Introduction to Computer Science with MakeCode for Minecraft

Lesson 9: Artificial Intelligence

In this chapter, we'll dive into the popular field of Artificial Intelligence, or "AI". From driverless cars, to robots who beat humans at Chess and Jeopardy, the field of artificial intelligence is one of the most exciting and promising areas of computer science. The art and science of crafting programs that mimic, and even surpass human intelligence, is tremendously important. But there are also some ethical questions, and fears when it comes to AI. There have been many science fiction books and futuristic movies made about machines and robots that take over the world (i.e. The Terminator, The Matrix, etc.)

	IBM's Deep Blue computer was the first computer to beat a human at chess – in 1997 Deep Blue defeated the World Chess Champion Garry Kasparov. There was a documentary film called Game Over: Kasparov and the Machine that was made about this match.

Here are some good prompts to start a class discussion around Artificial Intelligence:

1. What are some criteria you would use to classify a computer as "intelligent"?
   - There is no one definition, but researchers agree on some common traits:
     - Ability to make 'smart' decisions
     - Ability to learn and increase knowledge
     - Ability to imitate humans (language/speech, vision/image recognition)
   - The computer scientist, Alan Turing invented the "Turing Test" that was an effort to create a test that would determine if a machine was intelligent
2. What are some things that computers could do if we had better AI programs?
   - Some examples: diagnose diseases, drive our cars, fly airplanes, order groceries for us, do our laundry, be our personal translator when we travel, do our banking and money management, etc.
3. If a computer is intelligent, does that mean it has its own consciousness, meaning, is it self-aware?

- AI doesn't necessarily mean that a computer can feel, or that it has its own personality. However, many researchers have been studying whether there are "robot rights" that we need to be aware of.

4. What are some fears that humans have about AI? How real do you think these fears are?
    - Some examples: robots will take away all human jobs, computers will become smarter than humans and humans will become their slaves, AI robots "cyborgs" will become an enhanced race of humans

Side bar: Artificial Intelligence Research

Companies like Microsoft are investing in Artificial Intelligence Research. Ofer Dekel is an AI Researcher at Microsoft who focuses on a field of AI called "Machine Learning". Here are some of his thoughts.

What are the kinds of AI research you do in your job at Microsoft?

What is the most rewarding part of your work?

How did you get involved in artificial intelligence? What were your favorite subjects in school?

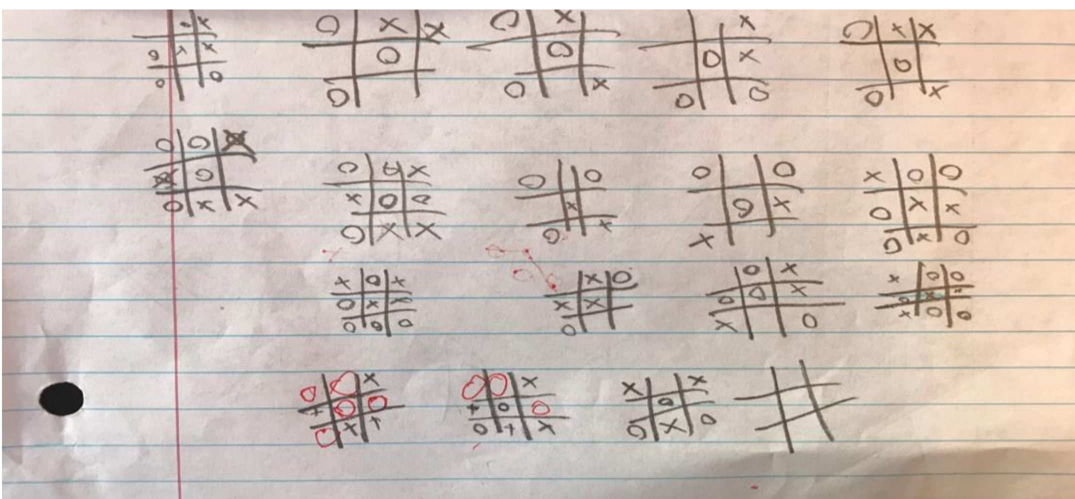
What advice do you have for students who are interested in coding and computer science?

Unplugged activity: Paper AI

Tell the students that they are going to play a game that they are probably all so familiar with that they may have stopped playing it altogether because it's not a challenge anymore. It's Tic-Tac-Toe!

- Have the students pair with a classmate and play a few rounds of tic-tac-toe. They usually enjoy this since for most of them it has been a while since they've played.
- Ask the students how much they thought about the decisions they were making while they played—decisions about where to place their mark next and how their decisions may change depending on the other player's moves. For most students, since they are so familiar with the game, they feel like they aren't thinking too much about their moves.

