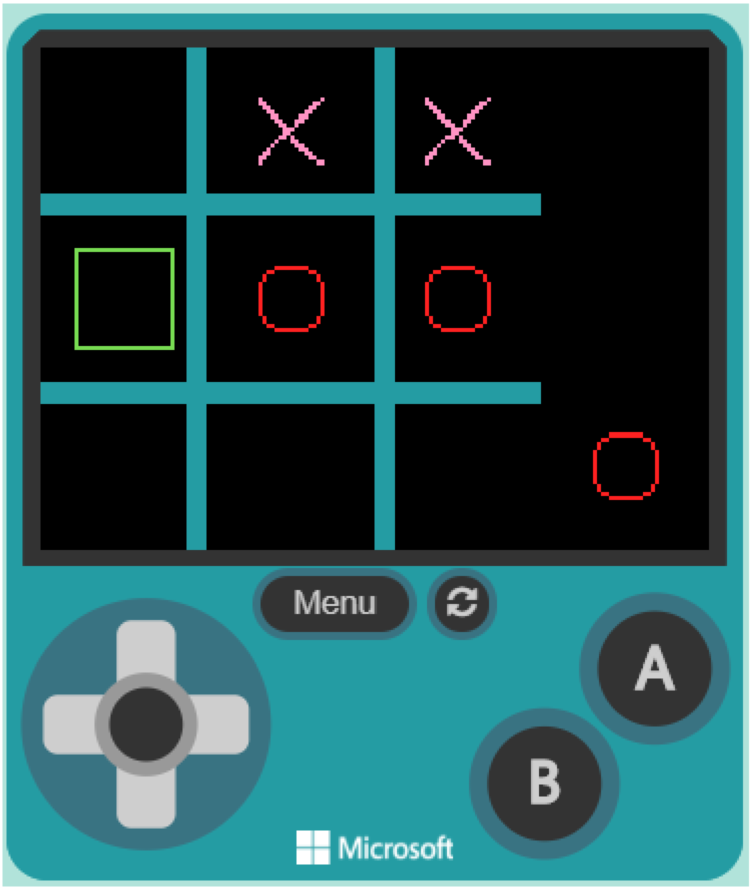
OXO - Python – Microsoft Arcade

The following instructions will take you through the steps of creating a game where you play Noughts and Crosses against the computer. The computer will play as Crosses leaving you to play as Noughts and always go first.

Use the D-Pad to select where to place your piece and then press A to play it. The computer will then think for a few moments before making its own move.



Controls

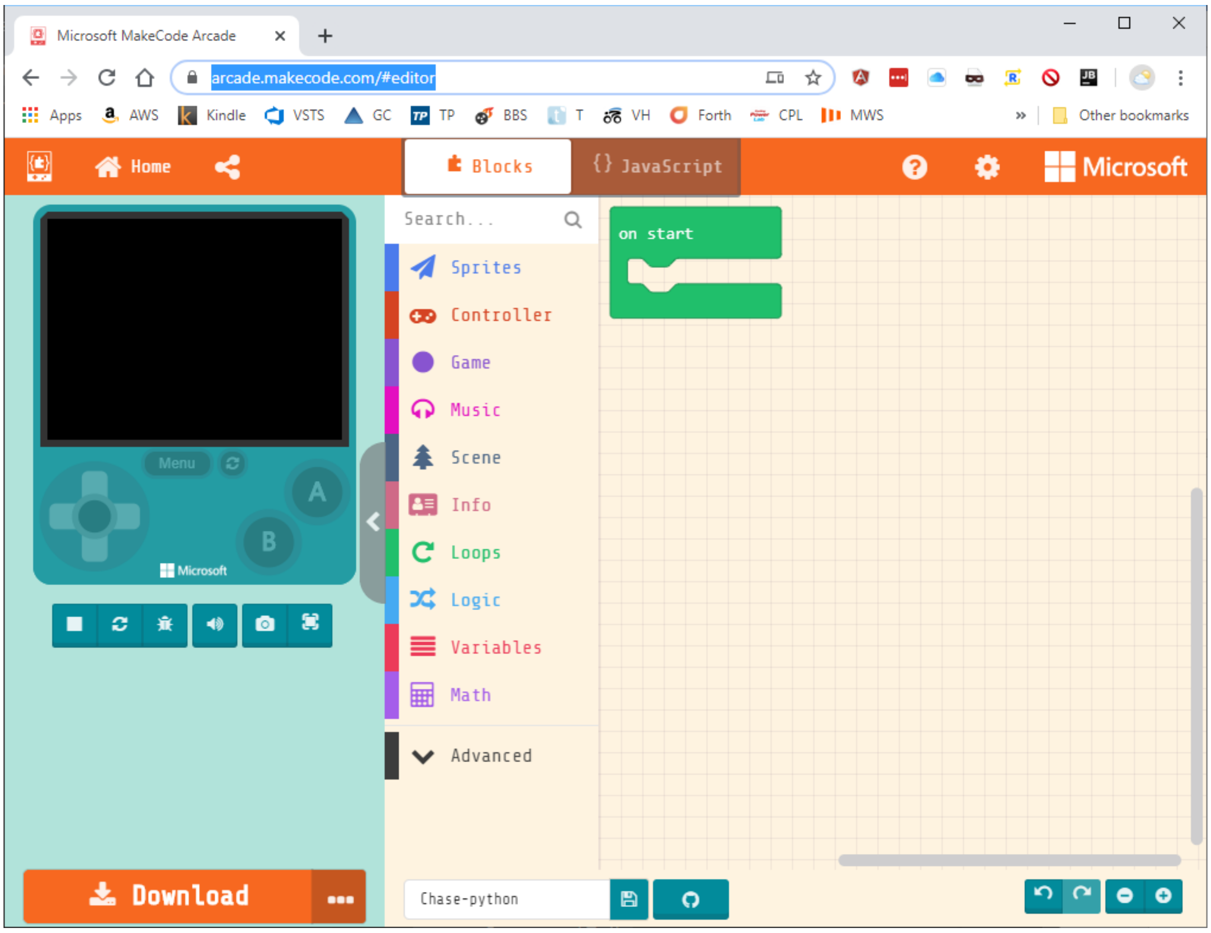
To control the players character, use the direction keys on the console. Alternatively, it may be easier to use the keyboard mappings for the keys as follows

