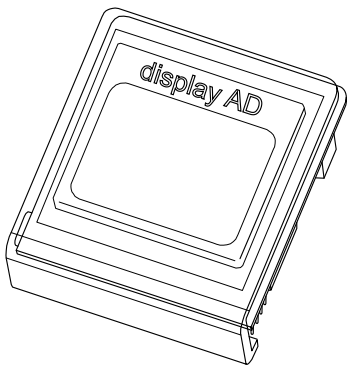
# Arduino - C programming

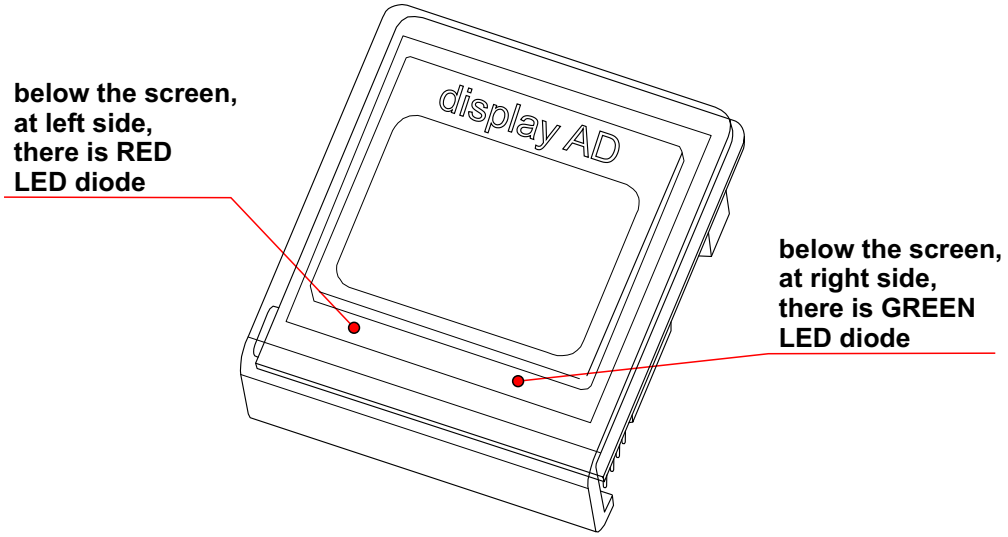
## display AD



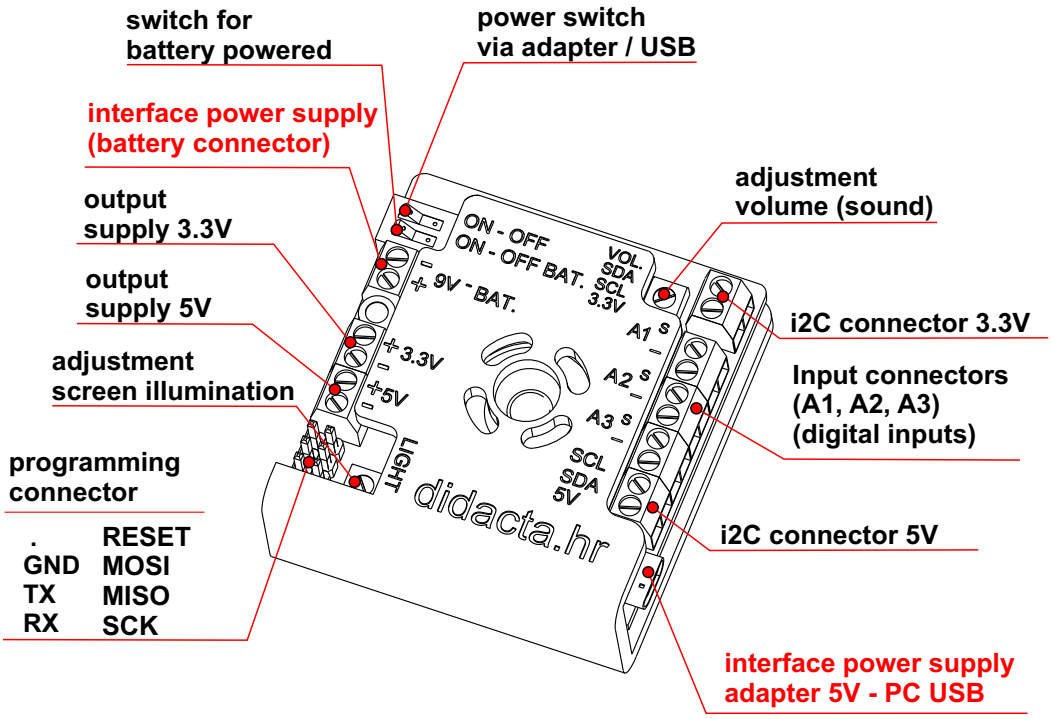

**DIDACTA ADVANCE d.o.o.**
**Čakovec, Croatia (EU)**
**www.didacta.hr**

# 1. BEGINNING

## 1.1. Upper side interface display AD




## 1.2. Power supply of display 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 display AD interface screen.
The interface is ready to work.
**If the program is already loaded in Arduino, you need to press the RESET button, to restart program.**

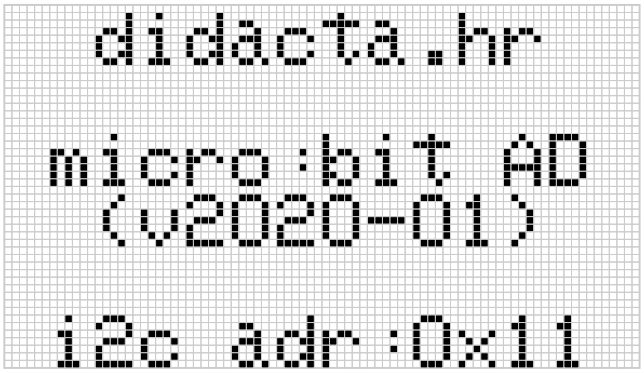



Figure 1.

