FIRST LOOP TASK

Description.

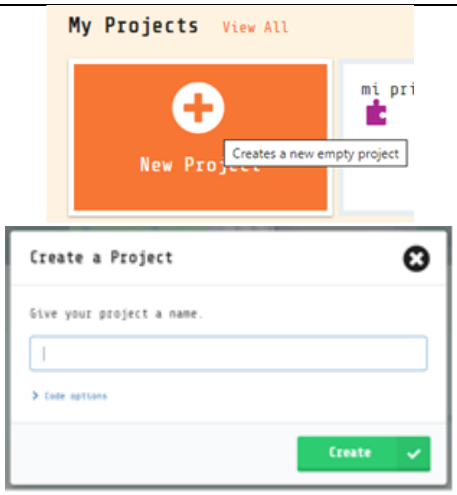
In this project, we will create a classic game called "Block Out". The main objective of the video game is to destroy blocks using a ball while preventing it from falling using a horizontal bar.

We acces to **MakeCode Arcade** and we do the necessary operations.

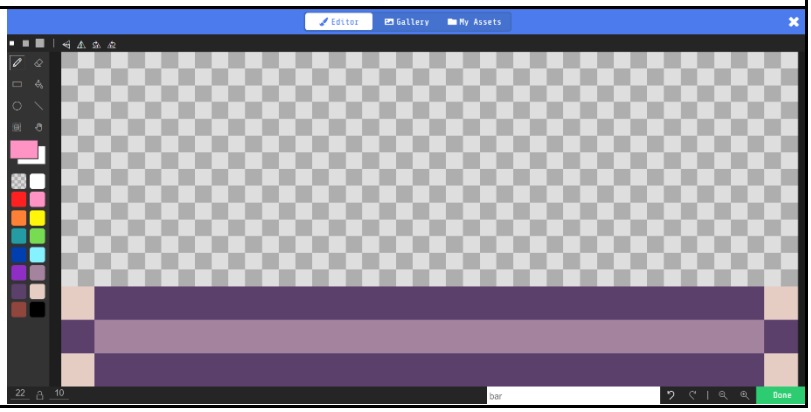
Goals.

