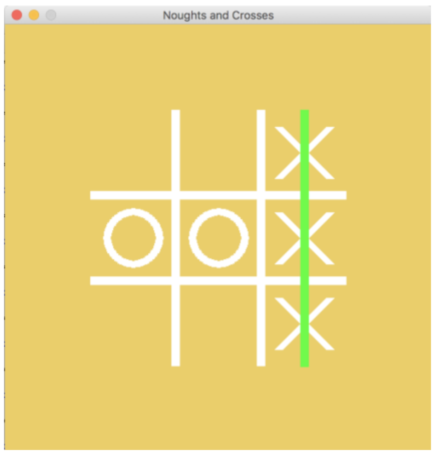
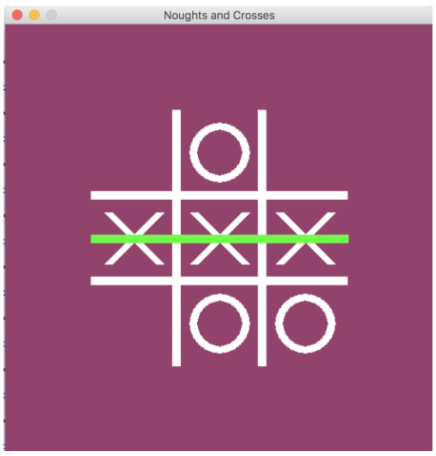
Noughts & Crosses

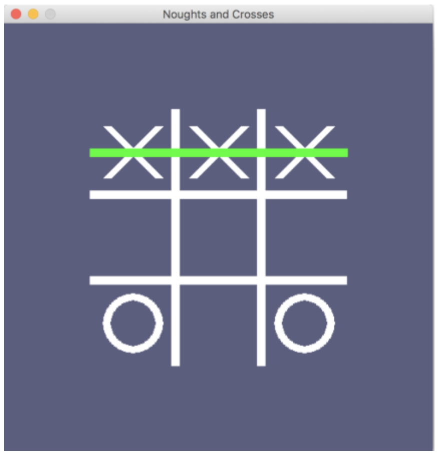
In this project you will create a noughts and crosses game in Python that is able to learn from how you play.

You won't give it instructions for how to play, or tell it what the objective or rules of the game are.

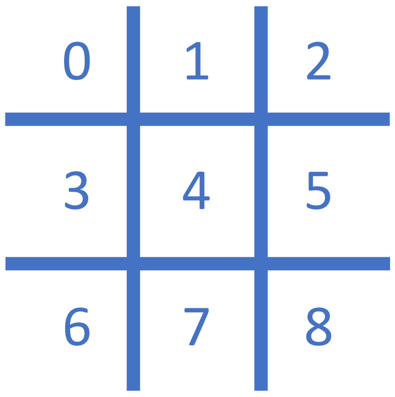
Instead, you'll show it examples of you playing the game. When it's seen enough examples to start trying to play for itself, you'll tell when it beats you.





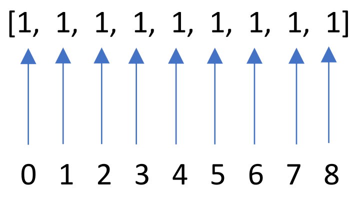


Representing noughts and crosses in Python



The positions of spaces on the noughts and crosses board are numbered from 0 to 8.

They are then stored in a list.

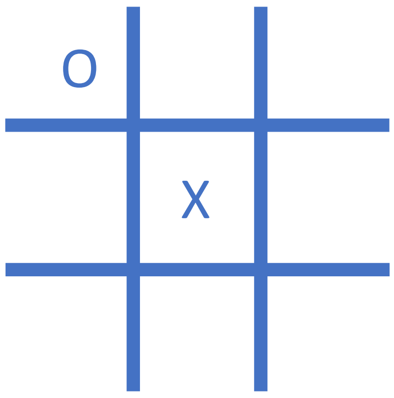


Empty = 1

O = 2

X = 3

An empty space is shown as a 1.
A nought O is shown as a 2.
A cross X is shown as a 3.



The board is represented as a list of 9 1's, 2's and 3's, one for each space

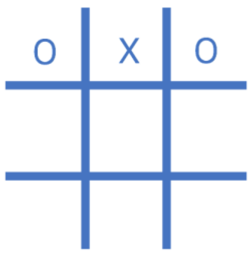
For example, at this point:

[2, 1, 1, 1, 3, 1, 1, 1, 1]

What are you going to do?

You're going to train a computer to play noughts and crosses. You'll do this by showing it examples of how you play the game.

Imagine the board looks like this and it's X's turn.

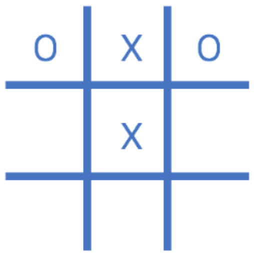


Imagine you decide to put your X in the centre space.

top-left	opponent
top-middle	player
top-right	opponent
middle-left	empty
middle-middle	empty
middle-right	empty
bottom-left	empty
bottom-middle	empty
bottom-right	empty

choice : middle-middle

Imagine the board looks like this and it's O's turn.



Imagine you decide to put your O in the bottom middle space.

top-left	player
top-middle	opponent
top-right	player
middle-left	empty
middle-middle	opponent
middle-right	empty
bottom-left	empty
bottom-middle	empty
bottom-right	empty

choice : bottom-middle

Using "opponent" and "player" instead of "nought" and "cross" means the computer can learn from both nought and cross moves.

You'll use examples of moves from the player that wins the game to train the computer.