## 1.4. Interface screen

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

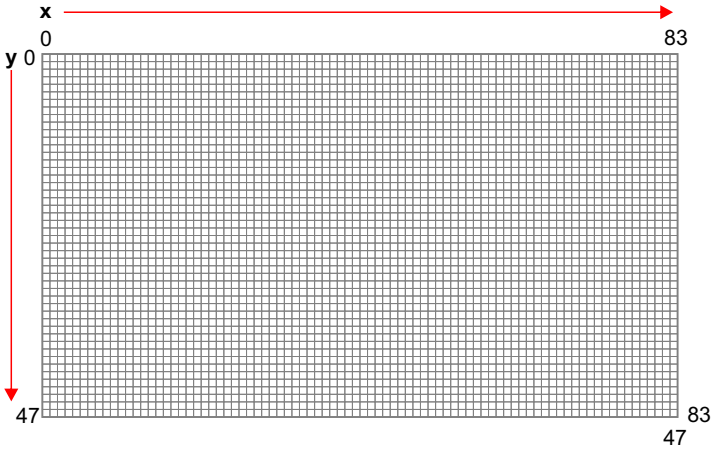



Figure 2. Graphics mode - resolution 48 x 84.

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.

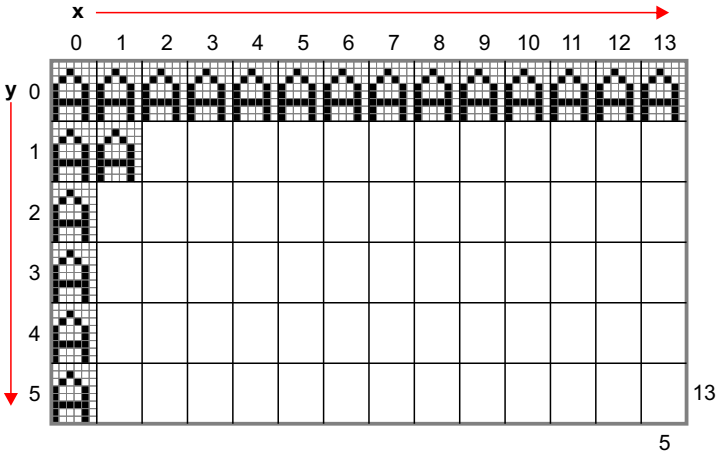



Figure 3. Text mode - 6 x 14 resolution.

# 2. Arduino - C++

## 2.1. Basic Modules

### 2.1.1. Modules for communication between Arduino and AD interface. (DO NOT CHANGE)

These modules define how data is transferred between Arduino and the AD interface, and should be used **without changing**. If you change some values or parts of the module, **communication may occur** between the Arduino and the AD interface stops working. **THIS MODULE IS REQUIRED.**

```
void salji(String poruka)
{
  Serial.println(poruka);
  poruka = poruka + ";";   // extra ; at end
  int duz = poruka.length();
  if (duz > 30)
  {
    String por1 = poruka.substring(0,30)+"+";
    char dio1[31];
    por1.toCharArray(dio1,duz);
    Wire.beginTransmission(0x11);
    Wire.write(dio1);
    Wire.endTransmission();
    String por2 = poruka.substring(30);
    duz = por2.length();
    char dio2[duz];
    por2.toCharArray(dio2,duz);
    Wire.beginTransmission(0x11);
    Wire.write(dio2);
    Wire.endTransmission();
    delay(duz*spd);
  }
  else
  {
     char copy[duz];
     poruka.toCharArray(copy,duz);
     Wire.beginTransmission(0x11);
     Wire.write(copy);
     Wire.endTransmission();
     delay(duz*spd);
   }
}
```

```
//--------------- VARIABLES USED IN MODULE
//-- MUST BE
char pit = "";
int spd = 8;
//---------------
```

### 2.2.2. RESET program module (DO NOT CHANGE)

This module resets (reset) a program running on the AD interface. It is **recommended to use** this module (run - rest()) at the beginning of each program, so as not to mix the old ones data stored in the AD interface memory with the new data. **NOT REQUIRED.**

```
void rest(){
   salji("RST");
   while(true)
   {
      trazi();
      if (pit == 5){
         break;
      }
      delay(20);
   }
}
```

### 2.2.3. Module for reading data from the AD interface (DO NOT CHANGE)

This module requests a value from the AD interface. Values are used to determine events in the game, such as: winning points, losing a life, or falling. If you do not play with sound or light effects this module is **NOT REQUIRED**.

```
void trazi(){
  Wire.requestFrom(0x11,1);
  while (Wire.available()) {
    pit = Wire.read();
  }
}
```

### 2.2.4. **Module for sound and light effects** (**CAN BE CHANGED**)

This module is used in game programs. The values tested in the module are specified and **cannot be changed**. Parts that are **GREEN** can be changed. If you want to use this module, you must include the previous module (trazi() - which reads values) in the program. **NOT REQUIRED**.

```
void sound_light(){
   if (pit == 2){  // POINTS
      ispis("BIP;200;50;");
      ispis("LED;G;50;");
   }
   if (pit == 3) {  // LIFES
      ispis("BIP;800;50;");
      ispis("LED;R;50;");
   }
   if (pit == 4){  // FALL
      ispis("BIP;1200;50;");
      ispis("LED;R;50;");
   }
}
```

### 2.2.5. **Message module at the BEGINNING of the program** (**CAN BE CHANGED**)

This module prints a message at the beginning of the program and can be modified according to your needs.

```
void begin_prog()
{
   ispis("CLS;");
   ispis("START;2;1;2;");
   ispis("G A M E;3;3;");
   delay(2000);
   ispis("CLS;");
}
```

### 2.2.6. **Message module at the END of the program** (**CAN BE CHANGED**)

This module requests a value from the AD interface. Values are used to display the number of points (Score) at the end of the program. If points are won in the program, only part of the green module can be changed. The rest of the module reads the number of points and displays it on the screen. You can change or supplement the print position variables, name, and end message as desired.

```
void end_prog(){
   delay(300); // must be if you read points (trazi())
   trazi(); //  read POINTS number (SCORE)
   delay(300);
   ispis("CLS;");
   ispis("E N D;2;1;2;");
   ispis("G A M E;3;3;");
   ispis("Score:"+String(int(pit))+";3;4;");
   while (true){
      delay(5000);
   }
}
```

# 3. PROGRAMMING - BASIC FUNCTIONS

## 3.1. The first program - RST function

We use the function at the beginning of the program, and after the basic modules. Running this function deletes the values in all fields used by the AD interface program. As part of the **rest()** module - 2.2.2 Page 4.

