

# Supplementary Material: Enabling End Users to Program Robots Using Reinforcement Learning

## Contents

### 1 User Interface

- 1.1 Blocks Used For the Different Programming Paradigms
- 1.2 *Sequential Programming*
- 1.3 *Trigger-Action Programming*
- 1.4 *Full MDP Programming*
- 1.5 *Goal-Only MDP Programming*

### 2 Tasks Participants Were Asked to Complete

### 3 Tutorial

- 3.1 *Sequential Programming*
  - 3.1.1 Part 1: Explanation of *Sequential Programming*
  - 3.1.2 Part 2: *Sequential Programming* Interactive Component
- 3.2 *Trigger-Action Programming*
  - 3.2.1 Part 1: Explanation of *Trigger-Action Programming*
  - 3.2.2 Part 2: *Trigger-Action Programming* Interactive Component
- 3.3 *Full MDP Programming*
  - 3.3.1 Part 1: Explanation of *Full MDP Programming*
  - 3.3.2 Part 2: *Full MDP Programming* Interactive Component
- 3.4 *Goal-Only MDP Programming*
  - 3.4.1 Part 1: Explanation of *Goal-Only MDP Programming*
  - 3.4.2 Part 2: *Goal-Only MDP Programming* Interactive Component

### 4 Survey Instrument

- 4.1 Pre-experiment Survey
- 4.2 Task Follow-up Survey
- 4.3 End of Experiment Survey

### 5 Program Correctness for Each Task

### 6 Types of Errors Made in Programming Paradigms

- 6.1 Incorrect Solution Analysis
- 6.2 Errors Made by Participants in Each Programming Paradigm
- 6.3 Common Types of Errors Made in *Sequential Programming*
- 6.4 Common Types of Errors Made in *Trigger-Action Programming*
- 6.5 Common Types of Errors Made in *Full MDP Programming*
- 6.6 Common Types of Errors Made in *Goal-Only MDP Programming*

# 1 User Interface

We present an end-user programming interface we designed that allows users to author programs that instruct a simulated robot to complete tasks in an everyday environment — the home. In this simple home environment, the user can program a TIAGo mobile manipulator robot to complete tasks as a household assistant. The home we designed has four rooms: kitchen, bedroom, playroom, and porch. Depending on the tasks, the environment includes a person and a range of everyday objects (e.g., toys, a cup of coffee, mail).

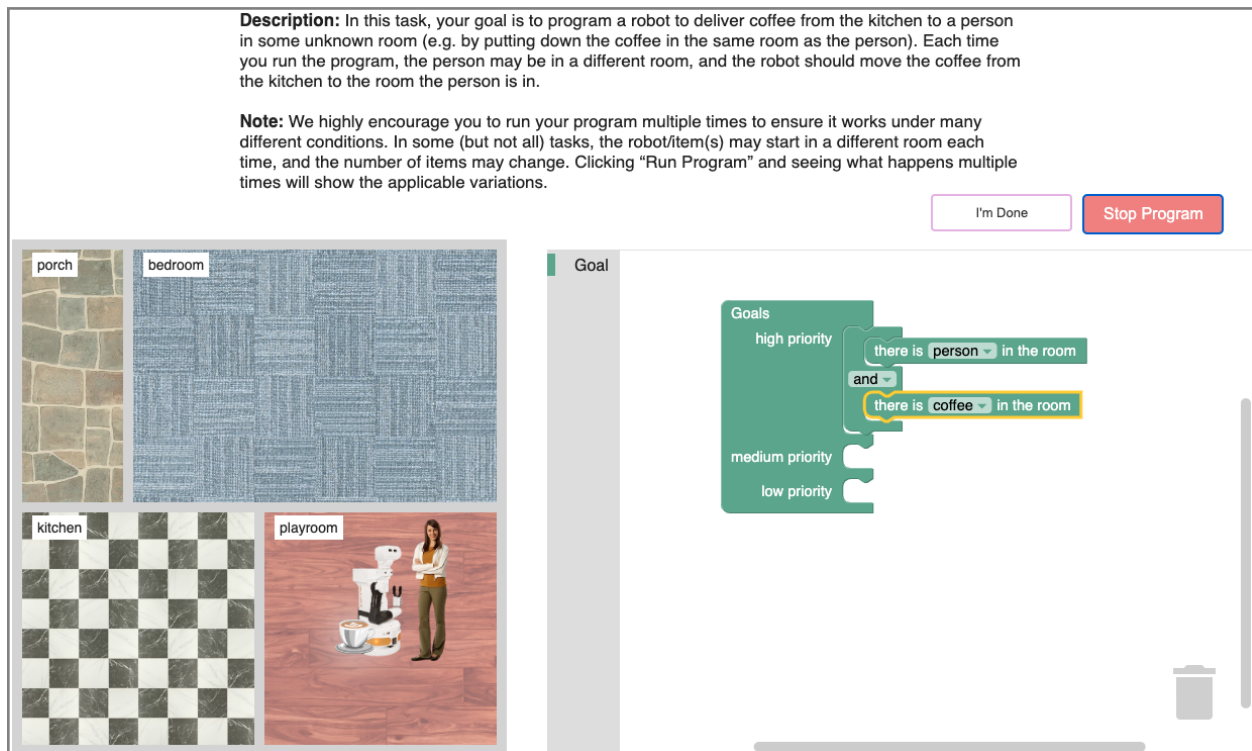


Figure 1: Corresponding to Figure 1 from the body of the paper, this is a larger screenshot of our graphical interface that also shows the task description as participants saw them. Here, the participant is programming the robot to deliver coffee to them (Task 4) using the *Goal-MDP* paradigm.

## 1.1 Blocks Used For the Different Programming Paradigms

Table 1: We present the different types of blocks participants used in each paradigm to solve *Task 4: Coffee Delivery*. We list the interfaces that share the same blocks.

Paradigm(s)	Block(s)
<b>Actions</b>	
<i>Seq</i> , <i>TAP</i> , <i>Full-MDP</i> , <i>Goal-MDP</i>	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="border: 1px solid black; padding: 2px; background-color: #d9ead3; border-radius: 5px;">go to a random room</div> <div style="border: 1px solid black; padding: 2px; background-color: #d9ead3; border-radius: 5px;">go to the <span style="border: 1px solid black; padding: 0 2px;">kitchen</span></div> <div style="border: 1px solid black; padding: 2px; background-color: #d9ead3; border-radius: 5px;">pick up <span style="border: 1px solid black; padding: 0 2px;">the coffee</span></div> </div> <div style="border: 1px solid black; padding: 2px; background-color: #d9ead3; border-radius: 5px; margin-top: 5px;">put down the <span style="border: 1px solid black; padding: 0 2px;">coffee</span></div>
<b>Goals</b>	

Full-MDP,  
Goal-MDP

there is person in the room

there is coffee in the room

---

**States**

---

Seq, TAP

there is person in the room

there is NO person in the room

there is coffee in the room

Full-MDP

Are my hands empty?

Am I in the bedroom ?

Am I in the porch ?

Am I in the kitchen ?

Am I in the playroom ?

Is there a coffee in the current room?

---

**Events**

---

TAP

I arrived at the kitchen

the program started

I picked up an item

I put down an item

I arrived at any room

---

**Organization Structure**

---

Seq

when the program starts

TAP

Rules

Full-MDP

Actions

States

Full-MDP,  
Goal-MDP

Goals  
high priority  
medium priority  
low priority

---

**Controls**

---

Seq

repeat while  
do

if  
do  
else

TAP

if  
then

if  
and  
then

*Seq, TAP*



*Full-MDP,  
Goal-MDP*



## 1.2 Sequential Programming

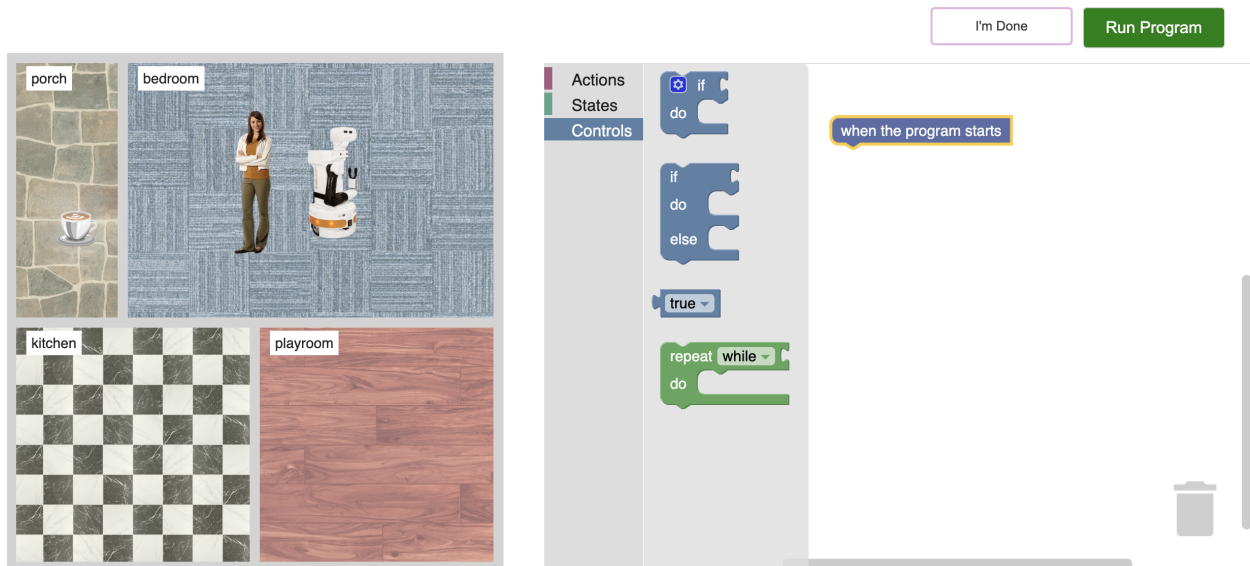


Figure 2: The graphical interface in Blockly for *Sequential Programming (Seq)* with the Control blocks displayed. Here, the user is programming the robot to deliver coffee to them (Task 4).

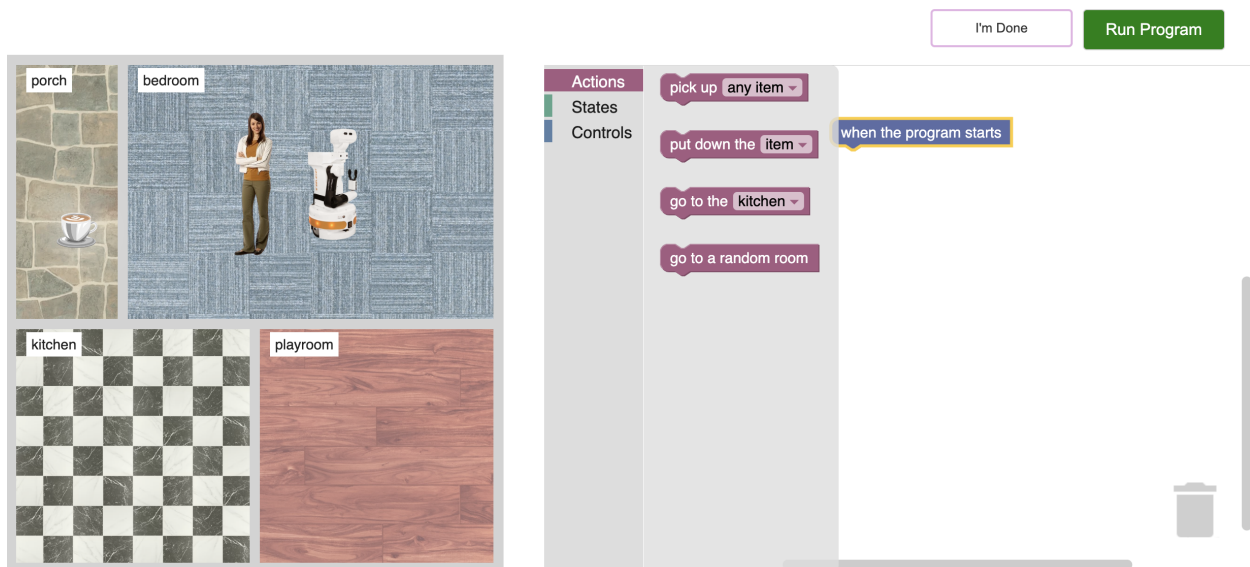


Figure 3: The graphical interface in Blockly for *Sequential Programming (Seq)* with the Actions blocks displayed in the coffee delivery task (Task 4).

