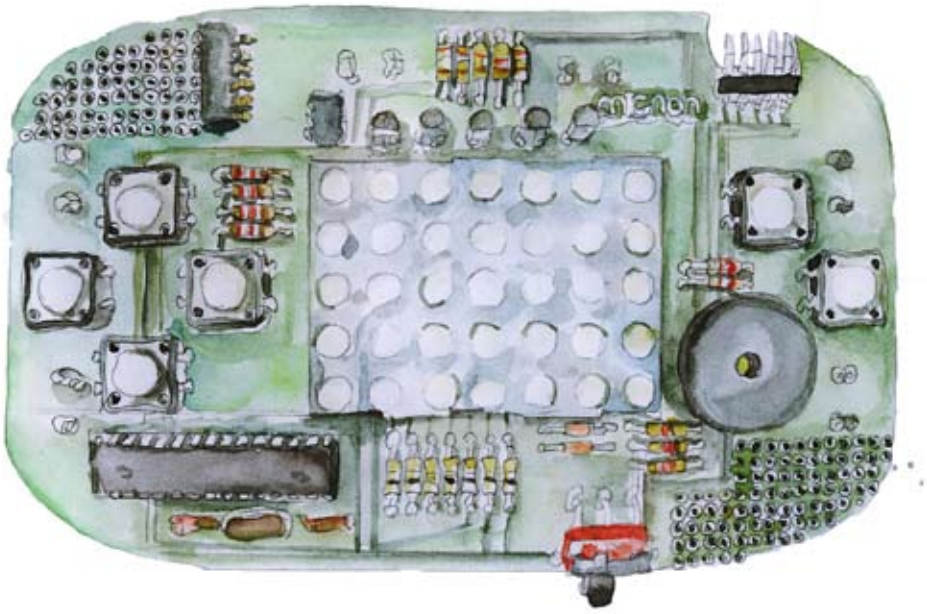
ALPHA VERSION

MIGNON

Mignon Game Kit

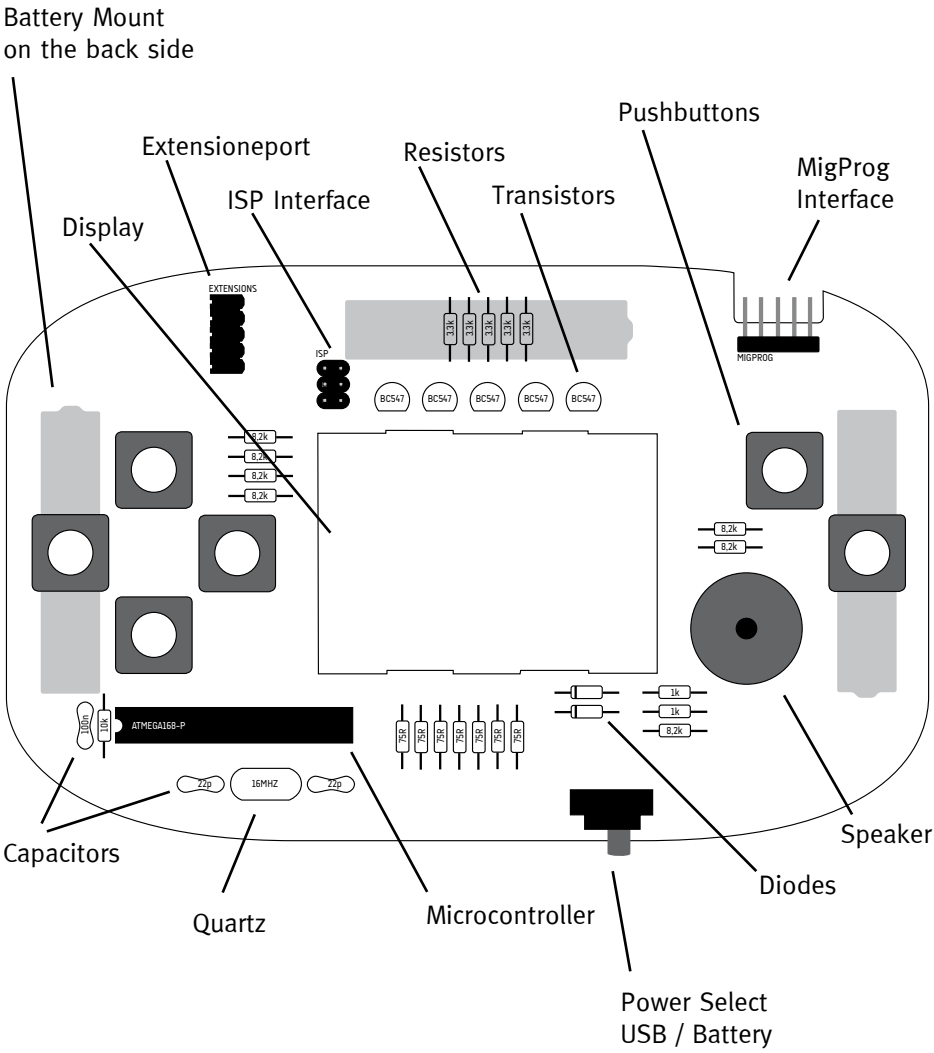
Hands on Digital Media



Version 0.2

Olaf Val

Bason Books


Battery Mount
on the back side
Extensioneport
ISP Interface
Display
Resistors
Transistors
Pushbuttons
MigProg
Interface
EXTENSIONS
ISP
MIGPROG
33k
BC547
8,2k
ATMEGA168-P
75R
1k
10n
18k
22p
16MHZ
Capacitors
Quartz
Microcontroller
Power Select
USB / Battery
Diodes
Speaker

Short Instructions

How to assemble the game kit, write your own code and publish you results?

Assembly

• Put all components – apart from the battery clamps into the circuit board and solder them to it. The battery clamps are fitted on the back.

• The construction guidelines help you to put all parts in the correct positions and the instructions for soldering show how to solder without creating bridges or dry solders

Programming

• Install the free programming tool Arduino.
http://arduino.cc

• Add the MIGNON GAME KIT Library. You find the download „gamekit.zip“ under files at www.mignongamekit.com

• Install the USB Treiber for MIGPROG. http://www.ftdichip.com/Drivers/VCP.htm.

• Now you are able to programme your own game! Help is available through the instructions for programming, the examples, and the overview of functions in the appendix.

Publishing

• You can publish your experiences and results on the MIGNON GAME KIT homepage. www.mignongamekit.com.

• Find a title for the programme you have written. If you describe it in a brief paragraph and design a logo, it can become a popular download.

Table of Contents

Construction Guidelines

Put all components – apart from the battery clamps onto the front of the circuit board. The front is the side with the assembly print. The wire legs of some parts, such as the diodes, transistors, and resistors need to be bend to fit through the quencher. Polarity is irrelevant in case of the sensors, resistors, capacitors, the speaker, and the quartz.

All other components, i.e. transistors, diodes, switch and microcontroller only function when soldered on in the right direction. Amazingly the display is constructed symmetrically thus can be fitted either way. The battery clamps are fitted to the back. And soldered from the front.

It is advisable to check the position of the parts once again before soldering, as soldering is an arduous process. Correct resistors? Correct polarity of all components? Attention! The micro-controller is not soldered directly to the circuit board but fitted with a base.

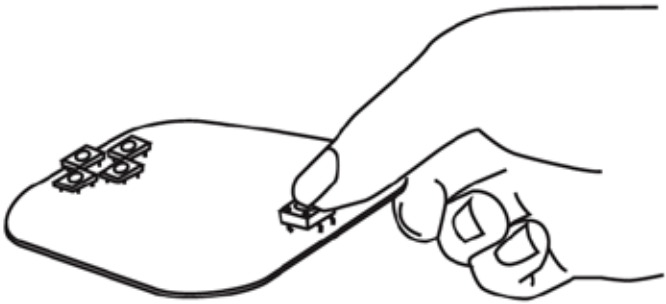
Neither the “extension port“ nor the “ISP interface“ form part of the standard equipment.

Step by step Instructions for Construction

This detailed version of the instructions also delivers an introduction to the foundations of electricity.