- Have them play a few more rounds of tic-tac-toe, but this time be aware of their thinking. Why are they making the moves that they are, and how do their opponent's moves affect their moves?
- Ask them to share some of the thinking that went into deciding where to place their next mark.
- Challenge them to come up with a list of rules/strategies that anyone could follow to win at tic-tac-toe. Tell them they can use if-else structures to describe their rules. What they're writing is pseudocode… a mix of English and code.
- To get them going with their list of rules, as a class, have them suggest a rule for the very first move. Most will say, if the center spot is open, place your mark in the center spot.
- Ask them to think about their next move, and then the one after that… continuing their list of rules/strategies until they think their rules would help anyone win every time.
- This is usually where the students realize just how much thinking actually goes into their decision making. This activity really challenges most students, even those that thought at first it would be easy!
- Have each student swap their finished list of rules with their classmate and play a few more rounds of tic-tac-toe against each other. The important difference in these rounds is that each student must use only the set of rules written by their classmate to determine their next move. They may not use their own rules or strategies. They enjoy this activity and they begin to realize more about the game and what it takes to write rules that result in a win every time.
- Ask them to imagine what it takes to program a computer to play checkers or chess!
- Optional challenge: Write a list of rules/strategies to play a connect four in a row type game. Try it out with a classmate.



Tic-Tac-Toe game

Pseudo-code Rules of Tic-Tac-Toe

Here are some Sample AI rules for Tic-Tac-Toe:

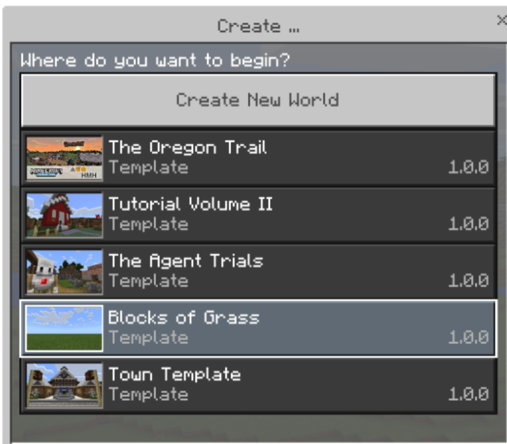
- Move 0: Place your mark in the center.
- Move 1:
    - If opponent places their mark adjacent to center, then place your next mark in a corner.
    - Else place your mark adjacent to center.
- Move 2:
    - If there are two of your marks in a row, column, or diagonal and the third space in that row, column, or diagonal is an empty space, place your mark there for the win.
    - Else, place your mark in the other corner.
- Move 3 and beyond:

- If there are two of your marks in a row, column, or diagonal and the third space in that row, column, or diagonal is an empty space, place your mark there for the win.
- Else, if your opponent has two of their marks in a row, column, or diagonal and the third space in that row, column, or diagonal is an empty space, place your mark there for the block.
- Else, go in any empty space.

Activity: Maze Generation

Learning how to read and learning how to write go hand in hand. It is good practice to look at other people's programs, and try to read and understand how they work, in order to become better at writing code yourself. For this activity, instead of writing code, we will explore an existing program for creating a maze.

