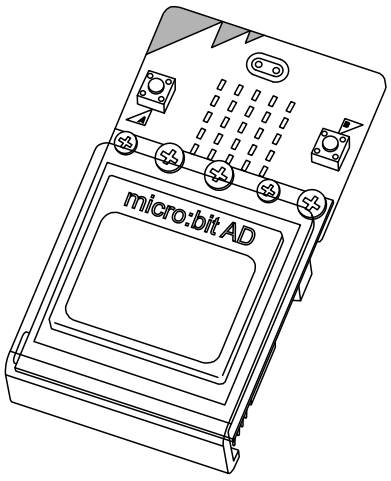
MakeCode programming

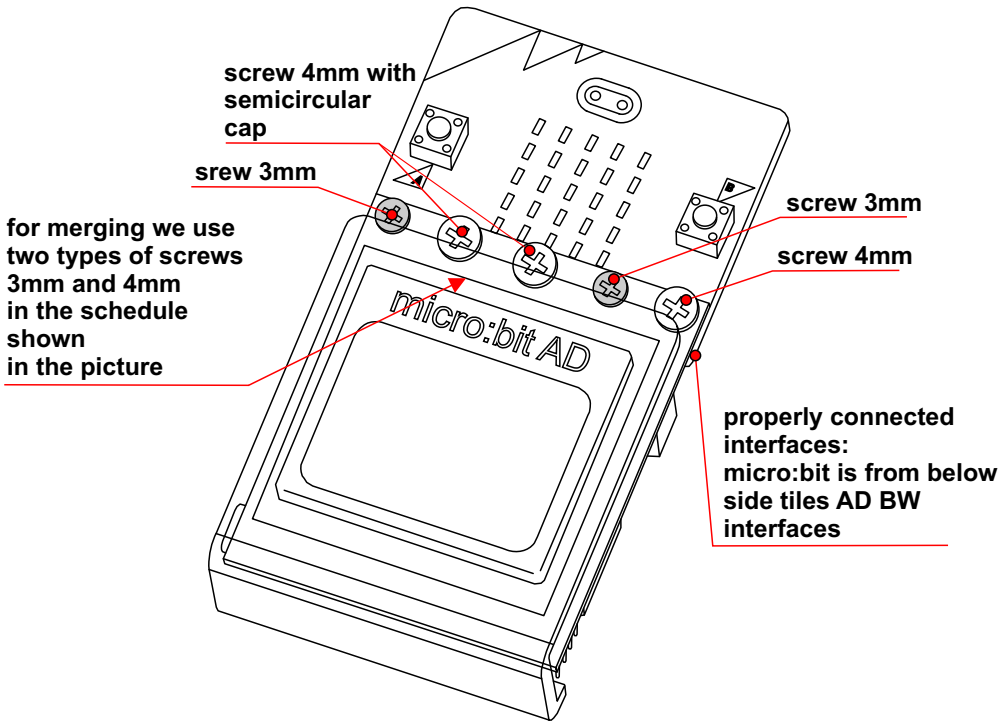
micro:bit AD BW



DIDACTA ADVANCE d.o.o.
Čakovec, Croatia (EU)
www.didacta.hr

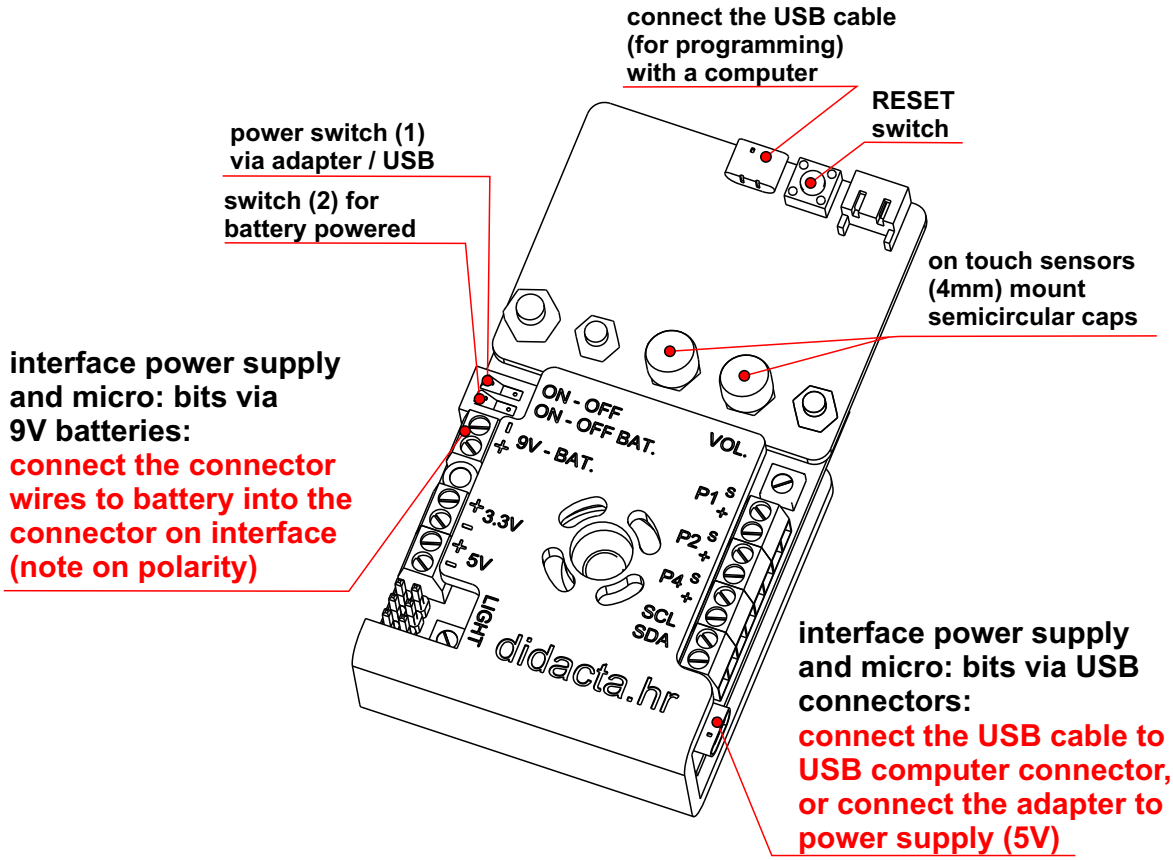
1. BEGINNING

1.1. Connecting micro:bit and AD interface with screws



We use smaller screws in positions where damage could occur elements on the micro: bit board, which are located near the connection opening.

1.2. Power supply of micro:bit and AD interface via battery, adapter or USB port



1.3. Interface launch

After startup, the text shown in Figure 1 will be displayed on the micro:bit AD BW interface screen.
The interface is ready to work.
If the program is already loaded in micro:bit, you need to press the RESET button on the micro:bit, to start program.

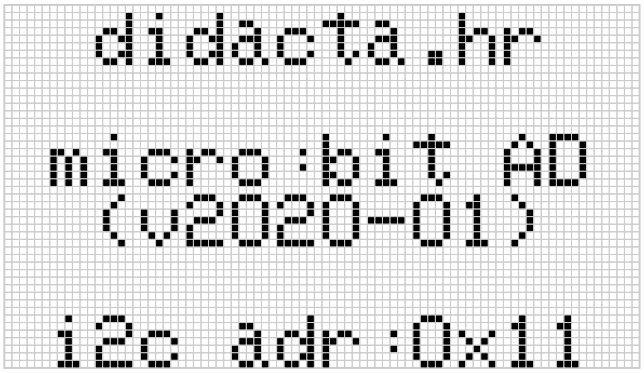


Figure 1.

1.4. Interface screen

The micro: bit AD BW interface has a black and white screen with a graphic resolution of 48 x 84 pixels (Figure 2).

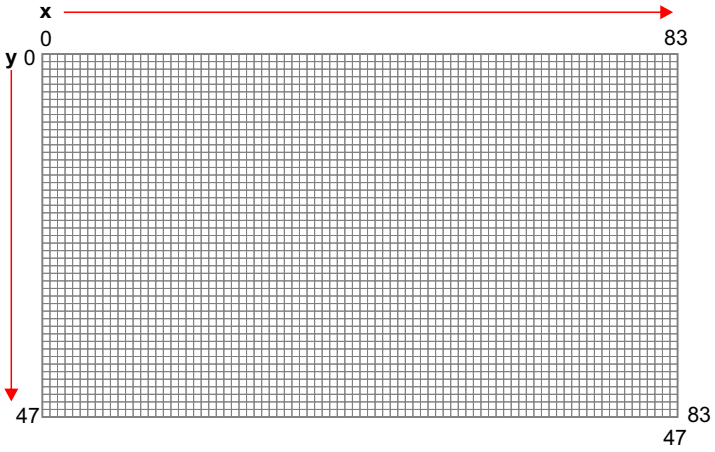


Figure 2. Graphics mode - resolution 48 x 84.

A resolution corresponding to the font size is used to print the text (text mode). Standard size text character (font font is 7 x 5 pixels) is 8 x 6 pixels with space pixels. That is why it is a resolution to print 6 x 14 characters (Figure 3). When creating a program, we must take into account which program commands we also use which mode of operation they are intended for.

Graphic functions use graphic resolution (line, circle, rectangle, ...), and text mode is used in standard text printing (not graphic) and when defining the game screen, and positioning the player object.

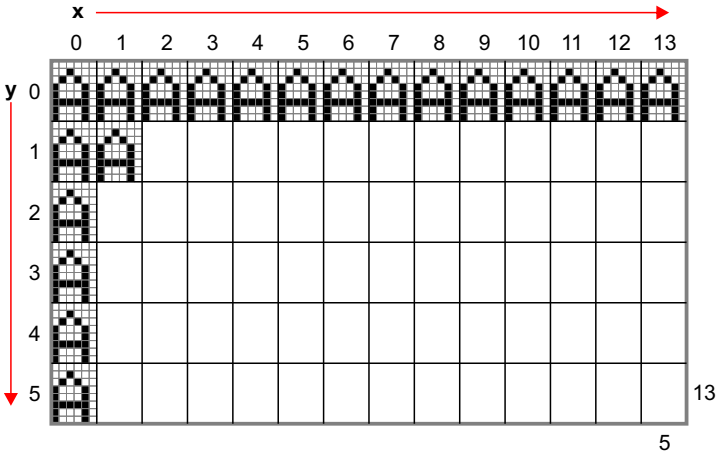


Figure 3. Text mode - 6 x 14 resolution.

2. MAKECODE INTERFACE AND LIBRARY

2.1. Launching the library programming and loading interface

Run the MakeCode programming interface in your internet explorer (link):

https://makecode.microbit.org/#editor

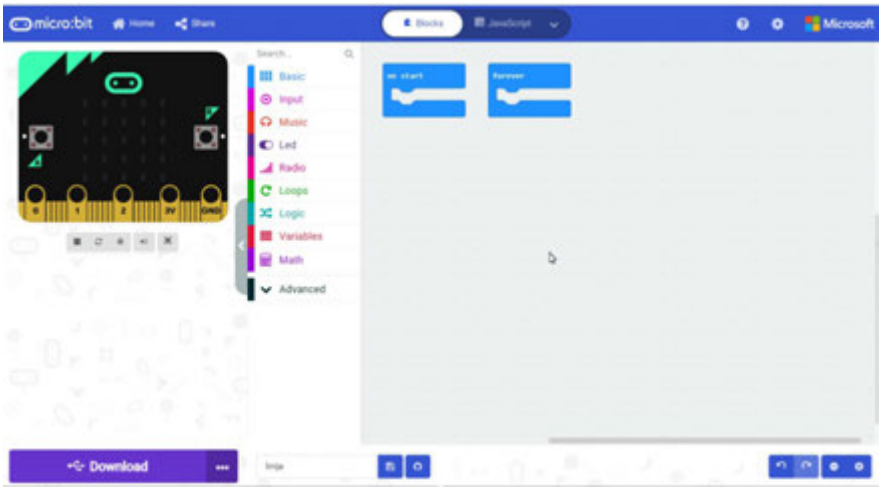


Figure 4. Makecode home screen interface

Download the library for the micro:bit AD interface:

https://github.com/didacta-advance/ADbw

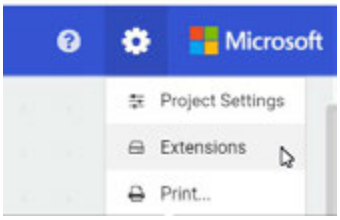


Figure 5. Select Extensions - to enter the library address

Click setup ( ), and then select **Extensions**.

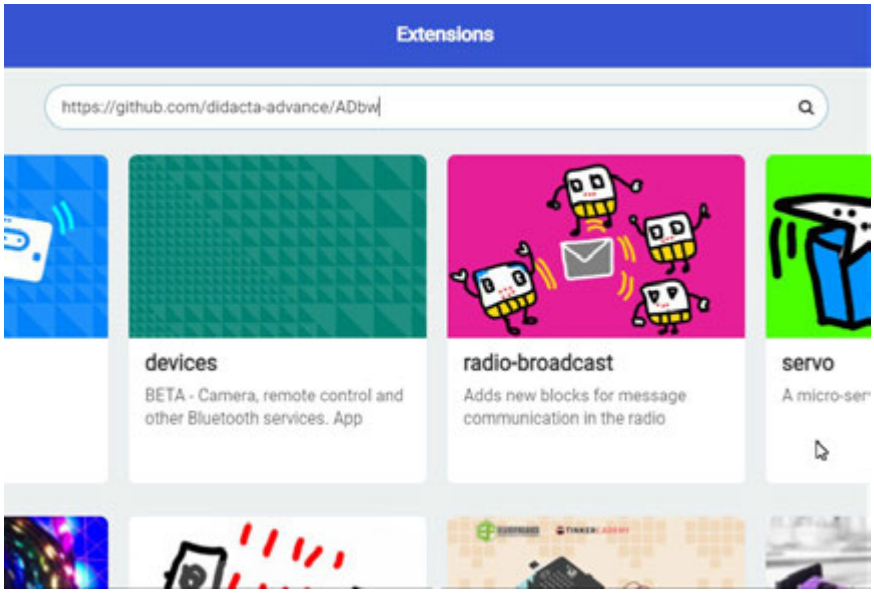


Figure 6. Entering the library address (the screen may look different)

Enter the address of the library.

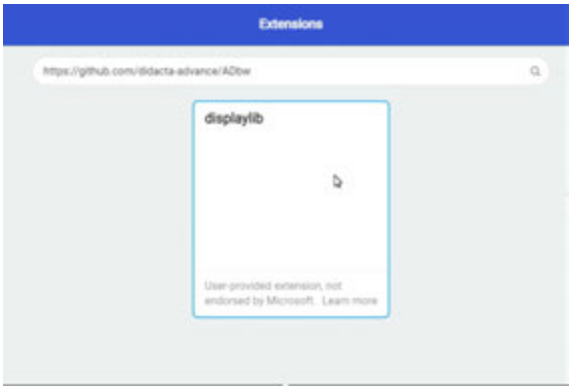


Figure 7. Selected library

Click on the **displaylib** library window to start loading (Figure 7).

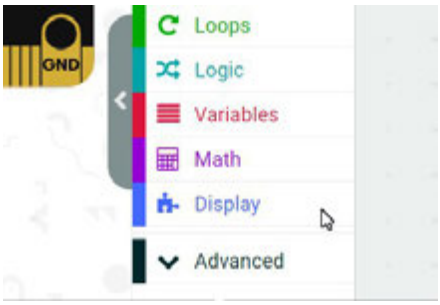


Figure 8. Library in the makecode interface menu

After loading the library, the name of the **Display** library should appear in the menu (Figure 8).