If you (X) win, you'll use your moves as examples to train the computer. If the computer (O) wins, you'll use the computer's moves to train with.

These **examples of moves that lead to winning** will teach the computer how to play to win!

1. Go to https://machinelearningforkids.co.uk in a browser.

2. Click on “**Get started**”

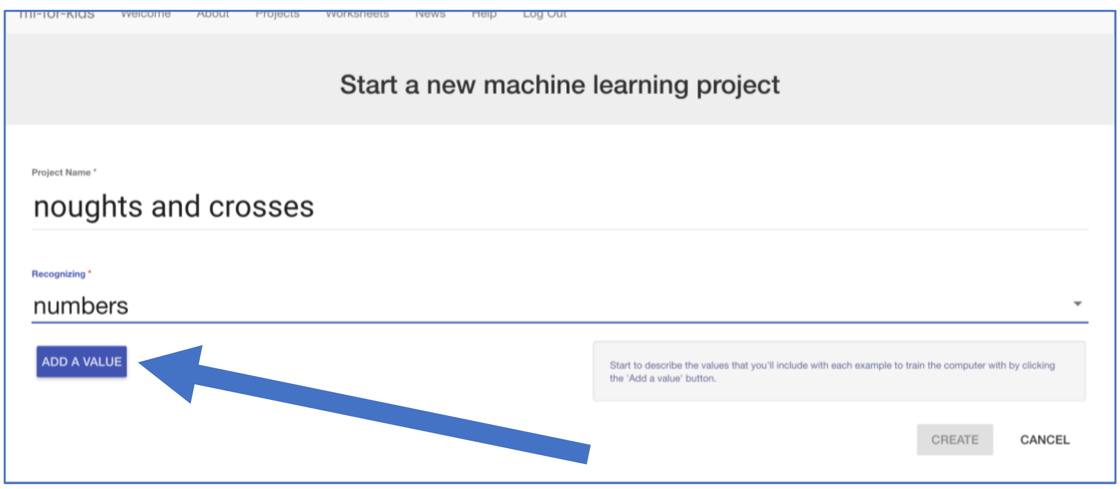
3. Click on “**Log In**” and type in your username and password
If you don’t have a username, ask your teacher to create one for you.
If you can’t remember your password, ask your teacher to reset it for you.

4. Click on “**Projects**” on the top menu bar

5. Click on the **“+ Add a new project**” button.

6. Name your project “noughts and crosses” and set it to learn how to recognise “**numbers**”

7. Click “**Add a value**”



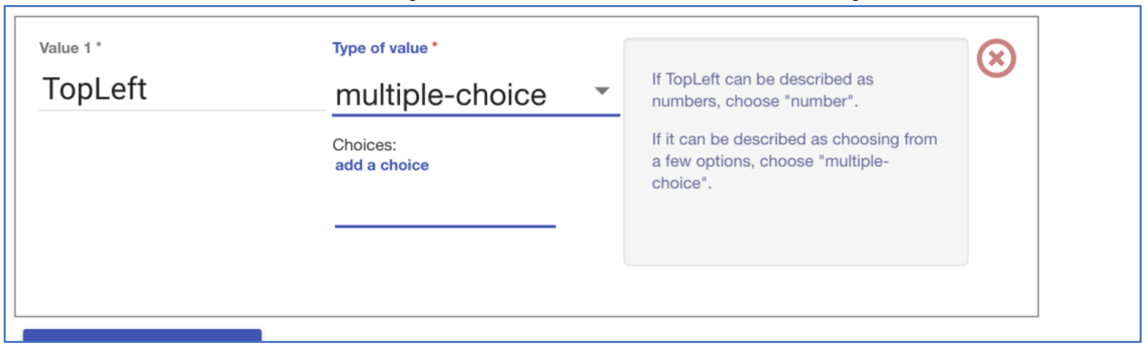
Start a new machine learning project

Project Name *
noughts and crosses

Recognizing *
numbers

Start to describe the values that you'll include with each example to train the computer with by clicking the 'Add a value' button.

8. Name a value “**TopLeft**” and make it “**multiple-choice**”



Value 1 *
TopLeft

Type of value *
multiple-choice

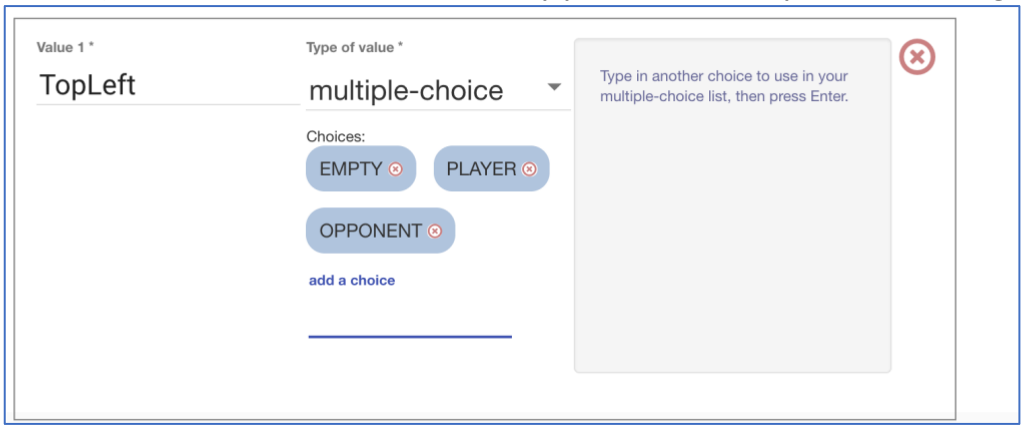
Choices:

If TopLeft can be described as numbers, choose "number".