	Up (↑ or W)				
Left (← or A)		**Right** (→ or D)		**B** (E, X or Enter)	**A** (Q, Z or Space)
	Down (↓ or S)				

OXO - Python – Microsoft Arcade

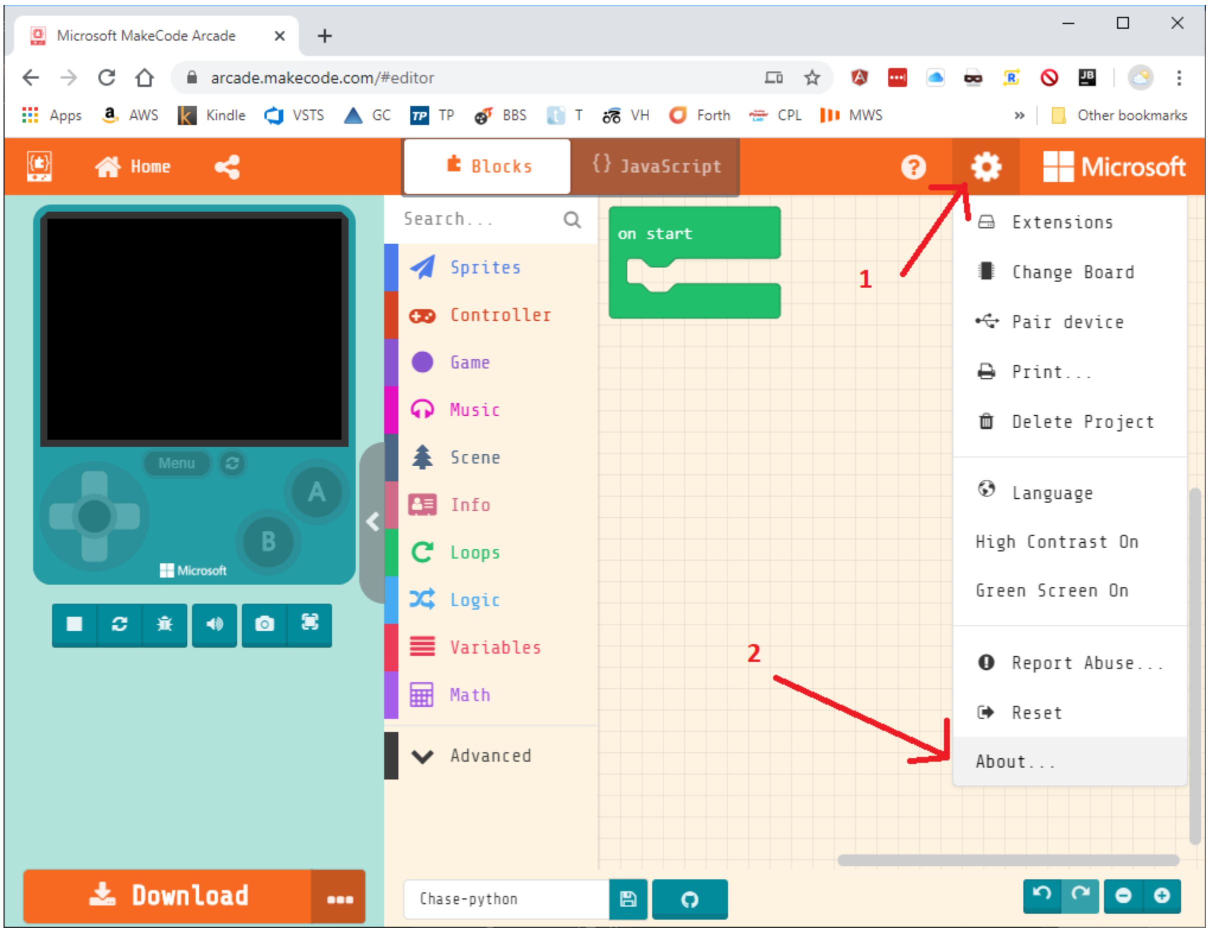
Step 1 – Setup MakeCode arcade for Python development