3. PROGRAMMING - BASIC FUNCTIONS

3.1. First program - text printing "HELLO" - TEXT function

In the first program we use the **TEXT** function (Figure 9).

Figure 9. RESET PROGRAM function

Figure 10. TEXT function (text mode)

At the beginning of each program it is good to use the **RESET PROGRAM** function which deletes the values that is program micro:bit AD used in previous work of the program on micro:bit.

We insert the **TEXT** function into the already existing program block **on start** which is located on the working surface of the interface. In the field ‘’ ‘’ enter the text **HELLO** and the print position (x, y) on the screen (text mode) according to the example in Figure 11. Load program in micro:bit via the **DOWNLOAD** command. The text **HELLO** in position should be displayed on the screen entered values (Figure 12).

Figure 11. HELLO text printing program

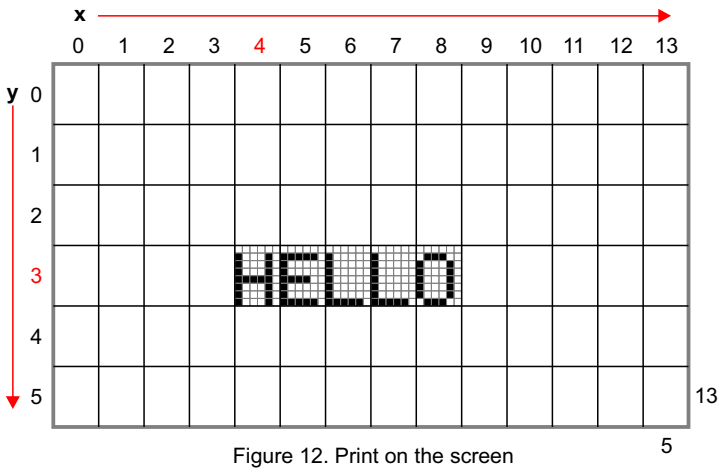


Figure 12. Print on the screen

Print on the screen after starting the first program Figure 12.
By entering a value of 2 or 3 in the **size** field, you can print text larger than the standard dimension (Figure 13).

Figure 13. Print on the screen

3.2. Multiple text printing via the repeat function

In this program we use the **repeat** function and the variable **brojac** that we use as **y** value in the **TEXT** function.

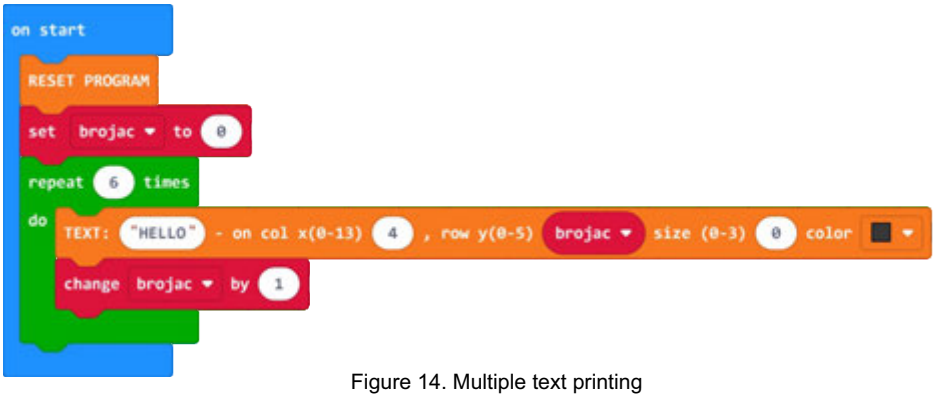


Figure 14. Multiple text printing

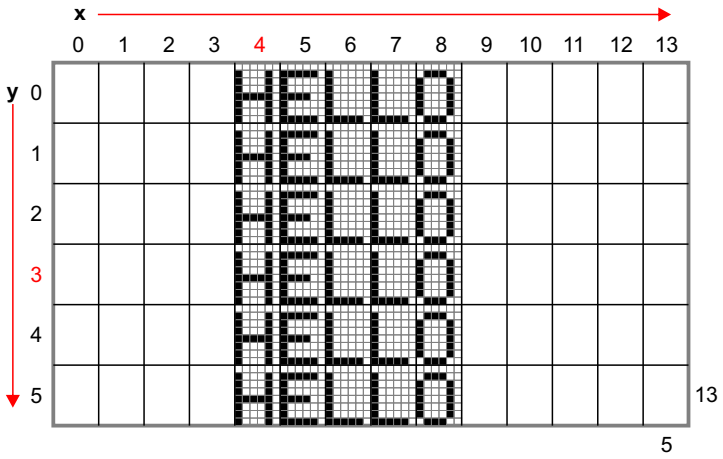


Figure 15. Print the previous program

3.3. Print text in two positions and clear the screen

In this program we use the **TEXT** function and the **CLEAR SCREEN** function (Figure 15).

Figure 16. CLEAR SCREEN function - clears the screen

In the **on start** block, leave the **RESET PROGRAM** function.

Move the **TEXT** function to the **forever** block and enter the values according to the example. After the first functions add a half-second **pause** (500 ms). Repeat this one more time and enter other values into a new function, according to the example in Figure 15.

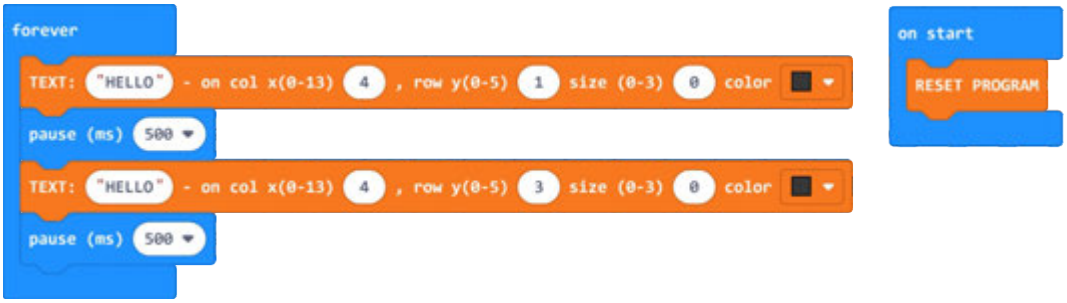


Figure 17. HELLO text printing program in two positions

Add the **CLEAR SCREEN** function to the program and load the program.

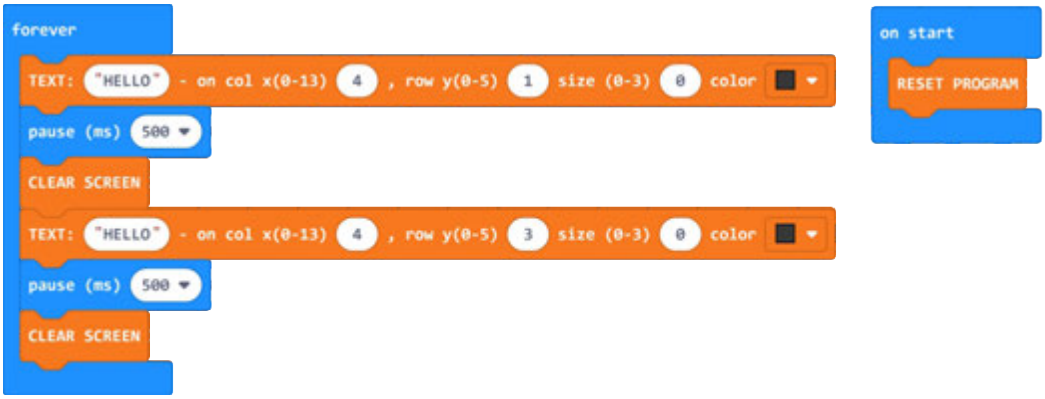


Figure 18. HELLO text printing program in two positions

What is the difference between the display on the screen ?

Remove the **CLEAR SCREEN** function from the program. Enter the same values for the position in both **TEXT** functions, and change the print color to another function.

What is the result of this change ?

3.4. Print text in graphic mode

In this program we use the **TEXT** function to print text in graphic resolution (graphic mode Figure 2.). With this function we can print text at any position on the screen.

This function does not print text directly on the screen but in the **BUFFER** (auxiliary memory), so after one or more The **TEXT (Graphics)** function must be executed by the **SHOW: buffer** function which displays a record from the auxiliary memory on the screen.

Figure 19. CLEAR SCREEN function - clears the screen

Figure 20. Function to print auxiliary memory on the screen

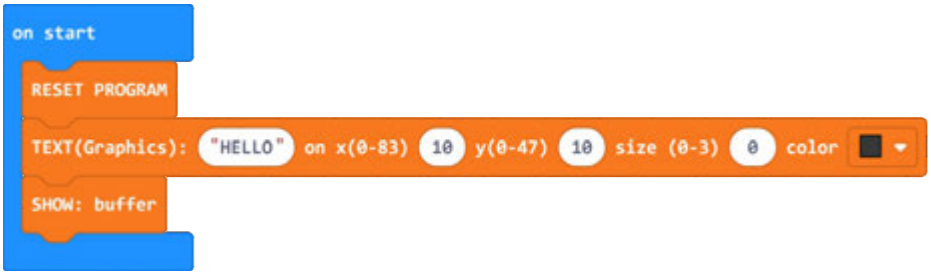


Figure 21. Example a program for printing a single line of text in graphic mode

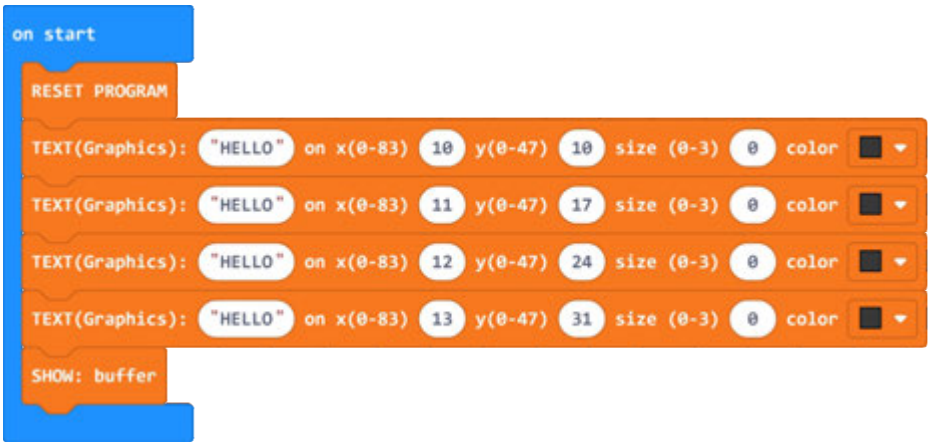


Figure 22. Example a program for printing multiple lines of text in graphic mode

An example of a program that prints numeric values at a specific position using the **TEXT** function.

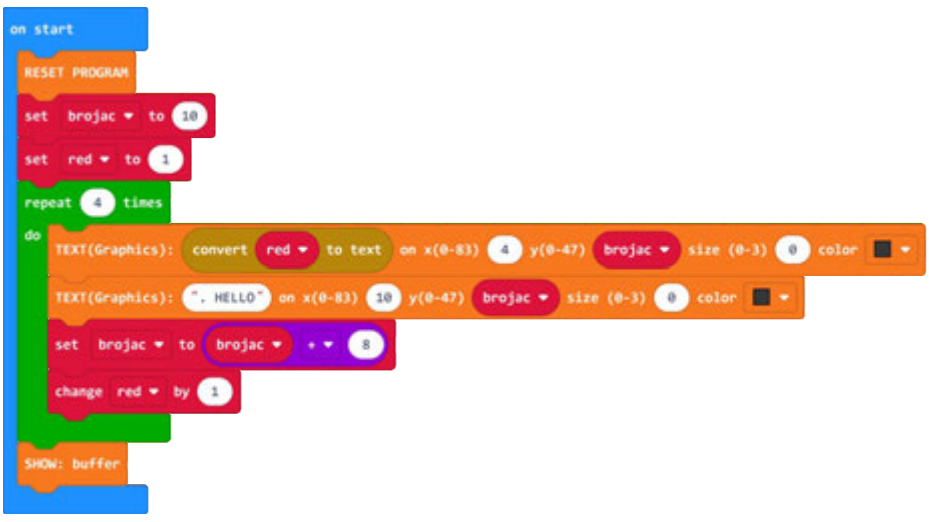


Figure 23. Example a program for printing multiple lines of text in graphic mode

3.5. Drawing a line

In this program, we use the **LINE** function to draw a line on the screen in graphics mode. We need to specify the start point (x1, y1) and end (x2, y2) point of line, on the screen. We can draw the line in black or white color. If we want to delete an already drawn black line, we need to draw a white line in the same position.

Give it a try!

Figure 24. Drawing a line in black

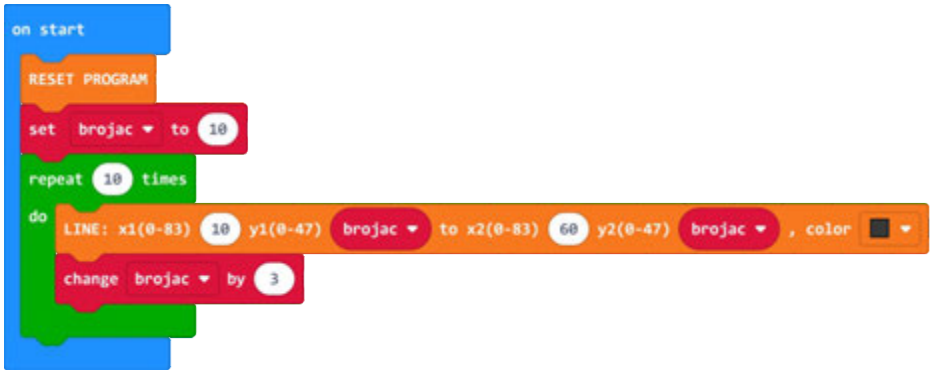


Figure 25. Drawing multiple lines in black

Figure 26. Draw multiple lines in black using the **PICK RANDOM** function

3.6. Drawing a circle or filled circle

The circle or filled circle is not the correct shape because of the screen resolution.

In these examples, we use the **CIRCLE** function to draw a circle, in graphical mode. At the circle we need to determine the position of the center (x, y) of the circle on the screen and the radius. We can draw a circle in black or white color. If we want to delete an already drawn black circle, we need to draw a circle at the same position in white color. To draw a filled circle, we use the **color filled**.

Give it a try!

Figure 27. Function for drawing a circle or filled circle

Figure 28.

Draw a circle of radius 20 pixels at position x = 40, y = 20 (Figure 28).

Figure 29.

Draw several circles of different radius on the same at position x = 40, y = 20 (Figure 29).

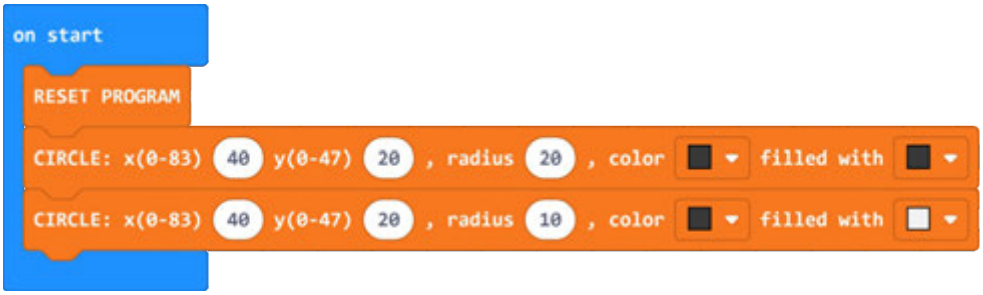
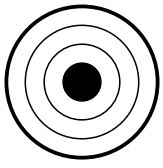


Figure 30.