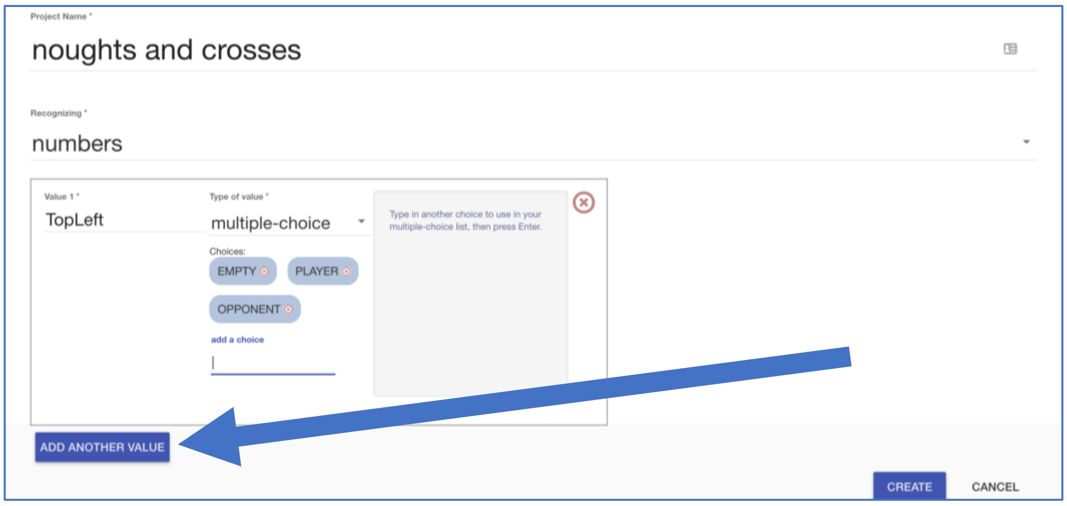
If it can be described as choosing from a few options, choose "multiple-choice".

9. Type "EMPTY" into the "**add a choice**" box and press Enter

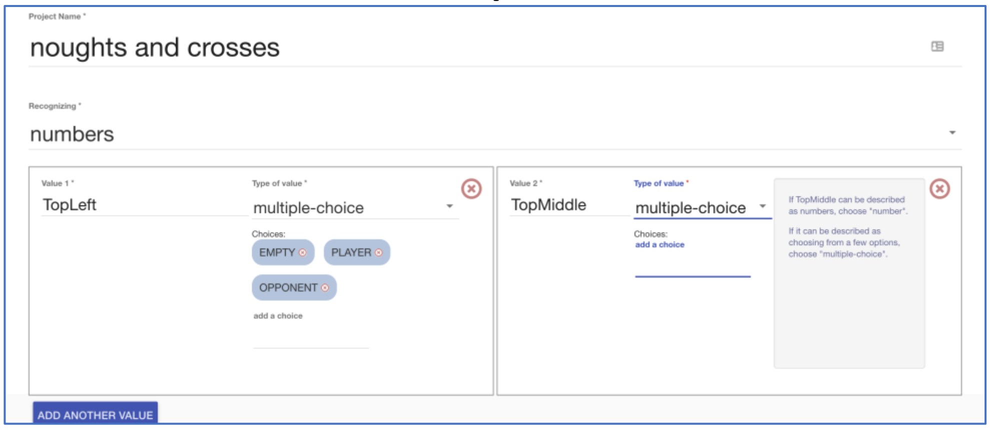
Type "PLAYER" into the "**add a choice**" box and press Enter
Type "OPPONENT" into the "**add a choice**" box and press Enter
These are the possible contents for the top-left space in the noughts and crosses board. It can be empty, or it can have the player's own shape (cross) in it, or it can have the opponent's shape in it (nought).