Navigate to https://arcade.makecode.com and create a new game using the name Chase-python. Your screen should like as follows.



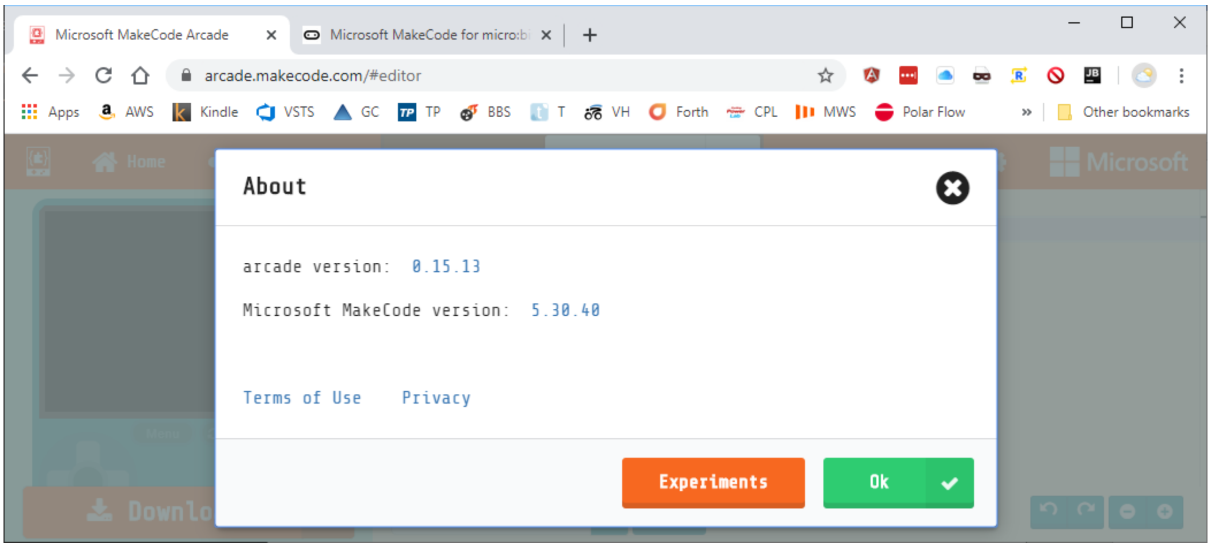
OXO - Python – Microsoft Arcade

Python is currently an experimental feature, so we have to turn it on. We do so by clicking the cog in the top right-hand corner of the screen and then clicking "About…" as illustrated in the screen shot below



OXO - Python – Microsoft Arcade

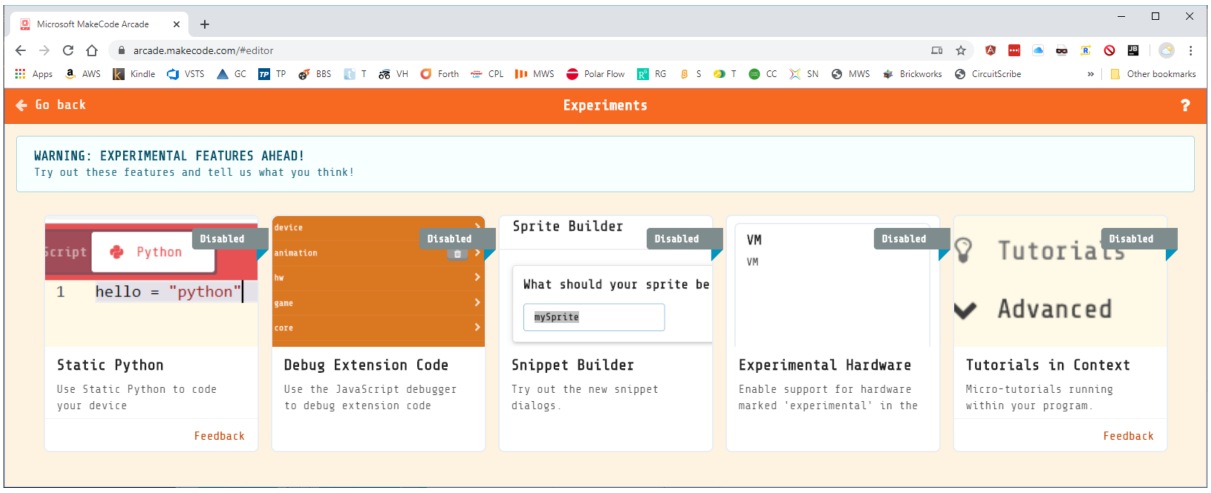
The About dialog should now be displayed as shown in the screen shot below.


About

arcade version: 0.15.13

Microsoft MakeCode version: 5.30.40

Click on Experiments to bring up the range of experimental features. A Static Python feature should be shown. Click on the Static Python icon which will enable it. Then click on the “Go back” button in the top left hand corner. This will take you back to the normal editor.



WARNING: EXPERIMENTAL FEATURES AHEAD!
Try out these features and tell us what you think!

Static Python
Use Static Python to code your device

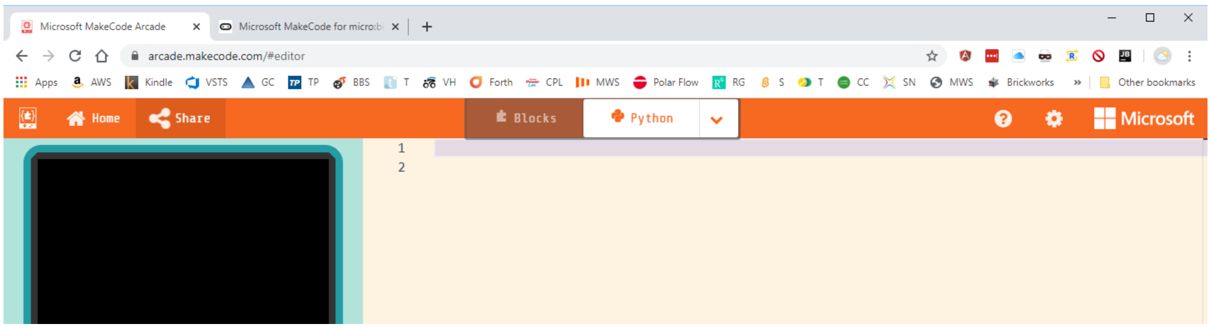
Debug Extension Code
Use the JavaScript debugger to debug extension code

Snippet Builder
Try out the new snippet dialogs.

Experimental Hardware
Enable support for hardware marked 'experimental' in the

Tutorials in Context
Micro-tutorials running within your program.

You will now see along the top of the screen the Python tab. Congratulations, Python is now eaabled for MakeCode Arcade.



OXO - Python – Microsoft Arcade

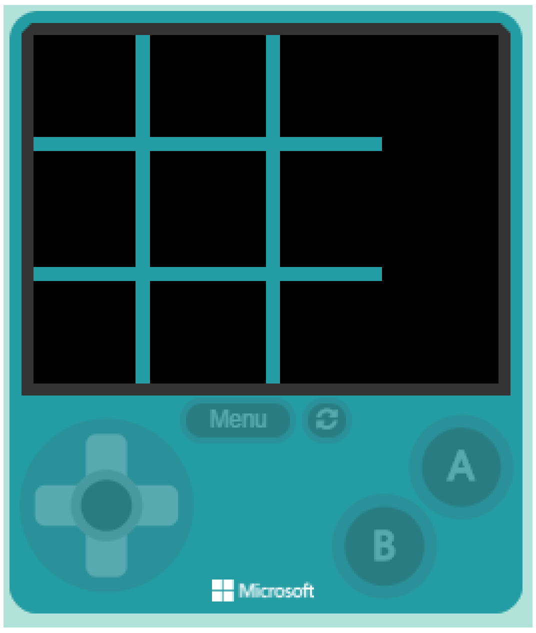
Step 2 – Draw the background

We will be using these values to control some aspects of the game, such as who's turn it is to play or what the result of the game is. These values are added as different kinds of sprite.

We also setup the background for the game and draw the board. The code below uses a background colour of black (colour number 15) and for the lines of the board the colour cyan is used (colour number 6). The colours that are available for use are numbered 0 to 15 so experiment to pick some colours that you like (colour zero is transparent).

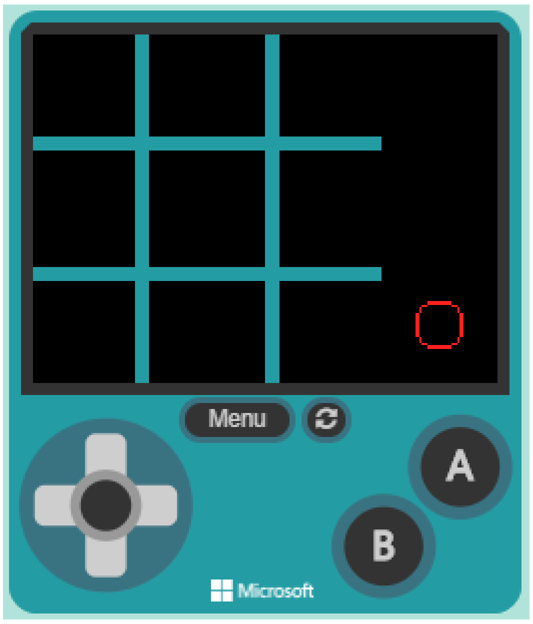
```
@namespace
class SpriteKind:
    Blank = SpriteKind.create()
    Nought = SpriteKind.create()
    Cross = SpriteKind.create()
    Draw = SpriteKind.create()
    Cursor = SpriteKind.create()

# Setup the background board by drawing the lines
scene.set_background_color(15)
background = image.create(120, 120)
background.fillRect(35, 0, 5, 120, 6)
background.fillRect(80, 0, 5, 120, 6)
background.fillRect(0, 35, 120, 5, 6)
background.fillRect(0, 80, 120, 5, 6)
scene.set_background_image(background)
```

Once you have finished don't forget to run your game to make sure that it works. You should see a black background with a blue Noughts and Crosses board like the one below.



OXO - Python – Microsoft Arcade

Step 3 – The player indicator



We are now going to add an indicator to the bottom right-hand corner of the screen which will show the piece (Nought or Cross) of who's turn it currently is to play. Once you've added this code and run the game, it should look like the picture below.

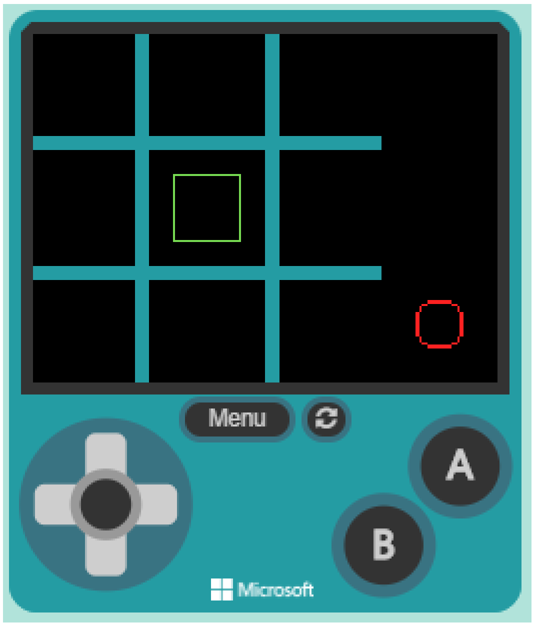
Remember to use the paint tool to draw the images that we will be using for the Noughts and Crosses as it will be much easier than doing it by hand. The images for the Noughts and Crosses should be 16 x 16 pixels in size. Don't feel constrained to use plain old Noughts or Crosses. Use your creativity and imagination to draw your own icons or pick some from the gallery. Tacos vs Ducks?

The code for tracking the current player makes use of a Python class with two items of static data. Static data is similar to global variables and placing these two items in the class as static data provides a useful way to group the data together.

```
# 16 x 16 pixel Nought
Nought = img("""
    . . . . 2 2 2 2 2 2 2 2 . . . .
    . . 2 2 . . . . . . . . 2 2 . .
    . 2 . . . . . . . . . . . . 2 .
    . 2 . . . . . . . . . . . . 2 .
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    2 . . . . . . . . . . . . . . 2
    . 2 . . . . . . . . . . . . 2 .
    . 2 . . . . . . . . . . . . 2 .
    . . 2 2 . . . . . . . . 2 2 . .
    . . . . 2 2 2 2 2 2 2 2 . . . .
""")

# 16 x 16 pixel cross
cross = img("""
    3 . . . . . . . . . . . . . . 3
    3 3 . . . . . . . . . . . 3 3 .
    . 3 3 . . . . . . . . . 3 3 . .
    . . 3 3 . . . . . . . 3 3 . . .
    . . . 3 3 . . . . . 3 3 . . . .
    . . . . 3 3 . . . 3 3 . . . . .
    . . . . . 3 3 . . 3 . . . . . .
    . . . . . . 3 3 3 . . . . . . .
    . . . . . . 3 3 . . . . . . . .
    . . . . . 3 3 . 3 . . . . . . .
    . . . . 3 . . . 3 3 . . . . . .
    . . . 3 . . . . . 3 3 . . . . .
    . . 3 . . . . . . . 3 3 . . . .
    . 3 3 . . . . . . . . 3 3 . . .
    3 3 . . . . . . . . . . 3 3 . .
    3 . . . . . . . . . . . . . 3 3
""")


class CurrentPlayer:
    kind = SpriteKind.Nought
    sprite = sprites.create(Nought, SpriteKind.player)

CurrentPlayer.sprite.set_position(140, 100)

def setup_nought_player():
    CurrentPlayer.kind = SpriteKind.Nought
    CurrentPlayer.sprite.set_image(Nought)

def setup_cross_player():
    CurrentPlayer.kind = SpriteKind.Cross
    CurrentPlayer.sprite.set_image(cross)

# Changes to the next player
def setup_next_player():
    if CurrentPlayer.kind == SpriteKind.Nought:
        setup_cross_player()
    else:
        setup_nought_player()
```

Step 4 – Selector

Now it is time to add in a selector which allows the player to choose which square to place their piece in. We will use a similar technique as we did for the current player of using a Class with static (i.e. global) variables is used to control the position of the selector. We also need to hook the selector into the d-pad button events.



We also provide for “wrapping” the selector. This is where pressing up when the selector is in the top row results in the selector showing on the bottom row as it has wrapped around. The same is true for pressing down when in the bottom row and left to right.

```
# 4 x 4 pixel blank
blank = img("""
    . . . .
    . . . .
    . . . .
    . . . .
""")

# 24 x 24 pixel cursor
cursor = img("""
    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 . . . . . . . . . . . . . . . . . . . . . . 7
    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
""")

class Selector:
    x = 1
    y = 1
    sprite = sprites.create(cursor, SpriteKind.Cursor)

update_selector()

def update_selector():
    Selector.sprite.set_position((Selector.x * 40) + 20, (Selector.y * 40) +
20)

def reset_selector():
    Selector.x = 1
    Selector.y = 1

def hide_selector():
    Selector.sprite.set_image(blank)
    update_selector()

def show_selector():
    Selector.sprite.set_image(cursor)
    update_selector()

def move_selector_left():
    Selector.x = Selector.x - 1
    if Selector.x < 0:
        Selector.x = 2
    update_selector()

def move_selector_right():
    Selector.x = Selector.x + 1
    if Selector.x > 2:
        Selector.x = 0
    update_selector()

def move_selector_up():
    Selector.y = Selector.y - 1
    if Selector.y < 0:
        Selector.y = 2
    update_selector()

def move_selector_down():
    Selector.y = Selector.y + 1
    if Selector.y > 2:
        Selector.y = 0
    update_selector()

def left_pressed():
    if CurrentPlayer.kind == SpriteKind.Nought:
        move_selector_left()
```

```
def right_pressed():
    if CurrentPlayer.kind == SpriteKind.Nought:
        move_selector_right()

def up_pressed():
    if CurrentPlayer.kind == SpriteKind.Nought:
        move_selector_up()

def down_pressed():
    if CurrentPlayer.kind == SpriteKind.Nought:
        move_selector_down()


controller.left.onEvent(ControllerButtonEvent.PRESSED, left_pressed)
controller.right.onEvent(ControllerButtonEvent.PRESSED, right_pressed)
controller.up.onEvent(ControllerButtonEvent.PRESSED, up_pressed)
controller.down.onEvent(ControllerButtonEvent.PRESSED, down_pressed)
```

Step 5 – Create the board sprites

This next step is to create a 3 x 3 array of sprites to represent each of the possible positions where a piece can be played. Each sprite is setup in the correct location and set it to a blank image. When a piece is played on the board (done in code in a later section) we can change the image of the sprite to the correct piece. Arrays are declared and their elements access with the [and] characters.

```
board = [[
            sprites.create(blank, SpriteKind.Blank),
            sprites.create(blank, SpriteKind.Blank),
            sprites.create(blank, SpriteKind.Blank)
        ],[
            sprites.create(blank, SpriteKind.Blank),
            sprites.create(blank, SpriteKind.Blank),
            sprites.create(blank, SpriteKind.Blank)
        ],[
            sprites.create(blank, SpriteKind.Blank),
            sprites.create(blank, SpriteKind.Blank),
            sprites.create(blank, SpriteKind.Blank)
        ]]

# Place the pieces in their correct location.
for x in range(3):
    for y in range(3):
        piece = CurrentPlayer.sprite
        piece = board[x][y]
        piece.set_position((x * 40) + 14, (y * 40) + 14)

# Reset all of the pieces to blanks as well as the board itself
def clear_board():
    piece = CurrentPlayer.sprite
    for x in range(3):
        for y in range(3):
            piece = board[x][y]
            piece.set_image(blank)
            piece.set_kind(SpriteKind.Blank)

# Returns if the board is full or not
def board_full():
    piece = CurrentPlayer.sprite
    for x in range(3):
        for y in range(3):
            piece = board[x][y]
            if piece.kind() == SpriteKind.Blank:
                return False
    return True
```

OXO - Python – Microsoft Arcade

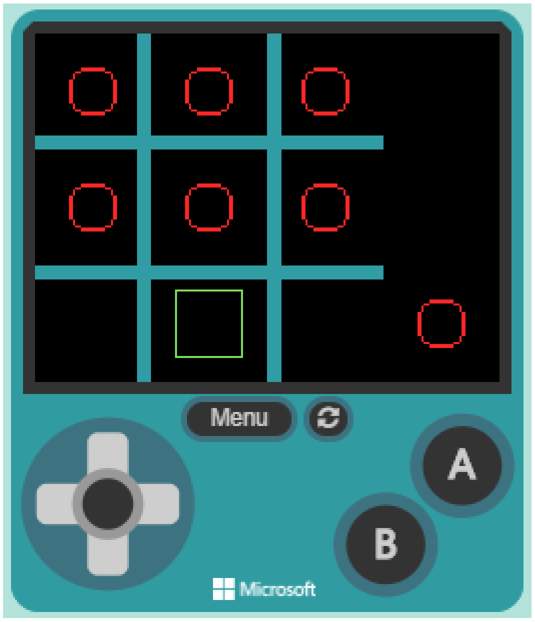
Step 6 – Placing the players pieces.

After this step you should be able to place noughts in every space. The screen shot opposite shows the top two rows filled up.

What we do is create two functions (can_play_here and play_move) that are hooked into the A-button press event. If the board is empty (contains a Blank) at the location indicated by the selector then we set the piece to the desired kind (Noughts in this case).

The play_move function does not yet contain the instructions for the computer to move yet so we will add to it in a later step.

We also provide a function to start a new game. Once you've added this code to your game, try it out.



```
# Returns true if the board is blank at the given location.
def can_play_here(x, y):
    piece = CurrentPlayer.sprite
    piece = board[x][y]
    if piece.kind() == SpriteKind.Blank:
        return True
    return False

def play_move(x, y, kind):
    if can_play_here(x, y):
        piece = CurrentPlayer.sprite
        piece = board[x][y]
        piece.set_kind(kind)
        if kind == SpriteKind.Nought:
            piece.set_image(Nought)
        else:
            piece.set_image(cross)

def a_pressed():
    if CurrentPlayer.kind == SpriteKind.Nought:
        if can_play_here(Selector.x, Selector.y):
            play_move(Selector.x, Selector.y, CurrentPlayer.kind)
controller.a.onEvent(ControllerButtonEvent.PRESSED, a_pressed)

# The actual game logic starts here.
def start_new_game():
    hide_selector()
    reset_selector()
    clear_board()
    setup_nought_player()
    show_selector()

# Start the game
start_new_game()
```

Step 7 – Working out if the game has finished

We now need to write the algorithm that works out who has won (if anyone). This method checks all of the possible combinations of vertical, horizontal or diagonal wins as well as a draw. If none of those conditions hold then the game is not finished, and the result is None. For a win, we return the type of piece that has won. The possible results from the get_result method are:

- SpriteKind.Nought
- SpriteKind.Cross
- SpriteKind.Draw
- None

```
# Returns the result of the board. This can be Nought or Cross for
# their win, Draw for a Draw or None for no result.
def get_result():
    # Check for a vertical win
    for x in range(3):
        top = CurrentPlayer.sprite
        middle = CurrentPlayer.sprite
        bottom = CurrentPlayer.sprite
        top = board[x][0]
        middle = board[x][1]
        bottom = board[x][2]
        if top.kind() != SpriteKind.Blank:
            if top.kind() == middle.kind()and top.kind() == bottom.kind():
                return top.kind()

    # Check for a horizontal win
    for y in range(3):
        left = CurrentPlayer.sprite
        middle = CurrentPlayer.sprite
        right = CurrentPlayer.sprite
        left = board[0][y]
        middle = board[1][y]
        right = board[2][y]
        if left.kind() != SpriteKind.Blank:
            if left.kind() == middle.kind() and left.kind() == right.kind():
                return left.kind()

    # Check for diagonal win
    centre = CurrentPlayer.sprite
    centre = board[1][1]
    if centre.kind() != SpriteKind.Blank:
        top_left = CurrentPlayer.sprite
        bottom_right = CurrentPlayer.sprite
        top_left = board[0][0]
        bottom_right = board[2][2]
        if top_left.kind() == centre.kind() and centre.kind() == bottom_right.kind():
            return centre.kind()

        bottom_left = CurrentPlayer.sprite
        top_right = CurrentPlayer.sprite
        bottom_left = board[0][2]
        top_right = board[2][0]
        if bottom_left.kind() == centre.kind()and centre.kind() == top_right.kind():
            return centre.kind()

    # Check for draw
    if board_full():
        return SpriteKind.Draw

    return None
```

Step 8 – Check for a result in play_move

We now can modify the play_move method that you create in step 6 to check for a result in the game (and start a new game if a result was found). Locate the play_move method that you already have and add the highlighted code to the end of that method.

```
def play_move(x, y, kind):
    if can_play_here(x, y):
        piece = CurrentPlayer.sprite
        piece = board[x][y]
        piece.set_kind(kind)
        if kind == SpriteKind.Nought:
            piece.set_image(Nought)
        else:
            piece.set_image(cross)

        result = get_result()
        if result == SpriteKind.Nought:
            game.show_long_text("Player wins", DialogLayout.BOTTOM)

        if result == SpriteKind.Cross:
            game.show_long_text("Computer wins", DialogLayout.BOTTOM)

        if result == SpriteKind.Draw:
            game.show_long_text("Game drawn", DialogLayout.BOTTOM)

        if result != None:
            start_new_game()
```

Step 9 – Getting the computer to play

All that remains now is to add the code to let the computer play. As in step 8, we will modify the play_move function to add the highlighted code as well as add the new play_computer function.

```
def play_move(x, y, kind):
    if can_play_here(x, y):
        piece = CurrentPlayer.sprite
        piece = board[x][y]
        piece.set_kind(kind)
        if kind == SpriteKind.Nought:
            piece.set_image(Nought)
        else:
            piece.set_image(cross)

        # If there was a result then start a new game otherwise
        # setup the next move and (if it is the computers turn)
        # play the computers move.
        result = get_result()
        if result == SpriteKind.Nought:
            game.show_long_text("Player wins", DialogLayout.BOTTOM)

        if result == SpriteKind.Cross:
            game.show_long_text("Computer wins", DialogLayout.BOTTOM)

        if result == SpriteKind.Draw:
            game.show_long_text("Game drawn", DialogLayout.BOTTOM)

        if result != None:
            start_new_game()
        else:
            setup_next_player()

            # Play computer
            if CurrentPlayer.kind == SpriteKind.Cross:
                play_computer()
```

```
# This plays the computers move.
def play_computer():
    hide_selector()

    # Simulate thinking
    pause(1000)

    # Check for computer player which is a random mover
    while CurrentPlayer.kind == SpriteKind.Cross:
        x = randint(0, 2)
        y = randint(0, 2)
        play_move(x, y, SpriteKind.Cross)

    show_selector()
```

Extending the game

There are many ways that this game can be extended. Just a few ideas are given below.

- Enhance the graphics to make the game look prettier
- Add in a title screen
- Add in sound effects
- Make the computer a smart AI that is impossible to beat
- Display the total number of games played and the Win/Draw/Lose amounts.
- Give the game a 2-Player (humans) option.
- Play as Crosses.

Extra – Getting the computer to play against itself

If you want a bit of fun, you can change the game so that the computer plays against itself. The code below shows how to modify the play_move function. Then there are the two copies of the play_computer method renamed to play_computer_x and play_computer_o that allow the computer to play as Crosses or Noughts. It plays fast and is only slowed down so you can see it (the pause(100)). See if you can replace the random move algorithm with a better strategy that wins more often.

```
def play_move(x, y, kind):
    if can_play_here(x, y):
        piece = CurrentPlayer.sprite
        piece = board[x][y]
        piece.set_kind(kind)
        if kind == SpriteKind.Nought:
            piece.set_image(Nought)
        else:
            piece.set_image(cross)

        result = get_result()

        if result != None:
            pause(100)
            start_new_game()
        else:
            setup_next_player()

# This plays the computers move.
def play_computer_x():
    hide_selector()

    # Check for computer player which is a random mover
    while CurrentPlayer.kind == SpriteKind.Cross:
        x = randint(0, 2)
        y = randint(0, 2)
        play_move(x, y, SpriteKind.Cross)

    show_selector()

def play_computer_o():
    hide_selector()

    # Check for computer player which is a random mover
    while CurrentPlayer.kind == SpriteKind.Nought:
        x = randint(0, 2)
        y = randint(0, 2)
        play_move(x, y, SpriteKind.Nought)

    show_selector()

while True:
    if CurrentPlayer.kind == SpriteKind.Cross:
        play_computer_x()
    else:
        play_computer_o()
```