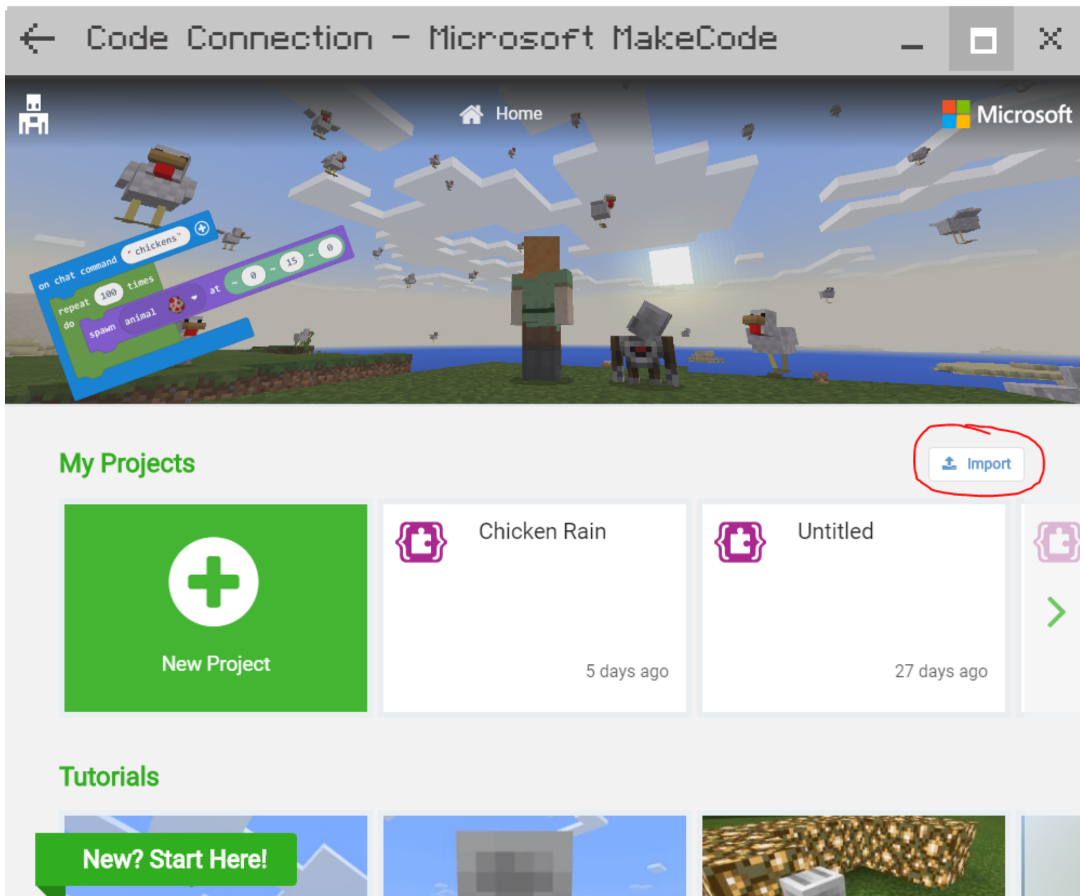
- Launch Minecraft and create a new Flat world in creative mode



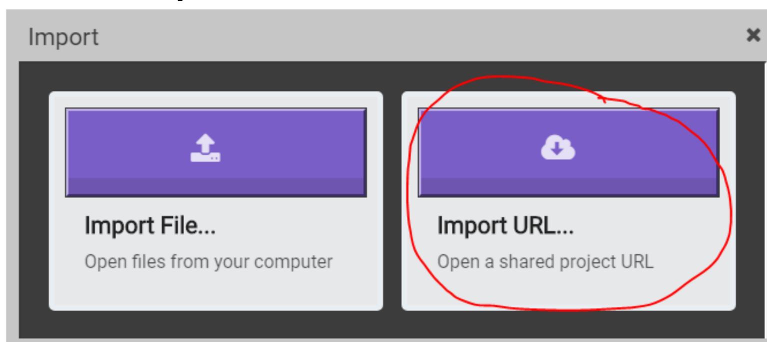
You can use the Blocks of Grass template in Minecraft: Education Edition

In Minecraft Windows 10, create a New World and set Game Mode to Creative, and World Type to Flat

- Open MakeCode in the Code Connection app
- Select the **Import** button in MakeCode



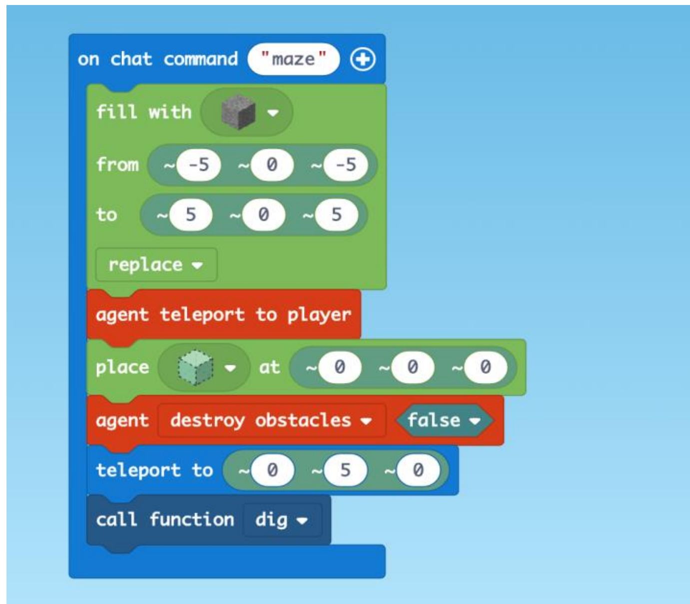
- Select the **Import URL** card



- Paste this URL in the text box: https://makecode.com/_U4aDvfCW665g
- Give the Agent some stone in its inventory, in the upper left inventory slot

- Type the command “maze” in the chat window to see the program run

Now let’s look at the code, and in a separate document, do your best to answer the following questions about this part of the program:



Questions:

1. Describe what the Fill With block does.
2. Why do we Place a block of Air at ~0 ~0 ~0?
3. Why do you think we set Agent destroy obstacles to false?
4. What is the difference between the two Teleport commands?