10. Click "**Add another value**"



11. Call the next value "**TopMiddle**" and make it "**multiple-choice**"

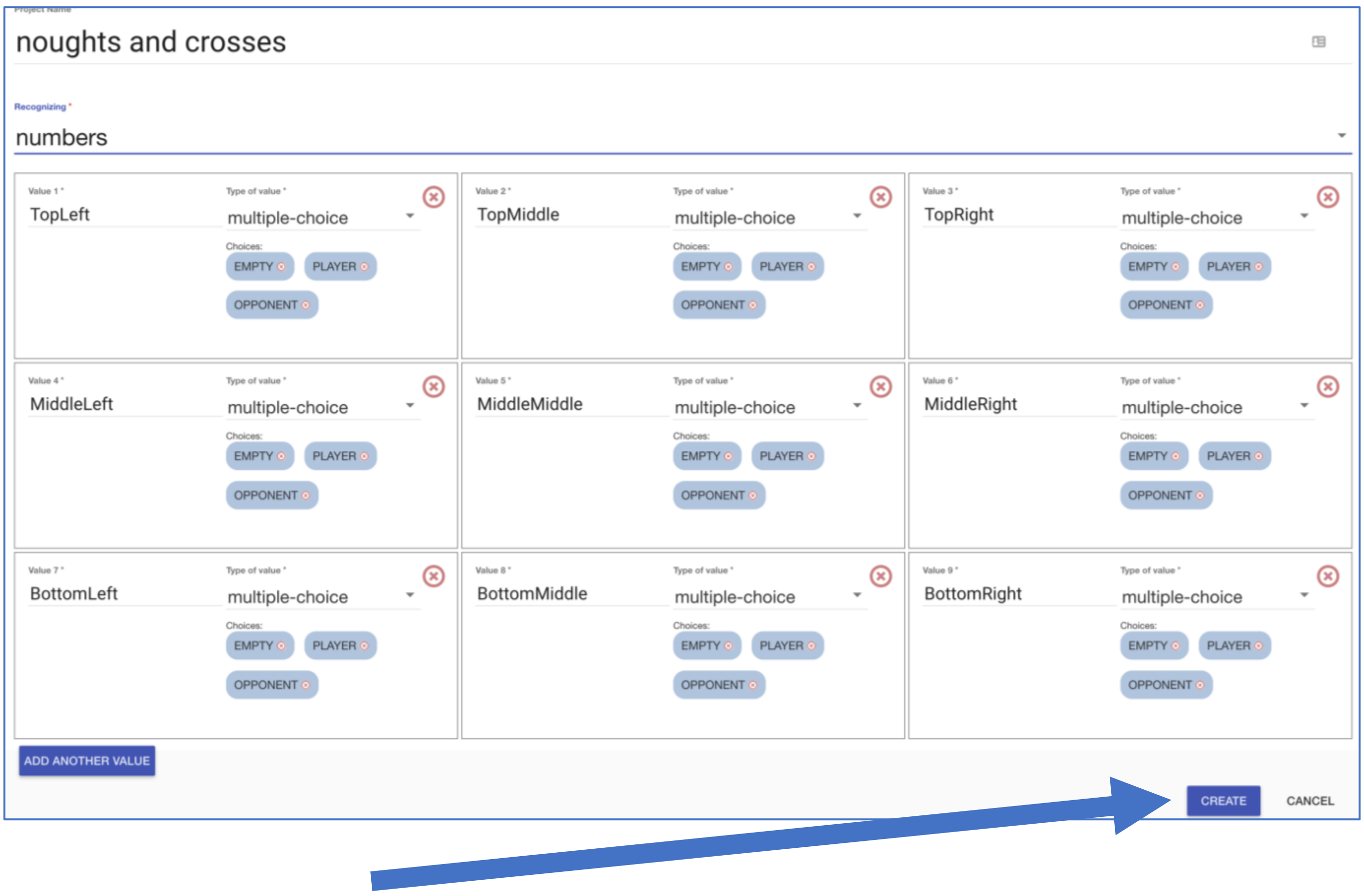


12. Add “EMPTY”, “PLAYER”, and “OPPONENT” to “TopMiddle” as you did to “TopLeft”

13. Repeat for the other positions on a noughts-and-crosses board
Each example is the state of the board before a move that led to a win.
TopLeft, TopMiddle, TopRight,
MiddleLeft, MiddleMiddle, MiddleRight,
BottomLeft, BottomMiddle, BottomRight

*It’s **very important** that you spell “EMPTY”, “PLAYER” and “OPPONENT” in the same way for all nine positions.*

14. Click “**Create**”



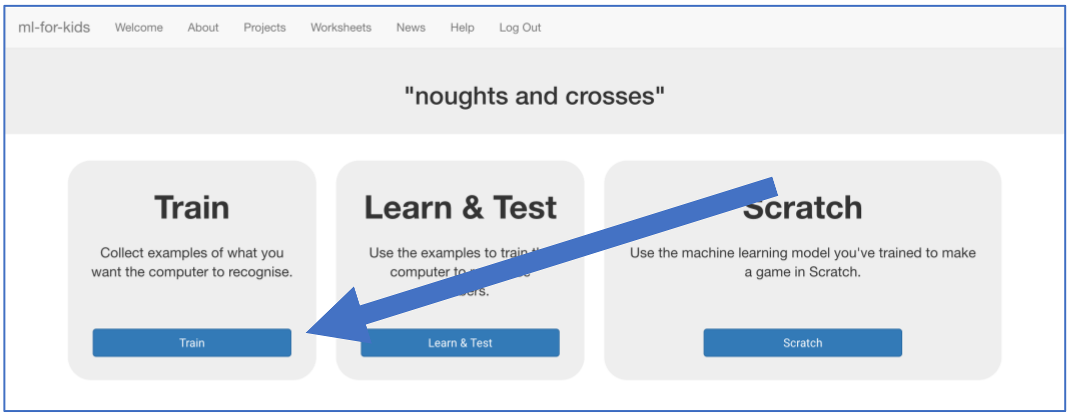
Project Name
noughts and crosses

Recognizing *
numbers

Value 1 * TopLeft; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT	Value 2 * TopMiddle; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT	Value 3 * TopRight; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT
Value 4 * MiddleLeft; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT	Value 5 * MiddleMiddle; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT	Value 6 * MiddleRight; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT
Value 7 * BottomLeft; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT	Value 8 * BottomMiddle; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT	Value 9 * BottomRight; Type of value * multiple-choice; Choices: EMPTY, PLAYER, OPPONENT

15. You should see “**noughts and crosses**” in your list of projects. Click on it.

16. Click the “**Train**” button



"noughts and crosses"

Train

Collect examples of what you want the computer to recognise.