From amber to electricity

[Coming soon...] Text text text



Pushbuttons

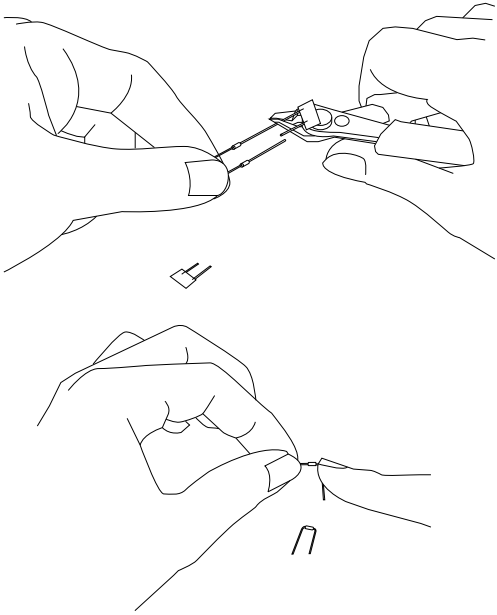
The six pushbuttons are easily fitted and soldered. Thus they are suited to beginners. It is important to cut off the two little pins on the back of the buttons. After that they can be pushed flat on to the board. When soldering on the back fully close the holes as the button are subject to strong mechanical pressure. (refer to the instructions for soldering page XX)

Everyone knows the function of a pushbutton. It turns the power on and off. It is easy to differentiate between the switch and the pushbutton. The switch stays in the on or off position while the pushbutton switches back and forth. But when we ask for a pushbutton in a store for electrical goods we will probably be confronted with the question: “An opener or a closer?” At this point it becomes apparent that electricity is a complex field. An opener interrupts the power circuit, when used; a closer lets the current pass when used. For the Game Kit we use a closer – as is most often the case.

Diodes

With the diodes it is most important that the small line is located on the left as on the assembly print. When the diodes are fitted the wrong way round no power reaches the circuit board. The diodes are divided by the cardboard strips, after that the small wires are bend so that they fit easily through the quencher on the board. In case you use a bending machine, use the second lowest setting.

Our senses cannot perceive the functioning of diodes. Thus we have to resort to models and analogies as always when we cannot understand something. In this case the most popular analogy is the water model. Imagine power is like water

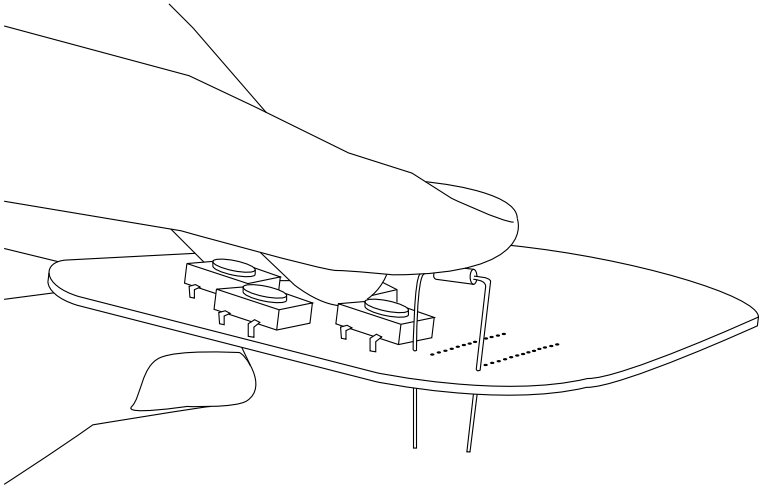


running from plus- the tap towards minus- the drainpipe. The diode is a valve causing the water to flow in one direction only. Thus the diodes prevent that something breaks, for example, in case we put the batteries in the wrong way. Light emitting diodes serve the same purpose and glow additionally.

Resistors

In principle the resistors are fitted in the same way as the diodes. The polarity is not important but the various values of the resistors such as 75 Ohm and 8,2 k must not be mixed up. You can compare them to the photograph (see image XX). The safest way is to measure the values is with a voltmeter.

To this end we set the measuring device to Ohm value, as „Ω“ Ohm are the units resistors are measured in. The best start is “10 k” i.e. a measuring sensitivity of 10,000 Ohm. You then place a resistor at the two test prods (image XXX) The display now shows how many kilo Ohm resistance the resistor has. If the value is less than zero the measuring sensitivity needs to be lowered. To do so the switch is turned anticlockwise by one level to “100”. Now the pure Ohm values, without the „k“ (Kilo) are displayed. The measured values can easily be assigned despite measuring discrepancies. The discrepancies are caused on the one hand during the process of measuring as factors such as body resistance, humidity influence the outcome. On the other hand, resistors



are inexact within a relatively large range of tolerance.

As the name suggests, resistors hinder or lower the circuit of power. They are used to protect parts of the construction such as the LEDs. As the microcontroller works at 5 V the LEDs, that are designed for 3,7 V would fuse without the resistors.

Quartz

As with the quartz watch the quartz of the Game Kit is responsible for the exact pulsing of the microcontroller at a frequency of 16 megahertz. The rhythm in which the processor processes data usually lies at about 8 MHZ according to the internal pulse generator. Through the quartz we connect an external pulse generator that causes the processor to work twice as fast and it runs so accurately that we could build a watch from the game kit.

Capacitors

On both sides of the quartz 22p capacitors are connected to ensure an exact swinging. The biggest of the capacitors is fitted behind the 10 k resistor at the top end of the microcontroller.

In all circuits interfering effects are possible that may lead to mistakes. For

example, if the Game Kit is connected to the power via the computer, then the electric current flows not always evenly but is subject to constant and quick fluctuations. This so called “noise” is reduced through capacitors. Like small batteries, capacitors take on voltage peaks, store the power, and release it when the voltage decreases for a moment.

Transistors

The transistors are fitted above the display with the flattened side turned to the bottom.

With a transistor it is possible to switch one power-signal on and off with another one. Thus the transistor is able to represent binary conditions – i.e. zero and one as on and off. This makes it the basic module of a computer. The first electronic computers worked with electro-mechanical relays and tubes, these are however slower, more prone to interferences, and much larger than transistors.

In the context of the Mignon Game Kit we use transistors to combine a relatively weak power signal with a somewhat stronger current. The five transistors operate the ground wires for the five rows of the display. As seven LEDs are connected in one row the microcontroller would overcharge, if it was connected directly to the pin of a controller.

Switch

The switch is fitted below the diodes. It switches back and forth between electrical power supply via USB and via battery. Before connecting the game kit to USB the batteries should be removed, otherwise it is not possible to switch it off.

Speaker

The speaker is an inexpensive Piezo element that is capsuled in a plastic case. The speaker is soldered beneath the two control keys. Here, too, polarity is irrelevant.

MigProg Interface

The interface of the Mignon programming adapter “MigProg” consists of a simple angled
multi-pin connector of five pins. The short pins are soldered, while the long pins protrude through the notches in the board.

The next two lines serve the purpose of communication between the computer and the game kit. In the process the data is transferred with the TX line (yellow LED) to the game kit and the game kit answers to the computer via the RX line (green LED).

The transfer of data works via a so-called “serial” interface (Com Port). As this connection is dated nowadays we work with the USB port and a driver that emulates the old serial interface (virtual comport). Serial communication has many applications. In connection with the extension port the game kit becomes an interface between computer and the outside world that may be used for a variety of experiments involving sensors, motor control units, and much more. However, the most important application of the communication with the computer is the programming of the game kits.

As the game kit needs to restart for programming the MigProg possesses a reset line on the fifth pin.

For programming a small programme called “bootloader” is located on the microcontroller. Immediately after being switched on- in a brief break before the game kit starts- the bootloader checks, if there is a new programme coming from the computer. If there is data being send from the computer the bootloader erases the memory of the game kit and automatically installs the new software that starts up shortly after.

Instead of the MigProg a USB to serial, or TTL Adapter can be used (also see: Programming Hardware page XX).

Display

Against all expectations the display is constructed symmetrically. The orientation of the
display does not matter when it is placed in the middle of the circuit board in landscape format.

It consists of a circuit board cast in plastic material. On the board LEDs are placed in a grid and they are wired in such a way that the five rows are each connected to the minus poles. The seven columns are each connected to the diodes at the plus poles. Through this grid it is possible to reach each of the 35 points –similar to the game “Battleships”- even though there are only 5 + 7 = 12 circuit points.

Battery Clamps

The Battery holding device is the only part of the construction set that is put on the circuit board from below and that is thus soldered from above. Small cardboard discs are placed between circuit board and holding device in order to prevent short-circuiting with ground wire. During the soldering the big sheet metal parts function as cooling elements therefore it takes a while until the battery holding device reaches a hot enough temperature to merge with the solder.

Microcontroller and Base

The IC-base, i.e. the base for the microcontroller is marked with a half- moon notch. Even though it is not relevant on a technical level the marking should be placed as indicated on the assembly print in order to assure the microcontroller is fitted the right way round. For soldering the base needs to be fixated as it easily slips from the quenchers when turned down slightly. Please use a fire resistant item to prevent burns.

If need be the IC-base could be omitted, but once the microcontroller is fixed to the board it cannot easily be exchanged. As the microcontroller may be damaged through mistakes in the soft- or hardware the base constitutes an extra security. Missing parts can be ordered using the Internet.

The microcontroller needs to be pressed into the base straight, from above with a bit of pressure- its best to use both thumbs. You have to pay attention that none of the 28 small legs buckles. It is best to wait to the end, till after a test as described in the chapter “Initial Start-Up” [page XX], to fit the microcontroller to the base.

ISP Interface

In order to programme the game kit via the MigProg interface you need a microcontroller already equipped with a Bootloader programme. The enclosed controller is fitted with such a Bootloader. When you buy a new controller or loose the Bootloader programme through a mistake it can be re-installed via the ISP interface. To do so you need an AVR-ISP-programmer (see: Programming Hardware, page XX). As this interface is only of interest to advanced users it does not belong to the standard equipment of the construction set. It can be ordered via the Internet.