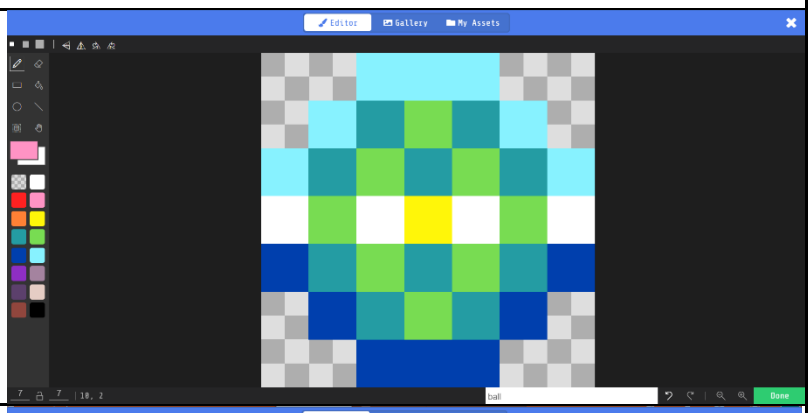
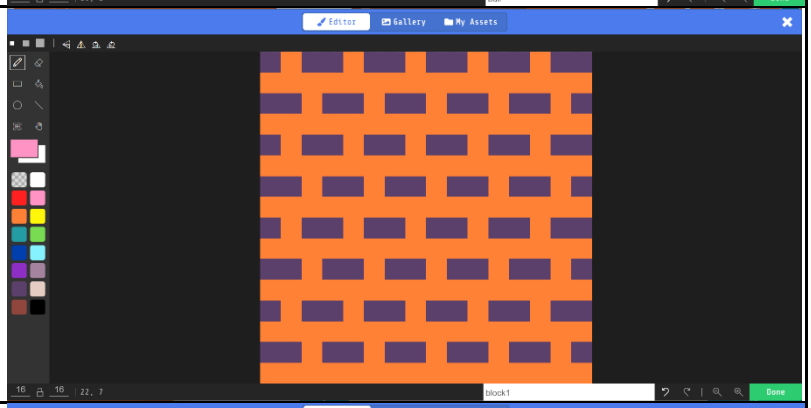
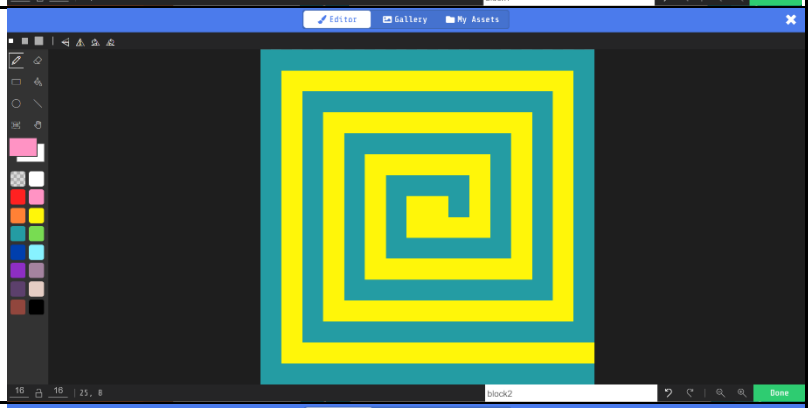
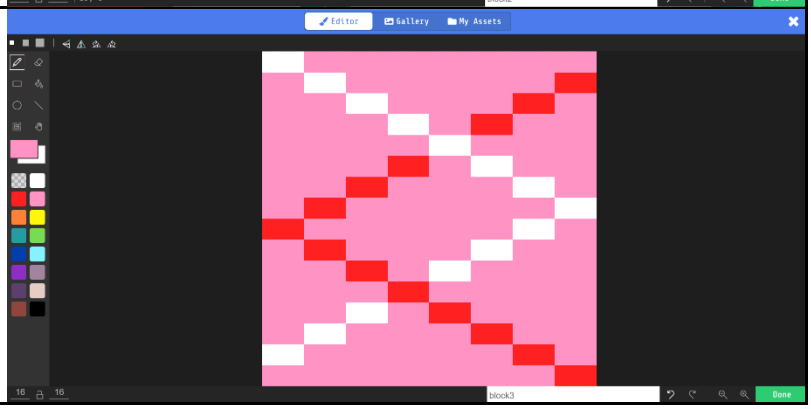
- Create a player bar sprite.
- Create a ball sprite.
- Create different sprite blocks.
- Implement ball bounce mechanics on blocks and win points.
- Implement ball bounce mechanics on the player bar.
- Implement automatic block creation and placement mechanics.

Programming guide.

NEW PROJECT	
We start creating a project, we should stablish the name, for example ¨Picking up food¨ and then press ¨create¨ button.	

Here you have the link with part of the programming and assets done. https://makecode.com/_Vgd64v9qAHAJ
ASSET CREATION
MAIN SPRITE CREATION

We recommend the use of a 22x10 grid for the Sprite of bar.	

ADITIONAL SPRITE CREATION	
We recommend the use of a 7x7 grid for the Sprite of ball.	
We recommend the use of a 16x16 grid for the Sprite of block1.	
We recommend the use of a 16x16 grid for the Sprite of block2.	
We recommend the use of a 16x16 grid for the Sprite of block3.	

MAIN PROGRAMMING

ON START GAME CREATION

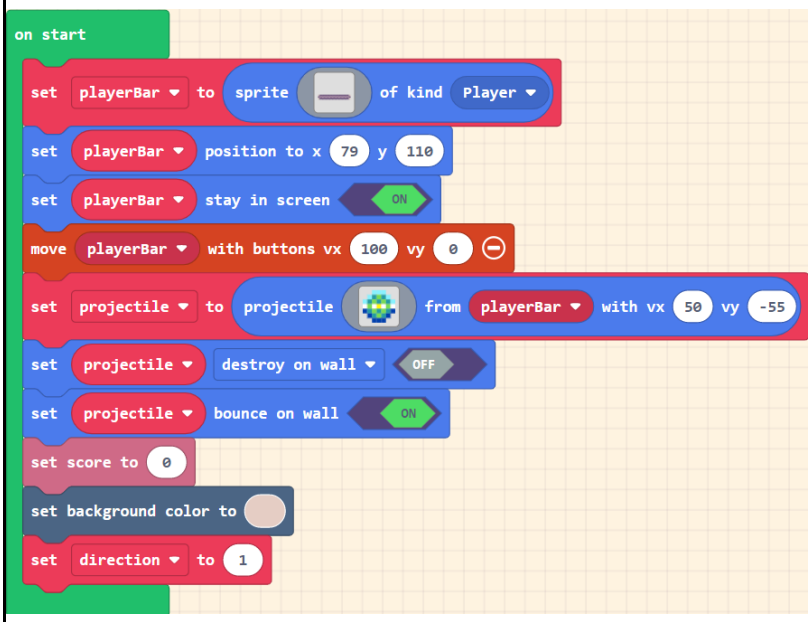
We create an on start block and we add the following blocks inside:

We create playerBar, add our Sprite bar and set it of kind Player. We place it on "x" 79 and "y" 110. We also say the sprite to stay in screen and set a vx of 100 and vy 0 so it only moves on the "x" axis.