Draw a white circle of radius 10 inside a black circle of radius 20 pixels at position x = 40, y = 20 (Figure 30).

For the exercise, you draw a target with a thick outer edge and a center in black.



3.7. Drawing a rectangle

In these examples, we use the **RECTANGLE** function to draw a rectangle in graphics mode. We need to determine the position of the upper left corner (x, y), width (0-83) and height (0-47). Rectangle we can draw in black or white color. If we want to delete an already drawn black rectangle, we need to draw, in the same position, a white rectangle. A rectangle can only be drawn with lines or filled filled with.

Give it a try!

Figure 31. Rectangle drawing function

Figure 32.

Draw a rectangle at position x = 5, y = 5 width 73 and height 37 pixels (Figure 32).

Figure 33.

Draw several rectangles of different positions and sizes (Figure 33).

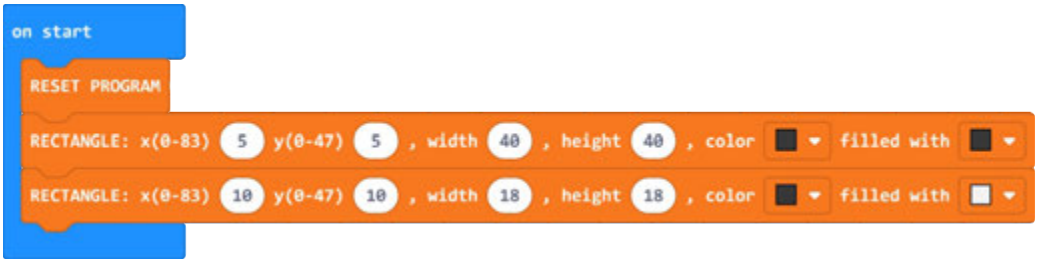
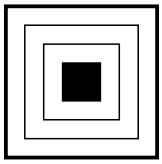


Figure 34.

Draw a white rectangle width and height 18, inside a black rectangle width and height 40 pixels (Figure 34).

For the exercise, you draw a rectangle with an outer thick edge and a center in black.



3.8. Screen coloring - PAINT display

The **PAINT** function fills the screen with the bytes of the value entered in the **color** field. The screen is filled with bytes that are laid vertically as in Figure 35.

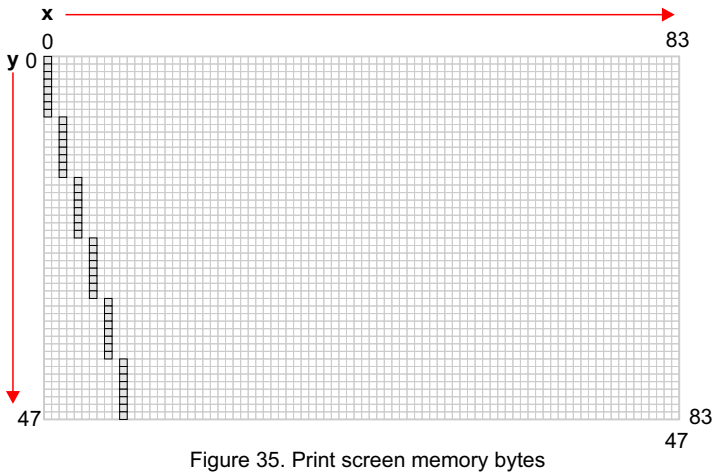


Figure 35. Print screen memory bytes

Example of filling (coloring) the screen with lines spaced one pixel apart. Color value calculation (bytes) you can see in Figure 36.

vrijednost = 1 + 4 + 16 + 64
vrijednost = 85

Figure 36. Color calculation (bytes)

Figure 37. Example program for PAINT function

3.9. Display black/white or white/black - SCREEN MODE

The screen can be set to "**normal**" mode (0) - white screen with black print, or in **inverse** (reverse) mode (1) - black screen with white print. By default, the screen is set to "**normal**" mode (0). By change mode the complete screen content is changed via the **SCREEN MODE** function. Try the following example in Figure 37. You can supplement it with text.

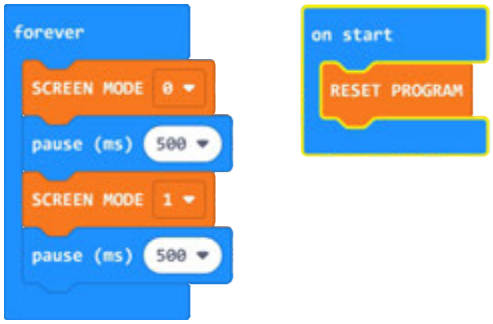


Figure 37. Example program for PAINT function

3.10. Drawing pixel

Figure 39.

Function for drawing pixels on the screen in graphic mode.

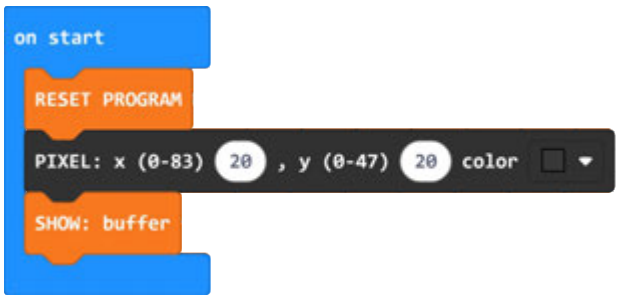


Figure 40.

Draw one pixel to the screen in graphic mode, at position x = 20, y = 20.

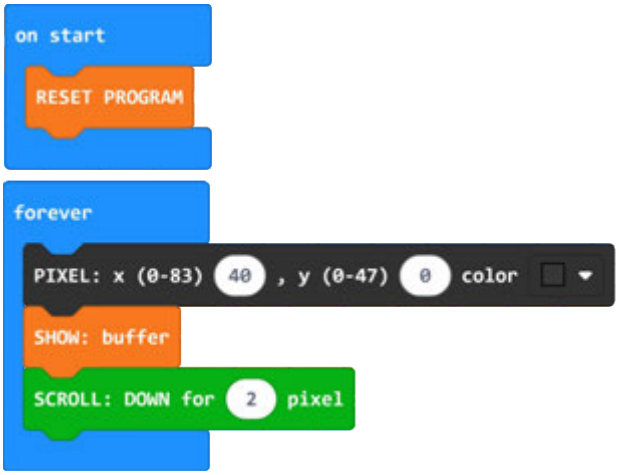


Figure 41.

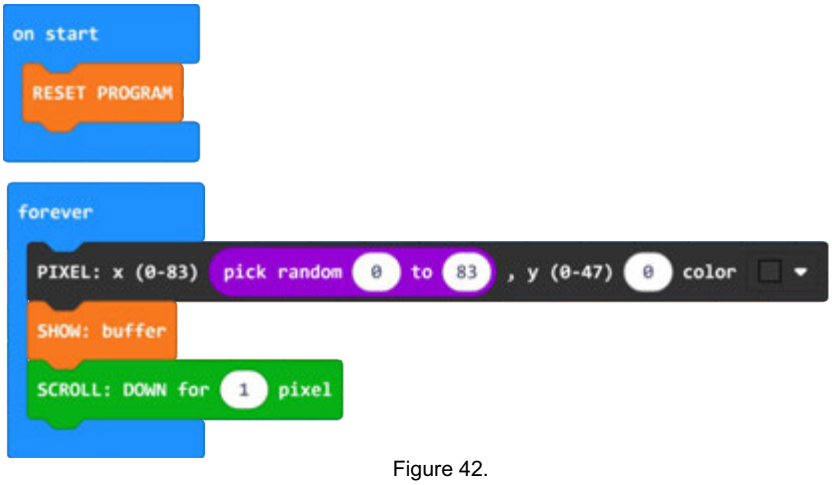


Figure 42.

Try the programs in previous figures 41 and 42.

4. PROGRAMMING - SHIFT FUNCTIONS - SCROLL

4.1. Move text UP by one line

Figure 43. SCROLL function of the text UP

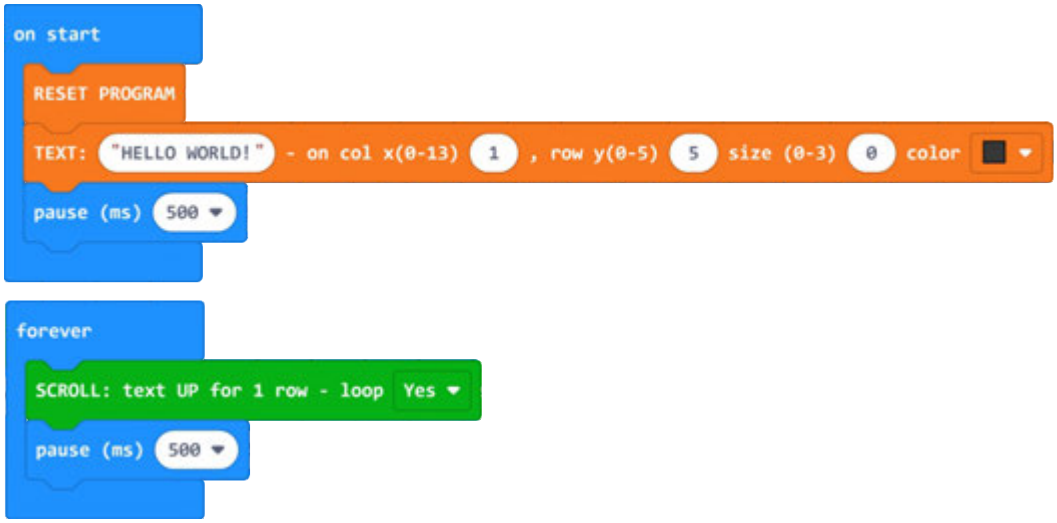


Figure 44. Example of vertical text shift

The text printed at the beginning is moved one line up, every half second (Figure 44). Shift text can be used for all text sizes.

The LOOP function has two states, YES and NO. Try changing the state to NO.
What is the difference in text offset (YES)?

4.2. Move text DOWN by one line

Figure 45. SCROLL function of text DOWN

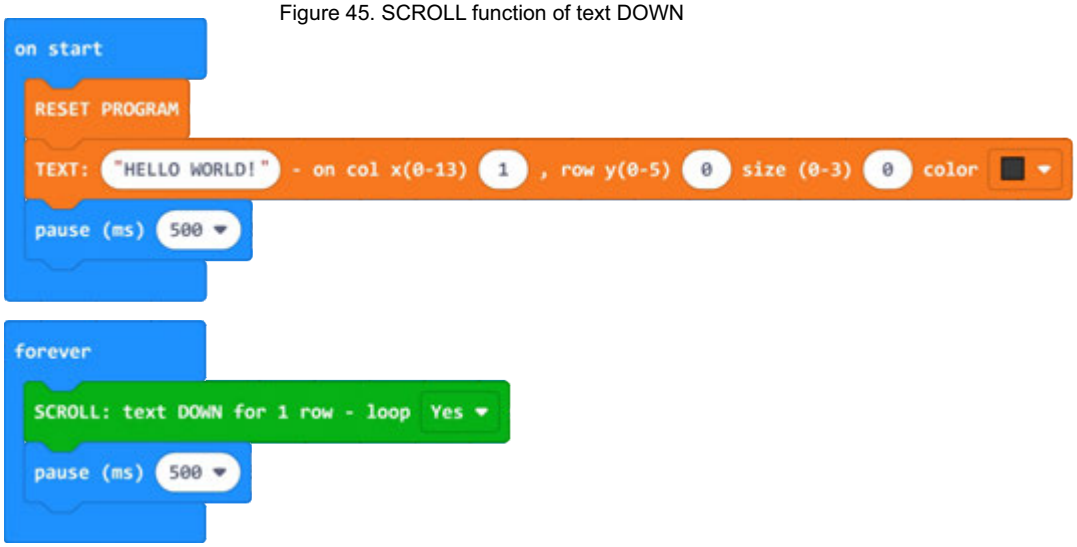


Figure 46. Primjer vertikalnog pomaka teksta

The text printed at the beginning scrolls down one line, every half second (Figure 46). Shift text can be used for all text sizes.

The LOOP function has two states, YES and NO. Try changing the state to NO.
What is the difference in text offset (YES)?

4.3. Move the screen (images) up by one or more pixels (pixel line)

Figure 47. SCROLL function of the screen up

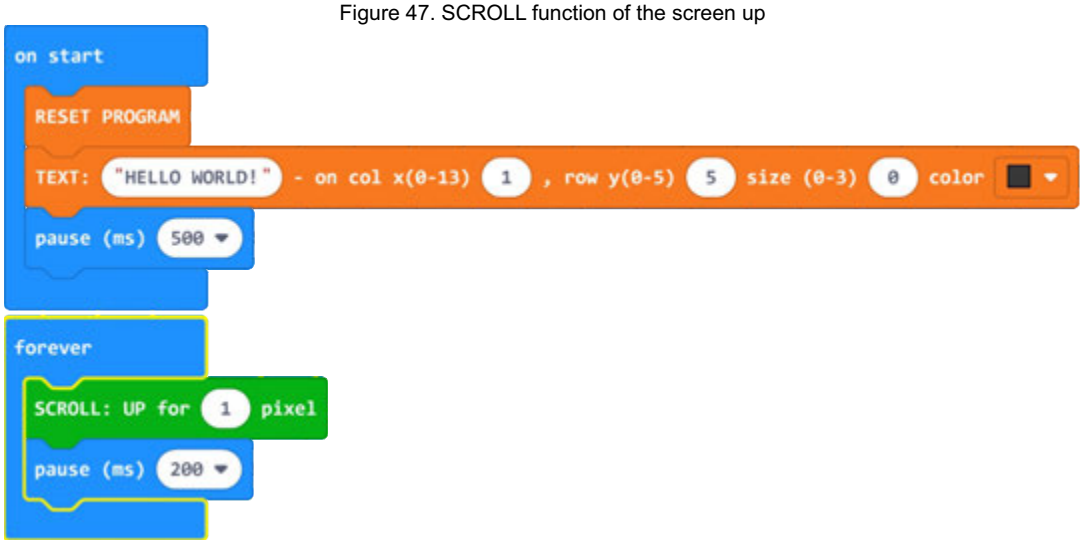


Figure 48. Example of moving the screen one pixel up

The text printed at the beginning moves one pixel line UP, every 200 milliseconds (Figure 48).
The screen offset (images) can be increased by entering a larger number in the **for** field.

4.4. Move the screen (images) DOWN by one or more pixels (pixel line)

Figure 49. SCROLL function of the screen DOWN

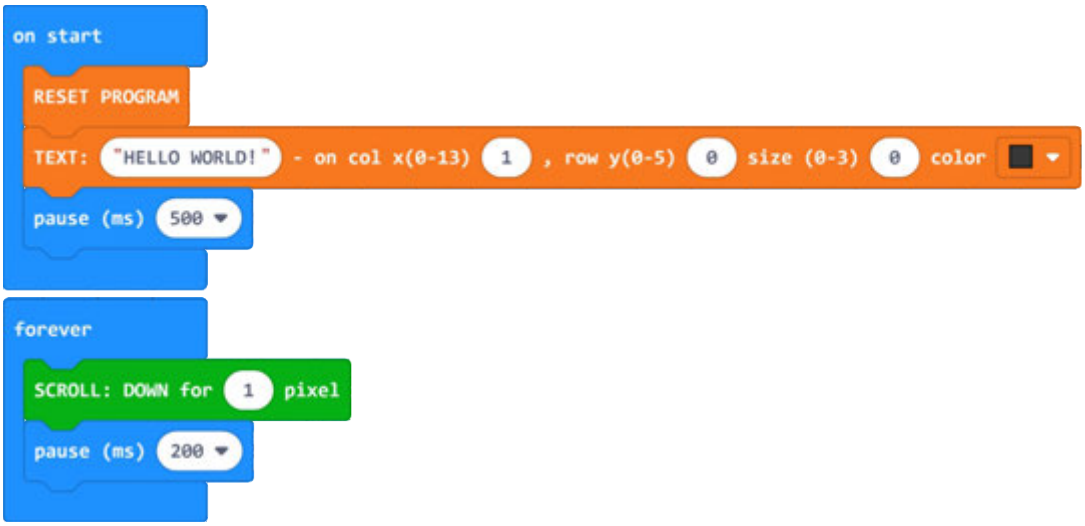


Figure 50. Example of moving the screen one pixel up

The text printed at the beginning scrolls one line of pixels DOWN, every 200 milliseconds (Figure 50).
The screen offset (images) can be increased by entering a larger number in the **for** field.