Answers:

1. The Fill With stone block creates a 10 x 10 block of stone around the Player's location. The Agent will carve the maze out of this stone.
2. The Agent starts at the center of the block and carves out, so first we teleport the Agent to the Player's coordinates, then place a block of air at that spot.
3. We set Destroy obstacles to false because we don't want the Agent to destroy the blocks in front of it until we have a chance to inspect them. This helps us determine whether we are in an existing maze pathway or not.
4. The first teleport block teleports the Agent to the Player's location, which is at the center of the maze. The second teleport block teleports the Player to a spot five blocks above the maze, for a bird's-eye view.

Answer the following questions about this section of code:

Questions

1. What is the purpose of the "shuffle" function?
2. What is the "dirs" variable?
3. Why do we use two different index variables – "index" and "index2"?
4. What is the purpose of the "temp" variable?

Answers

1. The "shuffle" function creates an array of directions and then shuffles them to create a random order of directions. This will create a random, meandering path through the maze.
2. The "dirs" variable is meant to specify a direction. It holds an array with the directions forward, left and right.
3. The index and index2 variables each specify a random element of the dirs array. The code then swaps those two values in the array to 'shuffle' them.
4. In order to properly swap two values, we need a third temporary variable to store one of them while we copy the other one over.

Maze generation algorithms are a popular programming exercise because there are as many different approaches as there are types of mazes! It's also fun to see the maze as it's being

created. The maze generation routine we are using here is adapted from an algorithm called "recursive backtracking".

Here is the pseudocode:
Shuffle the array:

- Create an array of three directions
- Randomize their order

For each value in the array:

- If it's left, turn left
- If it's right, turn right
- If there's a wall in front of us (empty on the other side), preserve the wall
- If there's no wall in front of us, dig a path forward (recursively by calling the same function "dig")
- Back up 2 spaces
- Turn back to original direction

The main idea is that the Agent will continue to carve an erratic path through the solid block until it reaches a blank space, trying not to destroy walls between passages. Notice that at the very end it calls the same function again in order to restart the maze carving. This is a special form of iteration known as recursion.

Optional Extension:
Pass in a parameter corresponding to the side length of the maze so that the Builder creates a maze of any size you want.

The Agent still gets stuck from time to time. Try to modify the main maze generating code to improve the algorithm.

Activity: Maze Pathfinding

Now let's teach the Agent how to navigate a maze on its own. Rather than giving the Agent a set series of directional commands to solve one particular maze, we are going to need to use conditional statements to teach the Agent how to find its through any maze, intelligently.

We will need a maze to start out with. You can either use the maze generation program above, and have the Agent create a maze for you, or you can create your own maze (which is lots of fun, too!) The Agent will need to know when it has successfully completed the maze, so figure out a starting point and an ending point, and place a redstone block underneath the ending point. We'll have the Agent stop when it detects redstone under its feet.

<u>The Algorithm</u>

In any maze or labyrinth, you can always find your way out by following one wall consistently. You may not find the most direct way out, but you will always come out on the other side. In a Minecraft maze, this basically means always turn left whenever you can, and when you reach a dead end, turn around. So, our simple maze following algorithm looks like:

- As long as you aren't standing on redstone:
    - If there is no block to the left of you, turn left and move forward.
    - Otherwise, if there is no block in front of you, move forward.

- Otherwise, if there is no block to the right of you, turn right and move forward.
- Otherwise, turn around and move forward.

Try to code this on your own before following the steps below if you get stuck.

Steps:

1. Create a new MakeCode project called "Pathfinder"
2. Rename the existing On chat command block to "maze"

We'll put the whole algorithm in a while loop, which means do this as long as you aren't standing on redstone.

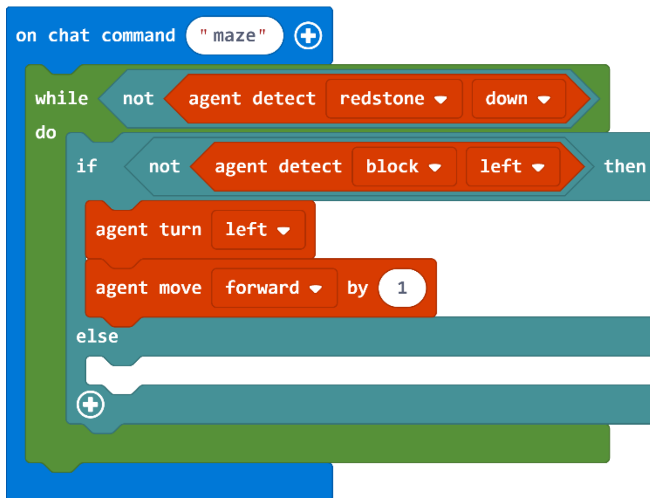
3. From the Loops Toolbox drawer, drag a While loop under the On chat command block
4. From the Logic Toolbox drawer, a Not block into the While loop replacing the true block
5. From the Agent Toolbox drawer, drag an Agent Detect block into the Not block
6. In the Agent Detect block, use the drop-down menu to select 'redstone' as the type of block to detect
7. In the Agent Detect block, use the drop-down menu to select 'down' as the direction

This essentially means that while we don't detect Redstone beneath us, we will run the following code. Next, we'll check to see if there is a path open to the left and if so, turn left and move forward.