**ispis("RST");**

## 3.2. The first program - text printing "HELLO" - TEXT function

In the first program we use the **TEXT** function. The default value for size (s) is **1**, so the function can be called without entering a value.

**ispis("TEXT;x;y;s;c");** **ili** **ispis("TEXT;x;y;c");**

TEXT = the text to be printed
x = 0-13 text mod
y = 0-5 text mod
s = size 1 - 3 (1)
c = color (B-black, W-white)

At the beginning of each program, it is a good idea to use the **rest()** module, which deletes the values used by the micro:bit AD program in the previous operation of the micro:bit program.

## 3.3. Clear the screen - CLS

CLS function to clear the contents of the screen.

**ispis("CLS");**

## 3.4. Print text in graphic mode (G)

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 the text directly on the screen but in the auxiliary memory (**BUFFER**), so after one or more **TEXT (Graphics)** functions must be executed by the **BUF** function that displays the record from the auxiliar screen memory. The default values are for **s = 1** and **c = B**, so they do not always need to be specified.

**ispis("TEXT;x;y;s;c;G");** **ili** **ispis("TEXT;x;y;G");**

TEXT = the text to be printed
x = 0-83 graphics mode
y = 0-47 graphics mode
s = size 1 - 3 (**1**)
c = color (B-black, W-white) (**B**)
**G** = graphics mode (84 x 48)

## 3.5. Print auxiliary memory - BUFFER

This function is used to display the contents of the auxiliary memory in which the graphic results are written function, such as printing text in graphic mode. This function displays the status of the full screen with all views previously written to the auxiliary memory. Using this function avoids vibration display for changes by deleting, and reprinting the new state.

**ispis("BUF");**

```
ispis("HELLO!;4;1;B;G");
ispis("BUF");
```

Example for printing a single line of text in graphics mode

```
ispis("HELLO!;10;10;G");
ispis("HELLO!;11;17;G");
ispis("HELLO!;12;24;1;B;G");
ispis("HELLO!;13;31;1;B;G");
ispis("BUF");
```

Example for printing multiple lines of text in graphics mode

## 3.6. Drawing a line - LIN

In this program, we use the **LIN** function to draw a line on the screen in graphics mode. Line code we need to specify the start (x1, y1) and end (x2, y2) point on the screen. We can draw the line in black or white colors. 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!

**ispis("LIN;x1;y1;x2;y2;c");**

LIN = function name
x1 = 0-83 graphics mode
y1 = 0-47 graphics mode
x2 = 0-83 graphics mode
y2 = 0-47 graphics mode
c = color (B-black, W-white)

## 3.7. Draw a circle or filled circle - CIR

The circle or circle is not the correct shape for the screen resolution.

In these examples, we use the **CIR** 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. If we want to delete an already drawn black circle, we need to draw a circle at the same position white. We use the **fill color** to draw a circle.

**ispis("CIR;x;y;r;c;f");**

CIR = function name
x = 0-83 graphic mode
y = 0-47 graphic mode
r = radius of the circle (pix)
c = Color (B-black, W-white)
f = Fill color (B, W)

## 3.8. Drawing a rectangle or filled rectangle - REC

In these examples, we use the **REC** function to draw a rectangle in graphical mode. Code rectangle 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. 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 with color.

**ispis("REC;x;y;w;h;c;f");**

REC = function name
x = 0-83 graphic mode
y = 0-47 graphic mode
w = width (pix)
h = height (pix)
c = Color (B-black, W-white)
f = Fill color (B, W)

## 3.9. Fill the screen - FIL

The **FIL** function fills the screen with bytes of the value entered in the color field.
The screen is filled with bytes that are laid vertically as in Figure 4.

**ispis("FIL;n");**

FIL = function name
n = 0 - 255

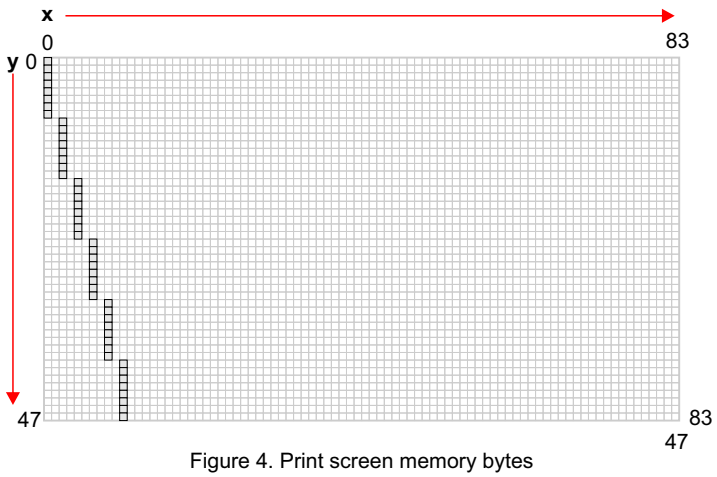



Figure 4. Print screen memory bytes

Example of filling (coloring) the screen with lines spaced one pixel apart. Color value calculation (bytes) you can see in Figure 5.


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

Figure 5. Calculate 'color' (bytes)

## 3.10. Display in black / white or white / black - DIS

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 **DIS** function.

**ispis("DIS;n");**

DIS = function name
n = 0 - black / white 1 - white / black

## 3.11. Print pixels - PIX

**ispis("PIX;x;y;c");**

PIX = function name
x = 0-83 graphic mode
y = 0-47 graphic mode
c = Color (B-black, W-white)

Function for printing pixels on the screen in graphic mode.