4.5. Horizontal screen shift (images) by one pixel

Figure 51.

Function to move the screen horizontally (images) by one pixel. The function allows you to select the direction of movement (LEFT - left or RIGHT - right), screen areas from line (text) to line or full screen (0-5).

As with text shift, the LOOP option can be turned on to move text or an image in a circle.

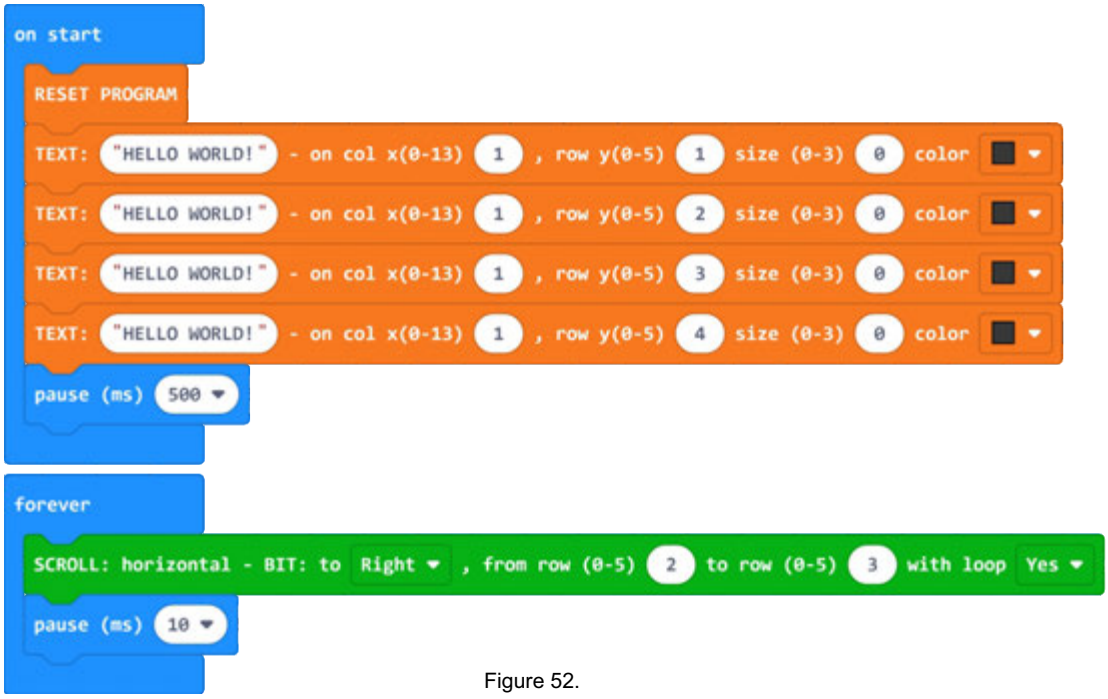


Figure 52.

Example of moving the image to the right (Right) of two middle lines of text with a circular display (Figure 52). Add lines to print the text in line 0 and 5, and change the SCROLL value to 2 in 0 and 3 in 5.

What change happened?

Try changing the direction of the shift.

4.5. Animation with scroll functions

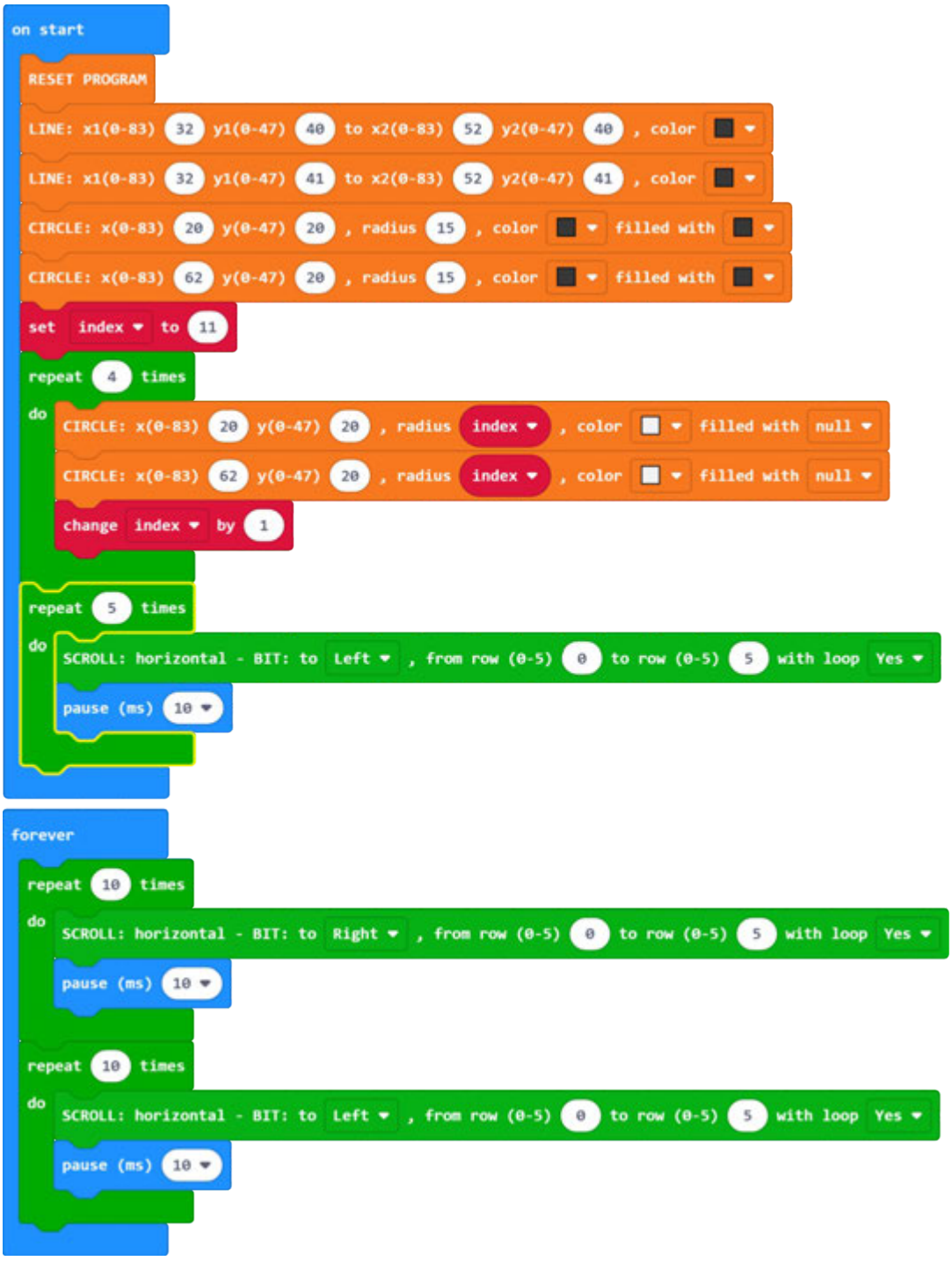


Figure 53.

Simple animation via various functions.

Try it!

5. PROGRAMMING - GAME FUNCTIONS

5.1. LED light control

The interface has two built-in LED lights, one RED and one GREEN. RED is below the screen on the left, and GREEN on the right.

Figure 56.

Figure 57.

An example of a program that alternately turns on RED and GREEN LED lights (Figure 57).

5.2. Creating BITMAPE objects

Creating graphic objects (BIT-maps or sprites) is performed in graphic mode (buffer - memory) for faster printing on the screen and avoiding certain bad effects (flickering). Therefore, the first object (one or more objects) are stored in a strong memory (buffer), and finally the memory is saved via the SHAW buffer function. displayed on the screen.

After creating the object (BITMAP), in this example CUSTOM 1, it is necessary to determine the position at which to draw the object and in which COLOR (DRAW BITMAP). A brief example with basic functions is shown in Figure 58.

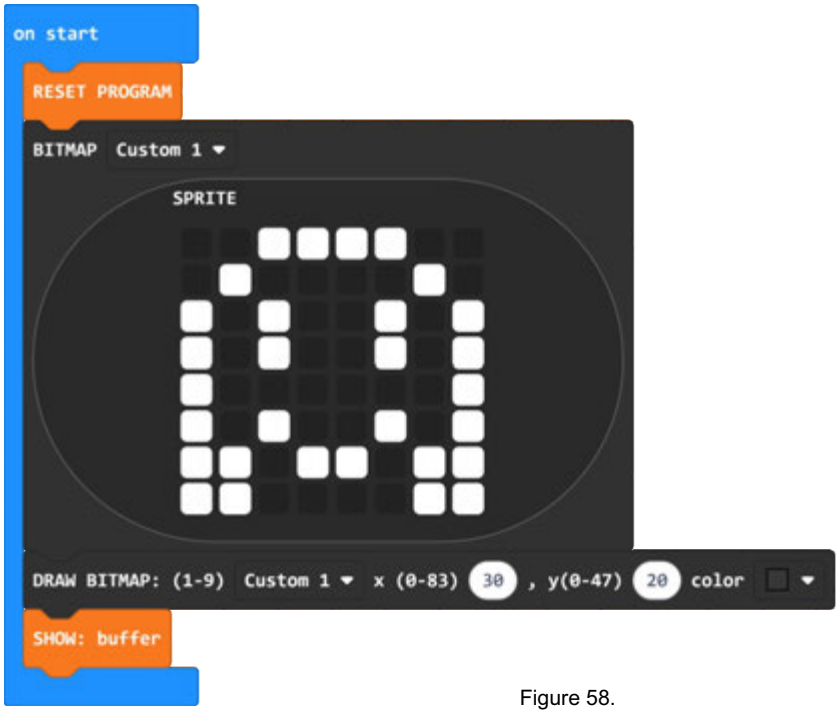


Figure 58.

The print color allows us to print the object in **BLACK** and delete it in **WHITE**.

Figure 59.

Complete the previous program shown in Figure 58 with the forever block shown in Figure 59.

5.3. PIAYER object

If we want BITMAP to be a player object in the menu we need to select **Player**. After creating the object it is necessary to run the function to display it on the screen. Text mode resolution is used for positioning (14 x 6). In the following example, the player object is plotted at position x = 5, y = 3.

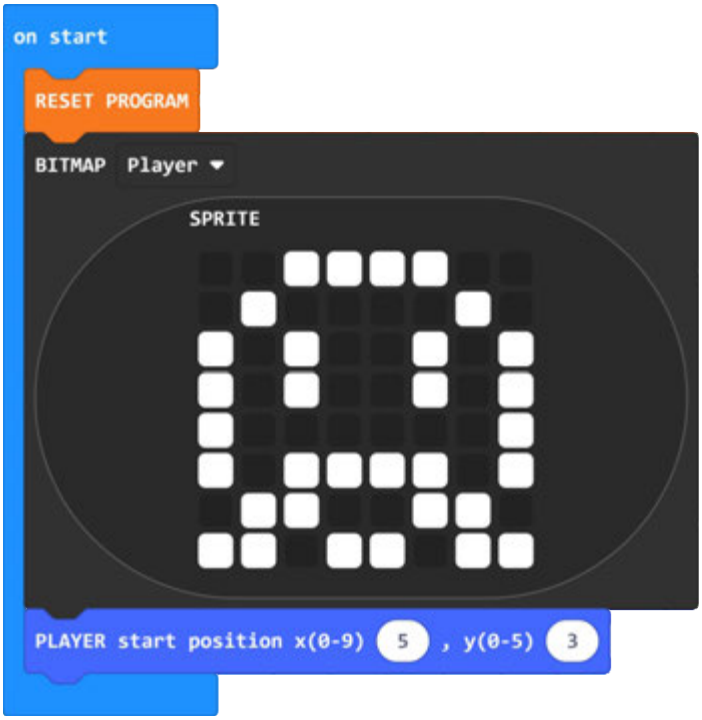


Figure 61.

The PLAYER function also displays the BUFFER status on the screen, so it is not necessary calling the **SHOW BUFFER** function.

5.4. Horizontal displacement controls

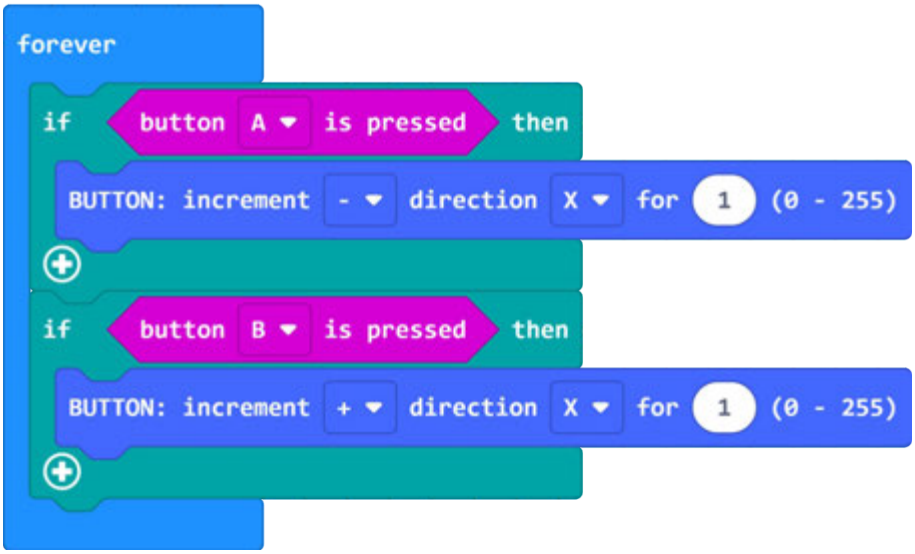


Figure 62.

In the previous program, add the **forever** block shown in Figure 62.

Test how the program works if you change the values for **direction** and **for** (pixel) with the **BUTTON** function.

5.5. Vertical controls (touch sensors)

To control the player object in all directions it is necessary to add and control the touch sensors that are located on the underside of the micro:bit. The analog reading of the touch sensors is not the same as the connected USB cable to micro:bit even when not connected. The example in Figure 63 shows the values with USB connected cable. The value without a USB cable connected to the micro:bit is **<100**.