8. From the Logic Toolbox drawer, drag an If then else block into the While loop
9. From the Logic Toolbox drawer, drag a Not block into the If then conditional slot replacing the true block
10. From the Agent Toolbox drawer, drag an Agent Detect block into the Not block
11. In the Agent Detect block, use the drop-down menu to select 'left' as the direction to look

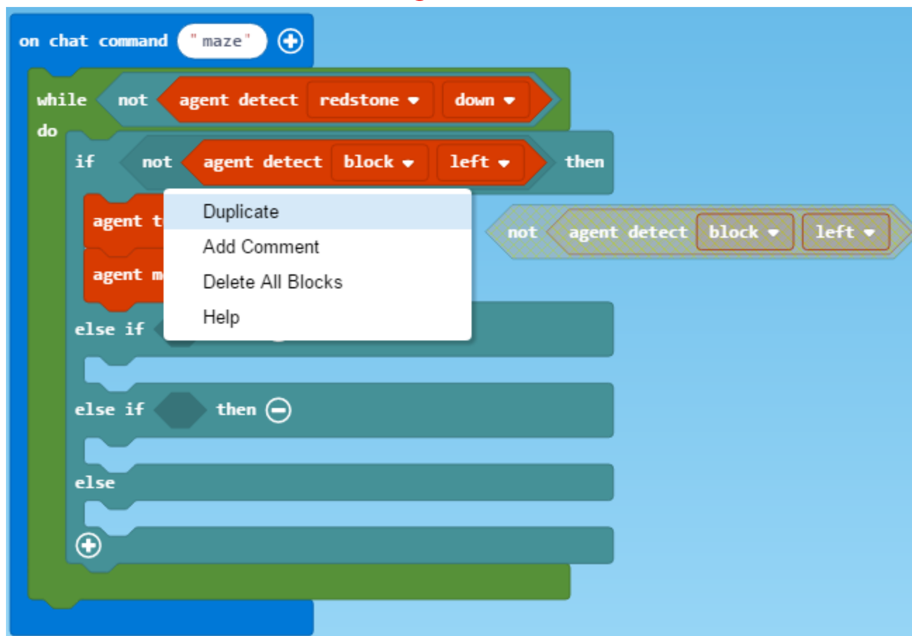
If the Agent doesn't detect a block to its left, then we want the Agent to turn left and move forward.

12. From the Agent Toolbox drawer, drag an Agent turn, and an Agent move block into the If then clause of our conditional block

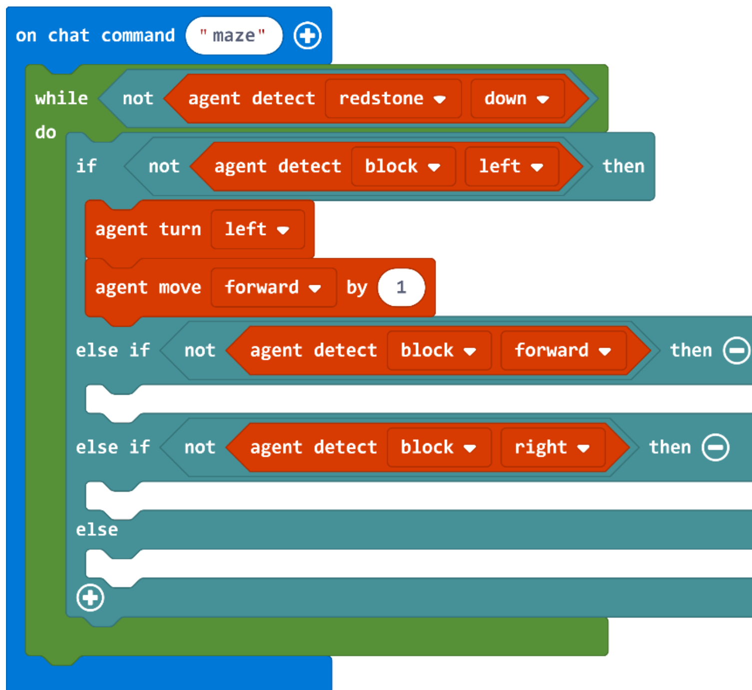


Now we need to add 2 more conditions to check forward and to the right of the Agent.