# 4. PROGRAMMING - SHIFT FUNCTIONS - SCROLL

## 4.1. Move text UP by one line - SCU

**ispis("SCU;r");**

SCU = function name
r = rotation (R = yes, "" = no)

## 4.2. Move text DOWN by one line - SCD

**ispis("SCD;r");**

SCD = function name
r = rotation (R = yes, "" = no)

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

**ispis("SBU;n");**

SBU = function name
n = number of pixels

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

**ispis("SBD;n");**

SBD = function name
n = number of pixels

## 4.5. Horizontal screen shift (images) by one pixel - SCC

**ispis("SCC;s;a;b;r");**

SCC = function name
s = direction of displacement (R, L)
a = from text line (0-5)
b = to text line (0-5)
r = rotation (R)

# 5. PROGRAMMING - GAME FUNCTIONS

## 5.1. LED light control - LED

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.

**ispis("LED;c;t");**

LED = function name
c = color (R - red, G - green)
t = time in milliseconds

## 5.2. Creating BITMAPE - objects - BIT

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 buffer, and finally the memory is displayed via the BUF function on the screen.

After creating the object (BITMAP), in this example 1, it is necessary to determine the position at which the object will be draw and what COLOR (SPR).

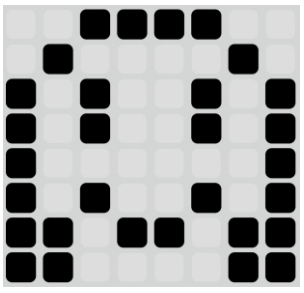
You can create a bitmap according to Excel. An Excel example with a template is shown in Figure 6. When transferring from an Excel file, you need to add a botmap number (BLACK - 1).

**YOU CAN USE ALL BITMAPES AS NEEDED IF YOU DO NOT USE THE FUNCTIONS WITH WHICH THEY ARE LINKED.**

**ispis("BIT;n;b;b;b;b;b;b;b;b");**

BIT = function name
n = bitmap number (1,2,3,4,5,8,9)
b = byte

RELATED BITMAPES: **2** - POINT + **3** - LIFE - **4** - DOOR **8** - ANIMA **9** - PLAYER
FREE BITMAPES: **1** and **5**



| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|---|---|
| 128 | | | 1 | 1 | 1 | 1 | | | 60 |
| 64 | | 1 | | | | | 1 | | 66 |
| 32 | 1 | | 1 | | | 1 | | 1 | 165 |
| 16 | 1 | | 1 | | | 1 | | 1 | 165 |
| 8 | 1 | | | | | | | 1 | 129 |
| 4 | 1 | | 1 | | | 1 | | 1 | 165 |
| 2 | 1 | 1 | | 1 | 1 | | 1 | 1 | 219 |
| 1 | 1 | 1 | | | | | 1 | 1 | 195 |
| | | | | | | | | | |
| | 60;66;165;165;129;165;219;195 | | | | | | | | |

Frame 6.

without a bitmap number

ispis("**BIT;1;60;66;165;165;129;165;219;195**");

ispis("SPR;1;40;20;B"); //-- print a bitmap in memory
ispis("BUF"); // -- print memory to the screen

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

## 5.5. Displacement controls - horizontal and vertical - BUT

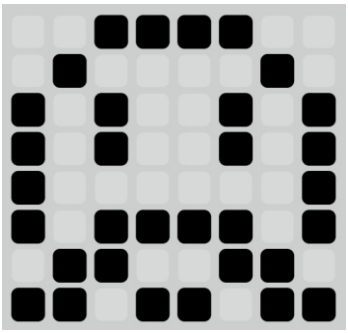
To control the player's facility in all directions, it is necessary to add entry controls on the Arduino.

**ispis("BUT;s;p;n");**

BUT = function name
s = direction + or -
p = X = horizontal Y = vertical
n = number of offset pixels

## 5.6. Player object (bitmap 9)

If we want BITMAPA to be a player object, select **9 (Player)** in the menu. 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 = 2.



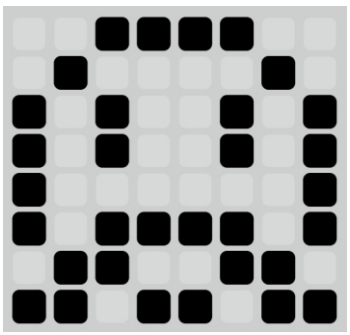
(9) Player

```
ispis("BIT;9;60;66;165;165;129;189;102;219"); // player
ispis("POZ;5;2");
```

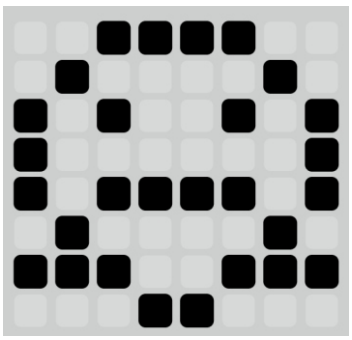
The PLAYER function also displays the BUFFER status on the screen, so it is not necessary calling the **BUF** function.

## 5.7. Animation of a player object (bitmap 8)

Another bitmap is required to animate the player object. By moving the **player bitmaps (9)** and the player **animation bitmaps (8)** are alternately drawn on the screen of the player object.



(9) Player



(8) Animation player bitmap

```
ispis("BIT;9;60;66;165;165;129;189;102;219"); // player
ispis("BIT;8;60;66;165;129;189;66;231;24"); // player anima map
```

## 5.8. Animation speed control - ANI

Animation of a player's object can be faster or slower. To control the speed, we use the **ANI** function (animation speed). If we want the animation (bitmap change) to be slower, we need to enter a higher value.

**ispis("ANI;n");** ANI = function name
n = speed (smaller number higher speed)

```
ispis("BIT;9;60;66;165;165;129;189;102;219"); // player
ispis("BIT;8;60;66;165;129;189;66;231;24"); // player anima map
ispis("POZ;5;2");
ispis("ANI;1");   //  smaller number higher speed
```

## 5.8. START game (2.1.5 - str. 5)

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. Insert the function for displaying the standard message **begin_prog()** at the beginning of the program, and after the communication module. After printing, the function must be started sleep, so that the message can be read, and then clear the screen with the **CLS** function. If we don't start screen clearing function, the player object will be drawn over the start text.