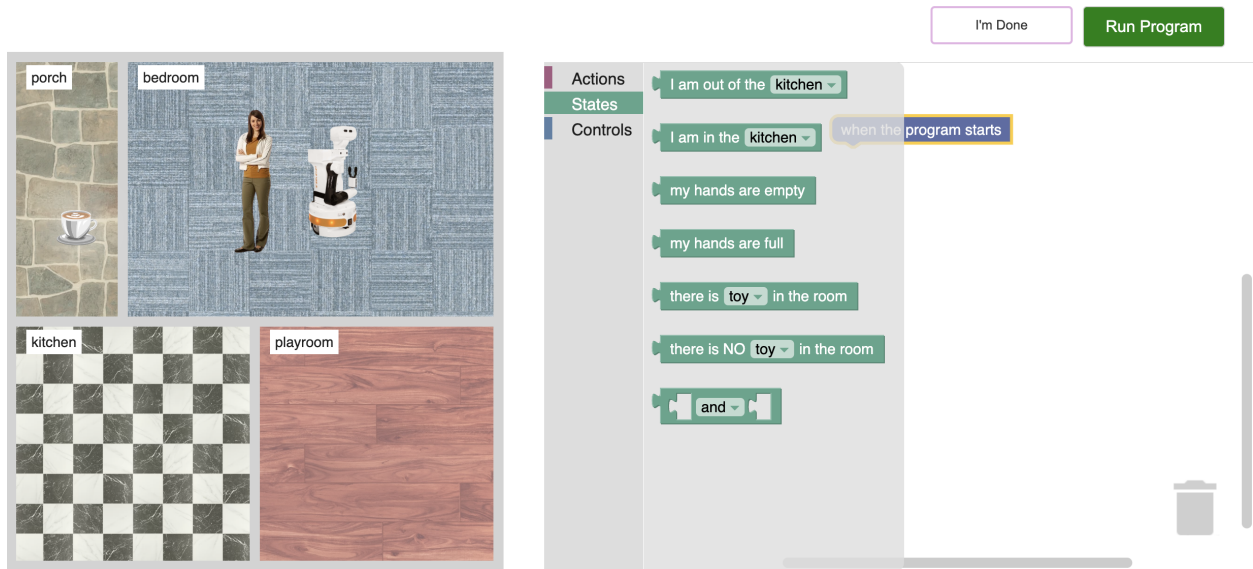


Figure 4: The graphical interface in Blockly for *Sequential Programming (Seq)* with the States blocks displayed in the coffee delivery task (Task 4).

### 1.3 *Trigger-Action Programming*

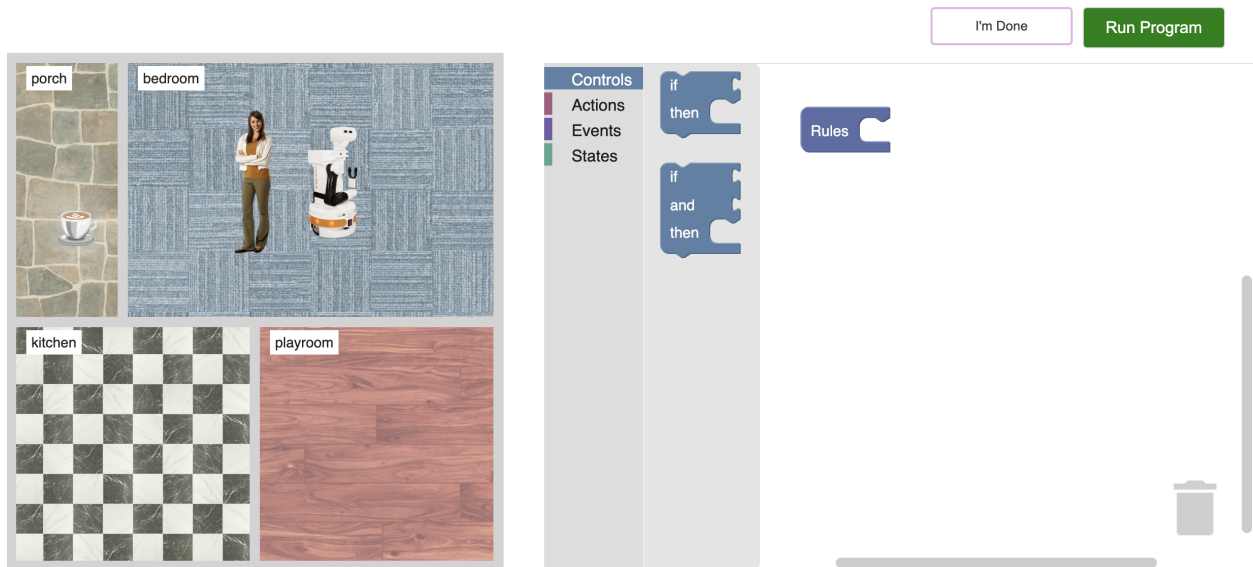


Figure 5: The graphical interface in Blockly for *Trigger-Action Programming (TAP)* with the Control blocks displayed. Here, the user is programming the robot to deliver coffee to them (Task 4).

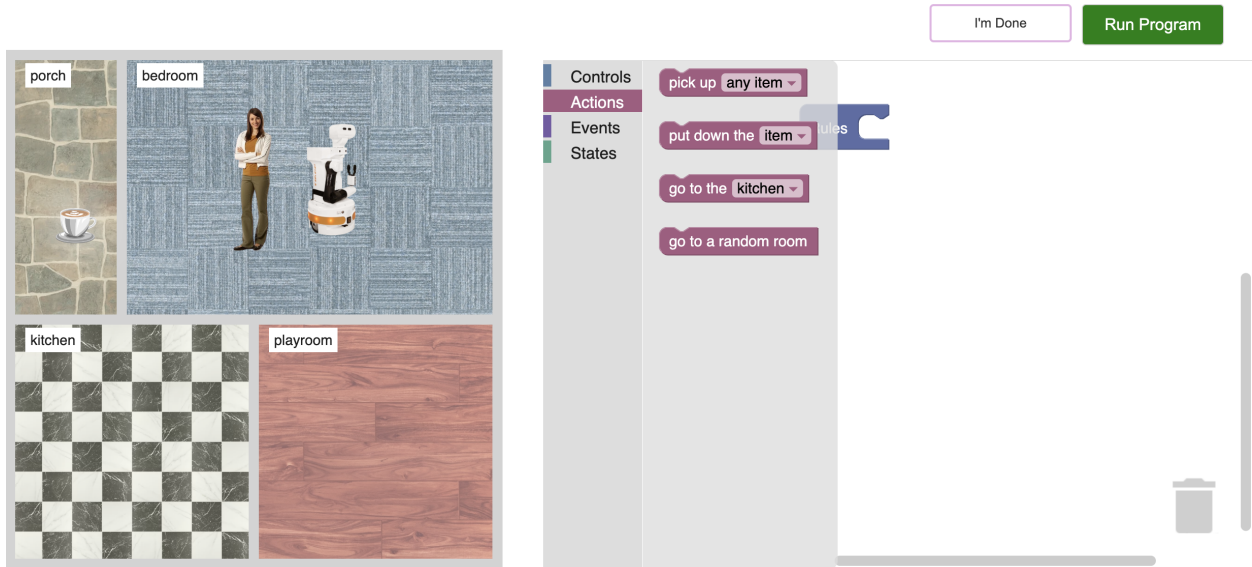


Figure 6: The graphical interface in Blockly for *Trigger-Action Programming (TAP)* with the Action blocks displayed in the coffee delivery task (Task 4).

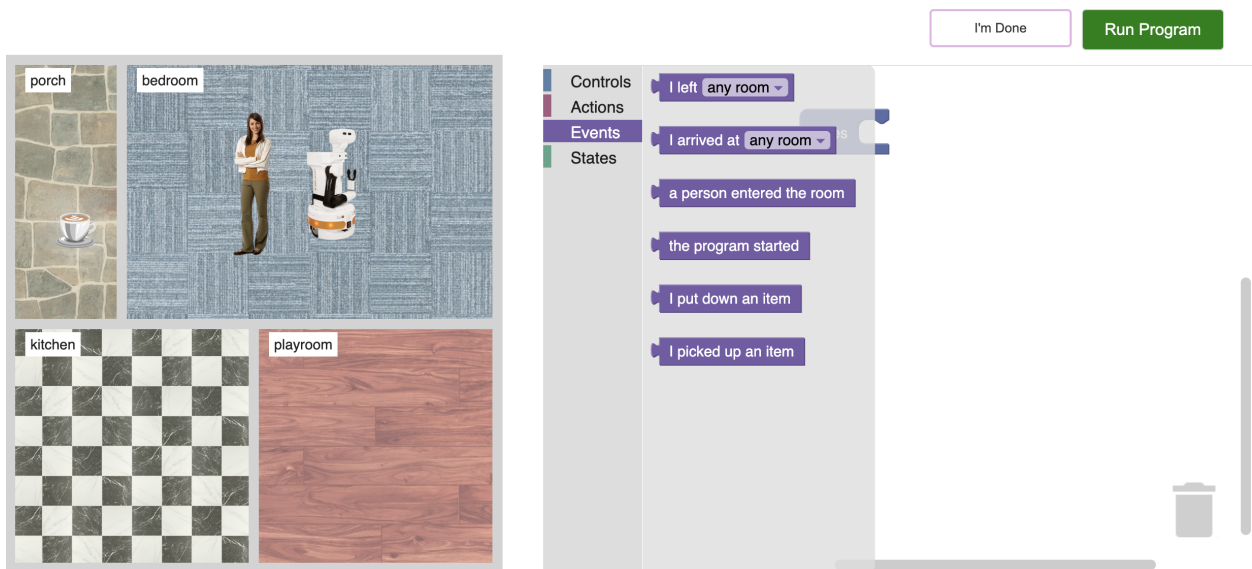


Figure 7: The graphical interface in Blockly for *Trigger-Action Programming (TAP)* with the Event blocks displayed in the coffee delivery task (Task 4).



Figure 8: The graphical interface in Blockly for *Trigger-Action Programming (TAP)* with the State blocks displayed in the coffee delivery task (Task 4).

### 1.4 Full MDP Programming

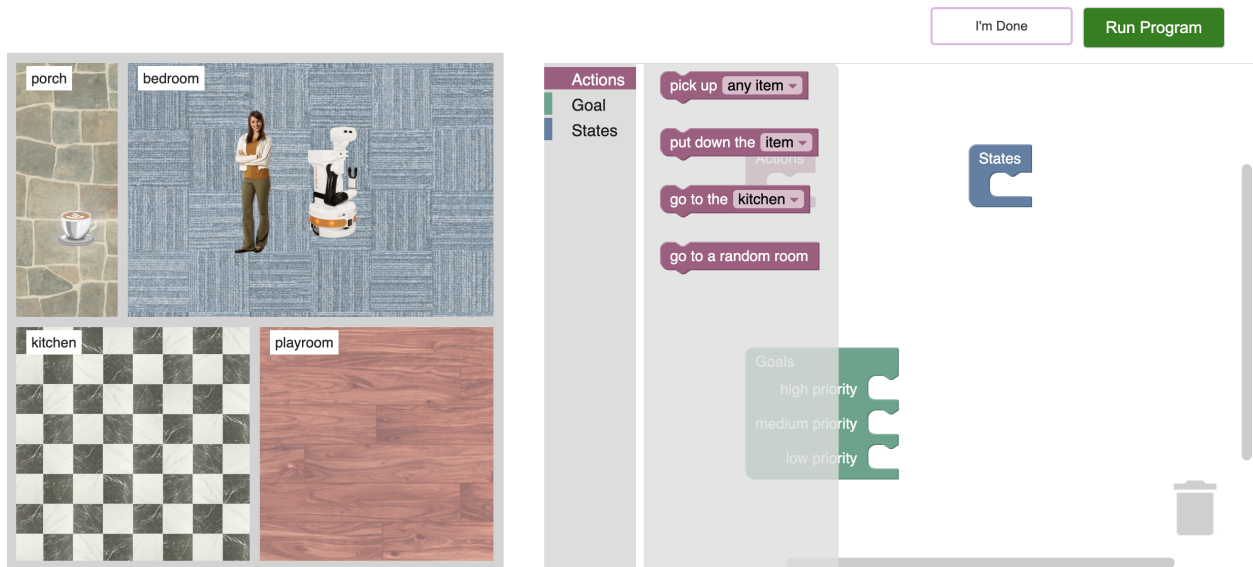


Figure 9: The graphical interface in Blockly for *Full MDP Programming (Full-MDP)* with the Action blocks displayed in the coffee delivery task (Task 4).



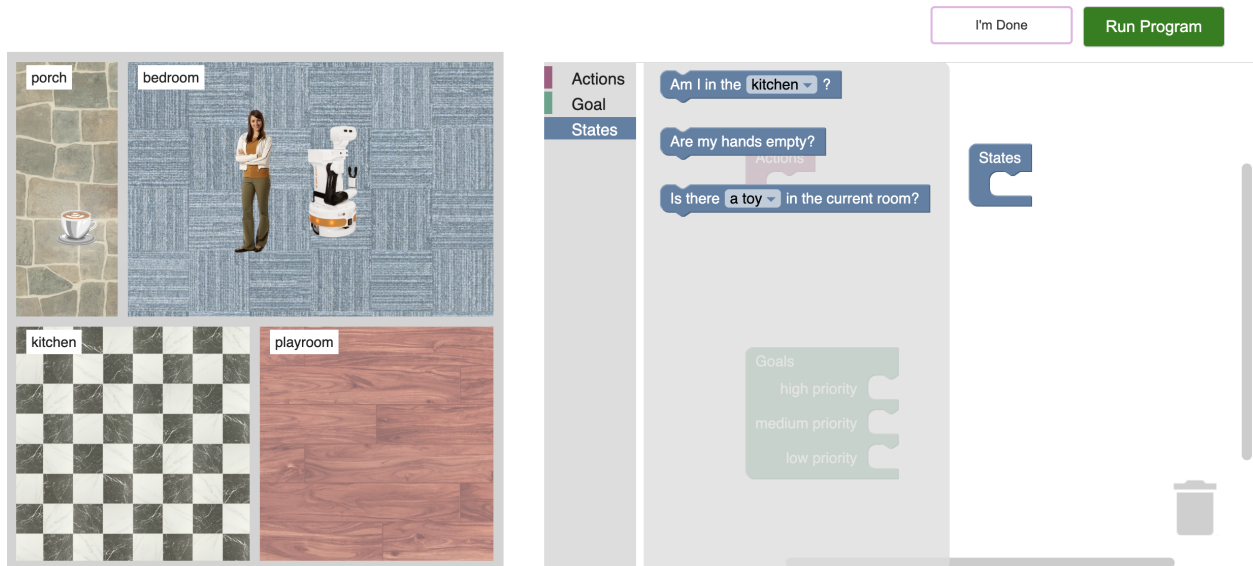


Figure 10: The graphical interface in Blockly for *Full MDP Programming (Full-MDP)* with the State blocks displayed in the coffee delivery task (Task 4).