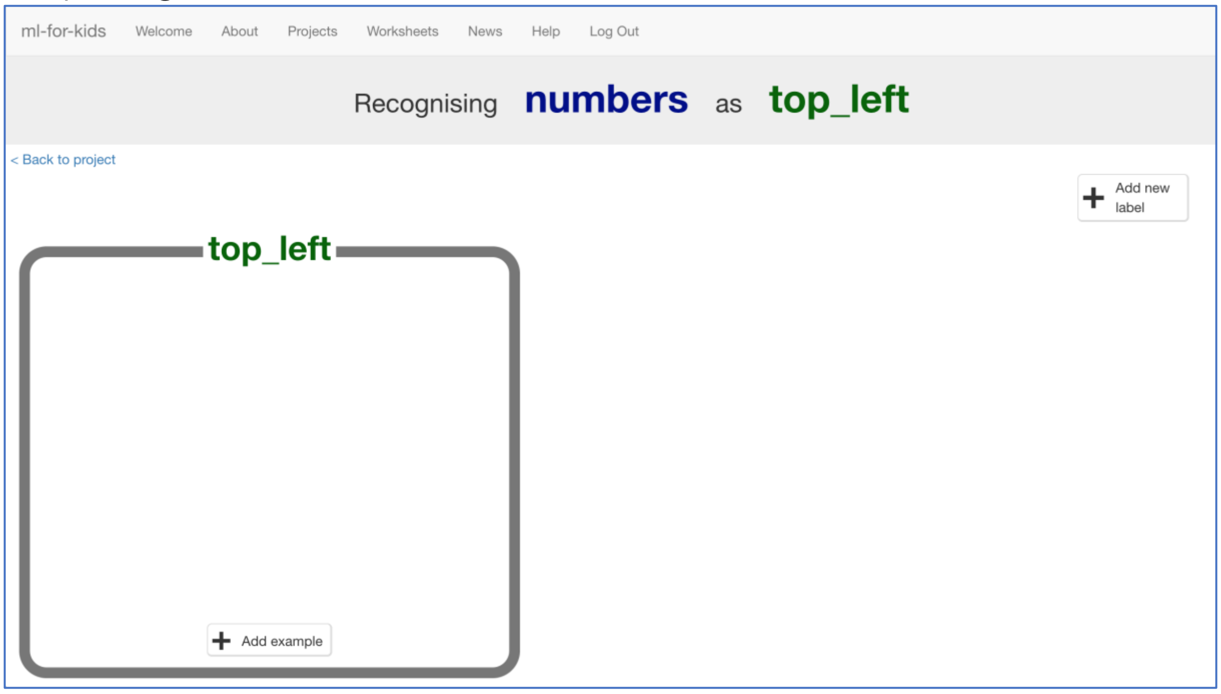
Learn & Test

Use the examples to train the computer to recognise numbers.

Scratch

Use the machine learning model you've trained to make a game in Scratch.

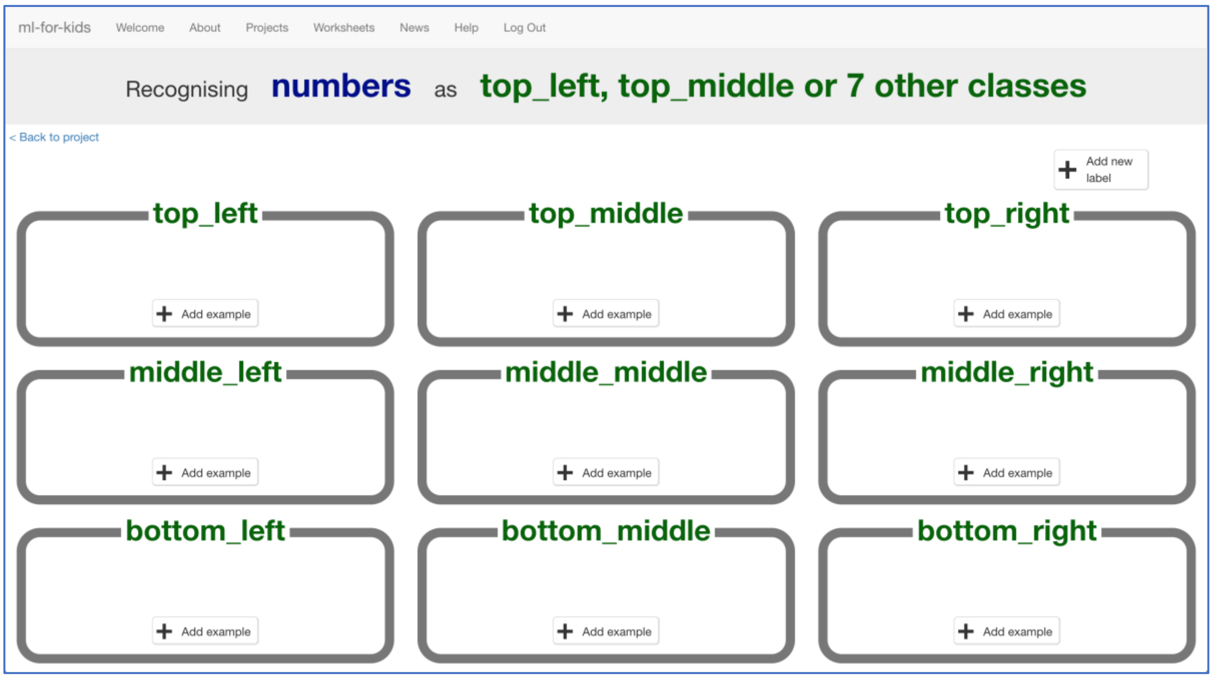
17. Click “**+ Add new label**” and create a label called “top left”
Examples of making a move in the top-left box (in games that lead to a win) will go in this bucket.