13. Click the Plus (+) sign at the bottom of the If then else block twice, to add 2 more 'Else if' clauses
14. Right-click on the Not block in the first If then conditional slot, and select Duplicate – do this twice to create two more Not Agent Detect conditions



15. Drop these Not Agent Detect conditions into the two additional Else if slots
16. In the first of these Not Agent Detect blocks, use the drop-down menu to select 'forward' as the direction to check
17. In the second Not Agent Detect block, use the drop-down menu to select 'right' as the direction to check

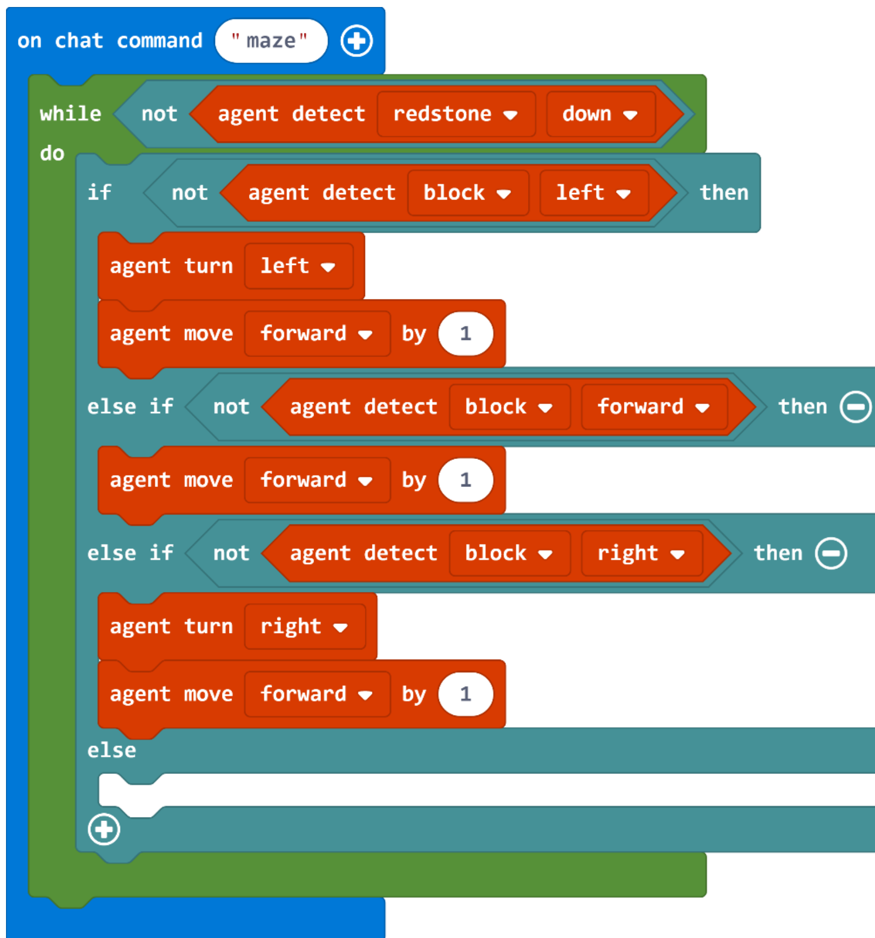


If the Agent doesn't detect a block in front of it, then we should simply move the Agent forward.

18. From the Agent Toolbox drawer, drag an Agent move block into the first Else if clause

If the Agent doesn't detect a block to its right, then we should turn the Agent right and move forward.

19. From the Agent Toolbox drawer, drag an Agent turn, and an Agent move block into the second Else if clause
20. In the Agent turn block, use the drop-down menu to select 'right' as the direction to turn