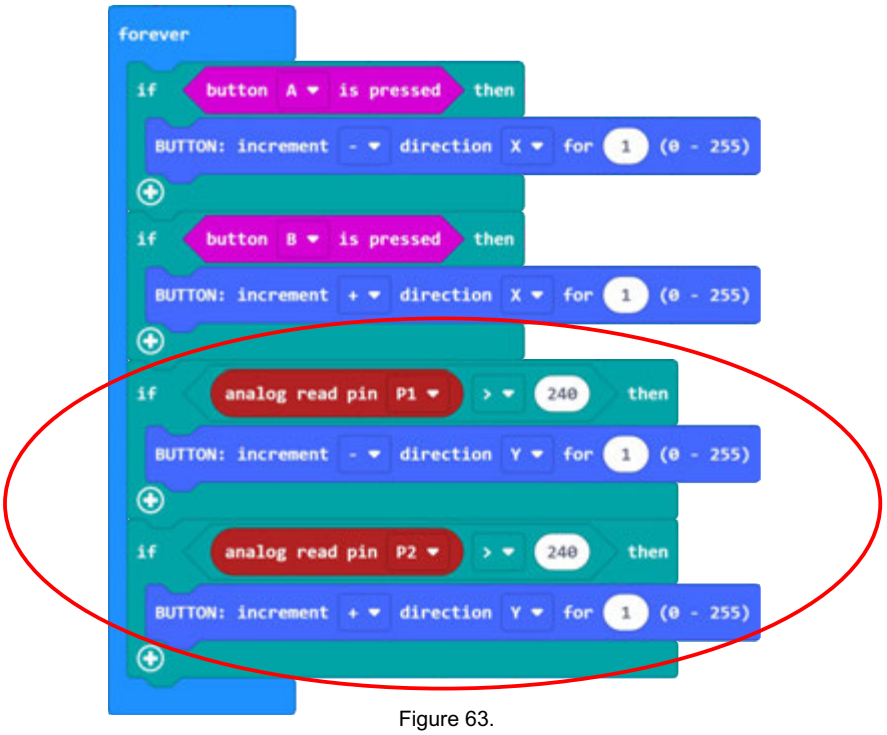


Figure 63.

5.6. Animation of a player object

To animate a player object required is another bitmap. Complete the program according to Figure 64. By moving the player object they are alternately drawn on the screen bitmap **Player** and **Player animation frame**.

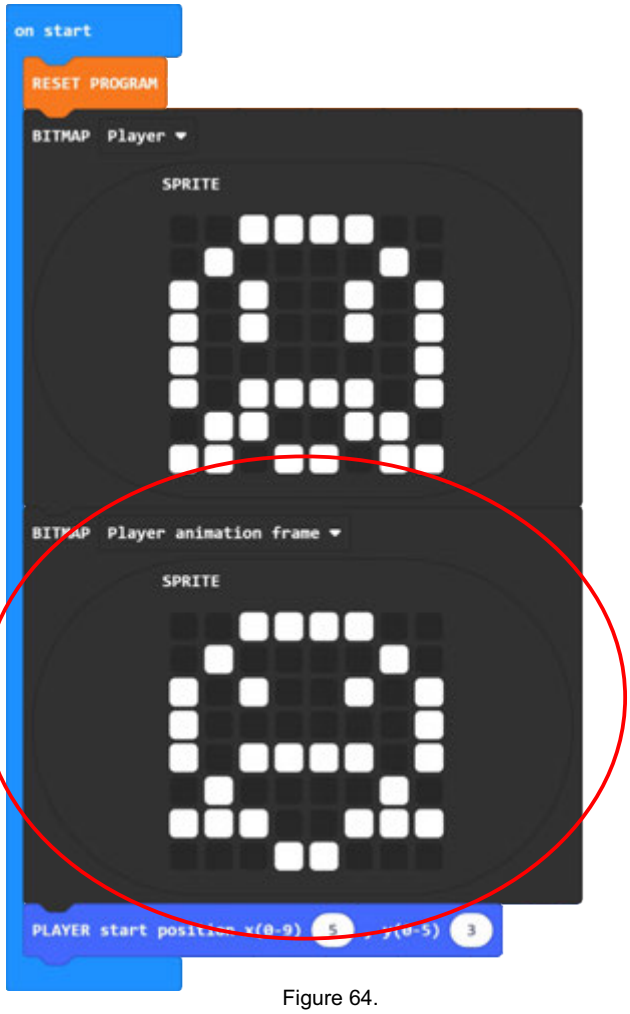


Figure 64.

5.7. Animation speed control

Animation of a player's object can be faster or slower. We use the **ANIMATION** function to control the speed (player) speed. If we want the animation (bitmap change) to be slower, we need to enter a larger one value in the speed field. The animation is performed only while the player's object is in motion.

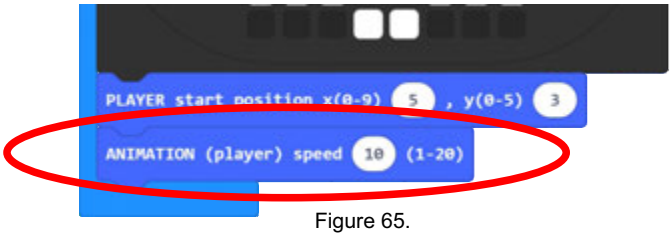


Figure 65.

Complete the program start block with the animation speed control function according to Figure 65. Try different speed values.

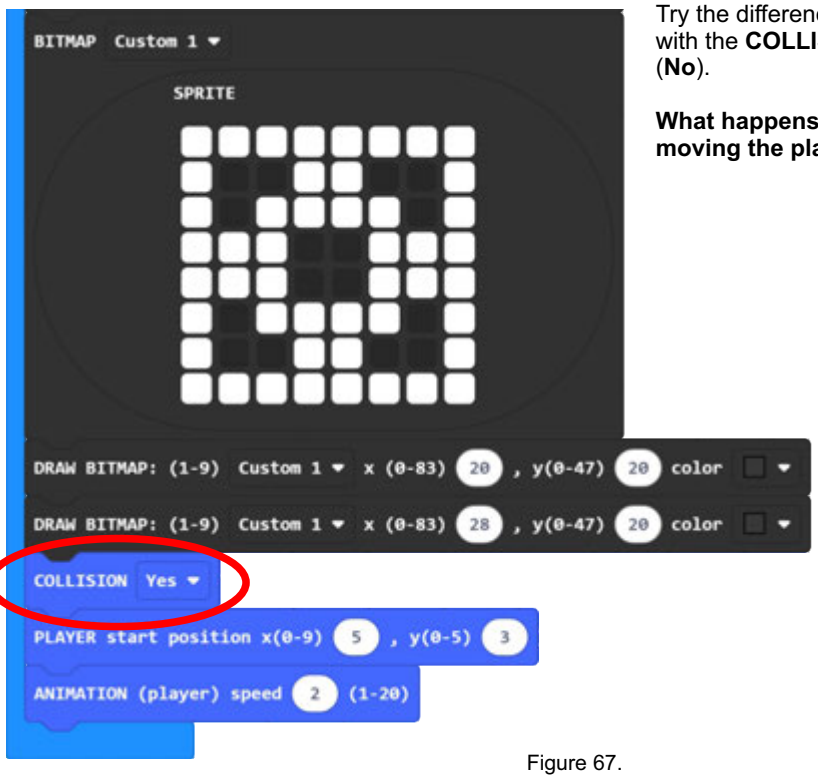
5.8. START game

We have the basic construction of the game with the control of the player's object movement. We need to put a message at the beginning which will be printed after the program starts. Standard message display function (**GAME: START message**) insert into the start block as shown in Figure 66. After printing, the function needs to be started pause so that the message can be read, and then clear the screen with the **CLEAR SCREEN** function. If we do not run the screen clear function, the player object will be drawn over the start text.

Figure 66.

5.9. COLLISION function

Fill the program with a new object (Custom 1). Draw it on the screen (**DRAW BITMAP**) several times, on different positions according to Figure 67 or arbitrarily. Complete the program with the **COLLISION** function.



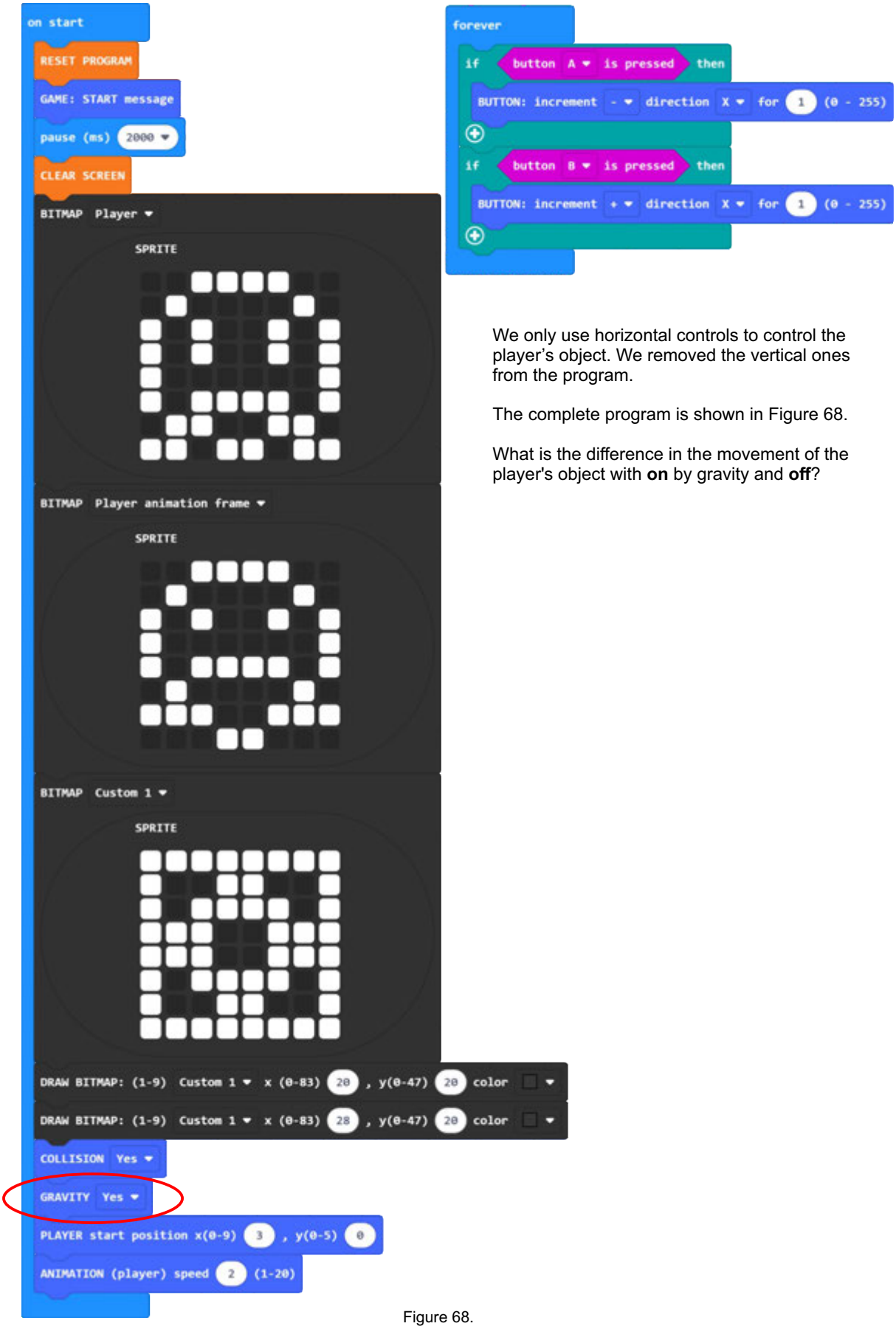
Try the difference with the included with the **COLLISION** function (**Yes**) and off (**No**).

What happens to objects on occasion moving the player object towards them?

Figure 67.

5.10. GRAVITY

In order for the movement of the player's object to be as natural as possible, and for him to be able to jump and fall, it is necessary to include him in the game gravity function - **GRAVITY** (Fig. 68 rounded).



We only use horizontal controls to control the player's object. We removed the vertical ones from the program.

The complete program is shown in Figure 68.

What is the difference in the movement of the player's object with **on** by gravity and **off**?

Figure 68.

5.11. Creating objects (horizontal and vertical) - max. 20 objects

Objects longer than one bitmap (8x8 pixels) can be placed horizontally or vertically.
Objects are created by repeating the same bitmap several times. **Objects through which points are earned or lost lives are usually the length of a bitmap, if they are longer, only the first position is active to obtain points or loss of life**.

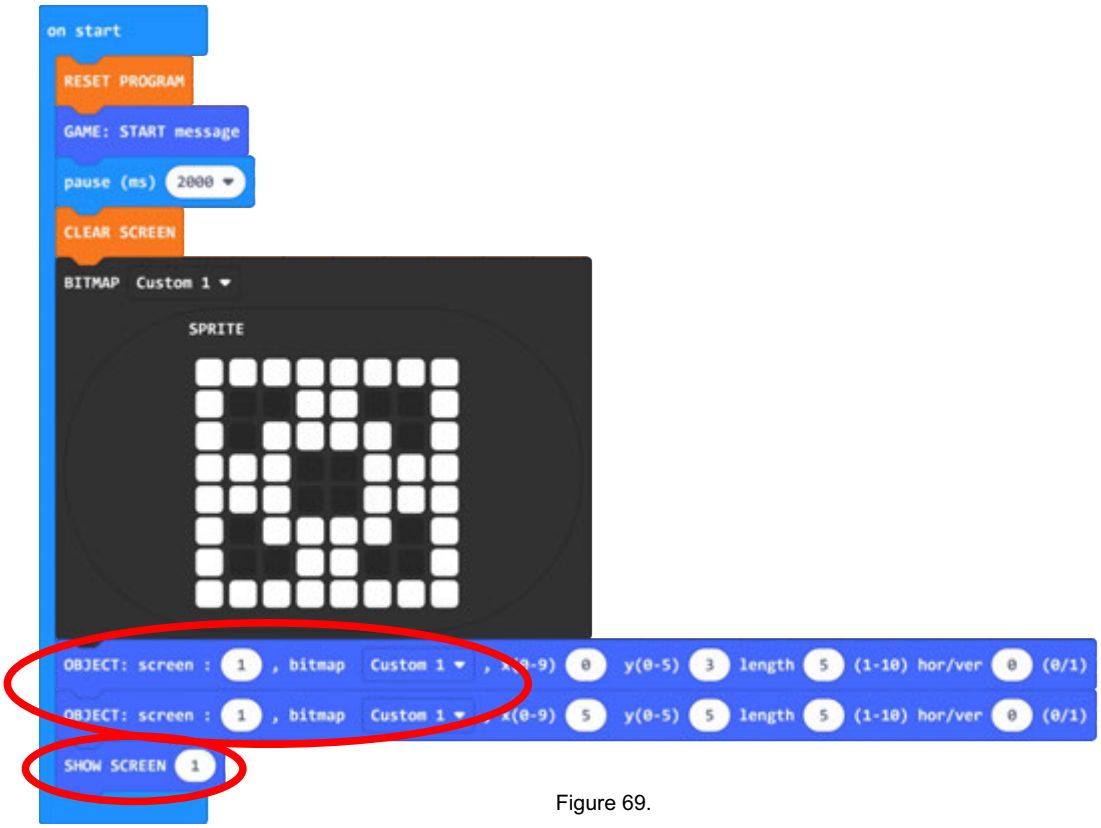


Figure 69.

We took part of the previous program (Figure 68) and added commands to create objects (Figure 69). It's the first it is necessary to define the objects that will be drawn on the screen via the OBJECT command. All commands are grouped to the «**screens**» that are displayed on the screen via the **SHOW SCREEN** command. In this example we define «**screen**» 1 with two objects. Both objects are composed of the same bitmap (custom1). For positioning it is necessary to determine the x and y position of the initial bitmap of the object. The length (number of repetitions) is determined by the entry values in the **length** field. The maximum horizontal length is 11 (84/8 = 10.5 bitmaps).

To read the bitmap vertically, it is necessary to change the value of the "**hor/ver**" field to **1**.

In the previous program (Figure 69) make changes to the values according to Figure 70.
Try the program.

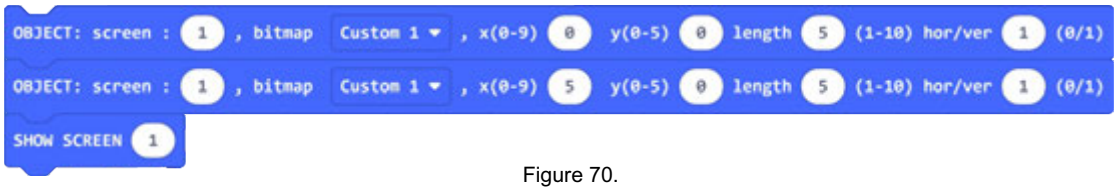


Figure 70.

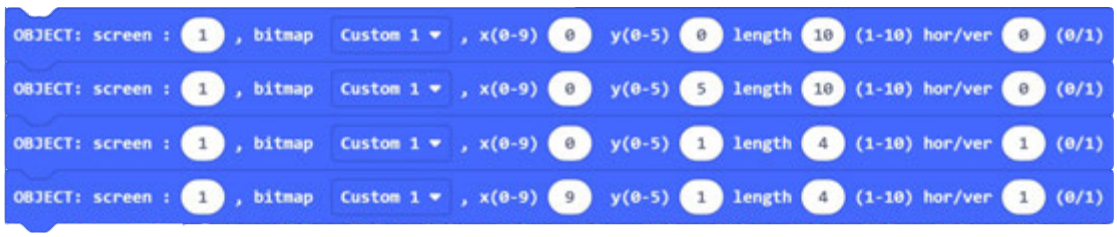


Figure 71.

Try the combination of functions according to Figure 71.
Don't forget the **SHOW SCREEN** command (1) that comes at the end.

5.12. Creating more than one "screen" - max. 5 "screens"

If we want to create more different "**screens**", it is necessary to create objects for each "**screen**". The following example is with two "**screens**" and three objects.

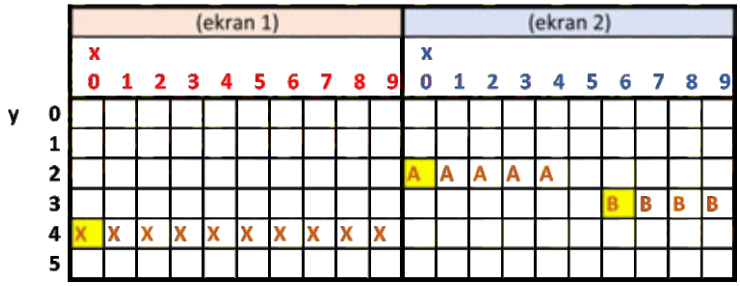


Figure 72.

When you want to create more "**screens**" it is good to make a sketch as shown in the example in Figure 72. Screens can be sketched using a spreadsheet in Excel or raster paper. This makes it much easier to visualize all screens, especially if horizontal and vertical objects are used.

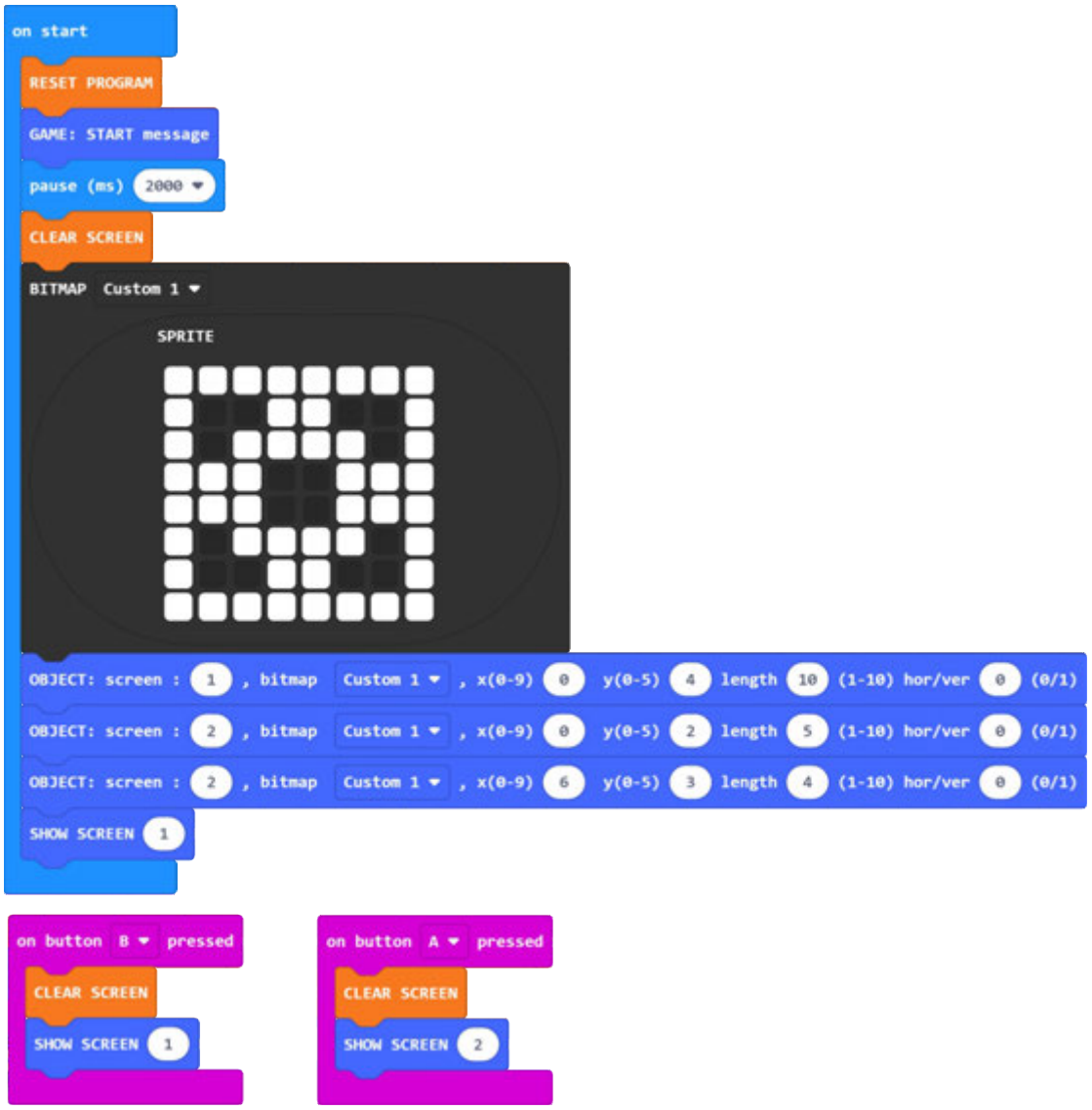


Figure 73.

The example in Figure 73 uses two “**screens**” that can be alternately displayed via the **A** or **B** key.

5.13. Sliding "screens" - display a screen with a horizontal scroll

When creating a platform game, we use mobile platforms that move from one side of the screen to the other. To run the platforms (objects) we created in the previous program (Figure 73) we need to add the **GAME SCROLL horizontal** (yes) command is activated (Figure 74). Objects on "**screens**" are printed on screen in a circular order of "**screen**" numbers (1,2,1,2,1,2, ...)

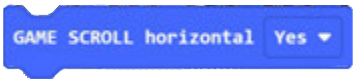


Figure 74.

The horizontal movement speed of objects can be changed with the **GAME SPEED** command (Figure 75). Smaller the value of the variable means a higher rate of shift. Maximum speed is limited to 10, and default is set to 100 (0 = 10). We can also increase the speed by shifting by 2 pixels (default 1), but shift more it won't be as 'fine' as a 1 pixel offset. If the program contains a lot of control objects, when the **maximum speed** (**10**) may stop working. **In that case you need to reduce the speed because the program does not manage to process all operations in too short a time.**

Figure 75.

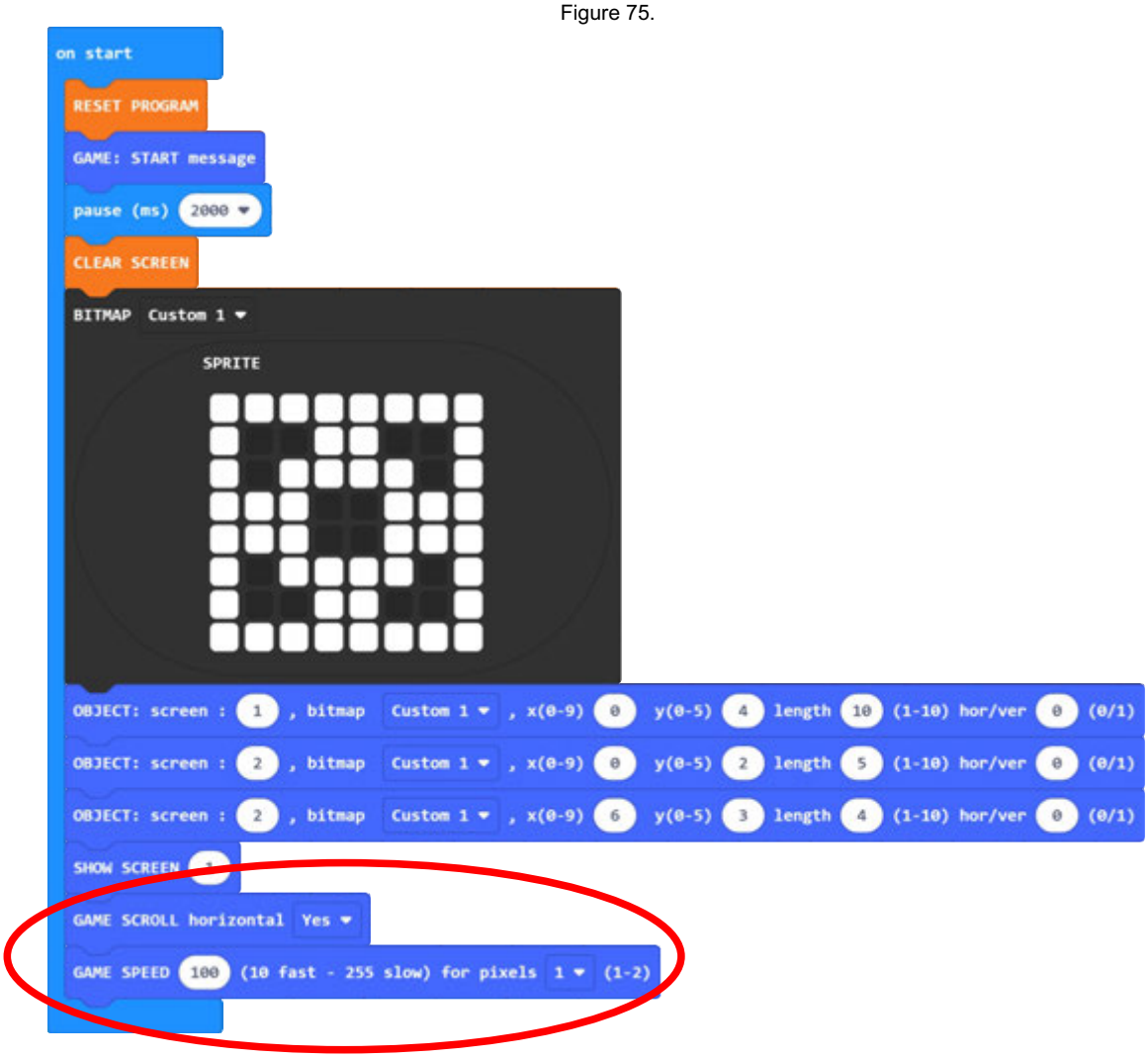


Figure 76.

To see what the differences are in the feed rate try a program with different speed values. You can expand the program to another "**screen**" (3).

5.14. Movement the player object at the scroll screen

The jump of the **''player''** directly upwards is controlled by the **JUMP UP** function, which is most often used in code platform games where the player is in the same position on the screen (horizontally). The jump height is determined in pixels. In order for the function to be active we had to add some more mandatory position and control functions movements of the **player's** object (**COLLISION, GRAVITY, PLAYER start position**).

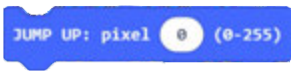


Figure 77.

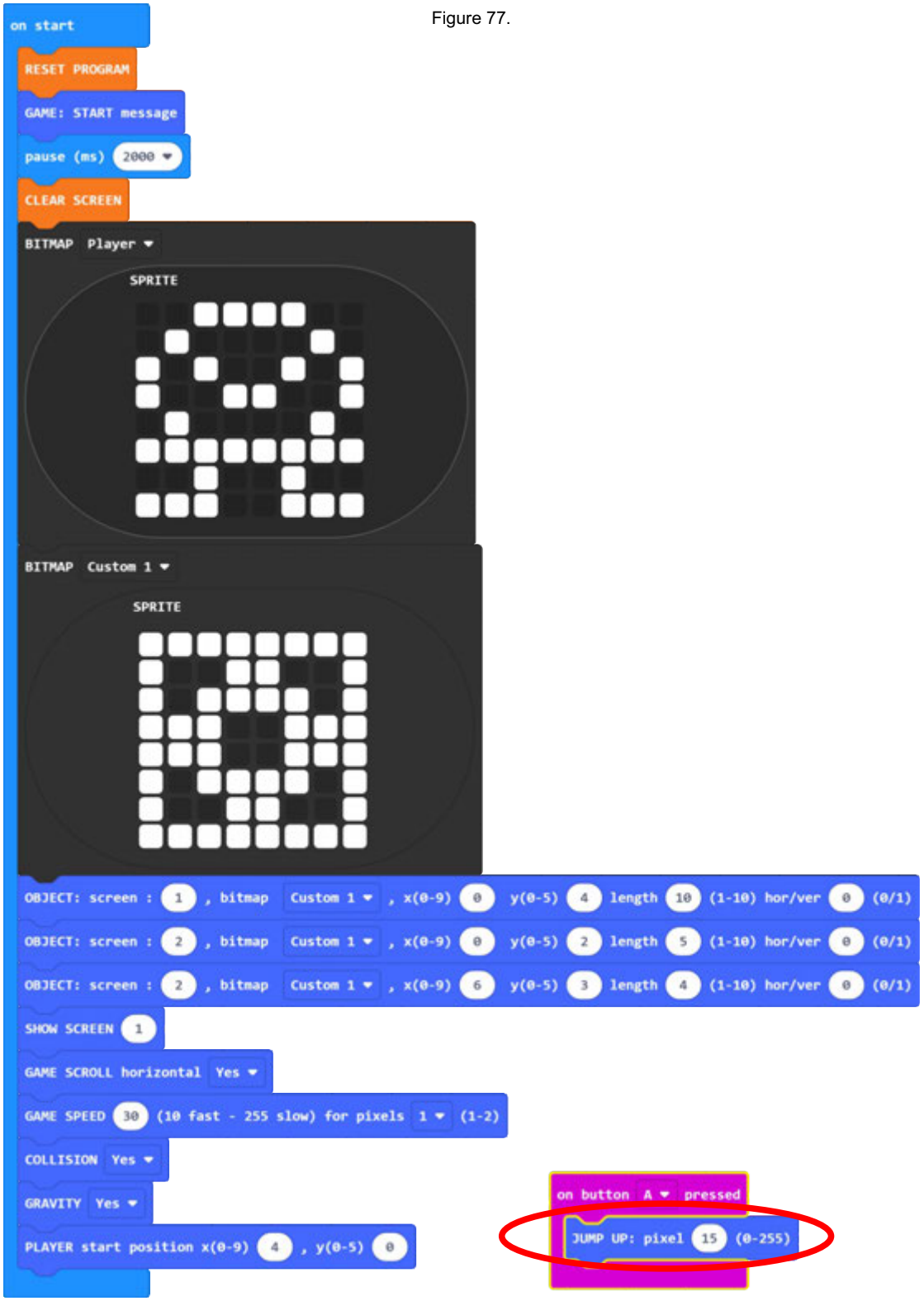


Figure 78.

We use the **JUMP** function to jump **players** to the right (**+**) or left (**-**) side. In addition to the height of the jump, as and with the JUMP UP command, we determine which way the player will move **+ = right** or **- = left** when jumping. By entering values from 0 to 5 we determine the jump angle. For a vertical jump (in place) the value is **0**, for a jump at a 45 degree angle the value (**length**) is 1. Values 2 - 5 increase the jump angle (**length**).

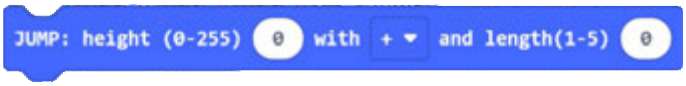


Figure 79.

To the previous program (Figure 78), change the button A function and add the button B function according to the figure below (Figure 80).

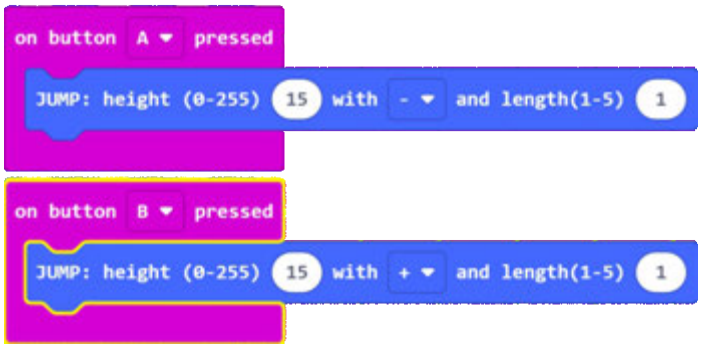


Figure 80.

5.15. Control functions

5.15.1. **Game status - GET GAME status**

In order for a program running in micro: bit to be able to perform some functions it is necessary read certain values used in the game. Game status is used for performance sound effects during the game and to know when the game is over. The function is called **ONLY ONCE** at the **beginning of the forever loop**.

Figure 81.

5.15.2. **Sound and light effects - GAME: all sounds**.

To perform certain sound and light effects associated with a particular event in In the game (point, loss of life, fall) we use the **GAME: all sounds** function. Mandatory previously run the **GET GAME status** function.

Figure 82.

5.15.3. **Message for the end of the game - GAME: END message**.

In order for the program to end the game with a message, you need to turn on the **GAME: END message** function. It is mandatory to run the **GET GAME status** function beforehand.

Figure 83.

5.15.4. **Points - POINTS at start**

To track the number of points won in the game, it is necessary to run the **POINTS at start** function with the starting number of points. In a game where we can only win plus points usually the starting value is 0. In a game with the possibility of winning and losing points, the initial value is greater than zero. In order for a **player** to get points, it is necessary to play include **Points (+)** objects.

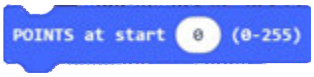


Figure 84.

5.15.5. **Lives - LIVES at start**

To lose a life, in the game, it is necessary to run the **LIVES at start** function that sets the initial value of the number of lives in the game. Loss of life occurs when falling (**FALL**) or touching the **Lives (-)** object.

Figure 85.

5.15.6. **Fall loss of life - FALL**

The game consists of platforms on which the **player** moves. When falling, the player's object can lose a life or just reappear in the starting position. By turning on the function **FALL** includes loss of life when falling off the screen.

Figure 86.

5.15.7. **Limited game duration - GAME DURATION**

To define the duration of the game in seconds, it is necessary to turn on the **GAME DURATION** function. We use the function in games where the goal is to collect as many points as possible equal time limit. In such games, the points-only function is used.

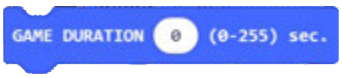


Figure 87.

5.15.8. **Negative points - POINTS negative**

If you want to enable point subtraction, you need to enable the **POINTS negative** function. In order for this function to be active, it is necessary to activate the **LIVES at start** function.

Figure 88.

5.15.9. **Display the “screen” by random selection - RANDOM displays flow**

If the game has **MORE THAN TWO "screens"**, to avoid repeated repetition of the same order we can turn on this feature. The **random** function creates a sequence **screen** display.

Figure 89.

5.16. Complete platform game

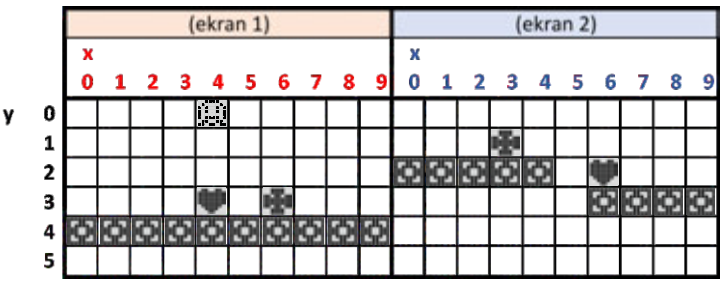
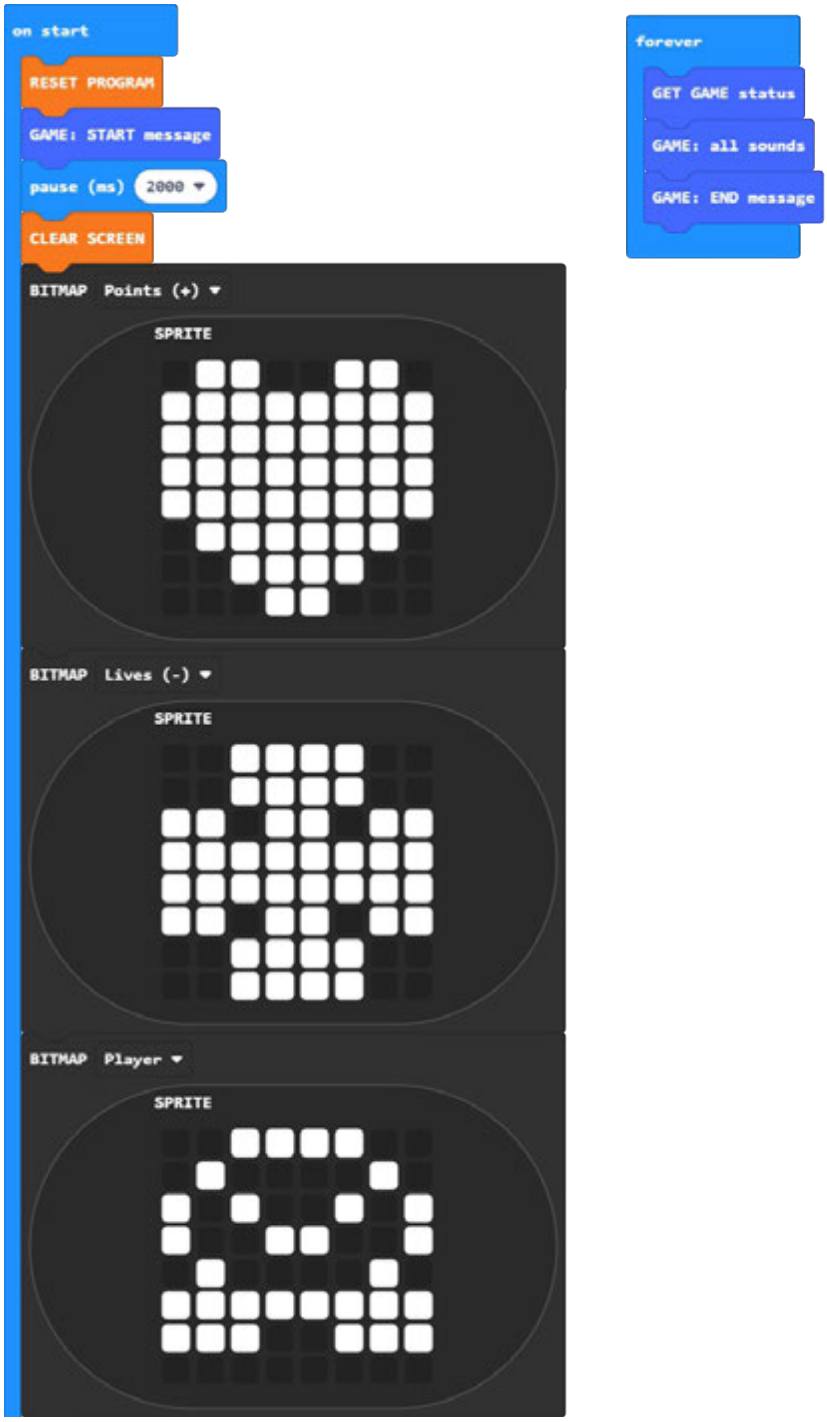


Figure 90.

To make the game complete, we added to each "**screen**" objects for gaining points ( ) and losing lives ( ) according to the positions on the sketch (Figure 90), and the **player** animation object.



(continuation of the program on the next page)

5.16.1. Erasing data from memory - DELETE past objects

During program creation and changing object definitions it can happen, that some objects that you delete from the program remain stored in the interface memory. In that In this case, '**phantom**' objects that you have deleted from the program may appear on the screen. To avoid this, you can add the **DELETE past objects** function at the beginning of the program objects.

This function can be removed from the program after the program is completed.

Figure 92.

5.16.2. Automatic level control (levels) of the game - AUTO LEVELS

To make the game more demanding we can add more weight new ones. The higher the level, the higher the speed of the game and therefore harder to finish. With automatic function control we can determine the values that determine the levels of the game. At the beginning (**speed max.**) enter the value that defines the maximum game speed (**last level**). After the **starting speed** at which we start the game. By how much it increases the speed of the game by moving to a higher level is entered in the field **change for**. Last value (**points for new level**) determines how many points it takes to get to higher level.

This feature may not support all game forms.

Delete the **GAME SPEED** function from the previous program and add **AUTO LEVELS** function with the values from the example in Figure 93.

Figure 93.

5.16.3. Data exchange rate (micro:bit <-> AD display) - SET COM FACTOR

At the beginning of the program, when a lot of data is sent to define different functions and objects need to be set **COM FACTOR** to **8** (initial value) or more. That way, the AD interface program has enough time to process all the data.
If the time is too short (speed too high), the program will not be able to process everything data sent to it by micro: bit and some objects will be missing or some will not work functions, or the program will stop working. Ako želite da se radnje (nakon dijela programa koji šalje postavke za objekte i funkcije) odvijaju brže, te da igra bude što brža, možete **COM FACTOR** postaviti na **4** (najmanja preporučena vrijednost, za najveću brzinu).

A value **less than 4** is not recommended (Python, JavaScript).

The AD interface program allows you to try with my values as well.

Figure 94.

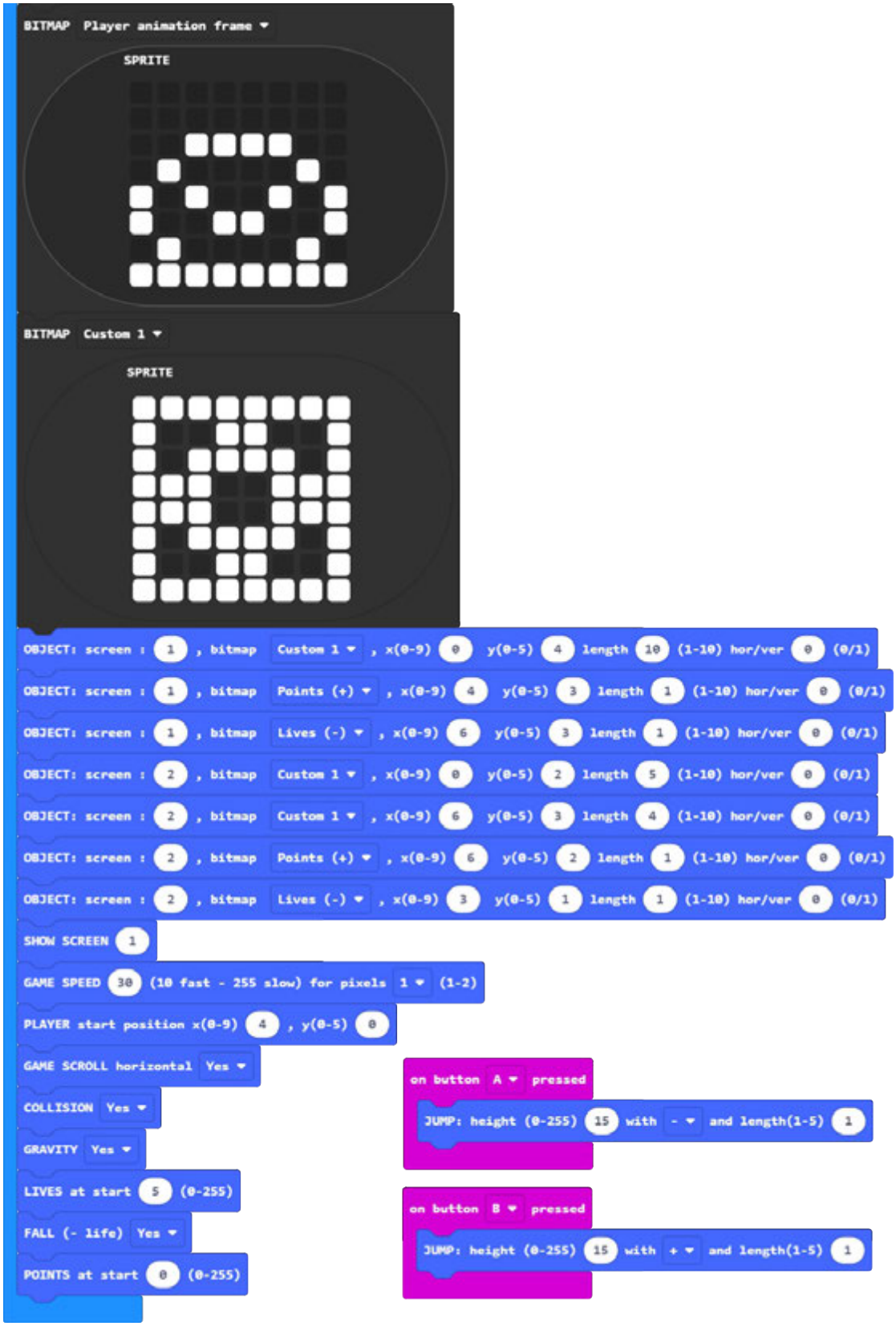

BITMAP Player animation frame
SPRITE
BITMAP Custom 1
SPRITE
OBJECT: screen : 1 , bitmap Custom 1 , x(0-9) 0 y(0-5) 4 length 10 (1-10) hor/ver 0 (0/1)
OBJECT: screen : 1 , bitmap Points (+) , x(0-9) 4 y(0-5) 3 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen : 1 , bitmap Lives (-) , x(0-9) 6 y(0-5) 3 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen : 2 , bitmap Custom 1 , x(0-9) 0 y(0-5) 2 length 5 (1-10) hor/ver 0 (0/1)
OBJECT: screen : 2 , bitmap Custom 1 , x(0-9) 6 y(0-5) 3 length 4 (1-10) hor/ver 0 (0/1)
OBJECT: screen : 2 , bitmap Points (+) , x(0-9) 6 y(0-5) 2 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen : 2 , bitmap Lives (-) , x(0-9) 3 y(0-5) 1 length 1 (1-10) hor/ver 0 (0/1)
SHOW SCREEN 1
GAME SPEED 30 (10 fast - 255 slow) for pixels 1 (1-2)
PLAYER start position x(0-9) 4 , y(0-5) 0
GAME SCROLL horizontal Yes
COLLISION Yes
GRAVITY Yes
LIVES at start 5 (0-255)
FALL (- life) Yes
POINTS at start 0 (0-255)
on button A pressed
JUMP: height (0-255) 15 with - and length(1-5) 1
on button B pressed
JUMP: height (0-255) 15 with + and length(1-5) 1

Figure 91.