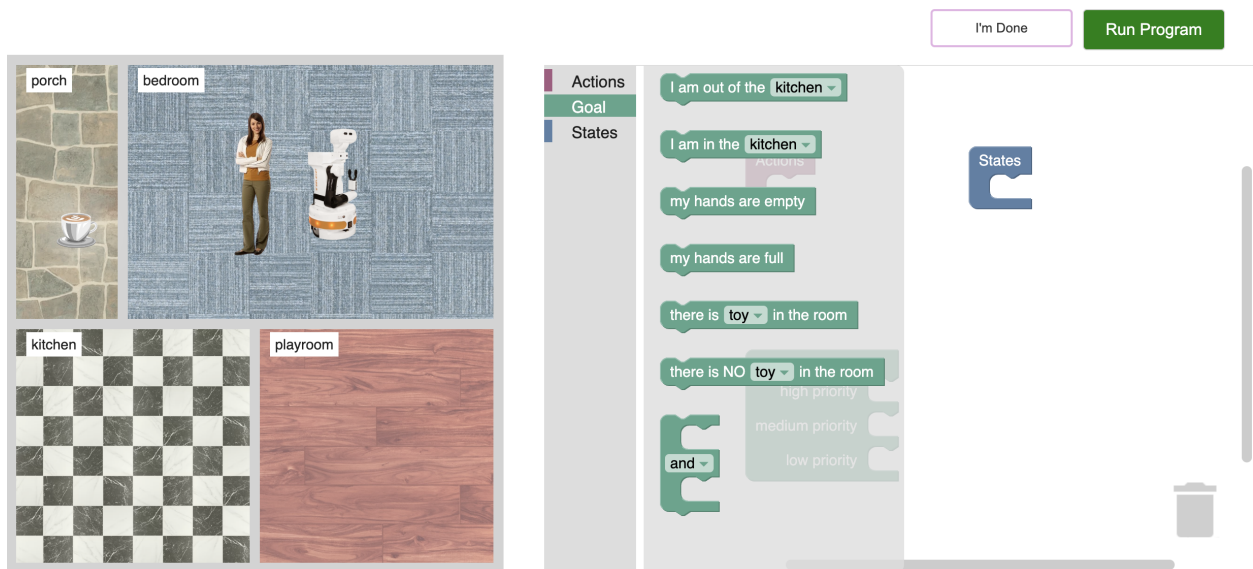


Figure 11: The graphical interface in Blockly for *Full MDP Programming (Full-MDP)* with the Goal blocks displayed in the coffee delivery task (Task 4).

## 1.5 Goal-Only MDP Programming

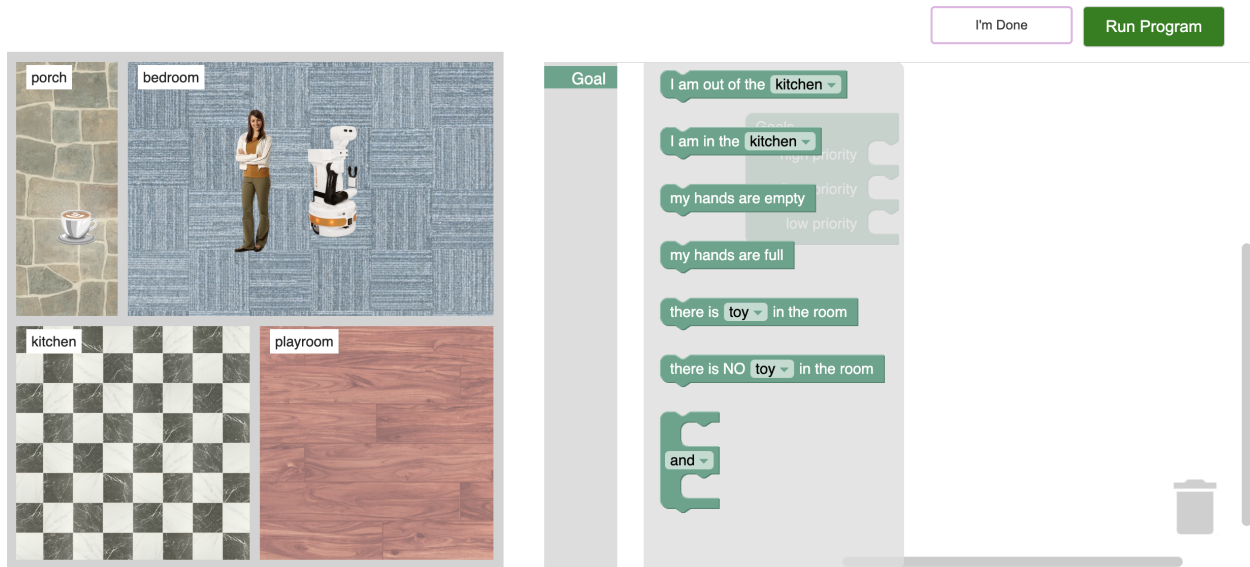


Figure 12: The graphical interface in Blockly for *Goal-Only MDP Programming (Goal-MDP)* with the Goal blocks displayed in the coffee delivery task (Task 4).

## 2 Tasks Participants Were Asked to Complete

For our user study, we designed 10 programming tasks where users would be asked to program a TIAGo mobile manipulator robot to achieve a desired goal within a home environment. These tasks can be performed using each of the Seq, TAP, Full-MDP, and Goal-MDP end-user programming paradigms. The 10 tasks, their descriptions, and the task features they contain are shown in Table 2.

## 3 Tutorial

For each end-user programming paradigm, participants are tasked with completing a tutorial that had two parts. The first part was introduced as part of the survey, where we explain how the general paradigm they are assigned to works through simple examples. After finishing this task, they are presented with comprehension check questions. Feedback is provided for any questions they answer incorrectly. Then, participants work on a second part of the tutorial, where they use the interface to construct programs to solve basic tasks step-by-step, guided by explanations. We maintained consistency across all paradigms by keeping the tutorial tasks uniform. In the following sections, we detail each of the four tutorials we used in the study verbatim.

### 3.1 Sequential Programming

#### 3.1.1 Part 1: Explanation of Sequential Programming

In this study, you will be provided with an interface for programming (virtual) robots. The interface lets you visually construct code by dragging and connecting various types of puzzle-like pieces and blocks, as illustrated below.

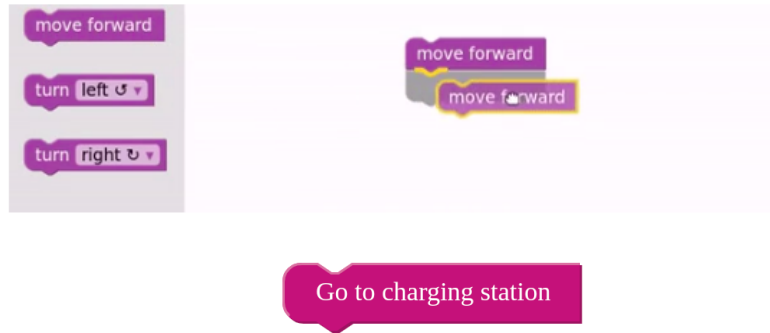
You will program a robot to accomplish various tasks by providing a sequence the robot will follow step-by-step.

We will start with a tutorial to help you understand how to create a program by specifying sequential actions. At the end of the tutorial, we will ask you to answer a few questions to make sure you understand the concepts.

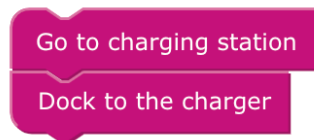
Actions are specific tasks or behaviors that the robot can perform, such as moving to a location, picking up objects, making sounds, or displaying information. For example:

Table 2: For each programming task we asked participants to complete, we list their titles, description, and features.

<b>Task</b>	<b>Description</b>	<b>Features</b>
<i>Task 0: Baseline</i>	In this task, your goal is to program the robot to move from the bedroom to the kitchen. At the start of the task, the robot will always be in the bedroom.	–
<i>Task 1: Person Avoidance</i>	In this task, your goal is to have a robot avoid a person. This means that whenever the person enters the same room as the robot, the robot should immediately exit that room. The person will be constantly moving from room to room within the house.	Loop Required
<i>Task 2: Toy in Random Room</i>	In this task, your goal is to program a robot to move a toy from some unknown room to the playroom. Each time you run the program, the toy may start in a different room, and the robot should move it to the playroom.	Multiple States and Actions, Initial Uncertainty, Unwieldy Number of Possibilities
<i>Task 3: Toys in Kitchen</i>	In this task, your goal is to program a robot to move all of the toys out of the kitchen. At the start of the task, the toys will be in the kitchen. The robot should move the toys out of the kitchen one at a time. Each time you run the program, there may be a different number of toys in the kitchen, and the robot should be able to move all the toys (regardless of how many there are) out of the kitchen.	Multiple States and Actions, Initial Uncertainty, Unintuitive Goals
<i>Task 4: Coffee Delivery</i>	In this task, your goal is to program a robot to deliver coffee from the kitchen to a person in some unknown room (e.g. by putting down the coffee in the same room as the person). Each time you run the program, the person may be in a different room, and the robot should move the coffee from the kitchen to the room the person is in.	Multiple States and Actions, End Uncertainty, Unwieldy Number of Possibilities
<i>Task 5: Coffee-or-Mail (Separate Rooms)</i>	In this task, your goal is to program a robot to move either mail or coffee to their respective destinations. At the start of the task, only the mail or the coffee will be moved to the porch. If the mail is on the porch, the robot should move it to the kitchen. If coffee appears, the robot should move it to the bedroom.	Multiple States and Actions, Initial Uncertainty, End Uncertainty
<i>Task 6: Mail on Porch</i>	In this task, your goal is to program a robot to move three pieces of mail off the porch. At the start of the task, the three pieces of mail will be on the porch. The robot should move the pieces of mail one at a time from the porch to any other room in the house so that there is no longer any mail on the porch.	Multiple States and Actions, Unintuitive Goals
<i>Task 7: Person Avoidance While in Kitchen</i>	In this task, your goal is to program the robot to remain in the kitchen and also avoid being in the same room as the person. If a person enters the kitchen, the robot should prioritize avoiding the person and exit the kitchen and should re-enter the kitchen soon after the person leaves the kitchen. The person will be constantly moving from room to room within the house.	Loop Required, Priority Levels
<i>Task 8: Coffee-or-Mail (Same Room)</i>	In this task, your goal is to program a robot to move either mail or coffee to the kitchen. Each time you run your program, only the mail or the coffee will be on the porch. If the mail is on the porch, the robot should move it to the kitchen. If the coffee appears, the robot should also move it to the kitchen.	Multiple States and Actions, Initial Uncertainty
<i>Task 9: Coffee to Kitchen</i>	In this task, your objective is to program a robot to move a cup of coffee from the bedroom to the kitchen. At the start of the task, the coffee will always appear in the bedroom, and the robot will be in the playroom.	Multiple States and Actions



The element above represents a single action that instructs the robot to navigate towards its charging station. A program must contain at least one action. If a program consists of multiple actions, actions will be executed in the order they appear from top to bottom in the program. Specifically, after the current action has been completed, the robot will begin the next action in the sequence.



In the program shown above, the robot will first execute the “Go to charging station” action completely. Only after completing that action by reaching the charging station, it will then proceed to perform the “Dock to the charger” action. This action instructs the robot to move and connect to the charger.

In certain scenarios, it is helpful to specify that actions depend on the state of the robot. One such control structure is the “IF [state] DO [action]” control. For example:



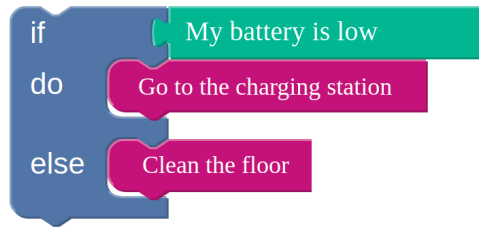
A state (following the “if”) refers to the current conditions or situations of a robot or its environment. The robots can detect these states, and they can be either true or false. For example, the state “battery charger is not in the room” evaluates as either true (indicating that the battery charger is indeed not in the room) or false (indicating that the battery charger actually is in the room).

If the state is found to be true (indicating that the charger is indeed not in the room), the action “make a beeping sound” is carried out by the robot. If, however, the state is found to be false, the action is not taken. In other words, in rules like the example here, whenever an event is triggered, the robot first verifies if the state is true before proceeding to execute the action. In the “IF [state] DO [action]” control, actions will only be executed if the “state” is evaluated as true.

Another handy tool is the “IF [state] DO [action] ELSE [action]” control. This control structure enables the robot to execute an action only if a specific condition is met, and execute an alternative action if the condition is not met. The “ELSE” part of the control structure refers to the alternative action that will be executed when the state in the “IF” statement is evaluated as false. For example:

In this example, the robot will check the state of its battery. If it determines that the battery is low, the robot will execute the “Go to the charging station” action. If the robot’s battery is not low (i.e., the state following the “IF” is false), the action attached to the “ELSE” is executed, so the robot will “Clean the floor” instead.

Note that the “DO” and “ELSE” components can contain more than one action each. You have the flexibility to include one or more actions within the “DO” block, which will be executed if the state associated with the “IF”



statement is satisfied. Similarly, you can attach one or more actions within the “ELSE” block, which will be executed if the state connected to the “IF” statement is false.