This module prints a message at the beginning of the program and can be modified according to your needs.

```
void begin_prog()
{
   ispis("CLS;");
   ispis("START;2;1;2;");
   ispis("G A M E;3;3;");
   delay(2000);
   ispis("CLS;");
}
```

## 5.9. COLLISION function - KOL

Fill the program with a new object (**1**). Draw it on the screen (**SPR**) several times, on different positions. Complete the program with the **KOL** function. Test the difference with the function on **KOL (1)** and off (**0**).

**What happens to objects when moving a player's object towards them in both case?**

**ispis("KOL;s");** KOL = function name
s = 0 - no, 1 - yes

## 5.10. Gravity - GRV

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 - GRV.

**ispis("GRV;s");** GRV = function name
s = 0 - no, 1 - yes

## 5.11. Creating objects (horizontal and vertical) - OBJ - 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 one bitmap, if they are longer, only the first position is active to obtain points or loss of life.**

**ispis("OBJ;e;b;x;y;n;s");**

OBJ = function name
e = screen number (1 - 5)
b = bitmap
x = horizontal position (0-10)
y = vertical position (0-5)
n = length (number of repetitions)
s = direction ("" or 0 = hor., 1 = ver.)

```
ispis("BIT;1;255;153;189;231;231;189;153;255");
ispis("BIT;9;60;66;165;165;129;189;102;219"); // player
ispis("BIT;8;60;66;165;129;189;66;231;24"); //player anima map
ispis("OBJ;1;1;0;3;6");
ispis("OBJ;1;1;8;0;5;1");
ispis("FX;1");    // -- display ”screen” 1
ispis("POZ;5;2");
ispis("ANI;1");
ispis("KOL;1");
ispis("GRV;1");
```

It needs to be defined first objects to be drawn on the screen via the **OBJ** command. All commands are grouped into "**screens**" that are via the **FX** command displayed on the screen. In this example, we define a “**screen**” 1 with two objects. Both objects are composed of the same bitmap (1). For positioning it is necessary to determine the x and y position of the initial object bitmaps. Length (number of repetitions) is determined by entering a value in the **length** field. Maximum the horizontal length is 11 (84/8 = 10.5 bitmaps).

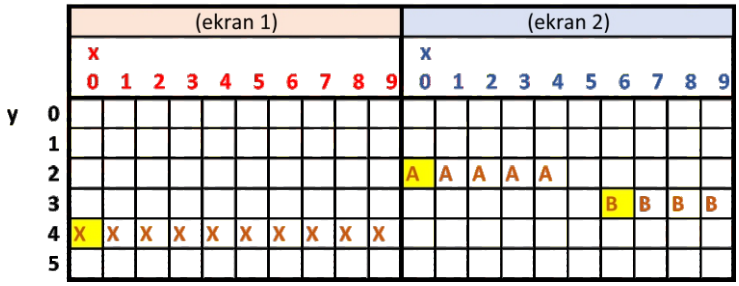
To vertically read a bitmap, it is necessary to change the value of the "hor / ver" field to 1.

```
ispis("OBJ;1;1;0;0;10");
ispis("OBJ;1;1;0;5;10");
ispis("OBJ;1;1;0;1;4;1");
ispis("OBJ;1;1;9;1;4;1");
```

**Don't forget the FX (1) command, which must come from defined objects.**

## 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.




Frame 7.

When you want to create multiple "**screens**" it is good to make a sketch as shown in the example in Figure 7. We can sketch the screens via a table in Excel or raster paper. This makes it much easier to visualize all screens, especially if horizontal and vertical objects are used.

```
ispis("BIT;1;255;153;189;231;231;189;153;255");
ispis("BIT;9;60;66;165;165;129;189;102;219"); // player
ispis("BIT;8;60;66;165;129;189;66;231;24"); // player anima map
ispis("OBJ;1;1;0;4;10");
ispis("OBJ;2;1;0;2;5");
ispis("OBJ;2;1;6;3;4;1");
```

GAME_display_AD_04.ino

## 5.13. Current "screens" - display a screen with a horizontal scroll - ASD

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 have created we need to add the activated **ASD** command (**1**). Objects on "screens" are printed on the screen in a circular order of "**screen**" numbers(1,2,1,2,1,2,...)

**ispis("ASD;s");**

ASD = function name
s = ("" or 0) = no, 1 = yes

The horizontal movement speed of objects can be changed with the **SPD** command. Lower value variables mean a higher shift rate. The maximum speed is limited to **10**, and the 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.**

**ispis("SPD;s;p");**

SPD = function name
s = speed max.recommended speed = 10
(higher number = lower speed)
p = number of pixels

## 5.14. Launch the player object at the offset screen - jump - JMP

In order for the game to function normally, we must replace the classic **player** movement controls with new ones. The jump of the 'player' directly upwards is controlled by the **JMP** command, 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. If we want, we determine the direction of the jump by entering the + or - variable and the jump angle. To function was active we had to add some more mandatory functions for the position and control of the **player's** object movement (COLLISION, GRAVITY, PLAYER start position).

**ispis("JMP;h;s;a");**

JMP = function name
x = jump height - pixels
s = direction (-, +)
a = jump angle (45%) 1 - 5

## 5.15. Control functions

5.15.1. **Game status - GAME status - trazi() ( 2.1.3 - page 4 )**

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** before testing the values obtained, or when querying for an X or Y position player.

```
void trazi(){
  Wire.requestFrom(0x11,1);
  while (Wire.available()) {
    pit = Wire.read();
  }
}
```

5.15.2. **Sound and light effects - sound_light() ( 2.1.4 - page 5 )**

To perform certain sound and light effects associated with a particular event in in the game (point, loss of life, fall) we use the **sound_light()** function. Mandatory previously run the function **trazi()**.

```
void sound_light(){
    if (pit == 2){  // POINTS
        ispis("BIP;200;50;");
        ispis("LED;G;50;");
    }
    if (pit == 3) {  // LIFES
        ispis("BIP;800;50;");
        ispis("LED;R;50;");
    }
    if (pit == 4){  // FALL
        ispis("BIP;1200;50;");
        ispis("LED;R;50;");
    }
}
```

5.15.3. **Message for the end of the game - end_prog() ( 2.1.5 - page 5 )**

In order for the program to end the game with a message, it is necessary to turn on the **end_prog()** function for the final one message. It is mandatory to run the function beforehand **trazi().**

```
void end_prog(){
    delay(300); // mora biti ako koristite funkciju trazi()
    trazi(); //  čita broj osvojenih BODOVA (SCORE)
    delay(300);
    ispis("CLS;");
    ispis("E N D;2;1;2;");
    ispis("G A M E;3;3;");
    ispis("Score:"+String(int(pit))+";3;4;");
    while (true){
        delay(5000);
    }
}
```

5.15.4. **Points - BOD**

To track the number of points won in the game, it is necessary to start the **BOD** function with initial 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.