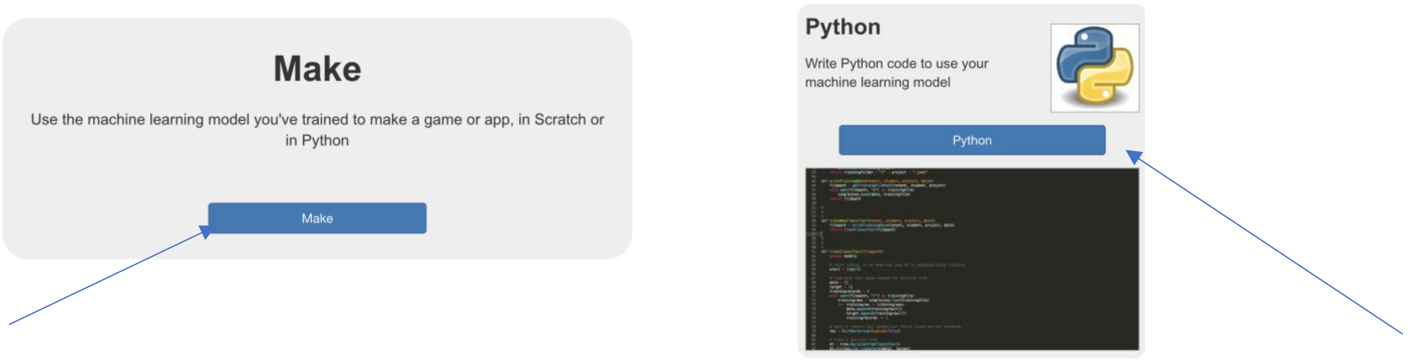
Recognising numbers as top_left

18. Click “**+ Add new label**” again and create labels for the other eight spaces on the board.
“top middle”, “top right”,
“middle left”, “middle middle”, “middle right”,
“bottom left”, “bottom middle”, “bottom right”

(see the next page for a picture)



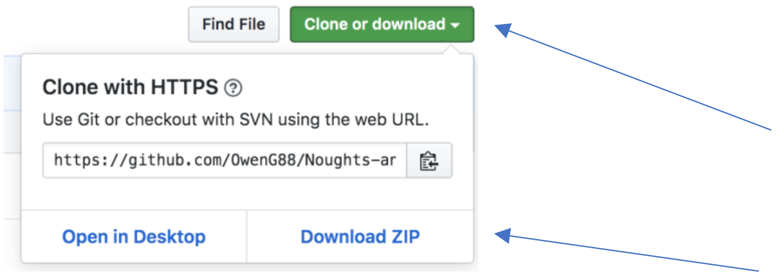
19. Click the "**< Back to project**" button. Click on Make. Click on Python.



Make
Use the machine learning model you've trained to make a game or app, in Scratch or in Python
Make
Python
Write Python code to use your machine learning model
Python

20. In a different tab open this link: https://github.com/OwenG88/Noughts-and-Crosses

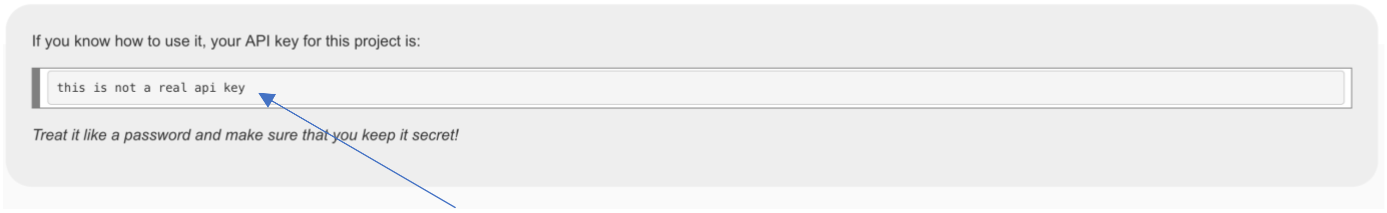
21. Click on the Clone or download button. Click download as ZIP.



Find File
Clone or download
Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/OwenG88/Noughts-ar
Open in Desktop
Download ZIP

22. Unpack the ZIP file and open the code in your Python editor of choice.

23. Go back to your Machine Learning for Kids tab. Find the API key and copy it.


If you know how to use it, your API key for this project is:

this is not a real api key

Treat it like a password and make sure that you keep it secret!

24. Paste the API key into the code into the empty string named key.

```
import pygame, random, time, sys, requests

convert = {1:"EMPTY", 2:"OPPONENT", 3:"PLAYER"}
deconvert = {"top_left":0, "top_middle":1, "top_right":2, "middle_left":3, "midd

global key
key = ""
```

25. Good job! Now we can finish the code.

26. Find the winner function.

```
def winner(board, savedmoves, movelocation, movelocations):
```

27. Find the Rows Section.

```
######## Rows ########
if board[deconvert["top_left"]] == i and board[deconvert["top_middle"]] == i and board[deconvert["top_right"]] == i:
    pygame.draw.line(screen, a, (100, 150), (400, 150), 10)
    whohaswon = i-1
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
    pygame.draw.line(screen, a, (100, 250), (400, 250), 10)
    whohaswon = i-1
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
    pygame.draw.line(screen, a, (100, 350), (400, 350), 10)
    whohaswon = i-1
```

28. This section of code checks to see if a player has successfully completed a row with three crosses or three noughts.

29. Change the second and third if statements so that they check for the middle and bottom rows.

From

```
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
    pygame.draw.line(screen, a, (100, 250), (400, 250), 10)
    whohaswon = i-1
```

To

```
if board[deconvert["middle_left"]] == i and board[deconvert["middle_middle"]] == i and board[deconvert["middle_right"]] == i:
    pygame.draw.line(screen, a, (100, 250), (400, 250), 10)
    whohaswon = i-1
```

From

```
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
    pygame.draw.line(screen, a, (100, 350), (400, 350), 10)
    whohaswon = i-1
```

To

```
if board[deconvert["bottom_left"]] == i and board[deconvert["bottom_middle"]] == i and board[deconvert["bottom_right"]] == i:
    pygame.draw.line(screen, a, (100, 350), (400, 350), 10)
    whohaswon = i-1
```

30. Now find the columns section.

```
######## Columns ########
if board[deconvert["top_left"]] == i and board[deconvert["middle_left"]] == i and board[deconvert["bottom_left"]] == i:
    pygame.draw.line(screen, a, (150, 100), (150, 400), 10)
    whohaswon = i-1
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] ==i:
    pygame.draw.line(screen, a, (250, 100), (250, 400), 10)
    whohaswon = i-1
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
    pygame.draw.line(screen, a, (350, 100), (350, 400), 10)
    whohaswon = i-1
```

31. This section of code checks to see if a player has successfully completed a column with three crosses or three noughts.