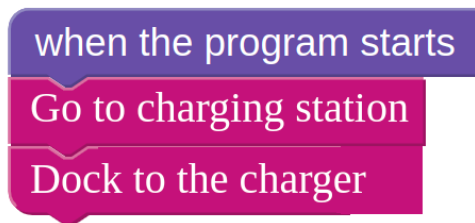
The final control structure enables the repetition of actions. It is represented by the “REPEAT WHILE [state] DO [action]” block. This construct allows for the continuous execution of the specified action as long as the “state” is true. For example:



In this given example, the robot will perform the “Clean the floor” action in a continuous manner, provided that the condition “the floor is dirty” is true. This means that the robot will keep cleaning the floor until it’s no longer dirty.

Note that without using “REPEAT WHILE [state] DO [action]”, all other control structures execute actions only once, depending on the evaluation of a particular state.

In the context of this study, a program refers to a series of sequential actions. A complete program looks like the following example:



The aim of this program is to ensure the robot’s battery is charged. To achieve the goal, the program specifies sequences of actions the robot needs to execute.

In this example, the robot executes two actions sequentially when the program is started: “Go to charging station” and “Dock to the charger.” The “Go to charging station” action allows the robot to navigate to the charging station. The “Dock to the charger” action prompts the robot to physically connect itself to the nearby charger.

Note that the order of the two actions in this block affects the order in which the robot executes those actions. Furthermore, because there is no “repeat” structure, these actions are only taken once. Finally, the actions are connected to the “when the program starts” block so that the robot knows when to begin, which will always be the case for your own programs.

- Q1 If the “state” is false in the “IF [state] DO [action] ELSE [action]” control structure
- the action associated with the “DO” will be executed by the robot
  - the action associated with the “ELSE” will be executed by the robot
  - the action associated with both the “ELSE and the “DO” will be executed by the robot
  - None of the actions will be executed
- Q2 Which of these control structures can execute actions multiple times?
- IF [state] DO [action] ELSE [action]

- IF [state] DO [action]
- REPEAT WHILE [state] DO [action]
- Do action A FOREVER

Q3 Which of the following statements is not correct?

- the actions in your program must be connected to the “when the program starts” block
- the order of actions in a block specifies the order in which the robot will take those actions
- when using an “IF” control structure, the robot will take either the “do” or “else” action, but not both
- all of the actions in a list are taken in parallel (i.e., at the same time) by the robot

Q4 In the context of this study, please select the device you are going to program. Choose “robot” as your response.

- Thermostat
- Robot
- Oven

### 3.1.2 Part 2: *Sequential Programming Interactive Component*

1. The goal of this part of the tutorial is to familiarize you with the robot programming interface so you can start creating custom behaviors for robotic tasks. Let’s get started!
2. To begin, let’s take a look at the environment our robot will be operating in. The interface on the left displays a visual representation of four rooms in a house: the bedroom, kitchen, porch, and playroom. This is where the robot will be navigating and performing its tasks.
3. The right half of the interface is where you can program the robot.
4. The robot requires you to specify the sequence of ACTIONS it needs to complete the task (for example, pick up a toy, move to the kitchen). The robot executes these sequences of actions one after the other towards completing its mission.

Optionally, you may specify STATES or conditions that a robot may need to evaluate before executing an action. For example, before trying to pick up a toy (action), the robot could first assess whether it is already holding a toy (state).

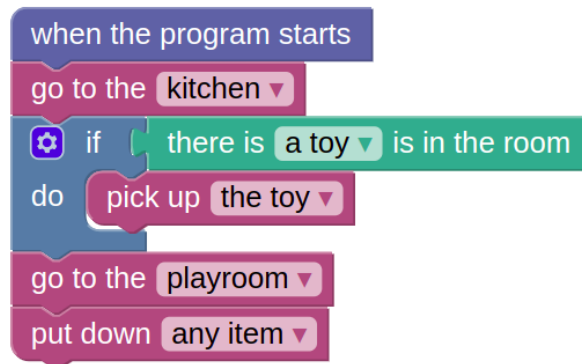
5. Now, you will construct a program to make the robot take a toy from the kitchen to the playroom. At the start the robot might be in any one of the four rooms. Therefore, we start by adding an action to move the robot to the kitchen. To achieve this, Click on the “Actions” tab and drag the “go to kitchen” piece and attach it to the “when the program starts” block on the programming interface. This action will make the robot move to the kitchen.
6. Once the robot arrives in the kitchen, it needs to pick up the toy. Before trying to pick up the toy, you will program the robot to check if there is a toy. This check may seem unnecessary for this task since we know that the toy will be in the kitchen. However, in other situations, the robot may not know where the toy is ahead of time, which makes performing this check to see if the toy is in the room helpful. To check if a toy is present in the room, click on the “Controls” tab and drag the “If do” block and attach it under the “Go to kitchen” piece. Then, drag “a toy is in the room” piece from the “States” tab and attach it to the if.
7. Click on the “Actions” tab and drag the “Pick up the toy” piece and attach it next to do. This block instructs the robot to pick up the toy only if it detects there is a toy in the room.

Once the robot picks up the toy, its next step is to take the toy to the kitchen. Drag the “Go to playroom” piece and attach it below the “if do” block.

8. The final step is to put down the toy once the robot reaches the kitchen. Click on the “Actions” menu and drag the “put down the item” piece below the last piece. Optionally, you can change “put down the item” to “put down the toy” (click on the caret to change the item name). This will also instruct the robot to put down the toy.

Note that the action “put down the item” directs the robot to put down whatever object it is currently holding. However, the action “put down the toy” specifically instructs the robot to put down the toy. If the robot is holding an item other than a toy, the instruction “put down the toy” would not prompt the robot to release the held object.

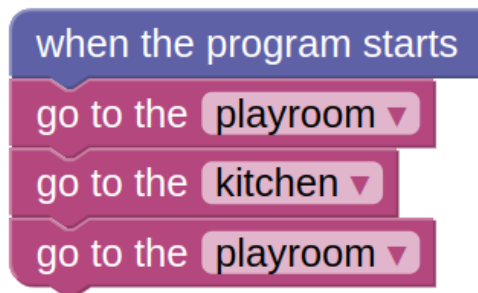
9. If your program looks like the image on the right, then press the “Run Program” button to see your program running.



Otherwise, move the blocks around to make them match the image and then run the program.

If the robot remains inactive after you click “Run Program,” double check that your program is correct. Click “Stop Program” to make any needed changes and then rerun the program.

10. Now you will work on a second task. Your task is to program the robot to go back and forth between the kitchen and the playroom forever. If you create a program like the one on the right, the robot will initially go to the playroom, then to the kitchen, then back to the playroom. After this third action, the robot will come to a stop in the kitchen and remain there.



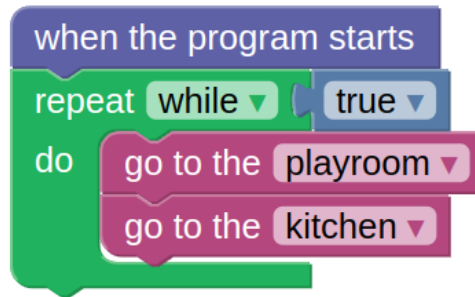
11. However, this task requires you to program the robot to move forever. While it is possible to include additional “Go to kitchen” and “Go to playroom” pieces, they will only contribute to one additional loop through the movements.

Note that the “go to playroom” piece is the same as the “go to kitchen” piece (click on the caret to change the room name).

12. To program the robot to move iteratively, first remove all the pieces you added to the programming interface by dragging each of them to the trash bin found on the bottom right corner.
13. Then click on the “Control” tab and drag the piece labeled “repeat while do” and drop it on the programming interface. Next from the “Control” tab drag the piece labeled “True” and attach it right next to while. The “repeat while do” block runs all the pieces inside it as long as the condition you attach next to while is true. Attaching True next to while means the block will be executed forever.
14. Next you need to put the actions that the robot needs to repeat inside the “repeat while do” block. For this task, the robot needs to move to the kitchen. Click on “Actions” tab and drag the “Go to kitchen” piece and attach it inside the “repeat while do” block.

Again, go to the “Actions” tab and drag “go to kitchen” piece inside the “repeat while do” block right below the previous piece. Now, change kitchen to playroom by clicking on the caret (downward pointing triangle) of the piece. This block makes sure the robot goes to the kitchen, then to the playroom, repeating forever.

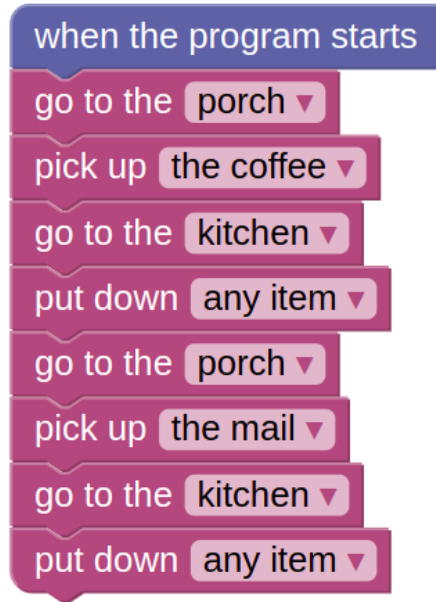
15. If your program looks like the image on the right, then press the “Run Program” button to see your program running.



Otherwise, move the blocks around to make them match the image and then run the program.

16. Now you will work on the final task. In this task, both a cup of coffee and a piece of mail will start on the porch, and your objective is to program a robot to move the coffee and mail from the porch to the kitchen. However, because the coffee is likely to get cold if outside too long, the robot should first move the coffee to the kitchen and then come back to move the mail to the kitchen. The robot should take the coffee to the kitchen and then the mail to the kitchen in this order.
17. First, remove all of the pieces added to the programming interface by dragging each one to the trash bin.
18. To have the robot perform the task, you’ll first need the robot to go to the porch. Click on the “Actions” tab and attach the “Go to kitchen” piece to “when the program starts” block. Then change kitchen to porch.
19. Once the robot arrives on the porch, it needs to pick up the coffee. Click on the “Actions” tab and drag the “Pick up any item” piece and attach it right below the previous piece. Then change the object to be picked up to coffee by clicking on the caret (downward pointing triangle) of the piece you just attached. This block instructs the robot to pick up the coffee.
20. Once the robot picks up the coffee, its next step is to take the coffee to the kitchen. Drag the “Go to kitchen” piece and attach it below the “Pick up the coffee” piece
21. The next step is to put down the coffee once the robot reaches the kitchen. Click on the “Actions” menu and drag the “put down the item” piece below the last piece. Again, you can change “the item” to “the coffee” to put down the coffee.
22. Once the coffee is brought to the kitchen, the next part of the task is bringing the mail to the kitchen. Click on the “Actions” tab and attach the “Go to kitchen” piece to the previously added piece and change kitchen to porch.
23. Once the robot arrives on the porch, it needs to pick up the mail. Click on the “Actions” tab and drag the “Pick up any item” piece and attach it right below the previous piece. Then change the object to be picked up to mail from the options on the piece you just attached. This block instructs the robot to pick up the mail.
24. Once the robot picks up the mail, its next step is to take the mail to the kitchen. Drag the “Go to kitchen” piece and attach it below the “Pick up the mail” block
25. The final step is to put down the mail once the robot reaches the kitchen. Click on the “Actions” menu and drag the “put down the item” piece below the last piece.





26. If your program looks like the image on the right, then press the “Run Program” button to see your program running.
27. Congratulations on completing the tutorial! Now click on the I’m done button.

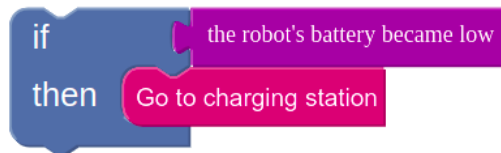
## 3.2 *Trigger-Action Programming*

### 3.2.1 Part 1: Explanation of *Trigger-Action Programming*

You will program a robot to accomplish various tasks by formulating rules. These rules serve as instructions to guide the robot’s actions and behavior.

We will start with a tutorial to help you understand how to create a program by specifying rules. At the end of the tutorial, we will ask you to answer a few questions to make sure you understand the concepts.

Each rule you will create to program the robot will be in the same format as the following example:



The element presented above is called a rule. A rule consists of two components: an IF statement and a THEN statement. The event “my battery became low” following the IF statement is called a trigger. Every time the trigger event occurs, it takes the corresponding action specified in the THEN part of the rule.

Actions are specific tasks or behaviors that the robot can perform, such as moving to a location, picking up objects, making sounds, or displaying information. For example:

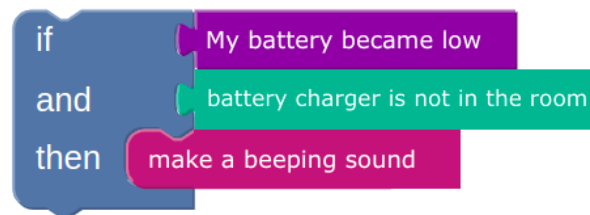


The element above represents a single action that instructs the robot to navigate towards its charging station. In the rule defined earlier, the action is “go to charging station”. Note that only a single action can be listed in each rule.

Note that triggers occur at a specific moment in time. At the instant “the robot’s battery became low” by falling below some predetermined threshold (e.g., when the battery falls below 10%), this rule activates once and sends the robot to its charging station. Even if the battery remains below 10%, the rule will only activate one time. The rule would only activate again after the robot’s battery has been charged and it once again drops below that threshold.

