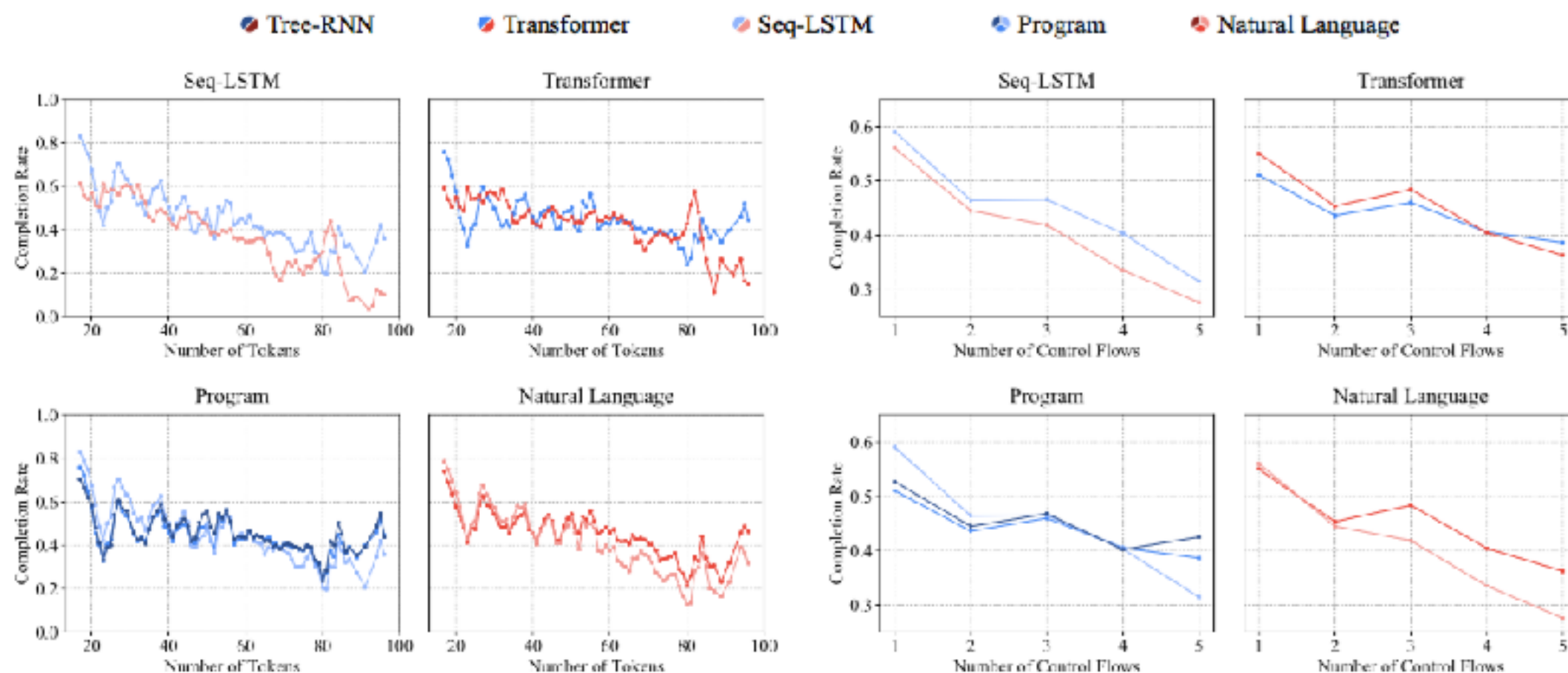
# Policy

- Take low-level actions in the environment for fulfilling a subtask
- **Input**
  - Symbolically represented subtask (goal)  $g$
  - State  $s$
- **Output**
  - Predicted action distribution



# Result

Instruction Method		Natural language descriptions		Programs			
		Seq-LSTM	Transformer	Seq-LSTM	Tree-RNN	Transformer	Ours (concat)
Dataset	test	$54.9 \pm 1.8\%$	$52.5 \pm 2.6\%$	$56.7 \pm 1.9\%$	$50.1 \pm 1.2\%$	$49.4 \pm 1.6\%$	$88.6 \pm 0.8\%$
	test-complex	$32.4 \pm 4.9\%$	$38.2 \pm 2.6\%$	$38.8 \pm 1.2\%$	$42.2 \pm 2.4\%$	$40.9 \pm 1.5\%$	$91.8 \pm 0.2\%$
Generalization gap		40.9%	27.2%	31.6%	15.8%	17.2%	3.8%