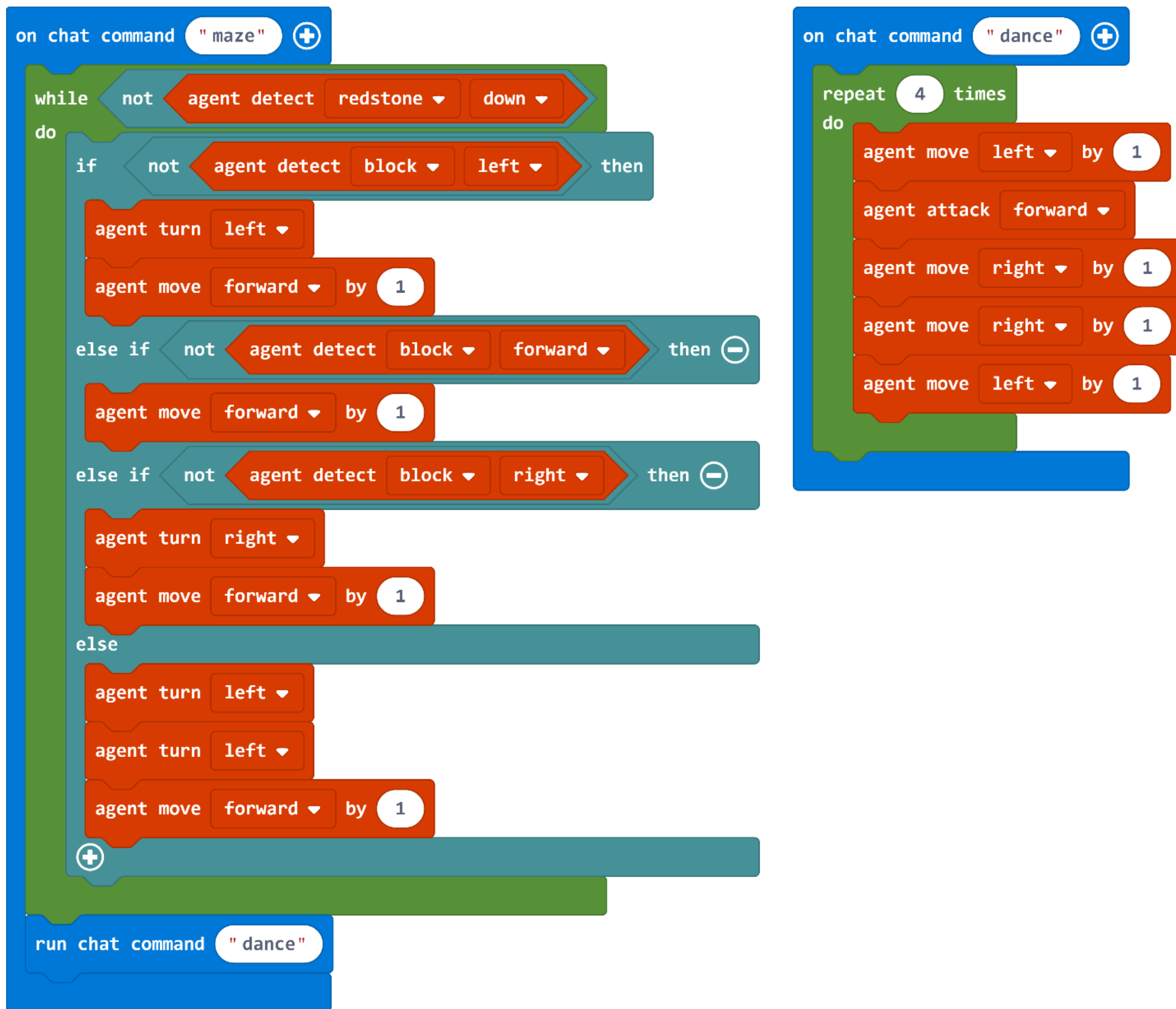


Finally, if the Agent is at a dead-end (none of the three directions are open), we want her to turn around and head back in the opposite direction.

21. From the Agent Toolbox drawer, drag the following 3 blocks into the last Else clause: 2 Agent turn blocks, and an Agent move block

If the Agent detects Redstone, it will know it has reached the finish, so you can even have it run a chat command “dance” and do a happy dance at the end of the maze! Feel free to reuse the code from the Dance Dance Agent activity in Lesson 5 Iteration.

Final program:



JavaScript:

```
player.onChat("maze", function () {
    while (!(agent.detect(AgentDetection.Redstone,
SixDirection.Down))) {
        if (!(agent.detect(AgentDetection.Block,
SixDirection.Left))) {
            agent.turn(TurnDirection.Left)
            agent.move(SixDirection.Forward, 1)
        } else if (!(agent.detect(AgentDetection.Block,
SixDirection.Forward))) {
            agent.move(SixDirection.Forward, 1)
```

```
        } else if (!(agent.detect(AgentDetection.Block,
SixDirection.Right))) {
            agent.turn(TurnDirection.Right)
            agent.move(SixDirection.Forward, 1)
        } else {
            agent.turn(TurnDirection.Left)
            agent.turn(TurnDirection.Left)
            agent.move(SixDirection.Forward, 1)
        }
    }
    player.runChatCommand("dance")
})
player.onChat("dance", function () {
    for (let i = 0; i < 4; i++) {
        agent.move(SixDirection.Left, 1)
        agent.attack(SixDirection.Forward)
        agent.move(SixDirection.Right, 1)
        agent.move(SixDirection.Right, 1)
        agent.move(SixDirection.Left, 1)
    }
})
```

Shared Program: https://makecode.com/_K5TeAxH6e1zk

Paired Independent Project: Build an AI

For an independent project, now it's your turn to work with another student to teach the Agent or the Builder to intelligently adapt to the Minecraft environment. You can use conditional statements to allow the Agent or the Builder to evaluate its environment and make its own decisions accordingly.