32. As before, change the second and third if statements so they check for the middle and bottom columns.

From

```
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] ==i:
    pygame.draw.line(screen, a, (250, 100), (250, 400), 10)
    whohaswon = i-1
```

To

```
if board[deconvert["top_middle"]] == i and board[deconvert["middle_middle"]] == i and board[deconvert["bottom_middle"]] ==i:
    pygame.draw.line(screen, a, (250, 100), (250, 400), 10)
    whohaswon = i-1
```

From

```python
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
    pygame.draw.line(screen, a, (350, 100), (350, 400), 10)
    whohaswon = i-1
```

To

```python
if board[deconvert["top_right"]] == i and board[deconvert["middle_right"]] == i and board[deconvert["bottom_right"]] == i:
    pygame.draw.line(screen, a, (350, 100), (350, 400), 10)
    whohaswon = i-1
```

33. All that is left is to fill in the diagonals. Locate the diagonals section.

```python
######## Diagonals #########
if board[deconvert["top_left"]] == i and board[deconvert["middle_middle"]] == i and board[deconvert["bottom_right"]] == i:
      pygame.draw.line(screen, a, (100, 100), (400, 400), 15)
      whohaswon = i-1
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
      pygame.draw.line(screen, a, (400, 100), (100, 400), 15)
      whohaswon = i-1
```

34. This is the positive diagonal. Fill in the gaps in the if statement as shown below.

From

```python
if board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i and board[deconvert["edit_here"]] == i:
      pygame.draw.line(screen, a, (400, 100), (100, 400), 15)
      whohaswon = i-1
```

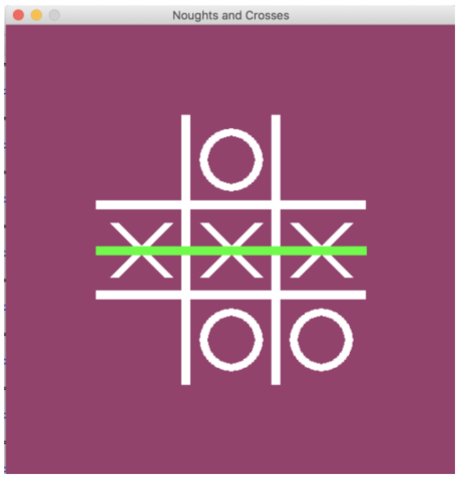
To

```python
if board[deconvert["top_right"]] == i and board[deconvert["middle_middle"]] == i and board[deconvert["bottom_left"]] == i:
      pygame.draw.line(screen, a, (400, 100), (100, 400), 15)
      whohaswon = i-1
```

35. Now that you have finished the code make sure you have **requests** and **pygame** installed. You can find out by trying to run the code.

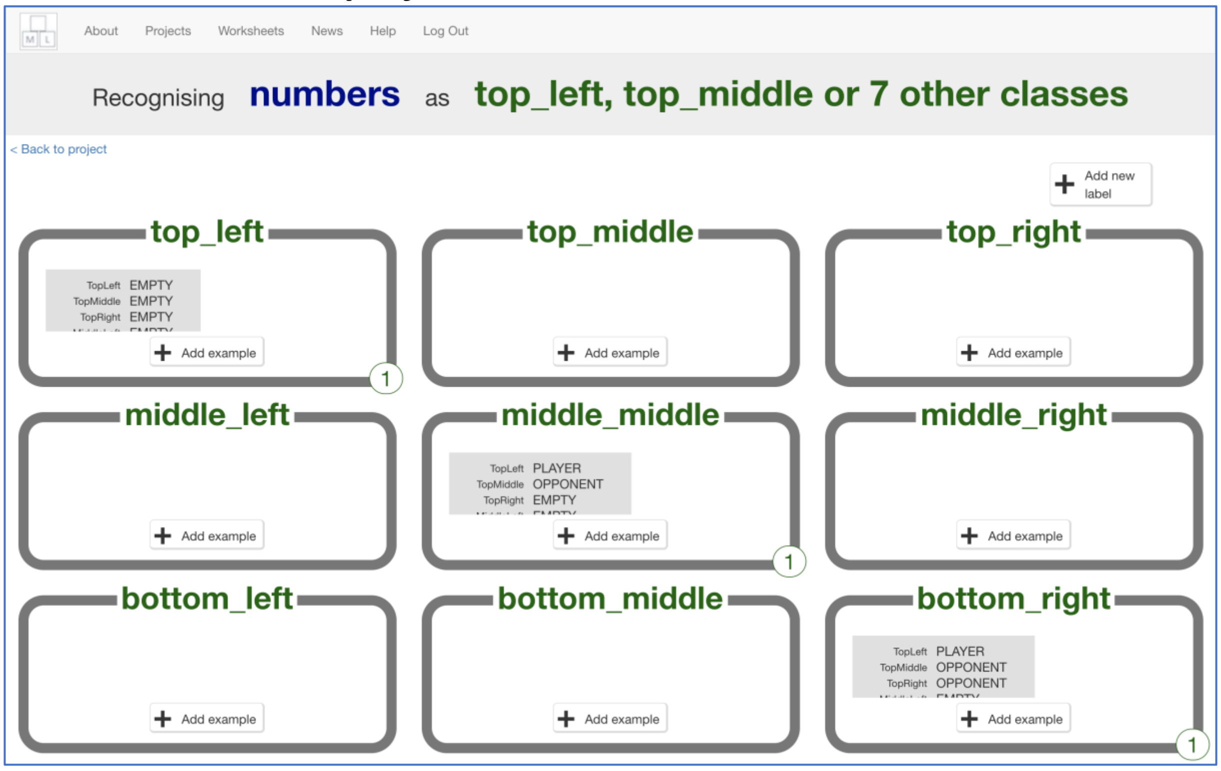
36. If you get an error, you will need to install requests and pygame. Ask your teacher whether or not they are installed.