**ispis("BOD;n");**

BOD = function name
n = number of points at the beginning of the game (0)

5.15.5. **Lives - LIV**

To lose a life, in the game, it is necessary to run the **LIV** function, which sets the initial one the value of the number of lives in the game. Loss of life occurs during a fall (**PAD**) or touches the **Lives (-)** object.

**ispis("LIV;n");**

LIV = function name
n = number of lives at the beginning of the game

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

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 **PAD** includes loss of life when falling off the screen.

**ispis("PAD;b");**

LIV = function name
b = 0 - no, 1 - yes

5.15.7. **Limited game duration - TIM**

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

**ispis("TIM;n");**

TIM = function name
n = time in seconds

5.15.8. **Negative points - BON**

If you want to turn on the deduction of points, you need to turn on the **BON** function. In order for this function to be active, the **LIV** function must also be activated. This feature takes away points by losing a life.

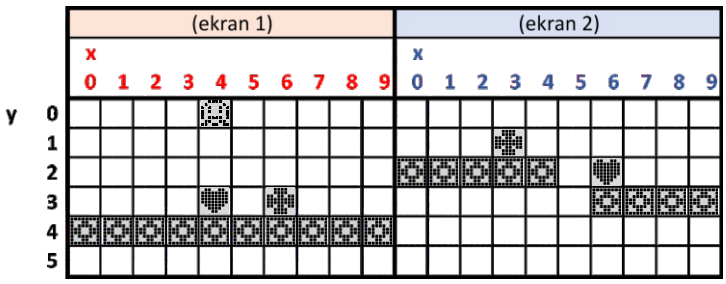
**ispis("BON");**

5.15.9. **The order in which the "screen" is displayed by random selection - RND**

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.

**ispis("RND");**

## 5.16. Complete platform game




Frame 8.

To make the game complete, we added to each "**screen**" objects for gaining points ( ) and losing lives ( ) according to the positions in the sketch (Figure 8), and the object for **player** animation.

```
//------- - 15.04.2021
// PLATFORM GAME LIKE
// PYTHON PROG40.PY
//-------
#include <Wire.h>

int tip1 = A0; // jump switch 1
int tip2 = A1; // jump switch 2
//--------------- VARIABLES - MUST BE
char pit = "";
int spd = 8;
//---------------

void setup() {
  pinMode(tip1, INPUT);
  pinMode(tip2, INPUT);
  //---------- START I2C COMM. - MUST BE
  Wire.setClock(100000);
  Wire.begin();  // join i2c bus
  //----------
 begin_def();
}

void loop() {
  trazi();
  if (pit == 9)
  {
    end_prog();
  }
  if (pit > 1 && pit < 5)
  {
    sound_light();
  }
  if (digitalRead(tip1) == LOW)
  {
    ispis("JMP;15;+;2");
  }
  if (digitalRead(tip2) == LOW)
  {
    ispis("JMP;15;-;2");
  }
    delay(100);
}
```

```
//---------------------------------------------- SETTINGS
void begin_def() // all settings you need in program
{
    delay(2000);
    ispis("RST");
    delay(200);
    ispis("CLS");
    delay(500);
    ispis("BIT;1;255;153;189;231;231;189;153;255");
    ispis("BIT;2;102;255;255;255;255;126;60;24");
    ispis("BIT;3;60;60;219;255;255;219;60;60");
    ispis("BIT;8;60;66;165;153;66;255;231;0"); // p.a.
    ispis("BIT;9;0;0;60;66;165;153;66;255"); // player
    ispis("OBJ;1;1;0;4;10");
    ispis("OBJ;1;2;4;3;1");
    ispis("OBJ;1;3;6;3;1");
    ispis("OBJ;2;1;0;2;5");
    ispis("OBJ;2;1;6;3;4");
    ispis("OBJ;2;2;6;2;1");
    ispis("OBJ;2;3;3;1;1");

    begin_prog();

    ispis("CLS");
    ispis("FX;1");
    ispis("POZ;4;0");
    ispis("ANI;3");
    ispis("KOL;1");
    ispis("GRV;1");
    ispis("ASD;1");
    ispis("SPD;10;1");
    ispis("PAD;1");
    ispis("BOD;0");
    ispis("LIV;5");
}
```

(continuation of the program on the next page)

```
//---------------------------------------- ALL MODULES
void ispis(String poruka)
{
  Serial.println(poruka);
  poruka = poruka + ";";
  int duz = poruka.length();
  if (duz > 30)
  {
    String por1 = poruka.substring(0,30)+"+";
    char dio1[31];
    por1.toCharArray(dio1,duz);
    Wire.beginTransmission(0x11);
    Wire.write(dio1);
    Wire.endTransmission();
    String por2 = poruka.substring(30);
    duz = por2.length();
    char dio2[duz];
    por2.toCharArray(dio2,duz);
    Wire.beginTransmission(0x11);
    Wire.write(dio2);
    Wire.endTransmission();
    delay(duz*spd);
  }
  else
  {
    char copy[duz];
    poruka.toCharArray(copy,duz);
    Wire.beginTransmission(0x11);
    Wire.write(copy);
    Wire.endTransmission();
    delay(duz*spd);
  }
}

void trazi(){
  Wire.requestFrom(0x11,1);
  while (Wire.available()) {
    pit = Wire.read();
  }
}

void begin_prog()
{
    ispis("CLS");
    ispis("START;2;1;2");
    ispis("G A M E;3;3");
    delay(2000);
}

void end_prog(){
    delay(300);
    trazi();
    delay(300);
    ispis("CLS");
    ispis("E N D;2;1;2");
    ispis("G A M E;3;3");
    ispis("Score:"+String(int(pit))+";3;4");
    while (true){
        delay(5000);
    }
}
void rest(){
    ispis("RST");
    while(true)
    {
        trazi();
        if (pit == 5){
            break;
        }
        delay(20);
    }
}

void sound_light(){
    if (pit == 2){  // POINTS
        ispis("BIP;200;50");
        ispis("LED;G;50");
    }
    if (pit == 3) {  // LIFES
        ispis("BIP;800;50");
        ispis("LED;R;50");
    }
    if (pit == 4){  // FALL
        ispis("BIP;1200;50");
        ispis("LED;R;50");
    }
}
```

5.16.1. **Erasing data from auxiliary memory - DEL**

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 at the beginning of the program you can turn off and on the AD or interface each time simply add the **DEL** function at the beginning of the program.