(a) Instruction Length

(b) Instruction Complexity

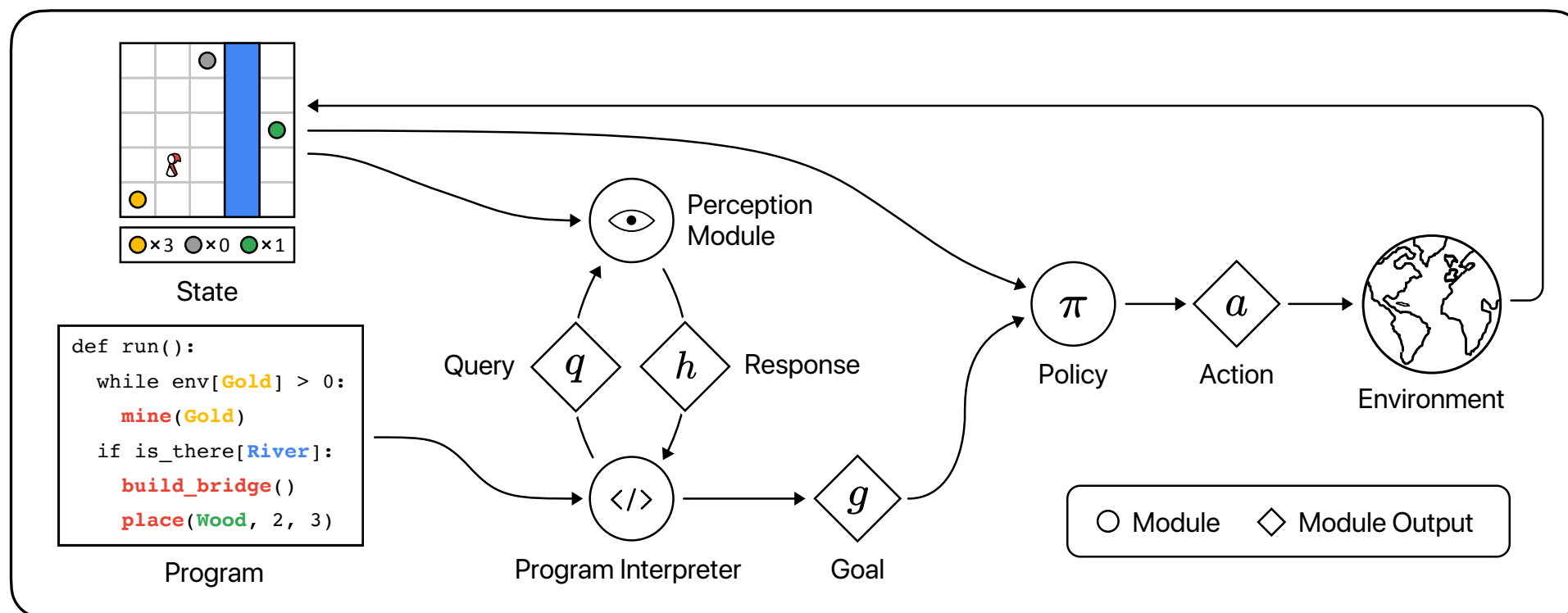
# Conclusion

- Specific tasks using **programs**

Program

```
def Task():  
    if is_there[River]:  
        mine(Wood)  
        build_bridge()  
        if agent[Iron] < 3:  
            mine(Iron)  
            place(Iron, 2, 3)  
        else:  
            goto(4, 2)  
    while env[Gold] > 0:  
        mine(Gold)
```

- Leverage the structure of programs with a modular framework

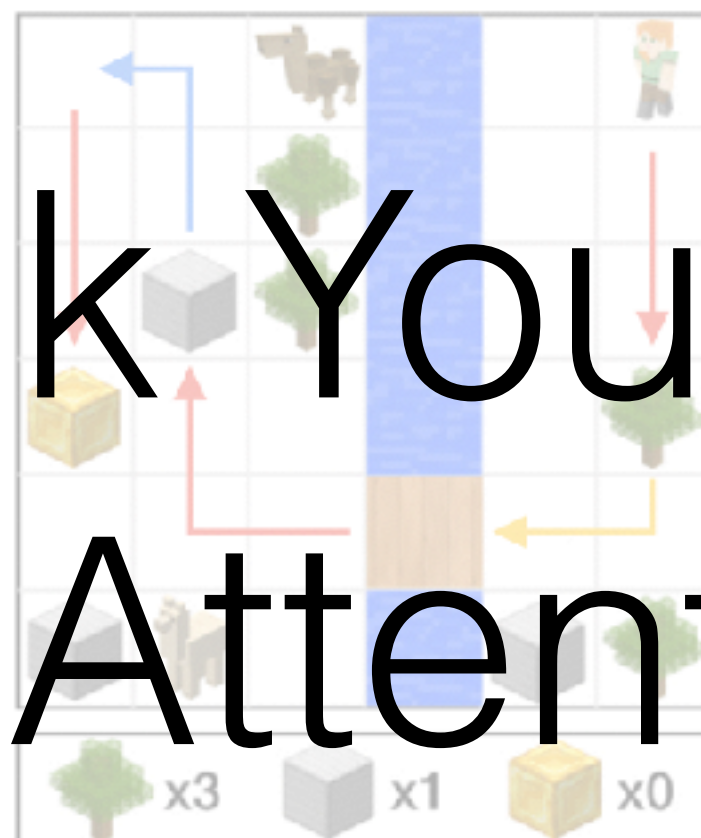


# Program Guided Agent

ICLR 2020 (Spotlight)

## Program

# Thank You



Shao-Hua Sun

Te-Lin Wu

Joseph J. Lim