6. EXAMPLE OF THE PROGRAM

6.1. METEORS

The program includes some of the functions described on page 28. The game has three "**screens**" that are displayed in random order using the **RANDOM** function (5.15.9) and is limited to **30 seconds** by the function **GAME DURATION** (5.15.7.). A sketch of the layout of the objects is shown in the figure below (Figure 95).

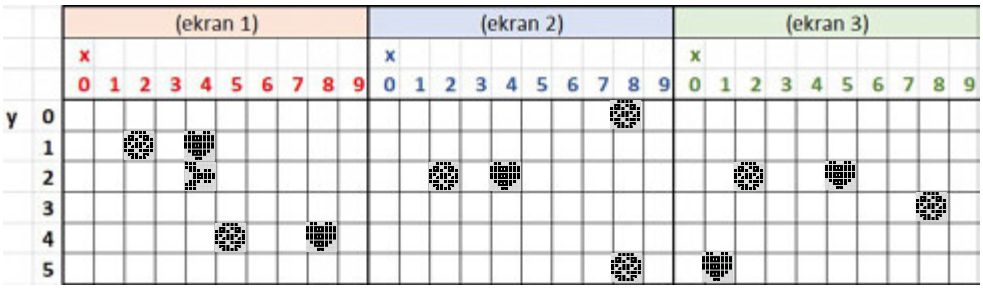


Figure 95.

If you want to make it harder to score points you can add more objects that will only make it harder for the **player** to move, as well which is shown in Figure 96.

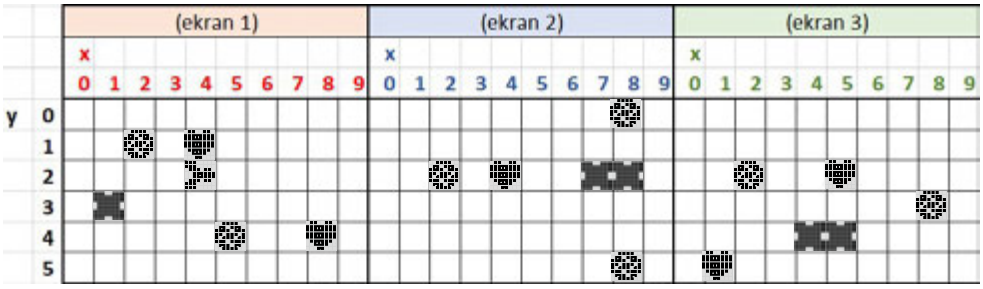
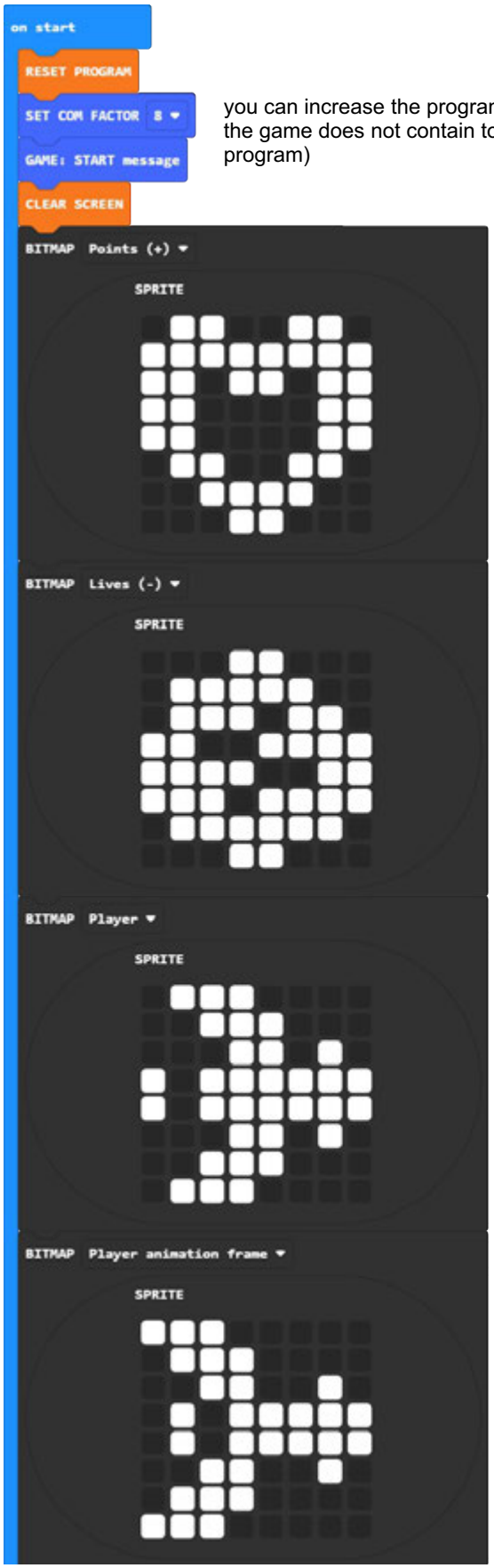


Figure 96.



you can increase the program speed if you set the **COM FACTOR** to **6 or 4**, if the game does not contain too many objects. (try which speed is right for your program)

(continuation of the program on the next page)


OBJECT: screen (1-5): 1 , bitmap Lives (-) , x(0-9) 2 y(0-5) 1 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 1 , bitmap Lives (-) , x(0-9) 5 y(0-5) 4 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 1 , bitmap Points (+) , x(0-9) 4 y(0-5) 1 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 1 , bitmap Points (+) , x(0-9) 8 y(0-5) 4 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 2 , bitmap Lives (-) , x(0-9) 2 y(0-5) 2 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 2 , bitmap Lives (-) , x(0-9) 8 y(0-5) 5 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 2 , bitmap Points (+) , x(0-9) 4 y(0-5) 2 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 2 , bitmap Points (+) , x(0-9) 4 y(0-5) 2 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 3 , bitmap Lives (-) , x(0-9) 2 y(0-5) 2 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 3 , bitmap Lives (-) , x(0-9) 8 y(0-5) 3 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 3 , bitmap Points (+) , x(0-9) 1 y(0-5) 5 length 1 (1-10) hor/ver 0 (0/1)
OBJECT: screen (1-5): 3 , bitmap Points (+) , x(0-9) 5 y(0-5) 2 length 1 (1-10) hor/ver 0 (0/1)
SHOW SCREEN 1
show first «screen»
GAME SPEED 30 (10 fast - 255 slow) for pixels 1 (1-2)
game speed and shift
PLAYER start position x(0-9) 4 , y(0-5) 2
the starting position of the player object
GAME SCROLL horizontal Yes
COLLISION Yes
POINTS at start 0 (0-255)
number of points at the beginning of the game = 0
LIVES at start 5 (0-255)
number of lives at the beginning of the game = 5
GAME DURATION 30 (0-255) sec.
game time limited to 30 seconds
RANDOM displays flow
selecting the “screen” display by randomly selecting the order
forever
GET GAME status
GAME: all sounds
GAME: END message
if button A is pressed then
BUTTON: increment + direction Y for 2 (0 - 255)
player control along the y axis down (+)
if button B is pressed then
BUTTON: increment - direction Y for 2 (0 - 255)
player control along the y axis up (-)

Figure 97.

7. EXAMPLE OF THE PROGRAM

7.1. OTHER FUNCTIONS

7.1.1. **Taking the position of a player - GAME get player position (x)**

Through this function you can retrieve the value for the horizontal (**x**) position of the **player** at screen. The value shows the graphical position of the player (**0-83**).
You can do this function use in any combination when creating a program.

Figure 98.

7.1.2. **Taking the position of a player - GAME get player position (y)**

Through this function you can take the value for the vertical (**y**) position of the **player** on screen. The value shows the graphical position of the player (**0-47**).
You can do this function use in any combination when creating a program.

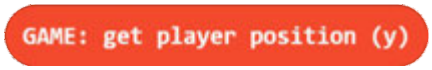
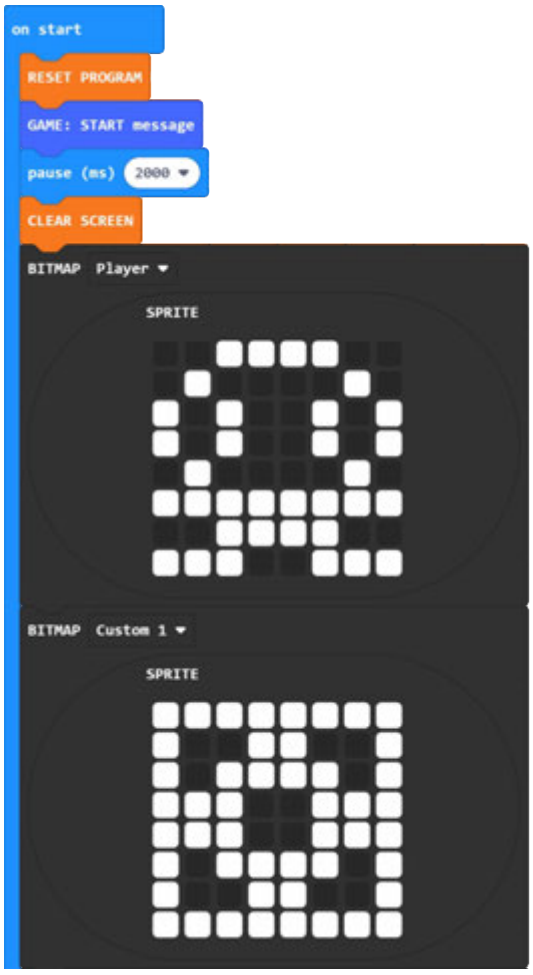


Figure 99.

7.2. Example program



(continuation of the program on the next page)

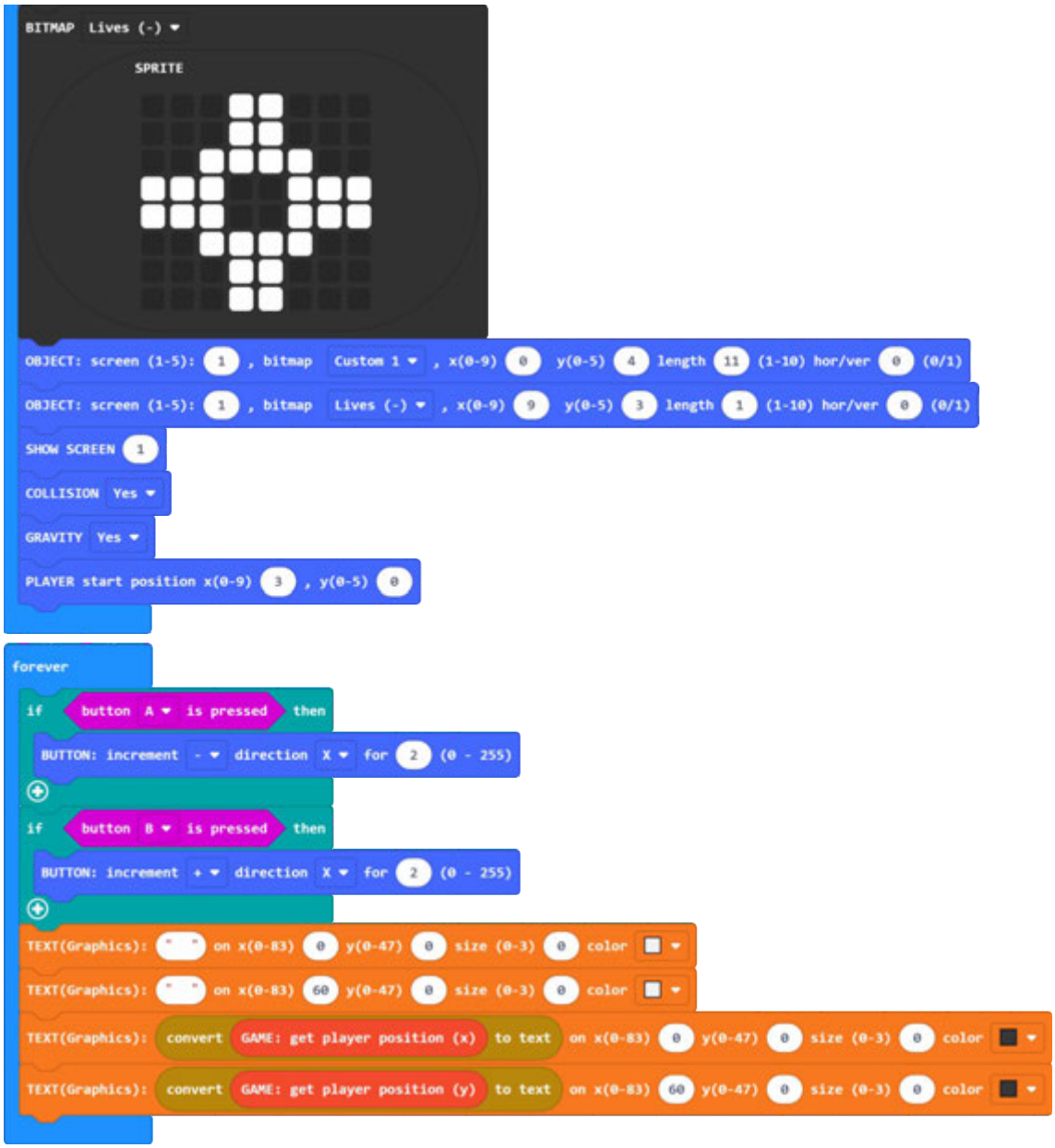


Figure 100.

In this example, we use the **player** position download functions just to display on the screen. Same functions you can use it to control the **player** or restrict his movement on the screen.

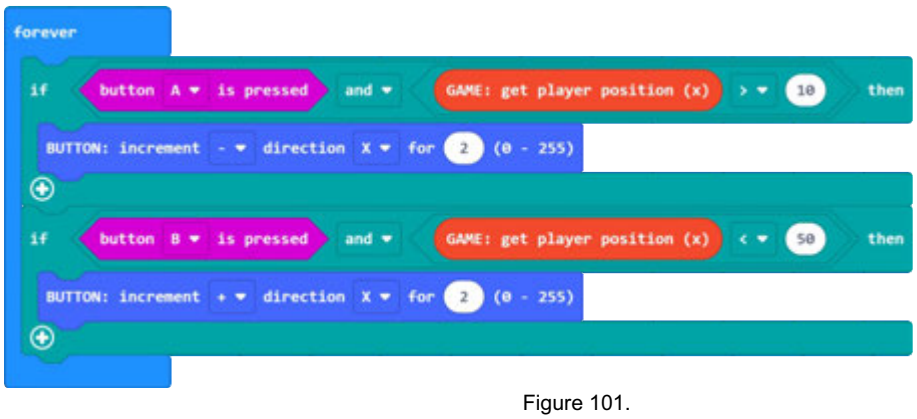


Figure 101.

Example of restricting the horizontal movement of players (Figure 101).
In the previous program (Figure 100), make the change according to the example in Figure 101.

8. CONCLUSION

We wanted to create a screen interface that would allow you to display data or create simple games. When creating a game, some functions are used to define the operation of the game, which we also have in real computer games (gravity). To allow for maximum creativity most functions have no limited value, which means they will happen errors such as program shutdown or printing of incorrect data on the screen.

We used an electronic translator to translate into English, because we wanted to complete these instructions as soon as possible.

We wish you a pleasant work.