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

**ispis("DEL");**

5.16.2. **Automatic level control (levels) of the game - LVL**

To make the game more demanding we can add more weight novelties. 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 we type a value that defines the maximum speed of the game (last level). After that the initial speed with which we begin the game. By how much the speed of the game increases by moving to a higher level. The last value determines how many points it takes to win to advance to a higher one level.

This feature may not support all game forms.

**ispis("LVL;s;f;d;b");**

LVL = function name
s = maximum speed (last level)
f = initial speed (first level)
d = increase in speed by the value of d
b = required number of points for a new level

5.16.3. **Data exchange rate (Arduino - AD display) - spd (value)**

At the beginning of the program, when a lot of data is sent to define different functions and objects need to **set the spd 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 Arduino and some objects will be missing or some will not work functions, or the program will stop working.

If you want to take action (after the part of the program that sends the settings for objects and functions) take place faster, and to make the game as fast as possible, you can set the **spd to 4** (lowest recommended value, for highest speed).
**A value less than 4 is not recommended.**

The AD interface program allows you to try even with smaller values.

**spd = 8;**

# 6. EXAMPLE OF THE PROGRAM

## 6.1. METEORS

The game has three "**screens**" that are displayed randomly using the **RND** function (5.15.9) and is limited to **30 seconds** by the **TIM** function (5.15.7.). A sketch of the layout of the objects is shown in the figure below (Figure 9).

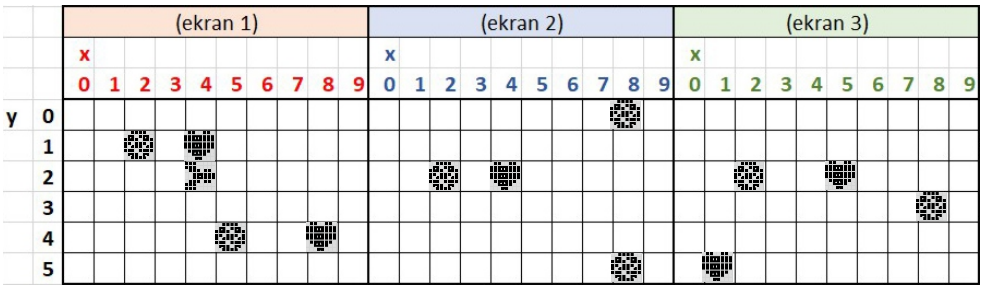



Figure 9.

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 10.

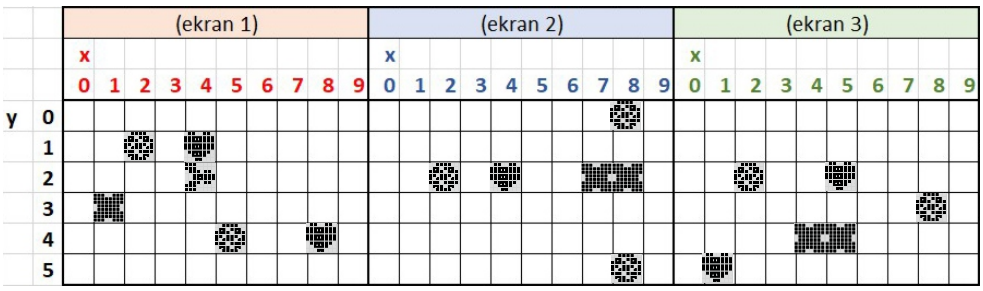
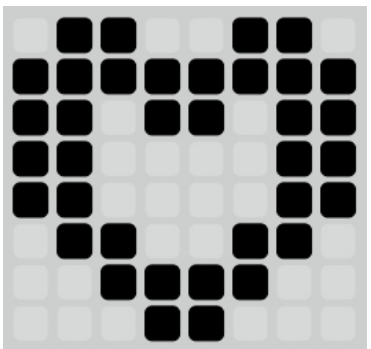
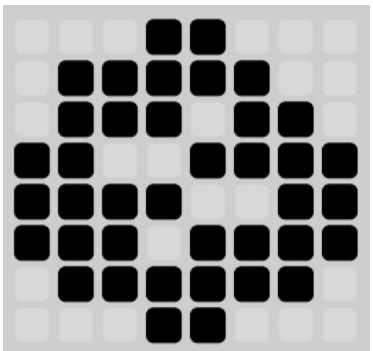


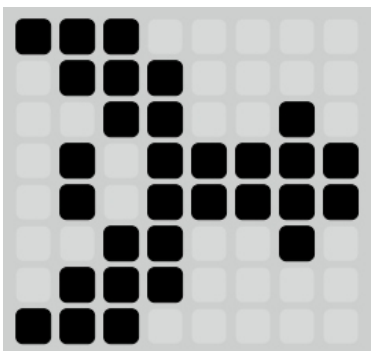

Figure 10.