For certain tasks, you may want to use the Builder instead of the Agent because the Builder is faster and can save its current position. However, the Builder is also invisible, so it can be difficult to see exactly what it is doing while it is working.

Pair Brainstorm:
First, sit with your partner and brainstorm about five or six common tasks in Minecraft that might be automated. Here are some ideas as a starting point:

- Finding temples
- Finding villages
- Finding dungeons
- Harvesting mature wheat
- Harvesting tall sugar canes
- Finding the way out of a cave

You and your partner should pick two different tasks from your list that would require the Agent or the Builder to evaluate its environment and react differently. Try to write out, in pseudocode, what the steps would be to accomplish each task.

Here is one example, for finding jungle temples. The Builder will fly over the treetops and examine a given search area beneath the tree line until it finds moss stone above ground, which is likely to be the site of a jungle temple. This is similar to the tree cover-penetrating radar that archaeologists use to find real temples in the Amazon!
<u>Pseudocode:</u>

- Create a variable for steps
- Create a variable for turns
- Teleport the Builder to my position
- Ascend to 20 blocks above my position
- While a test for moss stone in an area between 20 and 15 blocks below the Builder is negative:
    - Change the number of steps by 1
    - If the number of steps is greater than the search area:
        - Alternate turning right or left based on the value of turns
        - Reset steps to zero
        - Flip the value of turns (i.e., if it's true, make it false, and vice versa.)
    - Move forward to search a new area
- If you have found moss stone, report your current position since moss stone above ground level is the likely spot of a jungle temple.

You can use a similar algorithm for finding strongholds or dungeons. It is a good idea to have the Builder report its position every so often (using a Say block) so you have an idea of where it is and what it is doing.

As a pair, decide which one of these tasks seems the most promising, and then go ahead and build it in MakeCode. Be sure to test it out and make adjustments as necessary!

Minecraft Diary

Compose a diary entry addressing the following:

- What Minecraft problem did you decide to solve? What does your program do?
- Describe how your AI figures out how to perform the task or solve the problem
- Discuss one (or more) ways that working with a partner was different from just doing the project by yourself.
- Describe one point where you got stuck. Then discuss how you figured it out.
- Include at least one screenshot of your program in action.
- Share your project to the web and include the URL here.

NOTE: If you decided to improve one of this lesson's activities, please talk about the new code you wrote in addition to what was already provided in the lesson.

Assessment

	1	2	3	4
Diary	Minecraft Diary entry is missing 4 or more of the required prompts.	Minecraft Diary entry is missing 2 or 3 of the required prompts.	Minecraft Diary entry is missing 1 of the required prompts.	Minecraft Diary addresses all prompts.
Logic	Agent and/or Builder never completes its assigned task.	Agent and/or Builder completes its assigned task some of the time and in some situations.	Agent and/or Builder completes its assigned task most of the time in most situations.	Agent and/or Builder completes its assigned task in all circumstances all the time.

Project	Project code is greatly cumbersome and inefficient.	Project solves a specific problem but execution of the task is cumbersome or inefficient.	Project solves a specific problem mostly efficiently.	Project solves a specific problem, efficiently and effectively.

CSTA K-12 Computer Science Standards

- Identify factors that distinguish humans from machines.
- Recognize that computers model intelligent behavior. (Examples: computer opponents in gaming, speech and language recognition, robotics, computer animation of real world systems.)

(Resource: The CS Standards Crosswalk with CSTA K-12 Computer Science Standards for State/District/Course Standards http://csta.acm.org/Curriculum/sub/K12Standards.html)

Additional CSTA Standards:

- CL.L2-03 Collaborate with peers, experts, and others using collaborative practices such as pair programming, working in project teams, and participating in group active learning activities.
- CL.L2-04 Exhibit dispositions necessary for collaboration: providing useful feedback, integrating feedback, understanding and accepting multiple perspectives, socialization.
- CL.L3A-01 Work in a team to design and develop a software artifact.
- K-12 Computer Science Framework Core concept: Control Structures
- CT.L2-12 - Use abstraction to decompose a problem into sub problems
- CPP.L1:6-05 - Construct a program as a set of step-by-step instructions to be acted out
- CPP.L1:6-06 - Implement problem solutions using a block-based visual programming language
- NGSS 3-5-ETS1-2 - Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem