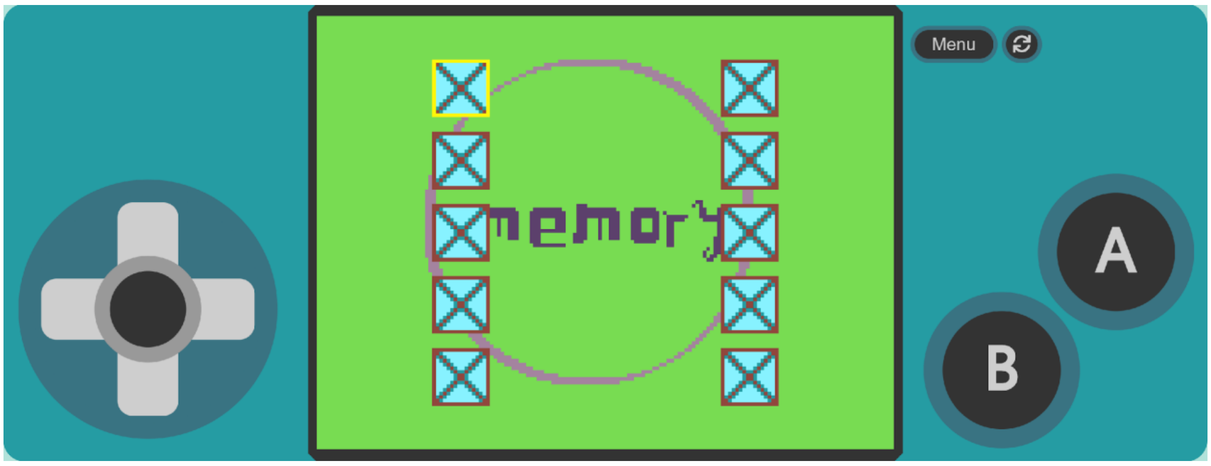
THIRD ARRAYS TASK



Description.

In this project, we will create a variation of the classic game "Memory," where the player has a table with cards and has to select two cards with the same image in consecutive turns.

We will use many of the concepts we have learned so far, with a primary focus on the "Array" section, where we will store the positions of each card, the related images, and more.

In order to do that, we use MakeCode Arcade to create the game.

Goals.

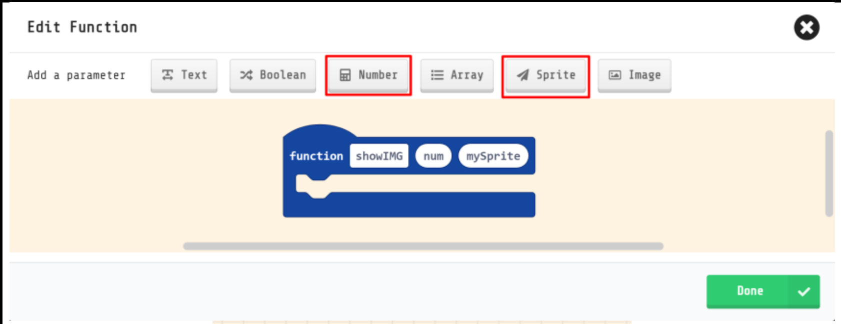
- Create a sprite that serves as a cursor for navigating between different options.
- Use arrays to achieve the desired result, both for generating random content and associating images with specific positions.

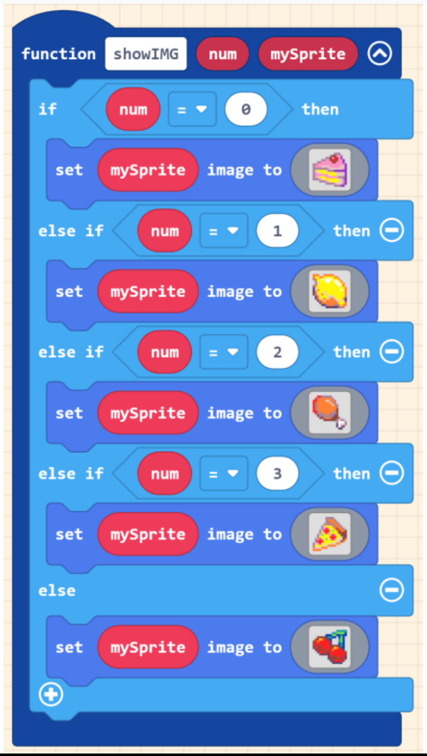
Programming guide.

ASSIGNMENT OF IMAGES IN THE ARRAY

We will create the "showIMG" function with two types of parameters: one numeric and one of type Sprite. Based on the number entered as the numeric parameter, we will assign a specific image to the Sprite passed as the other parameter.

We will use a separate condition for each pair of cards to assign a corresponding image.



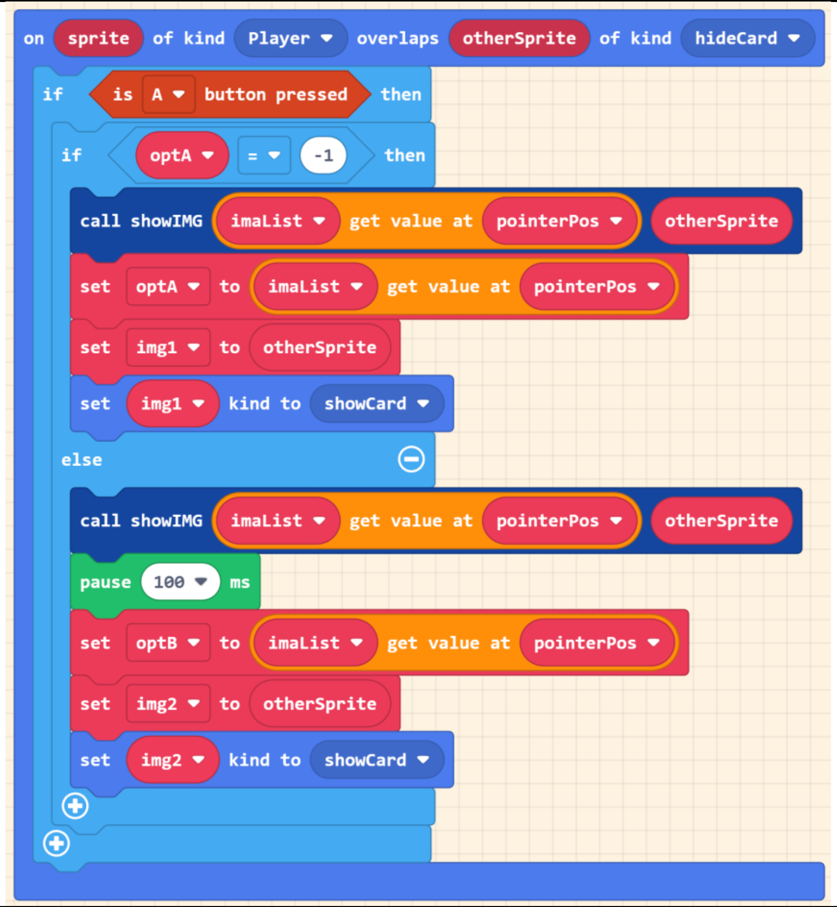


```
function showIMG num mySprite
    if num = 0 then
        set mySprite image to
    else if num = 1 then
        set mySprite image to
    else if num = 2 then
        set mySprite image to
    else if num = 3 then
        set mySprite image to
    else
        set mySprite image to
```

CHOOSING AN OPTION MECHANISM

We will program the interaction between the player and the hidden card. Here, we will check if the "A" button is pressed and based on whether one or both choices have been made, we will change the image of the hidden card and update the values of the variables "optA" or "optB" to reflect the selected card.
In this code, we assume that the variables "optA" and "optB" have been declared and set to -1 to indicate no choices made yet.

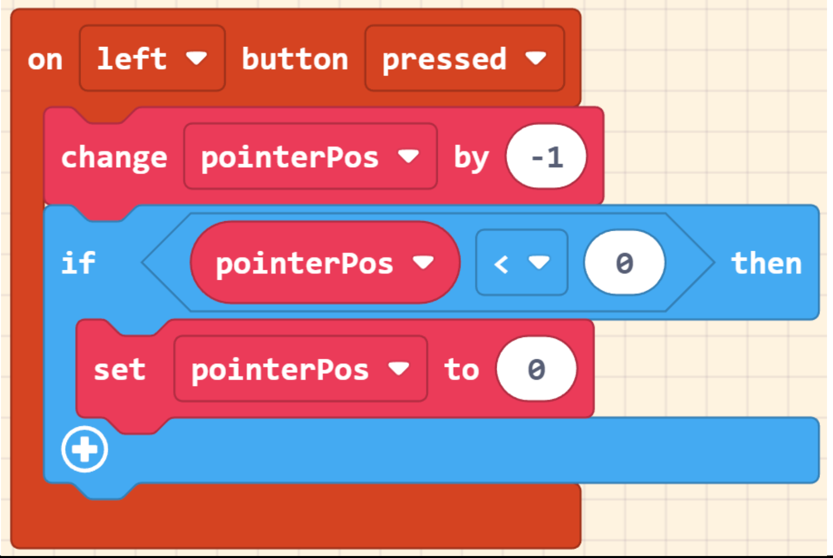
When the "A" button is pressed, we check if "optA" is -1. If it is, we assign the value of the selected card (from the "numberList") to "optA" and show the corresponding image on the card. If "optA" is not -1, we assume "optB" is -1, assign the value of the selected card to "optB", show the corresponding image, and then perform the comparison logic to check if the choices match.



```
on sprite of kind Player overlaps otherSprite of kind hideCard
    if is A button pressed then
        if optA = -1 then
            call showIMG imaList get value at pointerPos otherSprite
            set optA to imaList get value at pointerPos
            set img1 to otherSprite
            set img1 kind to showCard
        else
            call showIMG imaList get value at pointerPos otherSprite
            pause 100 ms
            set optB to imaList get value at pointerPos
            set img2 to otherSprite
            set img2 kind to showCard
```

CURSOR MOVEMENT MECHANISM

We will program the left button to decrease the value of "pointerPos". We will also add a condition to prevent the value from going below the minimum value, which in this case is 0.

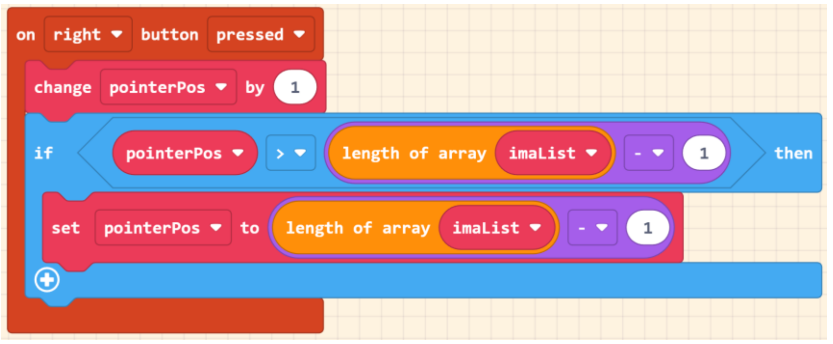


```
on left ▾ button pressed ▾
    change pointerPos ▾ by -1
    if pointerPos ▾ < ▾ 0 then
        set pointerPos ▾ to 0
```

We will program the right button to increase the value of "pointerPos". The maximum value allowed will be 9, which corresponds to the number of cards being used (assuming there are 10 cards, the positions will range from 0 to 9).

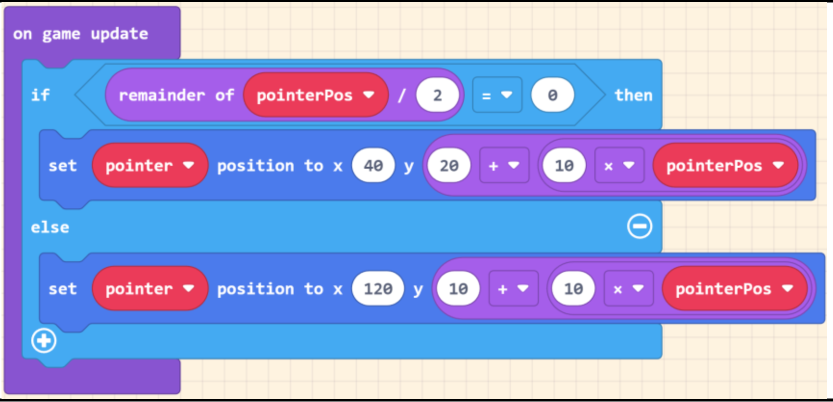
In this code, when the right button is pressed, we check if the current value of "pointerPos" is less than 9. If it is, we increment the value of "pointerPos" by 1.

This condition ensures that the value of "pointerPos" doesn't exceed 9, which is the maximum allowed value based on the number of cards being used.



```
on right ▾ button pressed ▾
    change pointerPos ▾ by 1
    if pointerPos ▾ > ▾ length of array imaList ▾ - ▾ 1 then
        set pointerPos ▾ to length of array imaList ▾ - ▾ 1
```

We will drag an "on game update" block to the workspace. With this instruction, we will change the position of the "pointer" sprite. We will also check if the value of "pointerPos" is even.



```
on game update
    if remainder of pointerPos ▾ / 2 = ▾ 0 then
        set pointer ▾ position to x 40 y 20 + ▾ 10 × ▾ pointerPos ▾
    else
        set pointer ▾ position to x 120 y 10 + ▾ 10 × ▾ pointerPos ▾
```

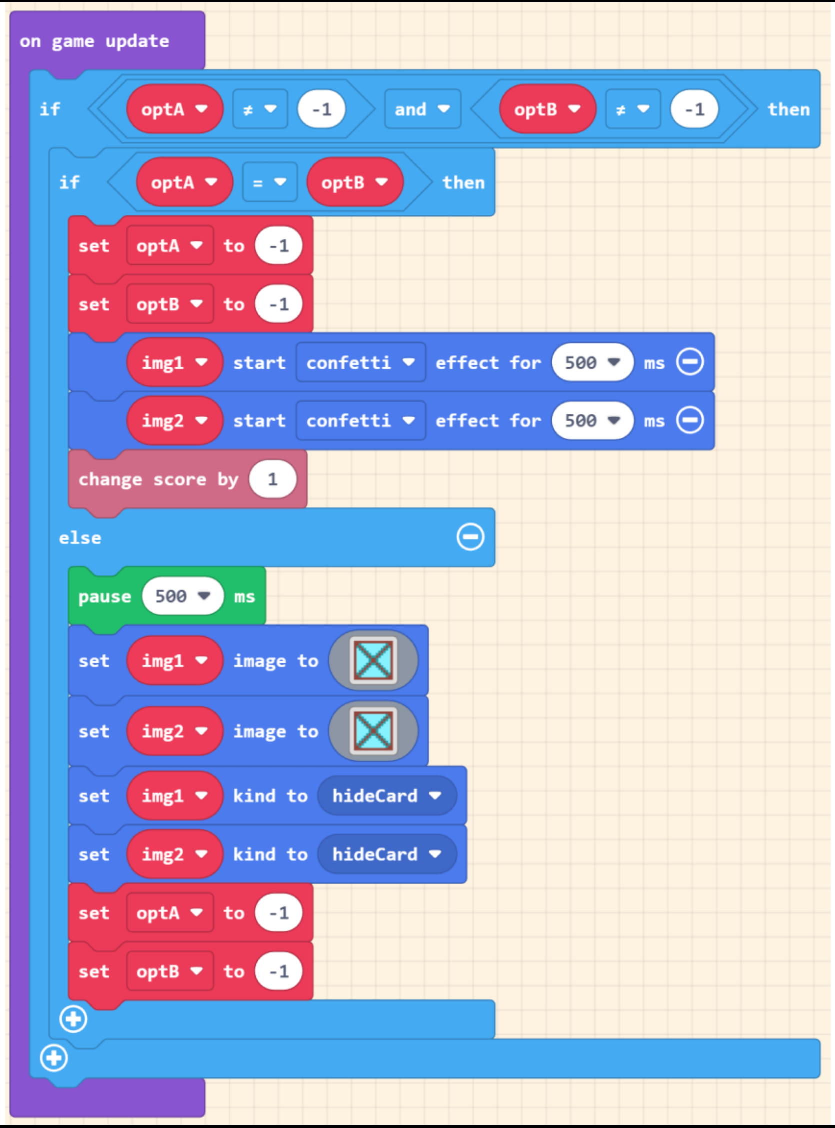
CORRECT/INCORRECT RESPONSE MECHANISM

We will drag another "on game update" block to check if the choices of the two cards are correct or not.

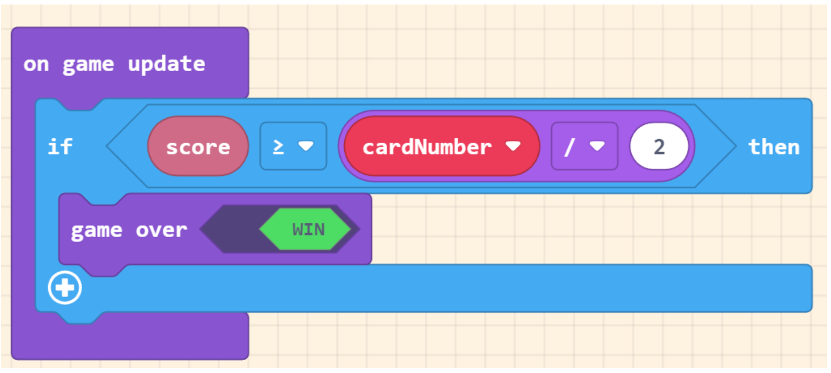
In this code, we use the "on game update" block to continuously check if both choices have been made. We do this by checking if the values of "optA" and "optB" are not equal to -1, which indicates that a choice has been made for both cards.

If both choices have been made, we compare the values of "optA" and "optB". If they are equal, it means the choices match. In this case, we reset the choices by setting "optA" and "optB" back to -1, increase the score by 1, and you can add your code to display a celebration effect.

If the choices do not match, we change the displayed images, reset the sprites to the "hideCard" type, and reset "optA" and "optB" back to -1.



```
on game update
    if optA ≠ -1 and optB ≠ -1 then
        if optA = optB then
            set optA to -1
            set optB to -1
            img1 start confetti effect for 500 ms
            img2 start confetti effect for 500 ms
            change score by 1
        else
            pause 500 ms
            set img1 image to
            set img2 image to
            set img1 kind to hideCard
            set img2 kind to hideCard
            set optA to -1
            set optB to -1
```

GAME OVER MECHANISM	
We will add one more "on game update" block to finish the game. In this code, we use the "on game update" block to continuously check if the score is equal to or greater than the number of card pairs. We calculate the number of card pairs by dividing the total number of cards (stored in "cardNumber") by 2. If the score meets or exceeds the number of card pairs, it means the game is over.	

```
on game update
    if score ≥ cardNumber / 2 then
        game over WIN
```

With the programming we have done so far, you will be able to navigate the cursor among the different cards, select them to uncover and see which cards they are. If you match a pair, you will earn a point, and when you have found all the pairs, you win the game.