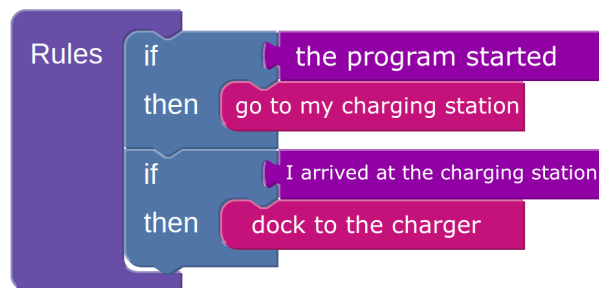
Note that each “THEN” statement in a rule can only have one action. The action associated with a trigger is executed when the trigger event occurs.

Sometimes it is helpful to specify that the robot should take an action only in certain circumstances. To enable such behaviors, rules can combine a trigger, a state and an action using the word “and.” For example:



A state (following the “and”) refers to the current conditions or situations of a robot or its environment. The robots can detect these states, and they can be either true or false. For example, the state “battery charger is not in the room” evaluates as either true (indicating that the battery charger is indeed not in the room) or false (indicating that the battery charger actually is in the room). If the state is found to be true (indicating that the charger is indeed not in the room), the action “make a beeping sound” is carried out by the robot. If, however, the state is found to be false, the action is not taken. In other words, in rules like the example here, whenever an event is triggered, the robot first verifies if the state is true before proceeding to execute the action.

Note that you can write multiple, independent rules, where each is defined by an “IF” block. The order in which the rules are listed does not matter. In the context of this study, a program refers to a collection of rules that are composed of TRIGGERS, ACTIONS, and may optionally include STATES. A program containing two different rules looks as follows:



The aim of this program is to ensure the robot’s battery is charged. The program specifies rules the robot needs to follow in response to certain events. The rules also specify the actions that need to be taken in response to the triggered events.

In this example, the robot responds to two triggers: “the program started” and “I arrived at the charging station.” When the program starts, “the program started” event is triggered, prompting the robot to execute the “go to my charging station” action. Once the robot reaches the charging station, the “I arrived at the charging station” event is triggered, leading the robot to perform the “dock to the charger” action.

Note that the two rules in this example are effectively chained together since one rule’s trigger represents the other rule’s action being completed. This does not always need to be the case; each rule can have a completely independent trigger.

Q1 In this study, a trigger (the part following IF)

- prompts a robot to execute an action as soon as the event associated with it occurs
- is a condition sensed by the robot

- either of the above
- None of the above

Q2 In the context of this study, a rule can contain

- A trigger and an action
- A trigger, a state and an action
- Either of the above
- None of the above

Q3 Which of the following statements is not correct?

- all rules in a program are contained under a “rules” block
- the order in which rules are listed does not matter
- each rule can contain only a single action
- all rules in a program must be chained together (i.e., by having another rule trigger on the completion of another rule’s action)

Q4 In the context of this study, please select the device you are going to program. Choose “robot” as your response.

- Thermostat
- Robot
- Oven

### 3.2.2 Part 2: *Trigger-Action Programming Interactive Component*

1. The goal of this part of the tutorial is to familiarize you with the robot programming interface so you can start creating custom behaviors for robotic tasks. Let’s get started!
2. To begin, let’s take a look at the environment our robot will be operating in. The interface on the left displays a visual representation of four rooms in a house: the bedroom, kitchen, porch, and playroom. This is where the robot will be navigating and performing its tasks.
3. The right half of the interface is where you can program the robot.
4. The robot is designed to act based on a set of rules you provide in the programming interface. Rules consist of TRIGGERS, ACTIONS, and may optionally include STATES.

For each of the rules you create, you need to specify a **TRIGGER**, an event that, once it occurs, allows the robot to respond (for example, arriving to the kitchen or its hands becoming full by picking up a toy).

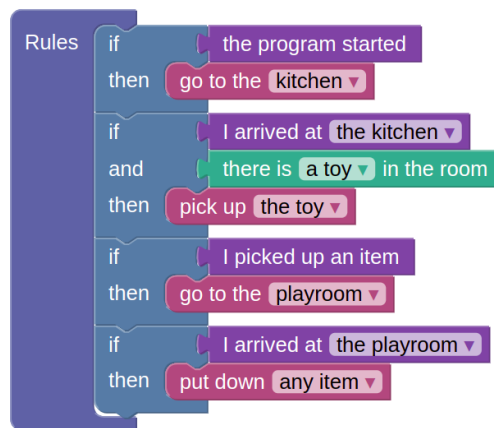
5. In addition to triggers, the robot requires you to specify **ACTIONS** it needs to perform following the activation of triggers (for example, pick up a toy, move to the kitchen).

Optionally, you may specify **STATES** or conditions that a robot may need to evaluate before executing an action within a rule. For example, to know whether or not to pick up a toy (action), the robot needs to know whether it is already holding a toy (state).

6. Once all the rules are provided to it, the robot executes actions based on these rules towards accomplishing a task.
7. Now, you will construct a program to make the robot take a toy from the kitchen to the playroom. At the start the robot might be in any one of the four rooms. Therefore, you need to specify a rule for the robot to go to the kitchen as soon as the program starts. To achieve this, click on the “Controls” tab and drag the piece labeled “if then” and attach it to the “Rules” block found in the programming interface. This block will create the “IF statement and a THEN statement” structure of the first rule.
8. Next, click on the “Events” tab and drag the piece labeled “program started” and attach it to the if in the “if then” piece in “Rules”. The “program started” piece is a trigger that activates only once when the program you constructed starts.
9. Next, from the “Actions” tab, drag the “Go to kitchen” piece and attach it to the “if program started” piece.

The rule you just created instructs the robot to move to the kitchen when the program starts.

10. Once the robot arrives in the kitchen, it needs to pick up the toy. Before trying to pick up the toy, you will program the robot to check if there is a toy. While this check may seem unnecessary, since we know that the toy will be in the kitchen, in other cases, the robot may not know where the toy is ahead of time, which makes performing this check to see if the toy is in the room helpful. To accomplish this, you need the **“if and then”** rule. Click on the **“Controls”** tab and drag the **“if and then”** piece and attach it to the **“Rules”** block below the previous rule.
11. You will need to have the robot detect the arriving to the kitchen event before it tries to pick up the toy. Now drag the **“I arrived at the kitchen”** piece from the **“Events”** tab and attach it next to **if**. Then, click on the **“States”** tab and drag **“there is a toy in the room”** and attach it next to **and**. This block would help the robot check if there is a toy in the kitchen as soon as it arrives in the room. Once the robot arrives at the kitchen and there is a toy in the room, the robot will now need to pick up the toy. For that, drag the **“pick up the toy”** piece from actions and attach it next to **then**.
12. Now that you have enabled the robot to pick up the toy at the kitchen, you’ll need to program the robot to go to the kitchen after having picked up the toy. To do this, we’ll rely on the event **“I picked up an item”** to instruct the robot to then head to the playroom. To do this, click on the **“Controls”** tab and drag the **“if then”** piece and attach it to the **“Rules”** below the last two blocks. Then, from the **“Events”** tab drag **“I picked up an item”** and attach it to the new **“if then”** piece. This piece helps the robot detect if it just picked up something.
13. Click on the **“Actions”** tab and drag the **“go to the kitchen”** piece inside the newly added **“if then”** block. Change the room from kitchen to playroom by clicking on the caret (downward pointing triangle) of the piece you just attached. This will allow the robot to move to the playroom once it has picked up the toy.
14. Next you will need to program the robot to put down the toy as soon as it gets to the playroom. To do that, drag the **“if and then”** block and attach it below the last block. Then you should add a way to detect if the robot arrives at the playroom: **“I have arrived at the playroom.”** Add the piece **“put down the item”** as the action.
15. If your program looks like the image on the right, then press the **“Run Program”** button to see your program running. Otherwise, move the blocks around to make them match the image and then run the program.  
If the robot remains inactive after you click **“Run Program,”** double check that your program is correct. Click **“Stop Program”** to make any needed changes and then rerun the program.



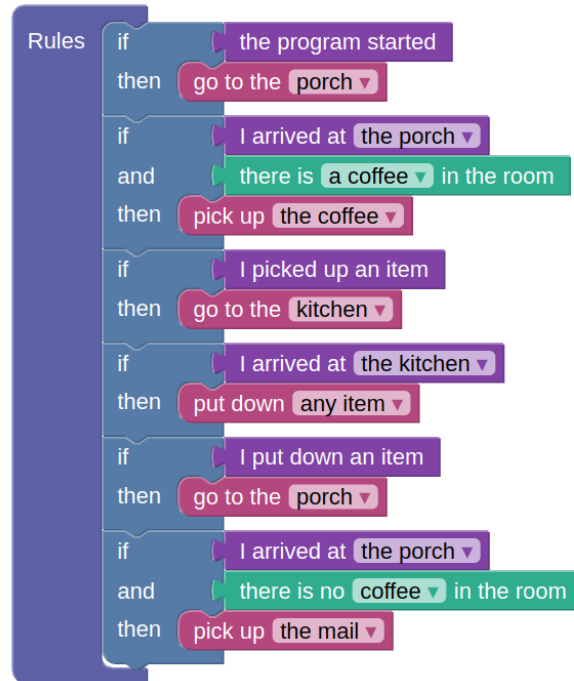
16. Note that, in this first example program, each rule triggered based off of another rule’s action completing (or the start of the program). This does not always need to be the case. Sometimes, rules should trigger based off of changes to the environment that the robot observes. For each task, you will need to consider what triggers make the most sense. However, this strategy of having other rules trigger based on another rule’s action completion is the most straightforward way to have the robot complete a sequence of actions since each rule can only contain one action.

17. Now let's work on the final task. In this task, both a cup of coffee and a piece of mail will start on the porch, and your objective is to program a robot to move the coffee and mail from the porch to the kitchen. However, because the coffee is likely to get cold if outside too long, the robot should first move the coffee to the kitchen and then come back to move the mail to the kitchen. The robot should take the coffee to the kitchen and then the mail to the kitchen in this order.
18. First remove all the pieces added to the "Rules" block by dragging them to the trash bin found on the bottom right corner.
19. To have the robot perform the task, you'll first need the robot to go to the porch. From the "Actions" tab, drag the "Go to kitchen" piece and attach it to the piece "If program started." Change the room from kitchen to porch by clicking on the caret (downward pointing triangle) of the piece you just attached. This will ensure that the robot will go to the porch once the program begins.  

Note that the "go to porch" piece is the same as the "go to kitchen" piece (click on the caret to change the room name).
20. Next, you need to program the robot to pick up the coffee once it gets to the playroom.  

For this second rule, you will need to have the robot detect the arriving to the porch event before it tries to pick up the coffee. To ensure that the robot executes each action appropriately, it is necessary to incorporate rules like these for every desired action. This is because the robot is designed to carry out actions only when it detects events being triggered. Click on the "Controls" tab and drag the "if and then" piece and attach it to the "Rules" below the "If program started" block.
21. Now drag the "I arrived at the porch" piece from the "Events" tab and attach it next to if. Then, click on the "States" tab and drag "there is coffee in the room" and attach it next to and. This block would help the robot check if there is a coffee in the playroom as soon as it arrives in the room. This check is necessary, since we want the robot to take the coffee to the kitchen before taking the mail. Once the robot arrives at the porch and there is coffee in the room, the robot will now need to pick up the coffee. For that, drag the "pick up any item" piece from actions and attach it next to then. Then change the object to be picked up to coffee by clicking on the caret (downward pointing triangle) of the piece you just attached.
22. Now that you have enabled the robot to pick up the coffee at the porch, you'll need to program the robot to go to the kitchen after having picked up the coffee. To do this, we'll rely on the event "I picked up an item" to instruct the robot to then head to the kitchen. To do this, click on the "Controls" tab and drag the "if then" piece and attach it to the "Rules" below the last two blocks. Then, from the "Events" tab drag "I picked up an item" and attach it to the new "if then" piece. This piece helps the robot detect if it just picked up something.
23. Click on the "Actions" tab and drag the "go to the kitchen" piece inside the newly added "if then" block. This will allow the robot to move to the kitchen once it has picked up the coffee.
24. Next you will need to program the robot to put down the coffee as soon as it gets to the kitchen. To do that, drag the "if and then" block and attach it below the last block. Then you should add a way to detect if the robot arrives at the kitchen: "I have arrived at the kitchen." Add the piece "put down the item" as an action.
25. Once the coffee is brought to the kitchen, the next part of the task is bringing the mail to the kitchen. To do this, click on the "Controls" tab and drag the "if then" piece and attach it to the "Rules" below the previous blocks. Then the event that was just triggered is associated with dropping the coffee specifically "I put down the coffee". Drag this block from the events tab and attach it to the new rule. Then attach the "go to porch block" action.
26. Once the robot is on the porch, it needs to pick up the mail. For that, drag a new "if and then" block and attach it below the last rule block. Then drag the "I arrived at the porch" piece from the "Events" tab and attach it next to if. Then, click on the "States" tab and drag "there is no coffee in the room" and attach it next to and. The check "there is no coffee in the room" is important because the first item the robot needs to move is the coffee. The robot should move the mail only when there is no coffee on the porch. From the action tab drag the "pick up the mail" action and attach it to the newly added rule. Note that this rule is very similar to one you created earlier, differing only in the "and" section.

27. You've already set a rule for the robot to go to the kitchen when it picks up something, so there's no need for a second rule. Similarly, you have set a rule for the robot's to drop what it is holding as soon as it arrives in the kitchen. Hence, there is no need to introduce another rule for the robot's behavior upon arrival in the kitchen.
28. If your program looks like the image on the right, then press the "Run Program" button to see your program running. Otherwise, move the blocks around to make them match the image and then run the program.