37. Once there are no errors a window should appear. This is where you will play noughts and crosses. Play one game against the computer. **Close** the window that appears by clicking the red cross after you have finished playing.



38. Go back to the training page

Leave the Python window to go back to the training tool window.
*Click the "**< Back to project**" link and then click "**Train**"*



Recognising **numbers** as **top_left, top_middle or 7 other classes**

< Back to project

Add new label

top_left

TopLeft EMPTY
TopMiddle EMPTY
TopRight EMPTY

Add example

1

top_middle

Add example

top_right

Add example

middle_left

Add example

middle_middle

TopLeft PLAYER
TopMiddle OPPONENT
TopRight EMPTY

Add example

1

middle_right

Add example

bottom_left

Add example

bottom_middle

Add example

bottom_right

TopLeft PLAYER
TopMiddle OPPONENT
TopRight OPPONENT

Add example

1

39. Look at your training so far

Each item is a move made by the winning player.
The details in each item describe the state of the board at the time the winning player made that move.

40. Go back to Python window

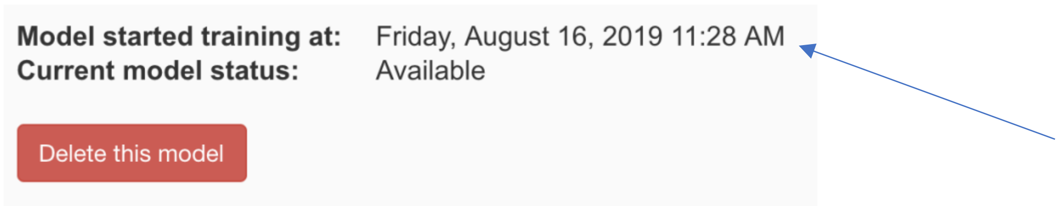
41. Play several more games – you want lots of training examples to teach the computer how to play the game

42. The code is set up so that it will automatically train a model for you about every 10 games.

43. As you carry on playing you should notice it start to get better.

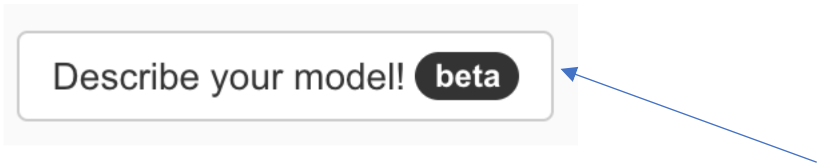
44. If you want to check when it last trained a model you can go back to the website and open the Learn & Test page. *From the Training window Click the "**< Back to project**" link and then click "**Learn and Test**"*

45. You should now be able to see the time it last trained a model.



Model started training at: Friday, August 16, 2019 11:28 AM
Current model status: Available

46. If you want to click the 'Describe your model!' button you can. It will help you understand how the computer makes its decisions.



47. Now just carry on playing until you start to lose!

What have you done?

You’ve trained a computer to play noughts and crosses.

You didn’t have to describe the rules to the computer.
You didn’t tell it that it should try to get three noughts in a row.
You didn’t describe the difference between rows, columns or diagonals.
(The rules are in the Python game, but that doesn’t count – that wasn’t used in the machine learning model).

Instead, you showed it how you play, by collecting examples of decisions that you made when you win.

When it makes decisions that leads to it winning, this is added to its training data, so it can be even more confident in that approach in future.

This is called “reinforcement learning” because when it does something good you are “reinforcing” this by rewarding it.

Tips

Don’t be kind!

You might be tempted to go easy on the computer when you’re playing against it, particularly when it’s just starting to learn and is playing very badly.

For example, you might have two crosses-in-a-row next to a blank space and could win. But instead, you might feel sorry for it doing badly and put a cross somewhere else instead to give it a chance.

Don’t.

It is learning from the way that you play. If you don’t complete a three-in-a-row when you can, you will be teaching it that it should do that.

If you want it to get better quickly, **play as well as you can**.

Mix things up with your examples

Try to come up with lots of different types of examples.

For example, start from a different position on the board on every turn.

Did you know?

People have been learning about machine learning by training a computer to play noughts and crosses for decades!

One famous example was **Donald Michie** – a British artificial intelligence researcher. During World War II, Michie worked at Bletchley Park as a code breaker.

In 1960, he developed "**MENACE**" – the Machine Educable Noughts And Crosses Engine. This was one of the first programs able to learn how to play noughts and crosses perfectly.

As he didn't have a computer he could use, Michie built MENACE using 304 matchboxes and coloured glass beads.

Each matchbox represented a possible state of the board – like the examples that you've been collecting in your training data.

He put beads in the matchboxes to show how often a choice led to a win – the number of beads in the matchbox was like the number of times an example shows up in one of the buckets you created for your training data.