We do the same with the ball but changing speed, the condition, the position, the bounce and the destroy off.

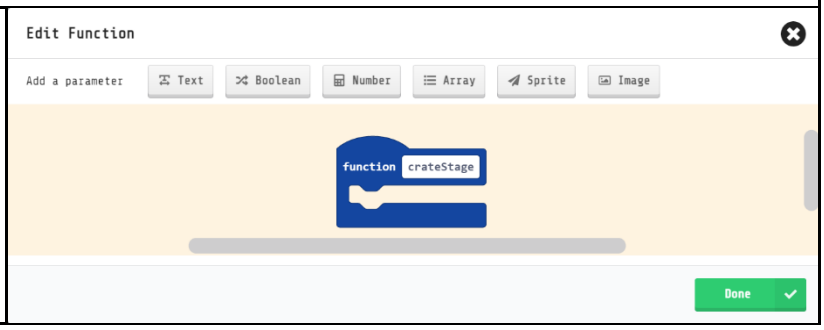
Set score to 0, set background color to so we can add a colour.

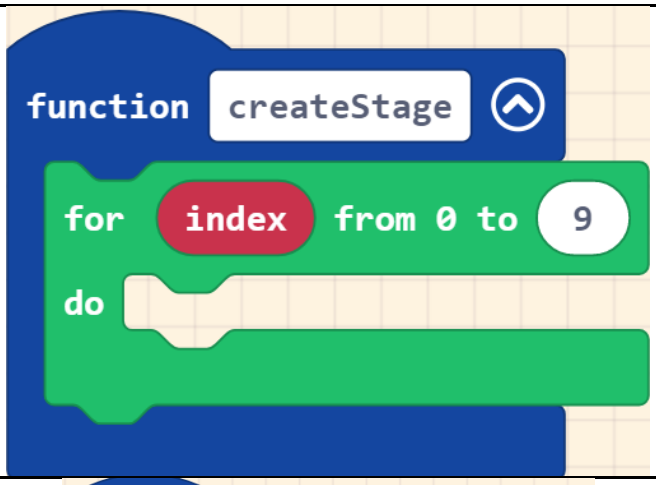
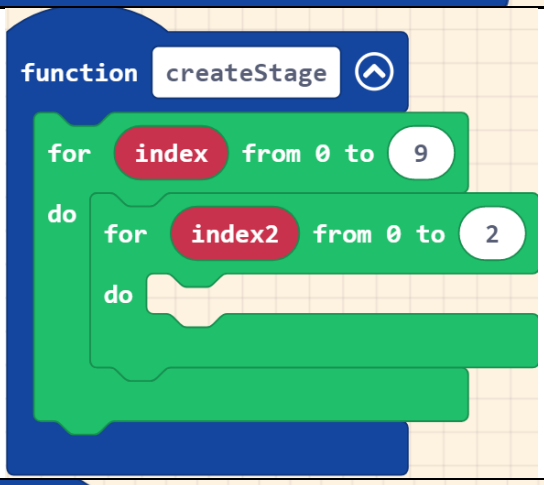
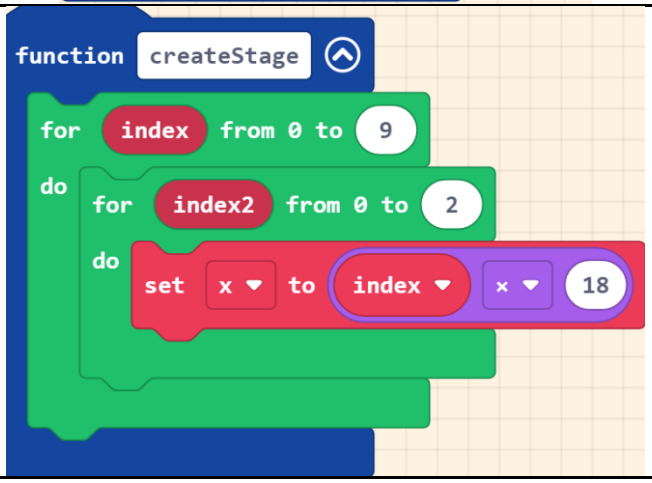
Last, we create the direction variable and set it to 1.

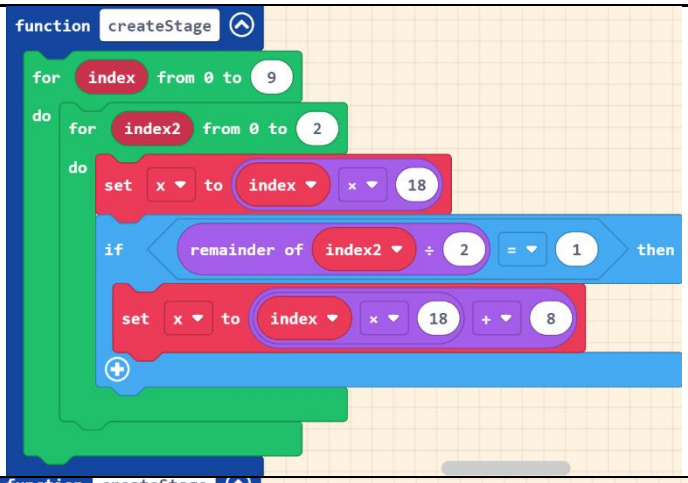
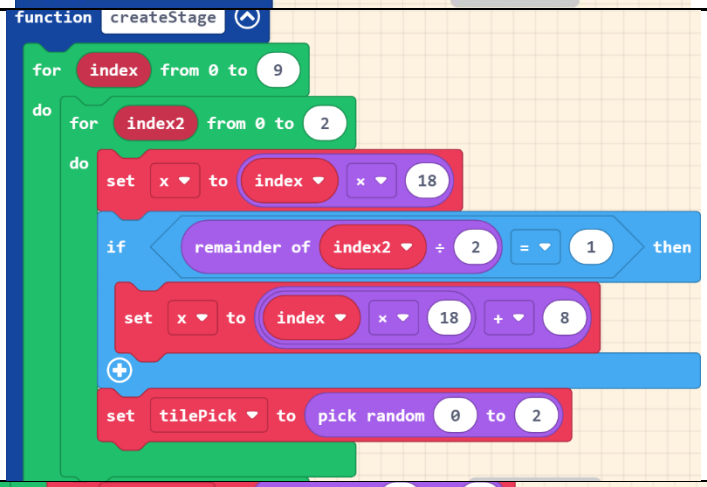
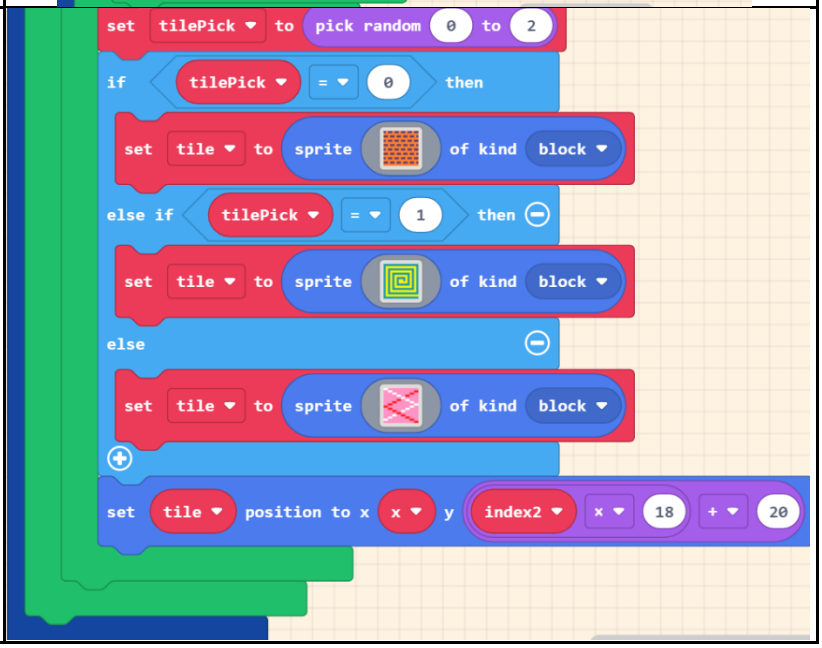


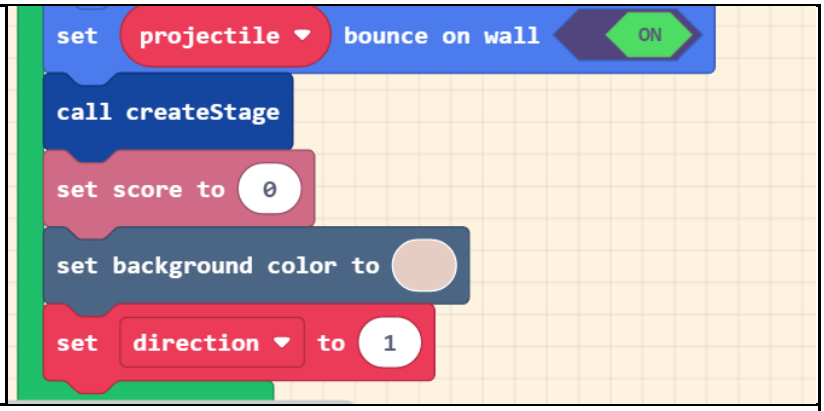
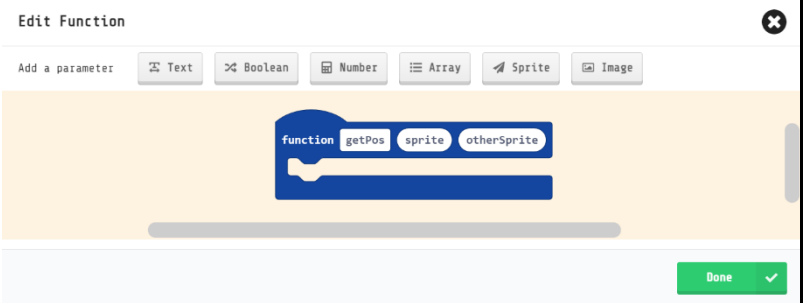
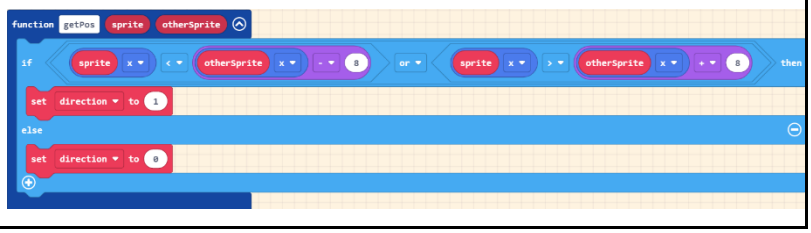
STAGE CREATION

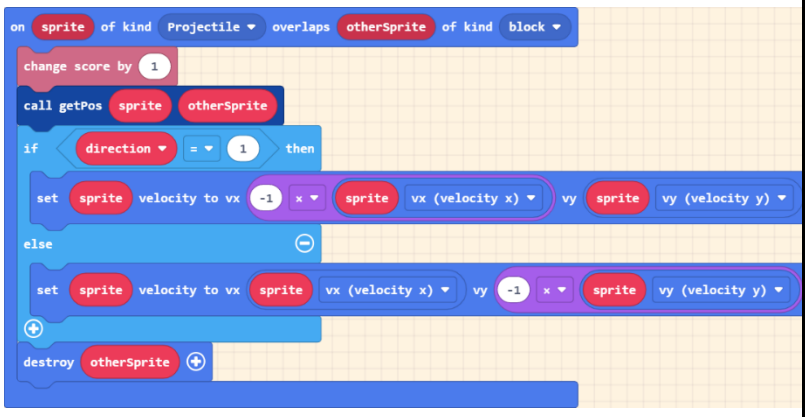
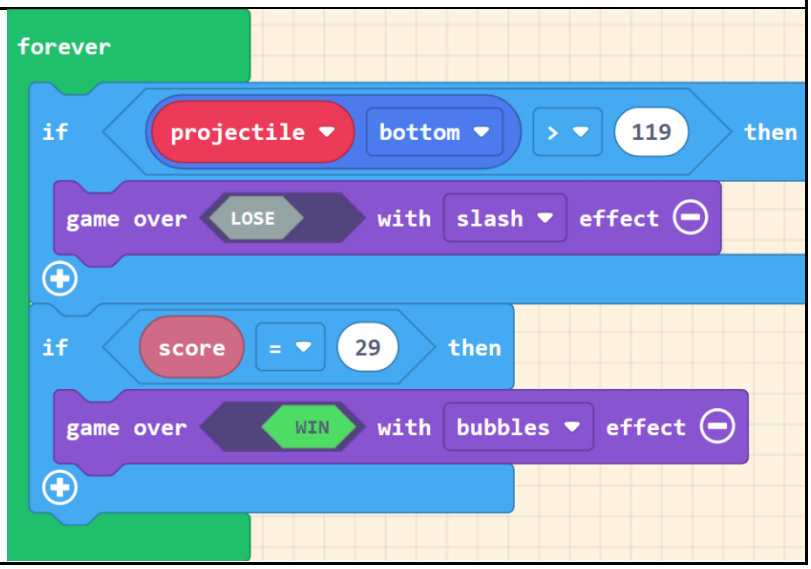
We are going to create our scenario, but instead of placing the blocks one by one, we will instruct the game to create them and then place them side by side. We add them in the function createStage.



We will use a loop, and the index will range from 0 to 9. With this, we will create the number of blocks that the rows will contain, a total of 10. In computer science, counting typically starts from 0 rather than 1, which is why the starts from 0 and ends at 9. These 10 numbers are included in the loop.	
Now, to create the number of rows in our level, we will add another for loop inside our existing loop. We will set the index2 to range from 0 to 2. This way, we are instructing the game to have 3 rows.	
First, we will instruct the game to multiply by 18 each index (16 for the size of our block and an additional 2 for spacing between blocks) to establish the spacing for our blocks.	

Next, we will add an if conditional statement and instruct the game that if the remainder of the division between index2 and 2 is equal to 1, we will multiply x by 18 and add 8. This will allow the blocks to move to the next row and have vertical spacing.	
With the foundation of creating rows and blocks per row, let's instruct the game to select blocks randomly. To do this, we will start by creating the variable " tilePick" and assign it a random value between 0 and 2. This will allow us to choose blocks randomly.	
Right after that, we instruct the game to assign a specific block based on the random number obtained in the variable. It chooses a block or another and place it in the position correctly in x based on the current value of the "index2" variable	

Finally, once the function is completed, we will include it at the beginning, right after programming the ball and before declaring the	
INTERACTION MECHANICS	
This mechanic is simple. We will instruct our game that when our (Projectile) touches the bar (Player), it should maintain its velocity in x direction but change its velocity in y direction to the opposite direction. To achieve this, we just need to multiply its velocity in the y direction by -1.	
BALL BOUNCING WITH WALL MECHANIC	
To begin, we are going to create a function that tells us where the ball has hit. For this, we will include 2 parameters of type Sprite named sprite (ball) and otherSprite (block) in the function.	
Inside the function, we will make the following checks: If the ball hits the left corner or the right corner of the block, we will set direction to . Otherwise, we will set it to 0.	

Once the function is implemented, we will instruct the game to perform the following actions when the ball hits with the block: First, we will add one point to the player. Then, we will perform the check inside the function. Lastly, if direction is equal to 1, we instruct the game to multiply vx by -1 to change the horizontal direction while maintaining the same vy. If direction is not equal to 1, we multiply by -1 the vy to change the vertical direction while keeping the same vx.	
END OF THE GAME MECHANIC	
For this mechanic, we will continuously check the following: First, if the ball descends more than 119, the game is lost. Second, if the score is equal to 29, the game is won.	

Thanks to this programming, we have created a very basic "Block Out" game. We have learned how to automatically generate a game scenario using loops, as well as how to perform collision checks and apply different effects to them. Now, it is your turn to customize it and add content. Here is ours for you to get inspired a bit: **https://makecode.com/_Pj54xtFvfddj**

Glossary

If-Else: A conditional statement that executes a sequence of instructions if it is true, and another sequence if it is false.

Comparison Operators: Operators that compare one value to another and are used within a condition.

Variables: A space associated with an identifier that contains a value that can be modified.

Functions: A subprogram that contains a set of instructions and can be executed from the main program by calling it.

Acceleration: The change in velocity per unit of time.

Velocity: A physical magnitude that relates position to the increment of time.

Walls: Objects or spaces where different elements of the game cannot pass through.

Score: The total points a player obtains from certain interactions.