If the robot remains inactive after you click "Run Program," double check that your program is correct. Click "Stop Program" to make any needed changes and then rerun the program.

29. Congratulations on completing the tutorial! Now click on the I'm done button.

### 3.3 Full MDP Programming

#### 3.3.1 Part 1: Explanation of Full MDP Programming

You will program a robot to accomplish various tasks by defining STATES, ACTIONS and GOALS the robot can reference to automatically come up with a plan to accomplish its tasks. A program refers to a collection of STATES, ACTIONS, and GOALS.

We will start with a tutorial to help you understand how to create a program by specifying states, actions, and goals. At the end of the tutorial, we will ask you to answer a few questions to make sure you understand the concepts.

First, we will introduce what STATES, ACTIONS, and GOALS are individually. Afterwards, we will explain how to combine them into a program.

A state refers to the current conditions or situations of a robot or its environment. The robot can detect these states, and they can be either true or false. For example, the state "is my battery low?" evaluates as either true (indicating that the robot's battery is indeed low) or false (indicating that the robot's battery is not low).The following element is an example of a state:



Actions are specific tasks or behaviors that the robot can perform, such as moving to a location, picking up objects, making sounds, or displaying information. For example:



Go to charging station

The element above represents a single action that instructs the robot to navigate towards its charging station.

A goal represents a desired state that we want the robot to learn how to achieve. A state refers to the current conditions or situations of a robot or its environment. The robot can detect these states, and they can be either true or false. For example, the state “is my battery low?” evaluates as either true (indicating that the robot’s battery is indeed low) or false (indicating that the robot’s battery is not low).

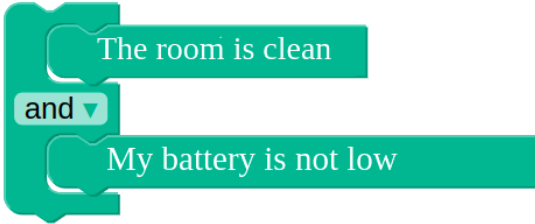
Based on the goal(s) your program indicates, the robot will automatically try different actions in its environment to learn how to achieve the specified goal(s) as quickly and as often as possible. Consider the following example goal:



My battery is charging

In the example, the goal “My battery is charging” means that the robot should make sure its battery is being charged. This goal directs the robot to try out its available actions, which in our situation typically leads to the robot finding a charging station and connecting to it to recharge its battery. The robot plans and executes actions that make progress towards achieving the goal(s) you specify.

Certain goals may involve combining two conditions using the word “and.” For instance:



The room is clean  
and  
My battery is not low

The goal “the room is clean and my battery is not low” combines two conditions that need to be met simultaneously. It means that the robot aims to have a clean room while ensuring that its battery is not low on power. Both the cleanliness of the room and the battery level need to be in a satisfactory state for the goal to be achieved; meeting only one condition of this combined goal is insufficient when the conditions are connected with “and.”

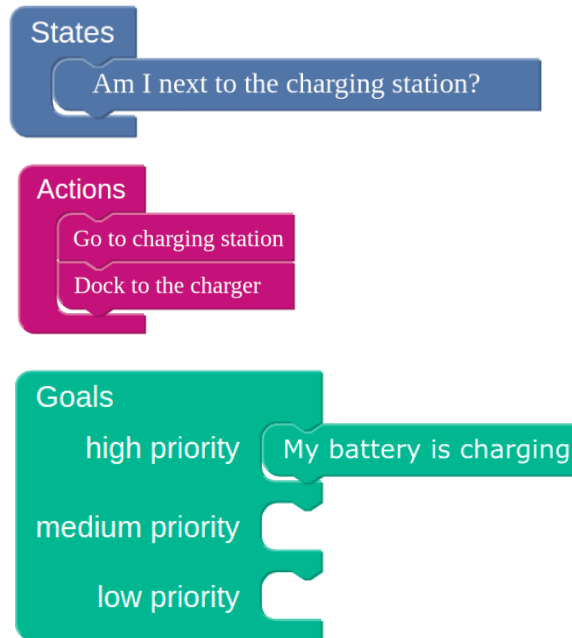
The collection of goals you specify may include multiple different goals for which you indicate different priorities. You can specify three different levels of priority for goals: high, medium, and low.

For instance, the goal “my battery is not low” might be assigned high priority, and the goal “the room is clean” might be assigned medium priority. This means that the robot will try both to keep its battery charged and to clean the room. If it cannot do both at once, it will elect to charge its battery. Two goals that are placed at the same priority level are given equal importance.

In the context of this study, a program refers to a collection of STATES, ACTIONS, and GOALS that you specify. A complete program looks like the following:

The aim of this program is to make sure that the robot’s battery is charging, which is reflected in the goal statement. To achieve the goal, the program specifies states that the robot needs to check when choosing actions. It also lists all possible actions that the robot should consider performing as it attempts to achieve the goal conditions. By executing appropriate actions and checking appropriate states, the robot can make progress towards achieving the stated goal.

In this example, the robot is provided with two actions it can use to make progress to the desired goal: “Go to charging station” and “Dock to the charger.” The “Go to charging station” action allows the robot to navigate to the



charging station. The “Dock to the charger” action prompts the robot to physically connect itself to the nearby charger. Note that the order of the two actions within the program’s action block does not affect how the robot achieves the goal—it is just a list of actions for the robot to consider. However, this list of actions must include all of the different actions the robot might need to use in the process of achieving the goal.

To achieve the goal of “My battery is charging,” the robot needs to consider the state “Am I next to the charging station?” This state helps the robot determine whether it is currently located near the charging station or not, which could be very important in deciding whether it should choose the “Go to charging station” or “Dock to the charger” actions.

Reminder: A program can have multiple states, actions, and goals. The order in which states and actions are listed does not matter. The order of goals within a given priority level does not matter.

Q1 In this study, a complete robot program requires

- States and Actions
- States only
- Goal only
- States, Actions, and Goal

Q2 The order in which actions are specified

- directly influences the sequence in which the robot executes them
- does not affect the sequence in which the robot executes them.
- sometimes it does sometimes it does not
- none of the above

Q3 Two conditions can be combined to form a goal using the word:

- Goals
- True
- And
- None of the above

Q4 In the context of this study, please select the device you are going to program. Choose “robot” as your response.

- Thermostat
- Robot
- Oven



### 3.3.2 Part 2: Full MDP Programming Interactive Component

1. The goal of this part of the tutorial is to familiarize you with the robot programming interface so you can start creating custom behaviors for robotic tasks. Let's get started!
2. To begin, let's take a look at the environment our robot will be operating in. The interface on the left displays a visual representation of four rooms in a house: the bedroom, kitchen, porch, and playroom. This is where the robot will be navigating and performing its tasks.
3. The right half of the interface is where you can program the robot.
4. The robot is designed to act based on **GOALS** that you specify. Goals are certain states or combinations of states that we desire the robot to reach or achieve to complete its mission (for example, a toy is in the playroom, the robot is in the kitchen) with specific levels of priorities. In addition to the goals, the robot requires you to specify all of the **ACTIONS** it might need to complete the task (for example, pick up a toy, move to the kitchen).
5. Finally, the robot must be informed of the set of distinct **STATES**. States are the different conditions that a robot may need to evaluate when choosing between different actions. For example, to know whether or not to pick up a toy (action), the robot needs to know whether it is already holding a toy (state). You will need to give the robot a comprehensive list of all of the states the robot needs to consider in learning how to achieve the goal. This list will include states that are directly relevant to the **GOAL**, as well as intermediate states needed to reach the goal. If a key state is missing, the robot won't be able to tell when to take an action that is needed to achieve the goal.

Once these are provided to it, the robot tries to find the most effective way to accomplish the task and then executes the determined course of action.

6. Now, you will construct a program to make the robot take a toy from the kitchen to the playroom. At the start the robot might be in any one of the four rooms.
7. To successfully perform this task, the robot needs a specification of the goals. The robot can only know it has achieved its goals by sensing the environment. For the current task, the goal can be stated as the robot seeing the toy in the playroom, which is how the robot would recognize that it has completed the task. Drag the "and" piece from the Goal tab and attach it to the "Goals" piece. Then drag the "I am in the kitchen" piece from the Goal tab and attach it to one of the empty spaces in "and". Then change kitchen to playroom by clicking on the caret (downward pointing triangle) of the piece you just added. Finally, drag the "a toy is in the room" piece and attach it to the remaining space of the "and" piece.
8. To successfully perform the task, the robot needs to have a set of actions to try: going to the kitchen, going to the playroom, picking up the toy, and putting down the toy. These are the full set of actions the robot needs to perform to pick up the toy from the playroom and put it down in the kitchen. Note that the robot will figure out the order in which to execute these actions to achieve this goal as long as you provide a complete list. Drag the "go to kitchen" from the "Actions" tab and attach it to the "Actions" block. In addition, add the pieces "go to playroom", "pick up the toy", and "put down the toy" to the "Actions" block.  
Note that "go to playroom" comes the same base piece as "go to kitchen" (click on the caret to change the room name).
9. Now the robot needs to know what states it will need to consider when learning which of these actions to take, in which order, to reach the goal. To correctly choose its actions, the robot needs to check the states of being in the kitchen, being in the playroom, seeing a toy in a room, and holding the toy. These are the relevant set of states the robot needs for choosing actions for reaching the goal. Drag the pieces that represent each state ("Am I in the kitchen?", "Am I in the playroom?", "Are my hands empty?", and "Is there a toy in the current room") from the "States" tab and attach them to the "States" block. Again, you need to use the caret to change the room name.
10. If your program looks like the image on the right, then press the "Run Program" button to see your program running. Otherwise, move the blocks around to make them match the image and then run the program.

Note that the order of the pieces in the "States" and "Actions" blocks do not matter.



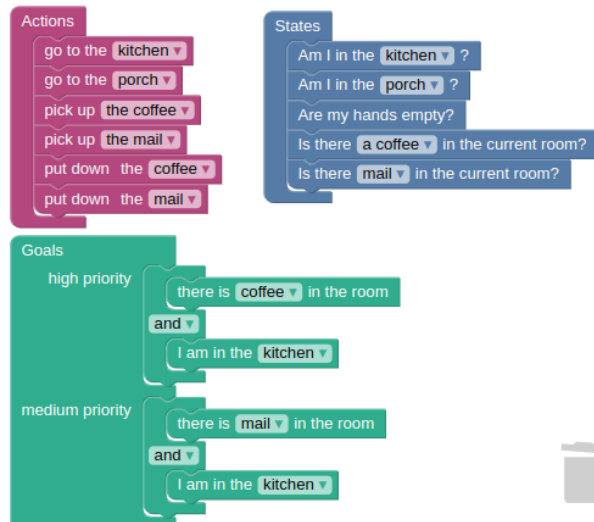
11. Now you will work on the final task. In this task, both coffee and mail will start on the porch, and your objective is to program a robot to move the coffee and mail from the porch to the kitchen. However, because the coffee is likely to get cold if outside too long, the robot should first move the coffee to the kitchen and then come back to move the mail to the kitchen. The robot should take the coffee to the kitchen and then the mail to the kitchen in this order.
12. First, remove all of the pieces added to the programming interface by dragging each one to the trash bin.
13. To have the robot perform the task, You need to specify two goals — the first is moving the coffee from the porch to the kitchen, and the second is moving the mail from the porch. To make sure the first goal is satisfied before the second, we will attach the first goal to the high priority slot in the “Goals” piece, and the second goal to the medium priority slot.
 

Again, because the goal is sensed from the robot’s perspective, we need the robot to be in the room when the goal is satisfied.
14. For the first goal, drag the “and” piece from the Goal tab and attach it to the slot for high priority in the “Goals” piece. Then, click on the Goal tab again, locate the “I am in the kitchen” piece, and drag it to connect it to one of the empty spaces in the “and” piece. Finally, add the piece “there is a toy in the room” and attach it to the remaining space of the “and” piece. Then change toy to coffee by clicking on the caret (downward pointing triangle) of the piece you just added.
15. For the second goal, drag the “and” piece from the Goal tab and attach it to the slot for medium priority in the “Goals” piece. Then, click on the Goal tab again, locate the “I am in the kitchen” piece, and drag it to connect it to one of the empty spaces in the “and” piece. Finally, add the piece “there is a toy in the room” and attach it to the remaining space of the “and” piece. Then change toy to mail by clicking on the caret (downward pointing triangle) of the piece you just added.
 

Note that the “there is a coffee in the room” piece is the same as the “there is mail in the room” (click on the caret to change the room name).
16. To successfully perform the task, the robot needs to try a set of actions: going to the kitchen, going to the porch, picking up the coffee, picking up the mail, putting down the coffee, and putting down the mail. Drag the “go to kitchen” piece from the “Actions” tab and attach it to the “Actions” block. In addition, also add the pieces “go to porch”, “pick up the coffee”, “pick up the mail”, and “put down the coffee”, and “put down the mail” to the “Actions” block.
17. Now the robot needs to know what states it will need to check when deciding which of these actions to take to reach the goal. To correctly choose its actions, the robot needs to check the states of being in the kitchen, being

on the porch, seeing a coffee, holding the coffee, seeing the mail, and holding the mail. These are the relevant set of states the robot needs for choosing which actions to take, and in which order, for achieving its goals. Drag the pieces that represent each state (“Am I in the kitchen?”, “Am I in the porch?”, “Are my hands empty?”, “Is there coffee in the current room?”, and “Is there mail in the current room?”) from the “States” tab and attach them to the “States” block.