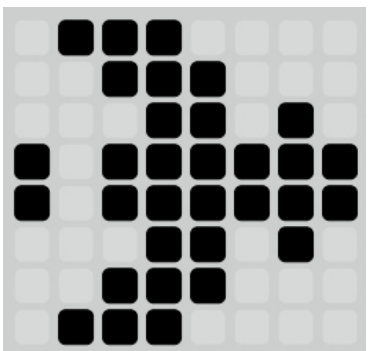
bitmap 2



bitmap 3



bitmap 9



bitmap 8

```
//------- - 15.04.2021
// DEMO METEORS GAME SAME LIKE PYTHON PROG42.PY
//-------
#include <Wire.h>

int tip1 = A0; // jump switch 1
int tip2 = A1; // jump switch 2
//--------------- VARIABLES - MUST BE
char pit = "";
int spd = 8;
//---------------

void setup() {
  pinMode(tip1, INPUT);
  pinMode(tip2, INPUT);
  //----------- START I2C COMM. - MUST BE
  Wire.setClock(100000);
  Wire.begin();  // join i2c bus
  //-----------
 begin_def();
}

void loop() {
  trazi();
  if (pit == 9)
  {
    end_prog();
  }
  if (pit > 1 && pit < 5)
  {
    sound_light();
  }
  if (digitalRead(tip1) == LOW)
  {
    ispis("BUT;+;Y;2");
  }
  if (digitalRead(tip2) == LOW)
  {
    ispis("BUT;-;Y;2");
  }
    delay(50);  // you can change this delay
}
```

```
//------------------------------------------  SETTINGS
void begin_def() // all settings you need in program
{
    delay(2000);
    ispis("RST");
    delay(200);
    ispis("CLS");
    delay(500);
    ispis("BIT;2;102;255;219;195;195;102;60;24");
    ispis("BIT;3;24;124;118;207;243;239;126;24");
    ispis("BIT;8;224;112;50;95;95;50;112;224"); // anim
    ispis("BIT;9;112;56;26;191;191;26;56;112"); // play.

    ispis("OBJ;1;3;2;1;1"); //-- screen 1
    ispis("OBJ;1;3;5;4;1");
    ispis("OBJ;1;2;4;1;1");
    ispis("OBJ;1;2;8;4;1");

    ispis("OBJ;2;3;2;2;1"); //-- screen 2
    ispis("OBJ;2;3;8;5;1");
    ispis("OBJ;2;2;4;2;1");

    ispis("OBJ;3;3;2;2;1"); //-- screen 3
    ispis("OBJ;3;3;8;3;1");
    ispis("OBJ;3;2;1;5;1");
    ispis("OBJ;3;2;5;2;1");

    begin_prog();
    ispis("CLS");

    ispis("FX;1");
    ispis("POZ;4;2");
    ispis("ANI;3"); // -- anima. speed
    ispis("KOL;1");
    // --------->                ispis("GRV;1") - not use
    ispis("ASD;1");
    ispis("SPD;10;1"); // -- brzina pomaka
    // --------->                ispis("PAD;1") - not use
    ispis("BOD;0");
    ispis("LIV;5");
    ispis("TIM;30"); //   -- game time 30 seconds
    ispis("RND"); //     -- random order
    spd = 4;

}
//------------------------------------------- ALL MODULES
// same as in the previous program
```

# 7. EXAMPLE OF THE PROGRAM

## 7.1. OTHER FUNCTIONS

7.1.1. **Take a player position or read digital inputs - GET (X, Y, 1, 2, 3)**

With this function you can take the value for horizontal (**X**) or vertical (**Y**) the position of the **player** on the screen. Value shows the graphical position of the player for x (0-83) or y (0-47) value. You can use this function in any combination when creating program. We also use this function to read the values of digital inputs.

**ispis("GET;n");**

GET = function name
n = variable (X, Y, A1 = 1, A2 = 2, A3 = 3)
A1-A3 digital inputs - value 0 - 1

## 7.2. Example program - input reading A1 - A3

```
// ------  - 15.04.2021
// DIGITAL READING INPUTS - A1,A2,A3
// ------
#include <Wire.h>

// variables for this sample
int I1 = 0;
int I2 = 0;
int I3 = 0;

int I1p = 0;
int I2p = 0;
int I3p = 0;
//--------------- VARIABLES - MUST BE
char pit = "";
int spd = 8;
//---------------

void setup() {
  //---------- START I2C COMM. - MUST BE
  Wire.setClock(100000);
  Wire.begin();  // join i2c bus
  //----------
  begin_def(); // start screen
}

void loop() {
    ispis("GET;1");
    trazi();
    I1 = String(int(pit)).toInt();
    if (I1 != I1p) {
        ispis("  ;5;2");
        ispis(String(I1)+";5;2");
        sound();
        I1p = I1;}
    ispis("GET;2");
    trazi();
    I2 = String(int(pit)).toInt();
    if (I2 != I2p) {
        ispis("  ;5;3");
        ispis(String(I2)+";5;3");
        sound();
        I2p = I2;}
    ispis("GET;3");
    trazi();
    I3 = String(int(pit)).toInt();
    if (I3 != I3p) {
        ispis("  ;5;4");
        ispis(String(I3)+";5;4");
        sound();
        I3p = I3;}
}

//-------------- for this sample
void sound(){
        ispis("BIP;800;50");
      //  salji("LED;G;50"); // LIGHT
}

//----------------------------------------  SETTINGS
void begin_def() // all settings you need in program
{
    ispis("DEL");
    delay(200);
    ispis("RST");
    delay(200);
    ispis("CLS");
    delay(500);
    ispis("INPUTS:;0;0");
    ispis("I1 =;0;2");
    ispis("I2 =;0;3");
    ispis("I3 =;0;4");
}
//---------------------------------------------- ALL MODULES
// same as in the previous program
```

GAME_display_AD_03.ino

```
ispis("   ;0;0;G");              //-- delete values
ispis("   ;65;0;G");
ispis("GET;X");                  //-- query for X value
trazi();                         //-- read X value
ispis(String(pit)+";0;0;G");   //-- print the X value to the screen
ispis("GET;Y");                  //-- query for Y value
trazi();                         //-- read Y value
ispis(String(pit)+";65;0;G"); //-- print the Y value to the screen
```

An example of reading and displaying the x and y positions of a player.

## 8. CONCLUSION

We wanted to create a screen interface that would allow data to be displayed or created simple games. When creating a game, some functions are used to define the operation of the game, a which we also have in real computer games (gravity). **To enable 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 wish you a pleasant work.**