18. If your program looks like the image on the right, then press the “Run Program” button to see your program running. Otherwise, move the blocks around to make them match the image and then run the program.



Note that the order of the pieces in the “States” and “Actions” blocks do not matter.

19. Congratulations on completing the tutorial! Now click on the I’m done button.

### 3.4 Goal-Only MDP Programming

#### 3.4.1 Part 1: Explanation of Goal-Only MDP Programming

You will program a robot to accomplish various tasks by providing it with one or more goals the robot can reference to automatically come up with a plan to accomplish its tasks. A program refers to a collection of one or more goals.

We will start with a tutorial to help you understand how to create a program by specifying goals. At the end of the tutorial, we will ask you to answer a few questions to make sure you understand the concepts.

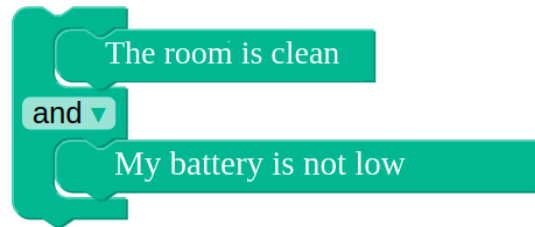
A goal represents a desired state that we want the robot to learn how to achieve. A state refers to the current conditions or situations of a robot or its environment. The robot can detect these states, and they can be either true or false. For example, the state “is my battery low?” evaluates as either true (indicating that the robot’s battery is indeed low) or false (indicating that the robot’s battery is not low).

Based on the goal(s) your program indicates, the robot will automatically try different actions in its environment to learn how to achieve the specified goal(s) as quickly and as often as possible. Consider the following example goal:



In the example, the goal “My battery is charging” means that the robot should make sure its battery is being charged. This goal directs the robot to try out its available actions, which in our situation typically leads to the robot finding a charging station and connecting to it to recharge its battery. The robot plans and executes actions that make progress towards achieving the goal(s) you specify.

Certain goals may involve combining two conditions using the word “and.” For instance:

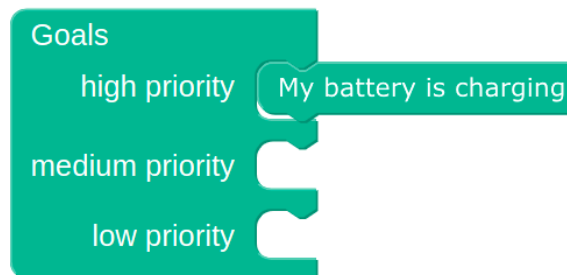


The goal “the room is clean and my battery is not low” combines two conditions that need to be met simultaneously. It means that the robot aims to have a clean room while ensuring that its battery is not low on power. Both the cleanliness of the room and the battery level need to be in a satisfactory state for the goal to be achieved; meeting only one condition of this combined goal is insufficient when the conditions are connected with “and.”

Your program (i.e., the collection of goals you specify) may include multiple different goals for which you indicate different priorities. You can specify three different levels of priority for goals: high, medium, and low.

For instance, the goal “my battery is not low” might be assigned high priority, and the goal “the room is clean” might be assigned medium priority. This means that the robot will try both to keep its battery charged and to clean the room. If it cannot do both at once, it will elect to charge its battery. Two goals that are placed at the same priority level are given equal importance.

A complete program with only a single goal could look like the following example:



The aim of this program is to make sure that the robot’s battery is charging, which is reflected in the goal statement “my battery is charging.” By specifying this goal, the robot will try to determine what actions to take to make progress towards achieving that goal.

- Q1 In this study, a robot program is defined by
- specifying a sequence of actions to be executed step by step
  - specifying a goal
  - can be either of the above
  - none of the above
- Q2 Two conditions can be combined to form a goal using the word:
- goals
  - true
  - and
  - none of the above
- Q3 Which of the following statements is not correct?
- a program may contain only a single goal, or it may contain multiple goals
  - goals that combine two conditions are considered satisfied as long as at least one condition is true
  - three different priority levels are possible for goals
  - all goals need to be given different priorities

- Q4 In the context of this study, please select the device you are going to program. Choose “robot” as your response.
- Thermostat
  - Robot
  - Oven

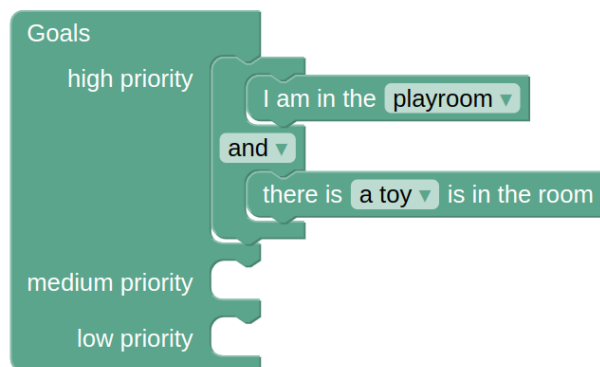
### 3.4.2 Part 2: Goal-Only MDP Programming Interactive Component

1. The goal of this part of the tutorial is to familiarize you with the robot programming interface so you can start creating custom behaviors for robotic tasks. Let’s get started!
2. To begin, let’s take a look at the environment our robot will be operating in. The interface on the left displays a visual representation of four rooms in a house: the bedroom, kitchen, porch, and playroom. This is where the robot will be navigating and performing its tasks.
3. The right half of the interface is where you can program the robot.
4. The robot is designed to act based on a GOAL that you specify. Goals are certain states or combinations of states that we desire the robot to reach or achieve to complete its mission (for example, a toy is in the playroom, the robot is in the kitchen).

The robot will try to find the most effective way to accomplish the goal you specify and then executes that course of action.

5. Now, you will construct a program to make the robot take a toy from the kitchen to the playroom. At the start the robot might be in any one of the four rooms.
6. To successfully perform this task, the robot needs a specification of the goals. The robot can only know it has achieved its goals by sensing the environment. For the current task, the goal can be stated as the robot seeing the toy in the playroom, which is how the robot would recognize that it has completed the task. Drag the “and” piece from the “Goal” tab and attach it to the “Goal” piece. Then drag “I am in the kitchen” piece from the Goal tab and attach it to one of the empty spaces in “and”. Then change kitchen to playroom by clicking on the caret (downward pointing triangle) of the piece you just added. Finally, drag “a toy is in the room” piece and attach it to the remaining space of the “and” piece.
7. If your program looks like the image on the right, then press the “Run Program” button to see your program running.

Otherwise, move the blocks around to make them match the image and then run the program.



8. Now you will work on the final task. In this task, both coffee and mail will start on the porch, and your objective is to program a robot to move the coffee and mail from the porch to the kitchen. However, because the coffee is likely to get cold if outside too long, the robot should first move the coffee to the kitchen and then come back to move the mail to the kitchen. The robot should take the coffee to the kitchen and then the mail to the kitchen in this order.

9. First, remove all of the pieces added to the programming interface by dragging each one to the trash bin.
  10. To do this, you need to specify two goals — the first is moving the coffee from the porch to the kitchen, and the second is moving the mail from the porch. To make sure the first goal is satisfied before the second, we will attach the first goal to the high priority slot in the “Goals” piece, and the second goal to the medium priority slot. Again, because the goal is sensed from the robot’s perspective, we need the robot to be in the room when the goal is satisfied.
  11. For the first goal, drag the “and” piece from the Goal tab and attach it to the slot for high priority in the “Goals” piece. Then, click on the Goal tab again, locate the “I am in the kitchen” piece, and drag it to connect it to one of the empty spaces in the “and” piece. Finally, add the piece “there is a toy in the room” and attach it to the remaining space of the “and” piece. Then change toy to coffee by clicking on the caret (downward pointing triangle) of the piece you just added.
  12. For the second goal, drag the “and” piece from the Goal tab and attach it to the slot for medium priority in the “Goals” piece. Then, click on the Goal tab again, locate the “I am in the kitchen” piece, and drag it to connect it to one of the empty spaces in the “and” piece. Finally, add the piece “there is a toy in the room” and attach it to the remaining space of the “and” piece. Then change toy to mail by clicking on the caret (downward pointing triangle) of the piece you just added.
- Note that the “there is a coffee in the room” piece is the same as the “there is mail in the room” (click on the caret to change the room name).
13. If your program looks like the image on the right, then press the “Run Program” button to see your program running. Otherwise, move the blocks around to make them match the image and then run the program.



14. Congratulations on completing the tutorial! Now click on the I’m done button.

## 4 Survey Instrument

### 4.1 Pre-experiment Survey

Thanks for participating in our study! The purpose of this study is to explore efficient methods for individuals to communicate tasks to future home robots. You will be requested to complete several tasks using the interface we have developed in order to assess its effectiveness and user-friendliness.

- Q1 Have you used a system in which you write if-this-then-that rules to control devices or services? IFTTT, Zapier, and NodeRed are common examples of such services.
- Yes  No  Other

- Q2 Are you majoring in, hold a degree in, or have held a job in any of the following fields: computer science; computer engineering; information technology; or a related field?  
 Yes  No  I prefer not to answer
- Q3 Have you ever completed a class or completed a class-length online tutorial about computer programming?  
 Yes  No  I prefer not to answer
- Q4 If you have experience programming, what programming languages have you used? (If you do not have experience with programming languages, please put "N/A" below)
- Q5 Have you ever taken a class about, or completed equivalent tutorials about, or had equivalent work/hobby experience in, creating machine learning models?  
 Yes  No  I prefer not to answer
- Q6 Have you ever completed a class or completed a class-length online tutorial or had equivalent work/hobby experience in, creating reinforcement learning algorithms?  
 Yes  No  I prefer not to answer

## 4.2 Task Follow-up Survey

- Q1 Please indicate the extent to which you agree or disagree with each statement about your experience with programming the robot.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I found the task to be easy.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I am confident that I have correctly programmed the robot to complete the task.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was able to program the robot to complete the task as specified	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
To this one question, please select 'Agree' as your response	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

- Q2 Please write around 4 sentences describing the process you took to solve the task
- Q3 Did you face a situation where you made a program, but later realized it was incorrect or the robot's behavior did not align with your expectations? Please describe the situation, the reasons for the error, and how you corrected it. Please provide 2-3 sentences for each situation you describe.

## 4.3 End of Experiment Survey

- Q1 What is your gender?  
 Female  Male  Non-binary  Prefer to self-describe  Prefer not to say
- Q2 What is your age?  
 18-24 years old  25-34 years old  35-44 years old  45-54 years old  55 years or older  Prefer not to answer
- Thank you for taking part in this study! Please click the button below to be redirected back to Prolific and register your submission.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I think that I would like to use this system frequently.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system unnecessarily complex.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought the system was easy to use.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think that I would need the support of a technical person to be able to use this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the various functions in this system were well integrated.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought there was too much inconsistency in this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I would imagine that most people would learn to use this system very quickly.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system very cumbersome to use.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt very confident using the system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I needed to learn a lot of things before I could get going with this system.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

## 5 Program Correctness for Each Task

As shown in Figure 13, we found differences across paradigms in *Task 3* ( $\chi^2 = 24.67, p < 0.001$ ), *Task 5* ( $\chi^2 = 13.66, p = 0.02$ ), *Task 6* ( $\chi^2 = 34.72, p < 0.001$ ), and *Task 8* ( $\chi^2 = 18.96, p = 0.002$ ). Pairwise comparisons for *Task 3* revealed that *TAP* ( $\chi^2 = 17.83, p < 0.001$ ) and *Seq* ( $\chi^2 = 14.14, p = 0.001$ ) had significantly higher correctness rate compared to *Full-MDP*. Pairwise comparisons for *Task 5* revealed that participants who were in *TAP* condition performed significantly worse than *Seq* ( $\chi^2 = 11.75, p = 0.004$ ). Pairwise comparisons for *Task 6* showed that participants in *TAP* had significantly higher average correctness compared to both *Full-MDP* ( $\chi^2 = 16.01, p < 0.001$ ) and *Goal-MDP* ( $\chi^2 = 9.42, p = 0.006$ ). Similarly, participants who used *Seq* had significantly higher correctness than *Full-MDP* ( $\chi^2 = 20.32, p < 0.001$ ) and *Goal-MDP* ( $\chi^2 = 13.84, p = 0.001$ ). Finally, on *Task 8*, pairwise comparisons revealed that *Goal-MDP* had a lower correctness rate than both *Seq* ( $\chi^2 = 14.07, p = 0.001$ ) and *TAP* ( $\chi^2 = 7.45, p = 0.03$ ). For all the remaining pairwise comparisons we did not find significant differences.

## 6 Types of Errors Made in Programming Paradigms

### 6.1 Incorrect Solution Analysis

To identify the types of errors featured in the incorrect solutions, the first and second authors reviewed all incorrect submissions together and agreed upon the error categories (listed below) based on their prevalence in each paradigm. We labeled each incorrect solution based on these categories and grouped the solutions that did not fall under these categories as miscellaneous. In our analysis, we counted each unique type of error that the participants made, irrespective of the number of times they made the same error.

1. *Anticipating Environment* is when participants neglect scenarios, often in tasks featuring Initial Uncertainty, End Uncertainty, and Unwieldy Number of Possibilities. For example, in *Task 5*, several participants programmed



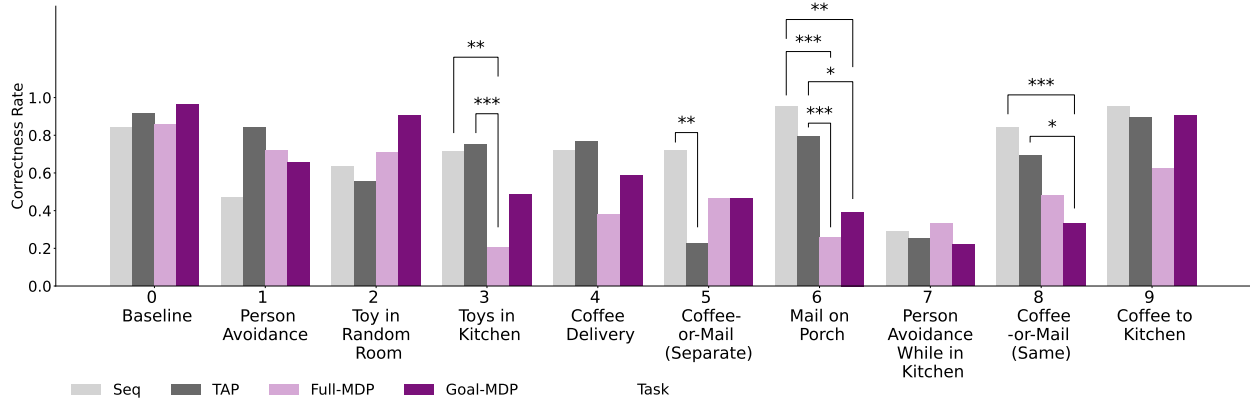


Figure 13: The proportion of participants whose program was correct by task and paradigm. \*  $p < 0.05$ , \*\*  $p < 0.01$ , \*\*\*  $p < 0.001$ .

the robot to move only one of the coffee or mail.

2. *No Priority* occurs in tasks where participants fail to establish a clear order for task completion.
3. *Missing Loop* occurs in Loop Required tasks when *Seq* participants fail to incorporate a loop in their solutions.
4. *Loop Required (Wrong Condition)* occurs in Loop Required tasks when *Seq* participants a loop but select the wrong condition.
5. *Conflicting Rules* occurs when *TAP* participants program two or more rules that prescribe different actions triggered by the same event or event-state combination.
6. *Missing Rule* occurs when *TAP* participants omit certain rules necessary for task completion.
7. *Wrong Event* occurs when *TAP* participants incorrectly define the event in the rule trigger. Notably, *TAP* requires a rule with the event “the program started” to specify the robot’s action at the start of each program.
8. *Wrong State* occurs when *TAP* participants incorrectly define the state attribute in the trigger in a rule.
9. *Underspecified States and Actions* are when *Full-MDP* participants provide a partial list of actions and states attributes necessary for the robot to complete the task.
10. *Sequential Goal* occurs when *Full-MDP* or *Goal-MDP* participants delineate the goals at every step of completing the task rather than stating the final desired outcome.
11. *Counterintuitive Goal* occurs when *Full-MDP* or *Goal-MDP* participants seem to misunderstand how the robot senses the goal state for task completion.
12. *Wrong Use of ‘AND’* occurs when *Full-MDP* or *Goal-MDP* participants seem to interpret the ‘and’ block by its semantic meaning in natural language instead of recognizing it as a Boolean logic operator.

## 6.2 Errors Made by Participants in Each Programming Paradigm

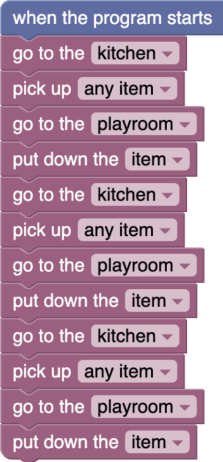

We examined the incorrect solutions submitted by 147 out of our 409 participants to identify distinct types of errors in each paradigm. In this section, we describe the most prevalent error types that more than 10% of the participants made in their incorrect programs. Table 3 provides a detailed breakdown of all error types, including the number of participants who made each mistake and the percentage of incorrect solutions within each end-user programming paradigm.

Table 3: For incorrect solutions, we report the types of errors participants made in each programming paradigm.

Paradigm	Errors Made
<i>Seq</i>	Anticipating Environment (N=13, 48.1%), Missing Loop (N=10, 37.0%), and Loop Required (Wrong Condition) (N=8, 29.6%), Substitution Error (N=5, 18.5%), Empty Solutions (N=2, 7.4%), Miscellaneous (N=2, 7.4%), Infinite Loop (N=1, 3.7%)
<i>TAP</i>	Anticipating Environment (N=18, 45.0%), Conflicting Rules (N=13, 32.5%), Missing Rule (N=12, 30.0%), Wrong Event (N=9, 22.5%), Wrong State (N=4, 10.0%), Start Misuse (N=3, 7.5%), No Priority (N=2, 5.0%), Miscellaneous (N=2, 5.0%), Infinite Loop (N=1, 2.5%), Substitution Error (N=1, 2.5%)
<i>Full-MDP</i>	Underspecified States and Actions (N=26, 49%), Sequential Goal (N=20, 37.7%), Anticipating Environment (N=13, 24.5%), Counterintuitive Goal (N=12, 22.6%), No Priority (N=12, 22.6%), Miscellaneous (N=6, 11.3%), Wrong Use of 'AND' (N=4, 7.5%), Substitution Error (N=2, 3.8%), Impossible Goal (N=1, 1.9%)
<i>Goal-MDP</i>	Anticipating Environment (N=23, 46%), Sequential Goal (N=17, 34.0%), No Priority (N=15, 30.0%), Counterintuitive Goal (N=10, 20.0%), Wrong Use of 'AND' (N=6, 12.0%), Empty Solutions (N=3, 6.0%), Substitution Error (N=3, 6.0%), Miscellaneous (N=3, 6%), Impossible Goal (N=1, 2.0%)

### 6.3 Common Types of Errors Made in *Sequential Programming*

Table 4: We present the incorrect programs that *Seq* participants constructed to solve different tasks. We list the tasks that these programs were aiming to achieve.

Error	Incorrect Program
Anticipating Environment	<p><i>Task 3: Toys in Kitchen</i></p> 
Missing Loop	<p><i>Task 1: Person Avoidance</i></p> 

Wrong Condition

Task 1: Person Avoidance

```
when the program starts
repeat while there is person in the room
do go to a random room
```

## 6.4 Common Types of Errors Made in *Trigger-Action Programming*

Table 5: We present the incorrect programs that *TAP* participants constructed to solve different tasks. We list the tasks that these programs were aiming to achieve.

Error	Incorrect Program
Anticipating Environment	<p>Task 2: Toy in Random Room</p> <pre>Rules if the program started then go to the kitchen if I arrived at the kitchen and there is toy in the room then pick up any item if I picked up an item then go to the playroom</pre>
Conflicting Rules	<p>Task 5: Coffee-or-Mail (Separate Rooms)</p> <pre>Rules if the program started then go to the porch if I arrived at the porch and there is mail in the room then pick up the mail if I picked up an item then go to the kitchen if I arrived at the porch and there is coffee in the room then pick up the coffee if I picked up an item then go to the bedroom</pre>

---

Missing Rule

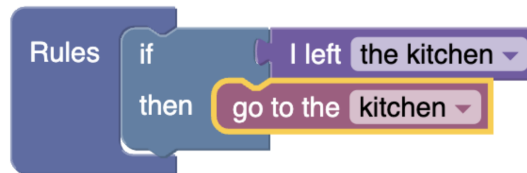
Task 5: Coffee-or-Mail (Separate Rooms)



---

Wrong Event

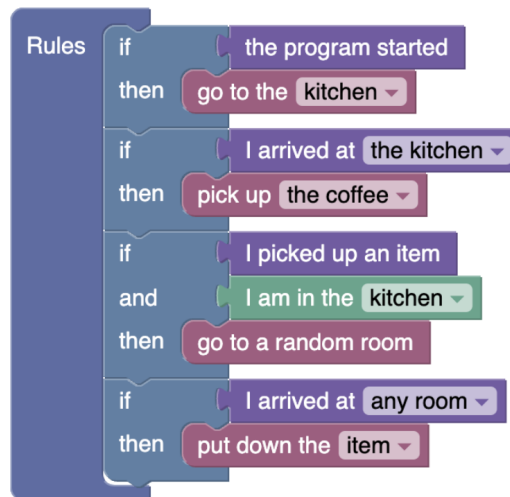
Task 0: Baseline



---

Wrong State

Task 4: Coffee Delivery



---

## 6.5 Common Types of Errors Made in Full MDP Programming

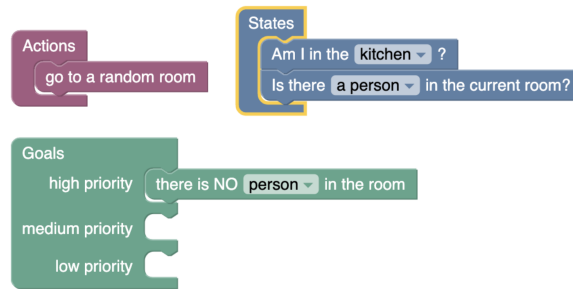
Table 6: We present the incorrect programs that *Full-MDP* participants constructed to solve different tasks. We list the tasks that these programs were aiming to achieve.

Error	Incorrect Program
Underspecified States and Actions	<p><i>Task 9: Coffee to Kitchen</i></p>
Anticipating Environment	<p><i>Task 4: Coffee Delivery</i></p>
Sequential Goal	<p><i>Task 6: Mail on Porch</i></p>

---

No Priority

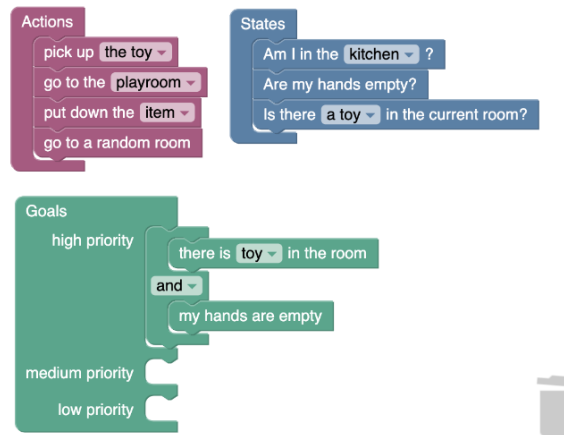
Task 7: Person Avoidance While in Kitchen



---

Counterintuitive Goal

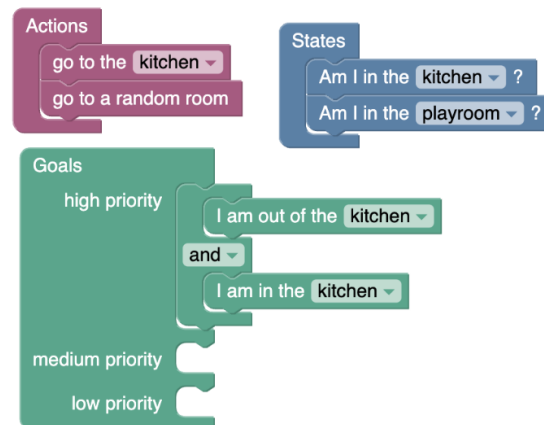
Task 3: Toys in Kitchen



---

Wrong Use of 'AND'

Task 7: Person Avoidance While in Kitchen



---

## 6.6 Common Types of Errors Made in Goal-Only MDP Programming

Table 7: We present the incorrect programs that *Goal-MDP* participants constructed to solve different tasks. We list the tasks that these programs were aiming to achieve.

---

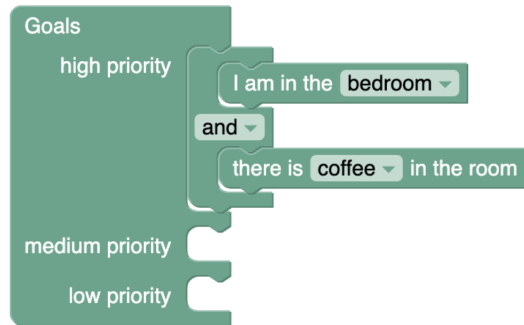
Error	Incorrect Program
-------	-------------------

---

---

Anticipating Environment

*Task 4: Coffee Delivery*



---

Sequential Goal

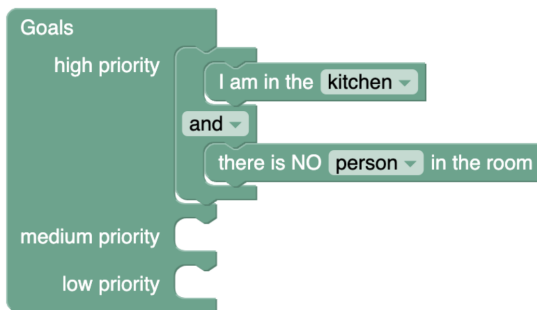
*Task 6: Mail on Porch*



---

No Priority

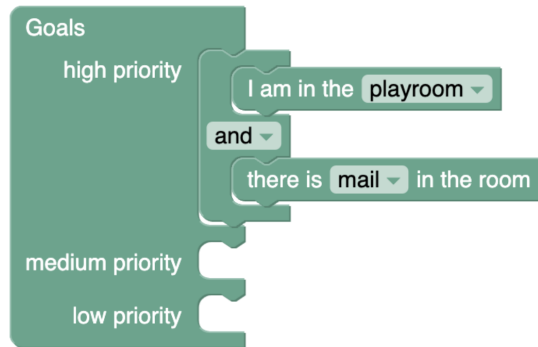
*Task 7: Person Avoidance While in Kitchen*



---

Counterintuitive Goal

Task 6: Mail on Porch



---

Wrong Use of 'AND'

Task 8: Coffee-or-Mail (Same Room)