Extension Port

Through the extension port further devices such as light-emitting diodes, motors, relays, or sensors can be connected to the game kit (see: Extension Port page XX). As this interface is only interesting for further experiments, this socket board, too, does not belong to the standard equipment of all construction sets. It can be ordered through the Internet.

Initial Start-Up

Before fitting the microcontroller you should check that the power supply has not short-circuited through a soldering mistake. To do so the batteries are inserted from below. Small symbols show the direction they need to be placed in. You then check with the tip of your fingers for about half a minute if they become warm. If the batteries stay cold everything is alright. In the case of a short-circuit a lot of power flows that may destroy the microcontroller and the batteries warm up. If you have a voltmeter you can check on pin 7 (plus) and 8 (minus) in the bottom row from the left, if the expected 4,5 Volt occur. That is a clear sign that everything is alright.

Please turn of the game kit after this test. Now the microcontroller can be fitted. After that turn the game kit on again. Now a test programme should start. If the frame around the display glows without gaps the display is soldered without mistakes. The six points in the middle of the display signify the six buttons if they turn of by applying pressure to the respective button the sensors are fully functional, too.

Finally a beeping noise should be audible to confirm that the speaker is also working. Afterwards the test programme turns itself off and a game begins. Which game depends on which version of the construction set it is.

In case the test programme discloses a mistake, you find detailed help in the chapter “debugging” on page XX.

Guide to Getting Started with Programming

• Install the Arduino software. You find the download on the Arduino homepage under „downloads“ (on the left top). Unzip the folder and move it to your folder “programmes”. Open the folder and move the logo to the dock.

• Download the Game Kit Library. The file is called “gamekit.zip“ and is linked to the homepage through “files”. Through unzipping the file you get a folder that you move to the Arduino folder: hardware/libraries/gamekit, so the Arduino software can locate it. In the process the Game Kit Examples are installed automatically.

• Look for the suitable Driver MIGPROG (FTDI Virtual Serial Port) for your computer at: http://www.ftdichip.com/Drivers/VCP.htm and install it. For Windows it is advisable to use the „setup executable“ on the right hand side.

• Connect MIGPROG with the computer via USB. The Power LED of the MIGPROG shows that the device is powered. Remove the batteries from the Game Kit and connect the MIGPROG. When you turn the switch to the left to USB the Game Kit runs with the computers power.

• Start the Arduino software and open one of the examples. You find that in the file menu “Sketchbook“. Allocate the Serial Port to the the software. After correct installation of the FTDI driver the “usbserial” adapter should come first in the “Tools/Serial Port” list”. Select it so a checkmark appears (Under Windows you can check in the “Device Manager” which com-port number has been assigned to the USB-to serial driver.)

• The command „Upload to I/O Board“ in the file menu (control + U) installs the example on the game kit. In the process the memory is overwritten and thus the previous file deleted. The LEDs of the MIGProg and the speaker of the game kit convey the programming process. The system should not be interrupted during programming, as the microcontroller may turn into a state where it is not programmable anymore otherwise.

• You find more games and programmes on the homepage. Download the files, unzip them and move them to the Arduino- Sketchbook- Folder. (Windows: Personal Files/ Arduino) (Mac:Documents/Arduino) (Linux: ~/sketchbook/)

• Now you are ready to begin writing your own programmes. The overview of functions and the examples as well as the step-by-step programming instructions help you. (See pages XXX, XXX, and XXX)

Step by Step Instructions for Programming

These Instructions combine an introduction to programming with an outline of the historic development of the first computer games.

All printed programming codes can be found on the Mignon Game Kit homepage (www.mignongamekit.com) under “How-To” for download. Beginners are advised to hand- copy small programmes to thus gain access to the working process of programming.

2006 Ralph Baer (left) receives the National Medal of Technology from president George W. Bush (right)

Background History of Computer and Games

Apart from realising military applications games were also developed on the very first computers. Computers did not yet feature screens and keyboards and

the developed games had very easy structures such as the “Nim Game” or the XOX Game” on the EDSAC (1949 UK). Twelve years later, after the invention of the transistor and microchips, more complex games such as “Space War” emerged. But these could only be played in the few laboratories in which computers were available. The situation changed when the television ingeneer Ralph Baer had the idea to sell inexpensive Video Games to private households in 1966.

At the time televisions generally had an inbuilt test image generator. While working on such a machine Bear observed how facinating the playful manipulation of the image through rotary switches was. His founding idea was to built a device that anyone could connect to their television. This device would draw simple synthetic forms on the display through analogue circuits- i.e. without a computer. From this games developed.

Exercise 1: “Creation of a synthetic audio signal”

In essence our first homemade programme consists of the basic structure setup and loop. Programming means writing commands that the computer can recognise underneath oneanother. When starting the programme the commands are executed from top to bottom. Firstly, there is an area for the presetting that is only exectuted once per start-up. After that a (generally endless) loop is processed. Both areas are usually framed by curly brackets.

• “File/New” is the start of a new Arduio file (Skatch)

• A commentary is the term for something that is supposed to happen. To mark such lines as commentary they begin with two forward slashes “// ”

• The connection for the speaker is to be defined as the output.

• At the exit the power is constantly turned on and off.

• We built the “Setup” and “Loop” structure:

```
void setup(){
// set sound port as output
}

void loop(){
// turn the power on and off
}
```

• Please note the differentiation between the round and curly brackets.

• Now we still need the appropriate commands to implement what is described in the commentaries.

• pinMode (number of connection, OUTPUT or INPUT);

• digitalWrite (number of connection, HIGH or LOW)

```
void setup(){
  // set sound port as output
  pinMode(9, OUTPUT);
}

void loop(){

  // turn the power on and off
  digitalWrite(9, HIGH);
  delay(1);
  digitalWrite(9, LOW);
  delay(1);
}
```

• Commands always end with a semikolon “;”

• Spacing and word wrap do not effect the functioning, but are important for the readability of the codes. The indents in particular keep interlockings lucid.

The first video game consisted of a dot placed on the screen. By use of a riffle endowed with a photocell the dot on the television could be “shot”.

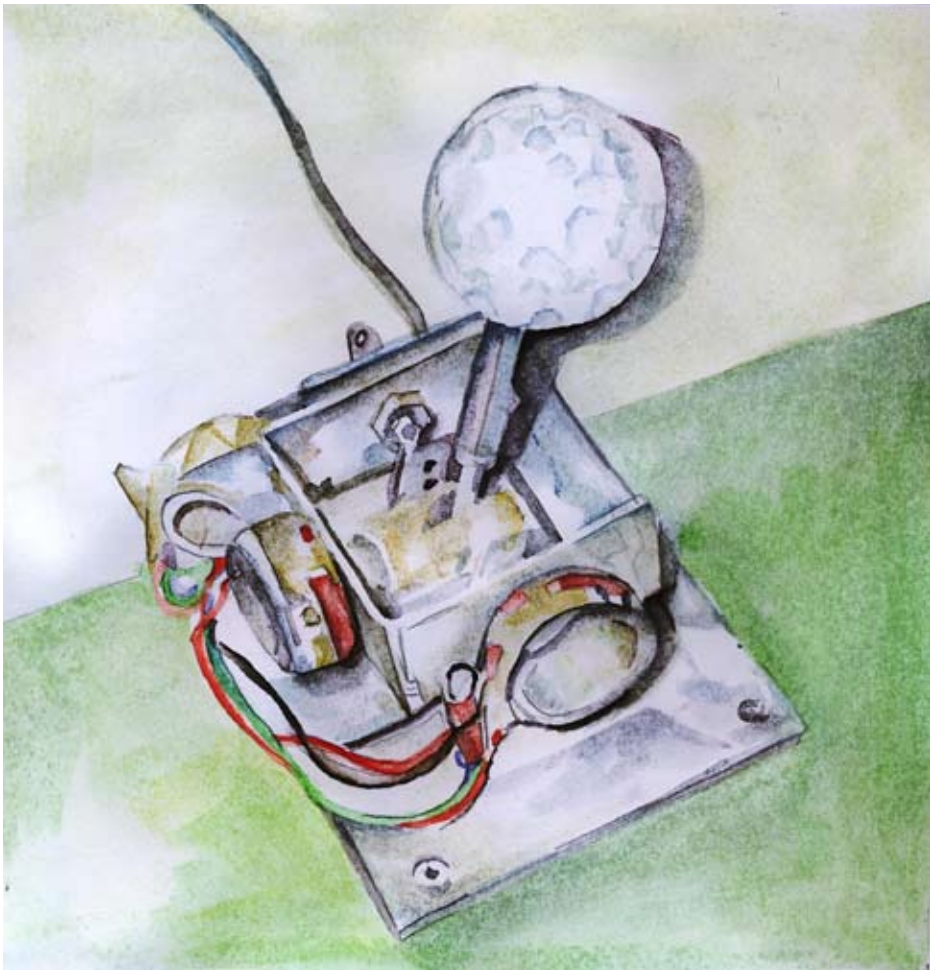
Excercise 2: “A Dot”

We put a dot on the display of the game kit. To do so we use the command „gamekit.set_pixel(row, collumn, value)“. As this command comes from the Game Kit Library, we need to first integrate and activate it. This happens with the following two programming rows: “#include ‹gamekit.h›” and “gamekit. Begin();”

```
//One Dot

#include <gamekit.h>

void setup(){
  gamekit.Begin();
}

void loop(){
  gamekit.set_pixel(2,3,15);
}
```

As we are missing the rifle sensitive to light we have to cease modelling the first video game at this point. But if one carries on putting points on the display it is possible to create signs e.g. Smileys with this little programme.

Ralph Baer continued his experiment “video game”. The next step involved moving the dot with two rotary switches and he produced first sketches of a Joystick for controlling the dots.



Excersise 3: “Move A Dot”

To move the dot the keys are queried with the following function:

```
if(gamekit.button_pressed(Konstante)){

}
```

When a key is operated the commands in between the curly brackets are executed.

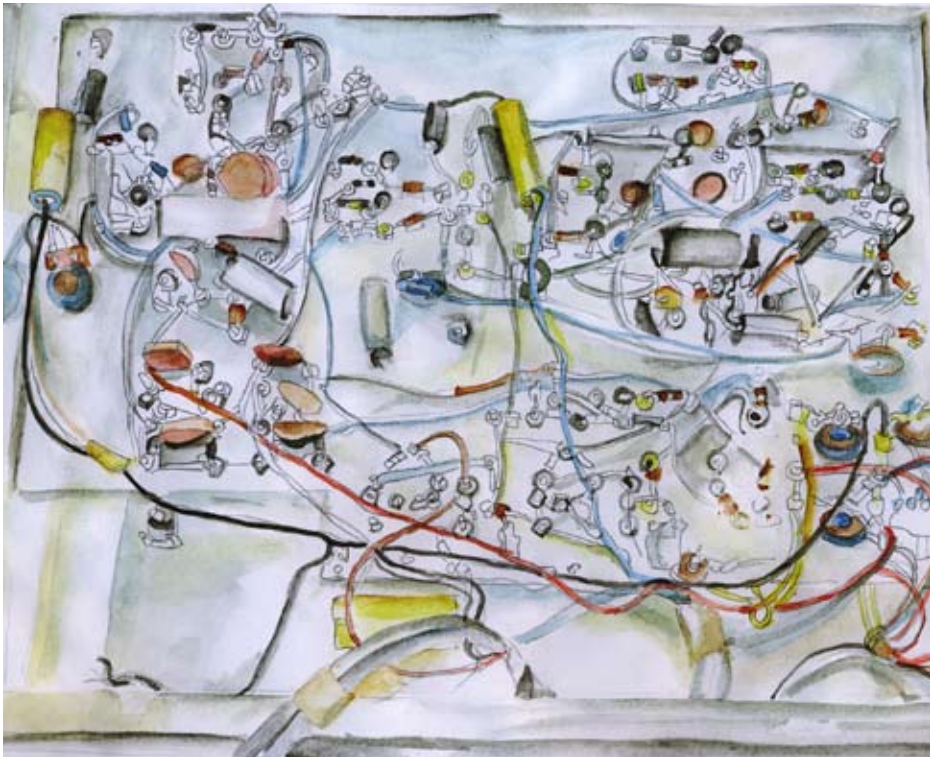
In order to make the dot moveable a variable is added to the „gamekit.set_ pixel()“ function for the position of the dot. These variables need to be defined with the command int (Integer) before setup. The value of these variables are lowered or raised by one with the orders “++” and “- -“. Thus the dot moves across the display. Lowering the brightness value from 15 to 0 erases the old dot.

```
// Move one Dot

#include <gamekit.h>

int row = 2;
int column = 3;

void setup(){
  gamekit.Begin();

}

void loop(){

  gamekit.set_pixel(row,column,15);

  if(gamekit.button_pressed(butt_UP)){
    gamekit.set_pixel(row,column,0);
    row--;
  }

  if(gamekit.button_pressed(butt_DOWN)){
    gamekit.set_pixel(row,column,0);
    row++;
  }
```

```
  if(gamekit.button_pressed(butt_LEFT)){
    gamekit.set_pixel(row,column,0);
    column--;
  }

  if(gamekit.button_pressed(butt_RIGHT)){
    gamekit.set_pixel(row,column,0);
    column++;
  }

}
```

This “Move One Dot” programme can easily create a “paint programme” by removing the erasure of the dots from the button query. In a further step the putting and erasing of dots can be coupled to the functioning keys.

The historic development now reached the collision query and thus a very substantial element of computer games. Ralph Baer expanded his circuits in a way that generated two dots. A second player could control the second dot. Now it was possible to play “catch”.



Exercise 4: “Chasing Game“

As the Game Kit only disposes of six sensors the second player runs diagonally in the chasing game. For the recognition of the collision we use an “if” query. Please note: to compare the two dots position the double equal sign is used. The single equal sign is used to allocate values to the variables. So, we differentiate between an equal sign for allocation and an equal sign for comparison. The comparisons of the “if”- query are set in round brackets. In this case two conditions need to be true at the same time. They are connected by “&&”.

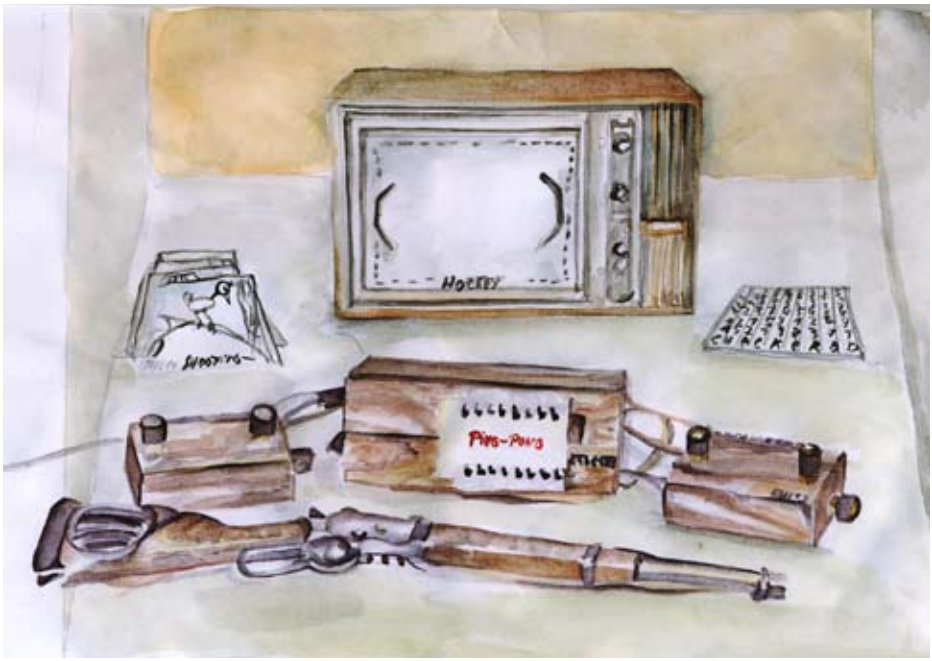
To move two dots on the display the variables for the second dot are doubled (row2, column2). And the second dot is allocated the blink value 18 in order to differentiate between the two dots.

For the collision the blink value is set to 20. The “delay()” function pauses the game for two seconds. After that the dots return to their starting positions.

```
// Chasing Game

#include <gamekit.h>

int row1 = 1; // Dot1
int column1 = 1;

int row2 = 3; // Dot2
int column2 = 5;

void setup(){
  gamekit.Begin();

}

void loop(){

  gamekit.set_pixel(row1,column1,15);
  gamekit.set_pixel(row2,column2,18);


  if(gamekit.button_pressed(butt_UP)){
    gamekit.set_pixel(row1,column1,0);
    row1--;
  }
```

```
  if(gamekit.button_pressed(butt_DOWN)){
    gamekit.set_pixel(row1,column1,0);
    row1++;
  }

  if(gamekit.button_pressed(butt_LEFT)){
    gamekit.set_pixel(row1,column1,0);
    column1--;
  }

  if(gamekit.button_pressed(butt_RIGHT)){
    gamekit.set_pixel(row1,column1,0);
    column1++;
  }

  if(gamekit.button_pressed(butt_FUNCA)){
    gamekit.set_pixel(row2,column2,0);
    column2--;
    row2--;
  }

  if(gamekit.button_pressed(butt_FUNCB)){
    gamekit.set_pixel(row2,column2,0);
    column2++;
    row2++;
  }

//collision detection
  if((row1 == row2)&&(column1==column2)){

    // blink for 2 seconds
    gamekit.set_pixel(row1,column1,20);
    delay(2000);

    // go back to start positions
    gamekit.set_pixel(row1,column1, 0);
    row1 = 1;
    column1 = 1;
    row2 = 3;
    column2 = 5;
  }
}
```

The casing game gives a first impression of the possibilities of a videogame, but even during pioneering times Ralph Baer and his co-workers realised, that the game was not entertaining enough to promise great marketing success. Shortly after, the breakthrough was brought on by a special circuit that enabled the presentation of a dot on the screen that moved without being moved by a player: the ball!



Exercise 5: “Mignon Pong”

We add a third dot and built the movement of the first two dots in a way that makes them move up and down on the left and the right like pong rackets. The magical programming row that takes care of the independent movement of the ball is very simple:

```
„column3 = column3 + columnM;„
```

The variable “column3” is allocated a value that corresponds to its current value plus the value of the variable “columnM” in this case +1 or -1. At every call of this function the ball moves one dot to the right or to the left.

To ensure the ball moves nicely and slowly we write the entire control of the ball in an if-loop that queries the system counter of the game kit and only

moves the ball at every hundreds rise of the counter. We need another variable “balltime” in which the current value of the counter is saved once, and than compared with the consecutive counter until it has moved on one hundred steps.

If the collision query is changed in such a way that the ball „hits“ the rackets and in doing so the value for the horizontal movement “columnM“ changes its sign the game almost looks like Pong. The ball moves from racket to racket but the vertical movement of the ball is missing. For this we use – for simplicities sake- random values, that are generated by the command “random()”. At the same time another if- query becomes necessary, that prevents the ball leaving the display on the top or bottom. Here again by the changing of the sign of the variable for movement “rowM” a rebound is generated.

To prevent the same random values to be generated every time the game kit is switched on the beginning of the series of random values, the so called “randomSeed”, is set to a changing value in the “Setup”. This value is best measured at the analogue input of the extension port. If no sensors such as photocells, microphone, or pressure sensor, are connected a noise is measured that delivers good coincidental values.

```
// Mignon Pong

#include <gamekit.h>

int row1 = 2; // Dot1
int column1 = 0;

int row2 = 2; // Dot2
int column2 = 6;

int row3 = 2; // Dot 3
int column3 = 3;
int rowM = 1;
int columnM = 1;
int balltime = 0;

void setup(){
  gamekit.Begin();
  //set the random seed to a noise value which is measured on the
analogue input pin 5 of the extensions port
  randomSeed(analogRead(5));
}
```

```
void loop(){

  gamekit.set_pixel(row1,column1,15);
  gamekit.set_pixel(row2,column2,15);
  gamekit.set_pixel(row3,column3,15);


  if(gamekit.button_pressed(butt_UP)){
    gamekit.set_pixel(row1,column1,0);
    row1--;
  }

  if(gamekit.button_pressed(butt_DOWN)){
    gamekit.set_pixel(row1,column1,0);
    row1++;
  }

  if(gamekit.button_pressed(butt_FUNCA)){
    gamekit.set_pixel(row2,column2,0);
    row2--;
  }

  if(gamekit.button_pressed(butt_FUNCB)){
    gamekit.set_pixel(row2,column2,0);
    row2++;
  }

  // move ball
  if(gamekit.get_systemcounter()> 100+balltime){
    balltime = gamekit.get_systemcounter();

    // collision detection 1
    if(((row1 == row3+rowM)&&(column1==column3+columnM))||(
(row2 == row3+rowM)&&(column2==column3+columnM))){
      columnM = columnM*-1;
      rowM = random(3)-1; // find a new angle for the balls path by
chance
      if(row3==4) rowM = random(2)-1;
      if(row3==0) rowM = random(2);
    }

    gamekit.set_pixel(row3, column3, 0); // turn the old dot off
```

```
    column3 = column3 + columnM; // move the dot further

    // bounce at top and bottom
    row3 = row3 + rowM;
    if(row3 +rowM <= -1){
      rowM = 1;
    }
    if(row3 +rowM >= 5){
      rowM = -1;
    }
  }
}
```

Now we reach a point at which the elementary principles of digital games have been worked through. With the learned functions new variations can be invented and games can be programmed easily.

At this stage of development Ralph Baer tried to market his product unsuccessfully in 1967. Only years later, in 1972 he was able to convince the company Magnavox to realize his vision with the “Magnavox Odyssey”. The first video console of the world was based on simple functions: Dots could be moved across the screen, moved by themselves, and it was noticeable when they hit each other. There were no sounds, no images, and no counter. Thus the “Magnavox Odyssey” included foils -with printed on motives of games that were stuck on the screen- and cards for counting scores.

So far, our exercises lack these components that turn an abstract game principle into an attractive videogame. In regard to this various possibilities exist that were discovered step-by-step in the history of computer games and lead to great commercial success. “Space Invaders” introduced the concept of several lives of the player in 1978. A figure in a game was given a name for the first time with “Pac-Man” in 1980, and the first character was established with “Mario” in 1981.

6. Exercise: „Intro Image“

We design a series of images with which we can represent small stories that are suitable to combine an abstract structure of a game with a motive. Images, sounds, and melodies raise the level of entertainment: There is identification with the avatar, an increase of sensuous appeal, and fantasy is stimulated.

To draw an image on the display we can work with the already used “set_pixel”

function. However, the “load_image” function from the Game Kit Library is more comfortable.

```
uint8_t myman[5][7] PROGMEM = {
// Dot Values
};

gamekit.load_image(myman);
```

As the images use a special type of variables the library “pgmspace” needs to be integrated. After that the images are immediately defined with the variables. This is a task for the advanced: One uses an array of Unsinde Integer 8 bit variables that are placed in the programme memory. You do not have to deal with that right now! The easiest way is to insert the respective codes using “copy and paste”. Instead of “myman” any chosen title can be used for the image. The five rows of numbers with the seven values represent the rows of the display with their seven LEDs. At the value of zero the diode is turned off. At the maximum value of 15 it shines the brightest. The values in between allow for working withnuances. The “load_imge” function loads the image onto the display.

```
// Intro Image

#include <gamekit.h>
#include <avr/pgmspace.h>

uint8_t myman[5][7] PROGMEM = {
  0 ,0 ,0 ,15,0 ,0 ,0 ,
  0 ,3 ,3 ,3 ,3 ,3 ,0 ,
  0 ,0 ,0 ,3 ,0 ,0 ,0 ,
  0 ,0 ,3 ,0 ,3 ,0 ,0 ,
  0 ,0 ,3 ,0 ,3 ,0 ,0 ,
};

void setup(){
  gamekit.Begin()
}

void loop(){
  gamekit.load_image(myman);
}
```

7. Exercise: „Intro Animation“

You can easily produce an animation by simply defining several images and open them one after another. The speed of the changing of the pictures is defined by the “delay” commands behind the “load_image” commands.

The following example follows a more elegant path. Again only one image is defined. However, it functions like a film reel. In the example the dancing man consists of a series of five images. This film reel is pushed from the bottom to the top across the display. Thus an animation is generated.

To load an image that is bigger than the 5 x 7 grid you use the “load_map” command. In this case the image is moved in its entirety, i.e. it jumps up 5 rows. For the line- pulling the variable “i” is used, that is raised with the command „i+=5“ at each round. An „if“ command returns „i“ to zero.

```
// Intro Animation

#include <gamekit.h>
#include <avr/pgmspace.h>
```

```
uint8_t dancingman [25][7] PROGMEM = {
  0 ,0 ,0 ,9 ,0 ,9 ,0 ,
  0 ,9 ,9 ,9 ,9 ,0 ,0 ,
  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,0 ,9 ,0 ,9 ,0 ,0 ,
  0 ,9 ,0 ,0 ,9 ,0 ,0 ,

  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,9 ,9 ,9 ,9 ,9 ,0 ,
  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,0 ,9 ,0 ,9 ,0 ,0 ,
  0 ,0 ,9 ,0 ,9 ,0 ,0 ,

  0 ,9 ,0 ,9 ,0 ,9 ,0 ,
  0 ,0 ,9 ,9 ,9 ,0 ,0 ,
  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,0 ,9 ,0 ,9 ,0 ,0 ,
  0 ,0 ,9 ,0 ,9 ,0 ,0 ,

  0 ,0 ,0 ,0 ,0 ,0 ,0 ,
  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,9 ,9 ,9 ,9 ,9 ,0 ,
  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,9 ,9 ,0 ,9 ,9 ,0 ,

  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,9 ,9 ,9 ,9 ,9 ,0 ,
  0 ,0 ,0 ,9 ,0 ,0 ,0 ,
  0 ,9 ,9 ,0 ,9 ,0 ,0 ,
  0 ,0 ,0 ,0 ,9 ,0 ,0 ,

};

void setup(){
  gamekit.Begin();
}

int i = 0;

void loop(){

  gamekit.load_map( (uint8_t *) dancingman, 25, 7, i, 0);
  delay(600);
```

```
  i=i+5;

  if( i >= 25 )
    i = 0;
}
```

8. Exercise: „Simon 6“

Similar to the historic development of computer games all exercises so far disregarded the sound. Sound design and musical design often took second place. The already mentioned first game console “Magnavox Odyssey” did not feature audio at all for the reason of cost. The first computer game sound appeared in Nolan Bushnell’s Pong version in 1972. At the time this artificial, electronically generated “Sonar-Blip” Sound was unusually fascinating and hypnotic. Six years later, in 1978 “Simon (“Senso” in Germany) was the first computer game on the market that integrated sounds and musicality in the game. The game invented by Ralph Baer generated random series of sounds that could be memorised and replayed like melodies.

[Coming soon...] Text text text Text

```
// Simon 6

#include <gamekit.h>
#include <avr/pgmspace.h>

Code is coming soon...
```

Code is coming soon...

9. Exercise: „Mignon Noise Generator“

In 1977, during the Christmas trade Nolan Bushnell marketed his computer game console “Atari VCS 2600”. It became a classic. This console was equipped with a simple audio processor “Atari TIA” that was able to play in two voices various undulations in mono through the speakers of the television set. Games like “Combat” featured interesting sounds, such as the experimental motor sound of the tanks.

[Coming soon...] Text text text Text

```
// Mignon Noise Generator

#include <gamekit.h>
#include <avr/pgmspace.h>

Code is coming soon...
```

```
Code is coming soon...
```

10. Exercise: „Intro Melody“

“Space Invaders” was probably the most important milestone in the history of computer game development through several groundbreaking innovations. The game by Toshihiro Nishikado developed strong dramaturgical characteristics. For example, the player possessed three lives for the first time. An adaptive soundtrack that underlined the increasing menace through the attacker supported the suspense; the quicker the Space Invaders moved the quicker the bass sounds played.

If you want to underscore your Mignon Game Kit game with sounds you encounter the problem that the playback of the audio by use of the “play_sound” function always pauses the programme just like a pause through “delay” would. The “play_melody” function remedies things. Through this function the Game Kit Library offers a comfortable possibility to accompany animations or games with music. You are spared programming a music player yourself. Once started the melody plays in the background while the normal programme continues.

```
// Intro Melody

#include <gamekit.h>
#include <avr/pgmspace.h>

uint16_t melody[][2] PROGMEM =
{
 m_set_volume, LOUD,

 //------------ 1
 m_16_dot_note, m_G4,
 m_rest       , m_16_rest,
 m_16_dot_note, m_D4,
 m_rest       , m_16_rest,
 m_16_dot_note, m_E4,
 m_rest       , m_16_rest,
 m_16_note    , m_F4,
```

```
  m_rest      , m_16_rest,
  m_16_note   , m_E4,
  m_rest      , m_16_rest,
  m_16_note   , m_D4,
  m_16_note   , m_C4,
  m_rest      , m_16_rest,
  m_16_note   , m_C4,
  m_rest      , m_16_rest,
  m_16_note   , m_E4,
  m_rest      , m_16_rest,
  m_16_note   , m_G4,
  m_rest      , m_16_rest,
  m_16_note   , m_E4,
  m_rest      , m_16_rest,
  m_16_note   , m_C4,
  m_rest      , m_16_rest,
  m_16_note   , m_D4,
  m_rest      , m_16_rest,
  m_16_note   , m_D4,
  m_rest      , m_16_rest,
  m_16_note   , m_E4,
  m_rest      , m_16_rest,
  m_16_note   , m_F4,
  m_rest      , m_16_rest,
  m_16_note   , m_G4,
  m_rest      , m_16_rest,
  m_16_note   , m_E4,
  m_rest      , m_16_dot_rest,
  m_16_note   , m_C4,
  m_rest      , m_16_dot_rest,
  m_16_note   , m_C4,
  m_rest      , m_16_dot_rest,

  m_stop      , 0
};

void setup(){
  gamekit.Begin();
  gamekit.play_melody((uint16_t *) melody);
}

void loop(){
}
```

Instructions for Soldering

Attention! Before turning on the machine: A soldering iron is a very dangerous tool! The iron becomes so hot that even the smallest and shortest contact can lead to sever burns. Everything that touches the soldering iron such as the wire, the construction parts and the stand of the soldering iron also get very hot! Children should be at least ten years old before beginning to solder. Accidents are best avoided by creating the optimum working conditions and a concentrated atmosphere. For that you need good lighting, a big fireproof worktop, time and quietude. In any case you need a secure space to place the soldering iron. If you do not have a stand it is possible to use a heavy ashtray, for example. When equipping the workstation it is important to make sure that the soldering iron is nowhere near a 230 Volt cable!

You need the right tools for a successful introduction to soldering. The soldering iron should posses a fine, thin top and the soldering wire should also be fine and thin (maximum diameter: 1 millimeter). For beginners a lead- containing electronics solder with the most possible flux is most suitable. (Lead- free solder would be less hazardous to health but is more difficult to use for now). DIY-Stores often feature fairly coarse tools only, which are totally unsuited for the work on the Mignon Game Kit. They are probably conceived for the work on rain gutters. The electrical goods store is where you will be properly advised. Fairly inexpensive soldering equipment is sufficient for the one-time use. However, it may break after only a few working days. Proper handling provided more expensive quality products are indestructible.

Soldering work is precession work. At best, you work in the middle of the table where both elbows can be firmly placed on the work- top. Clean the soldering tip frequently by swabbing it on a wet sponge or an handkerchief. Do avoid breathing in the rising smoke! Wash your hands -with which you touched the soldering wire- well after finishing the work.

The process of soldering is as follows: Both elements to be soldered are heated with the soldering iron at the same time. When the soldering wire is held to the tip of the soldering iron it melts. In the process the flux contained in the soldering tin flows to the soldering point and evaporates there. Thus the flux is a great help. It takes the surface tension of the solder. Cold solders- soldered connections that do not pass the power sufficiently- appear when not enough flux got to the respective part, when one of the two elements to be connected was not hot enough, or when to much solder was used. With the Mignon Game Kit the silvery surfaces; “eyelets” are soldered to the wires of the construction parts. The right dosage of solder just about closes every eyelet. If the solder

forms small pellets too much was used. Too much solder can cause unwanted bridges and thus lead to short circuits on the main board.

To remove the solder you heat the soldering point and try to liquefy the solder as much as possible and remove it with a suction pump as quickly as possible before it becomes solid. To unsolder a construction part you try to remove the solder with the pump as well as feasible. Often it remains impossible to remove the construction part. In such a case you fix the main board with a bench vice or a helping hand in a way that enables you to place the soldering iron from one side while you remove the construction part with pliers.

Genesis

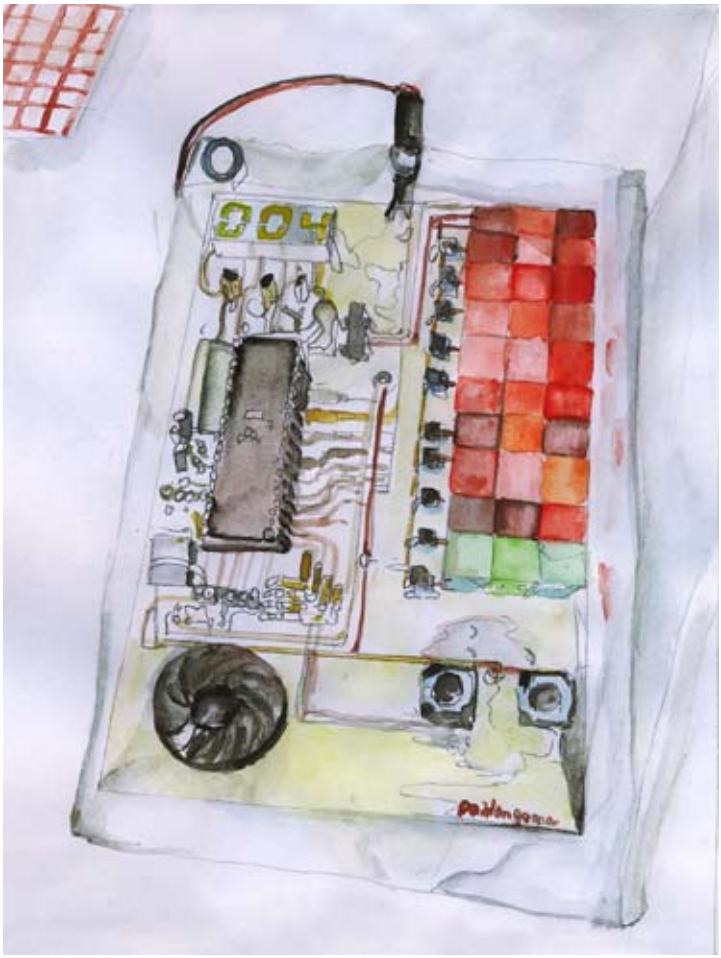
The “Mignon Game Kit” project developed from the two works “PaintOn_ Games” and “GameBolx”. Both of these precursors of the game kit drafted hard- and software visions of how to confer Friedrich Fröbel’s building blocks on the digital world. In the beginning the impulse for this was an aesthetic interest.

However, the multitude of reactions to these two works quickly referred to a pedagogical dimension of the approach and finally let to the development of the “Mignon Game Kit” construction set. Thus a hybrid project developed in the common ground of art, education, and technology. The artistic experiment became a product with a multitude of links to chances and problems that emerged in the areas of teaching and education in the course of the change to an information-society.

The “Mignon Game” reduces mobile game consoles such as the Game Boy or the “Play Station Portable” -and along with those interactive digital media per se - to their minimal level on which they function symbolically. The concept of the construction associates the introduction of electronics and information technology with computer games. This connection is advantageous for both

sides; the consumers of video games become interested in the structures behind the user interface, engineers and programmers in the making are challenged and motivated by the playful element. History can also justify the linkage of games and technology, as the improving performance of personal computers stands in close connection to the computer game industry.

The games of the “Mignon Game Kits” can also be deduced historically. The minimal technical specifications, such as 35 pixel resolution and 8 Kilobyte memory limit the game design. Even simple game motives such as “Pong” or “Pac Man” can only be realised through creative abstraction on this basis. Thus the game itself is a creative act that challenges imagination and fantasy.



Gamekit Library Reference

Short Overview of Functions

void Begin();
Initialises (i.e. starts) the Gamekit.

void set_pixel(row, column, value);
Allocates the pixel value in the row and the column.
Values between 0 and 15 are levels of brightness, higher values make pixels blink, or the like.

uint8_t get_pixel(row, column);
Reads the value of the pixel in the „row“ and the „column“.

void load_image(Bild);
Loads the image onto the display (see the example „load_image“).

void load_map(uint8_t *, uint8_t, uint8_t, uint8_t, uint8_t); (for the advanced)
Loads a detail of a large image to the display (see example „load_map“).

void assign_pixelfunction(uint8_t, uint8_t(*)(uint8_t, uint8_t, uint32_t)); (for the very advanced)
Allocates a value of pixels (has to be higher than 15) to a function.

uint32_t get_systemcounter();
Reads the system counter. The system counter rises by one when the display is redrawn.

uint8_t get_buttons(); (for the very advanced)
Reads all keys at the same time.

boolean button_pressed(button);
Checks if the key “button” is pressed.
„button“ may be:
butt_UP (Up)
butt_DOWN (Down)
butt_LEFT (Left)
butt_right (Right)
butt_FUNCA (function A)
butt_FUNCB (function B)

void wait_button_pressed(button);
Waits untill the key “button” is pressed.

void wait_button_released(button);
Waits untill the key “button” is released.

void set_button_timing(first, common); (for the advanced)
Changes the speed of the repetition of the keys (see example “button_ pressed”).

void play_tone(frequency, duration, volume);
Plays a sound of the pitch “frequency” over a period of time (in milliseconds) Sound level can be “LOUD” for loud and “SILENT” for silent.

void play_melody; (for the advanced)
Plays the melody (see example “play_melody”).

uint16_t get_current_melody_event(); (for the very advanced)
Reads which “play_melody” is playing currently.

void set_current_melody_event(number); (for the very advanced)
Jumps to the place “number” of the melody.

The Notes

Reads the notes in the “playMelody” function
The English notation is used: C, D, E, F, G, A, B

```
#define    m_C2        65
#define    m_Bt2        m_C2
#define    m_Ct2        69
#define    m_Db2        m_Ct2
#define    m_D2        73
#define    m_Dt2        78
#define    m_Eb2        m_Dt2
#define    m_E2        82
#define    m_Fb2        m_E2
#define    m_F2        87
#define    m_Et2        m_Et2
#define    m_Ft2        92
#define    m_Gb2        m_Ft2
#define    m_G2        98
#define    m_Gt2        104
#define    m_Ab2        m_Gt2
#define    m_A2        110
#define    m_At2        117    //46
#define    m_Bb2        m_At2   #define    m_B2        123
#define    m_Cb2        m_B2

#define    m_C3        131
#define    m_Bt3        m_C3
#define    m_Ct3        139    //49
#define    m_Db3        m_Ct3
#define    m_D3        147
#define    m_Dt3        156
#define    m_Eb3        m_Dt3
#define    m_E3        165
#define    m_Fb3        m_E3
#define    m_F3        175    //53
#define    m_Et3        m_E3
#define    m_Ft3        185
#define    m_Gb3        m_Ft3
#define    m_G3        196
#define    m_Gt3        208
#define    m_Ab3        m_Gt3
#define    m_A3        220    //57
```

```
#define    m_At3       233    //58
#define    m_Bb3       m_At3   #define    m_B3       247     //59
#define    m_Cb3       m_B3

#define    m_C4        262    //60
#define    m_Bt4       m_C4
#define    m_Ct4       277
#define    m_Db4       m_Ct4
#define    m_D4        294
#define    m_Dt4       311
#define    m_Eb4       m_Dt4
#define    m_E4        330
#define    m_Fb4       m_E4
#define    m_F4        349
#define    m_Et4       m_E4
#define    m_Ft4       370
#define    m_Gb4       m_Ft4
#define    m_G4        392
#define    m_Gt4       415
#define    m_Ab4       m_Gt4
#define    m_A4        440
#define    m_At4       466
#define    m_Bb4       m_At4   #define    m_B4       494    //71
#define    m_Cb4       m_B4

#define    m_C5        523    //72
#define    m_Bt5       m_C5
#define    m_Ct5       554
#define    m_Db5       m_Ct5
#define    m_D5        587
#define    m_Dt5       622
#define    m_Eb5       m_Dt5
#define    m_E5        659
#define    m_Fb5       m_E5
#define    m_F5        698
#define    m_Et5       m_E5
#define    m_Ft5       740
#define    m_Gb5       m_Ft5
#define    m_G5        784
#define    m_Gt5       831
#define    m_Ab5       m_Gt5
#define    m_A5        880
#define    m_At5       932
```

```
#define    m_Bb5       m_At5   #define    m_B5       988
#define    m_Cb5       m_B5   //83

#define    m_C6       1047    //84
#define    m_Bt6       m_C6
#define    m_Ct6       1109
#define    m_Db6       m_Ct6
#define    m_D6       1175
#define    m_Dt6       1245
#define    m_Eb6       m_Dt6
#define    m_E6       1319
#define    m_Fb6       m_E6
#define    m_F6       1397
#define    m_Et6       m_E6
#define    m_Ft6       1480
#define    m_Gb6       m_Ft6
#define    m_G6       1568
#define    m_Gt6       1661
#define    m_Ab6       m_Gt6
#define    m_A6       1760
#define    m_At6       1865
#define    m_Bb6       m_At6   #define    m_B6       1976    //95
#define    m_Cb6       m_B6

#define    m_C7       2093    //96
#define    m_Bt7       m_C7
#define    m_Ct7       2217
#define    m_Db7       m_Ct7
#define    m_D7       2349
#define    m_Dt7       2489
#define    m_Eb7       m_Dt7
#define    m_E7       2637
#define    m_Fb7       m_E7
#define    m_F7       2794
#define    m_Et7       m_E7
#define    m_Ft7       2960
#define    m_Gb7       m_Ft7
#define    m_G7       3136
#define    m_Gt7       3322
#define    m_Ab7       m_Gt7
#define    m_A7       3520
#define    m_At7       3792
#define    m_Bb7       m_At7   #define    m_B7       3951    //107
```

```
#define    m_Cb7        m_B7

#define    m_C8        4186    //108
#define    m_Bt8        m_C8
#define    m_Ct8        4439
#define    m_Db8        m_Ct8
#define    m_D8        4699
#define    m_Dt8        4978
#define    m_Eb8        m_Dt8
#define    m_E8        5274
#define    m_Fb8        m_E8
#define    m_F8        5588
#define    m_Et8        m_E8
#define    m_Ft8        5920
#define    m_Gb8        m_Ft8
#define    m_G8        6272
#define    m_Gt8        6645
#define    m_Ab8        m_Gt8
#define    m_A8        7040
#define    m_At8        7459
#define    m_Bb8        m_At8   #define    m_B8        7902    //119
#define    m_Cb8        m_B8
```

Debugging

[Coming soon...] Text text text Text

Text text text Text

Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text

Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text text text Text

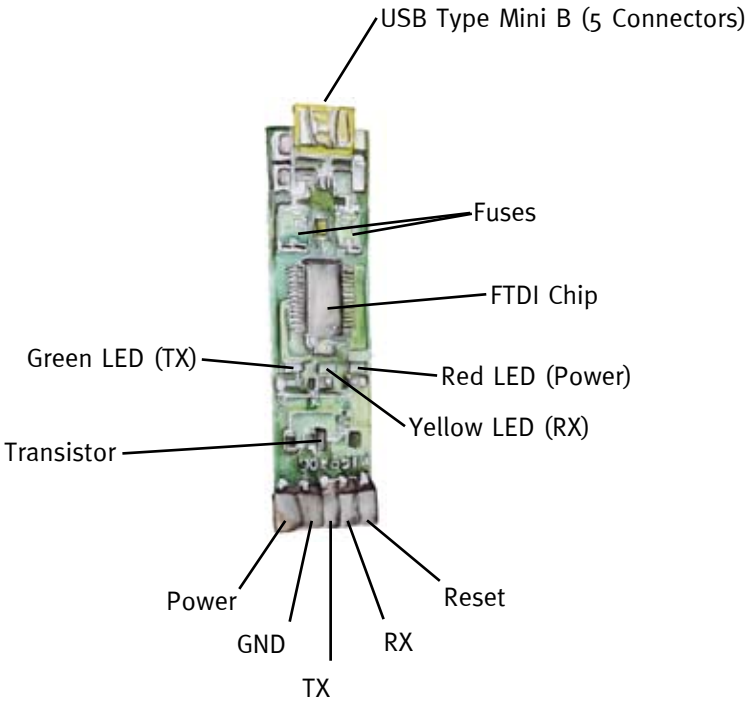
Text text text Text

Text text text Text

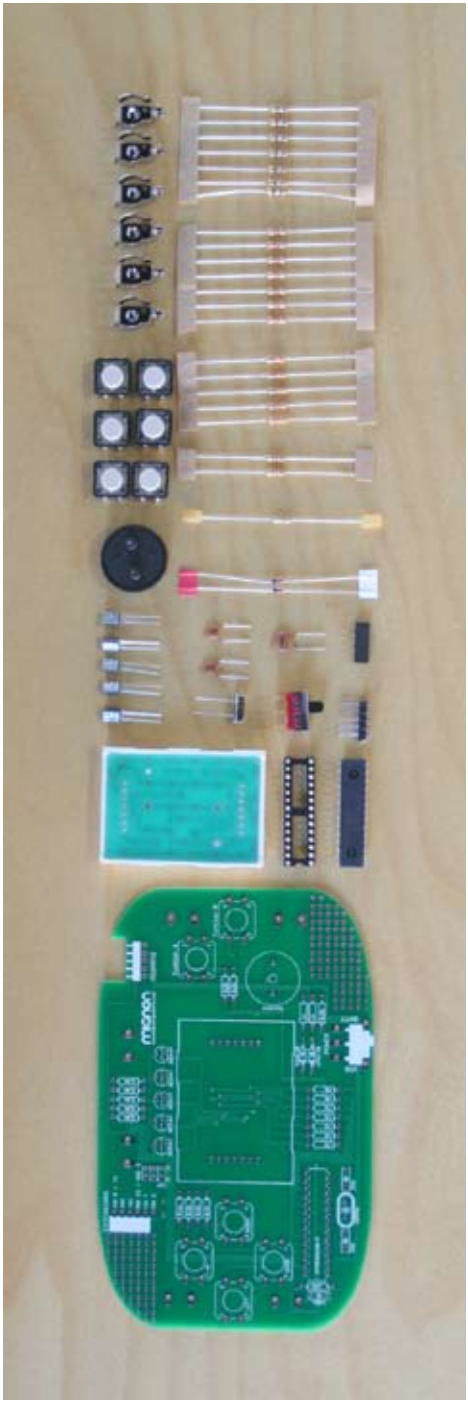
Text text text Text

MigProg

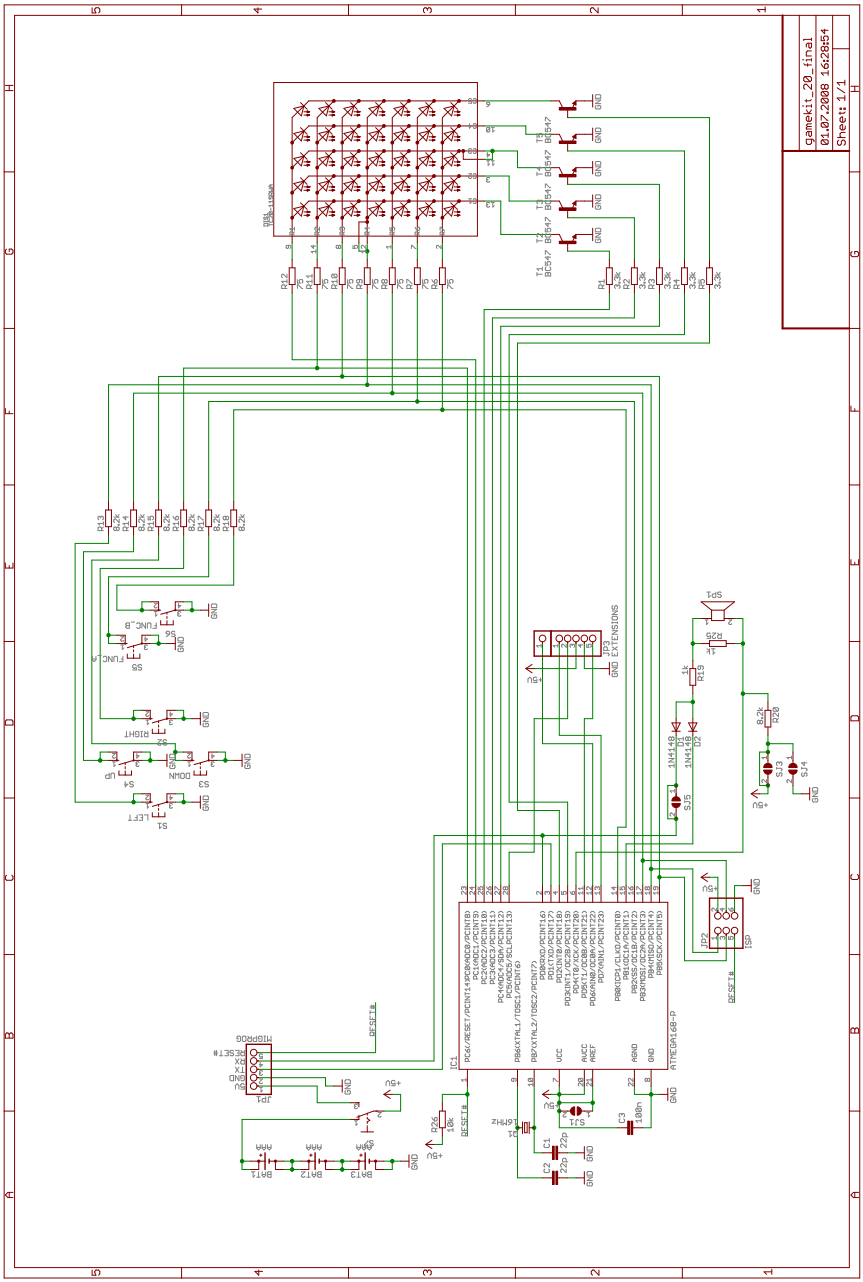
Text coming soon

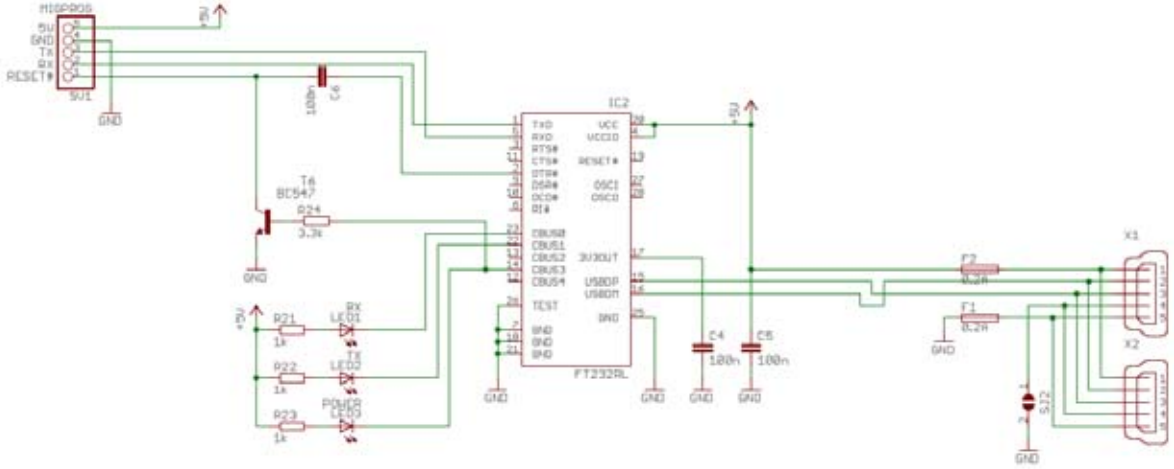


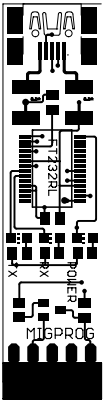
Content

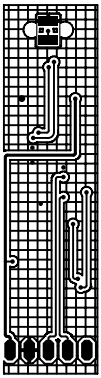


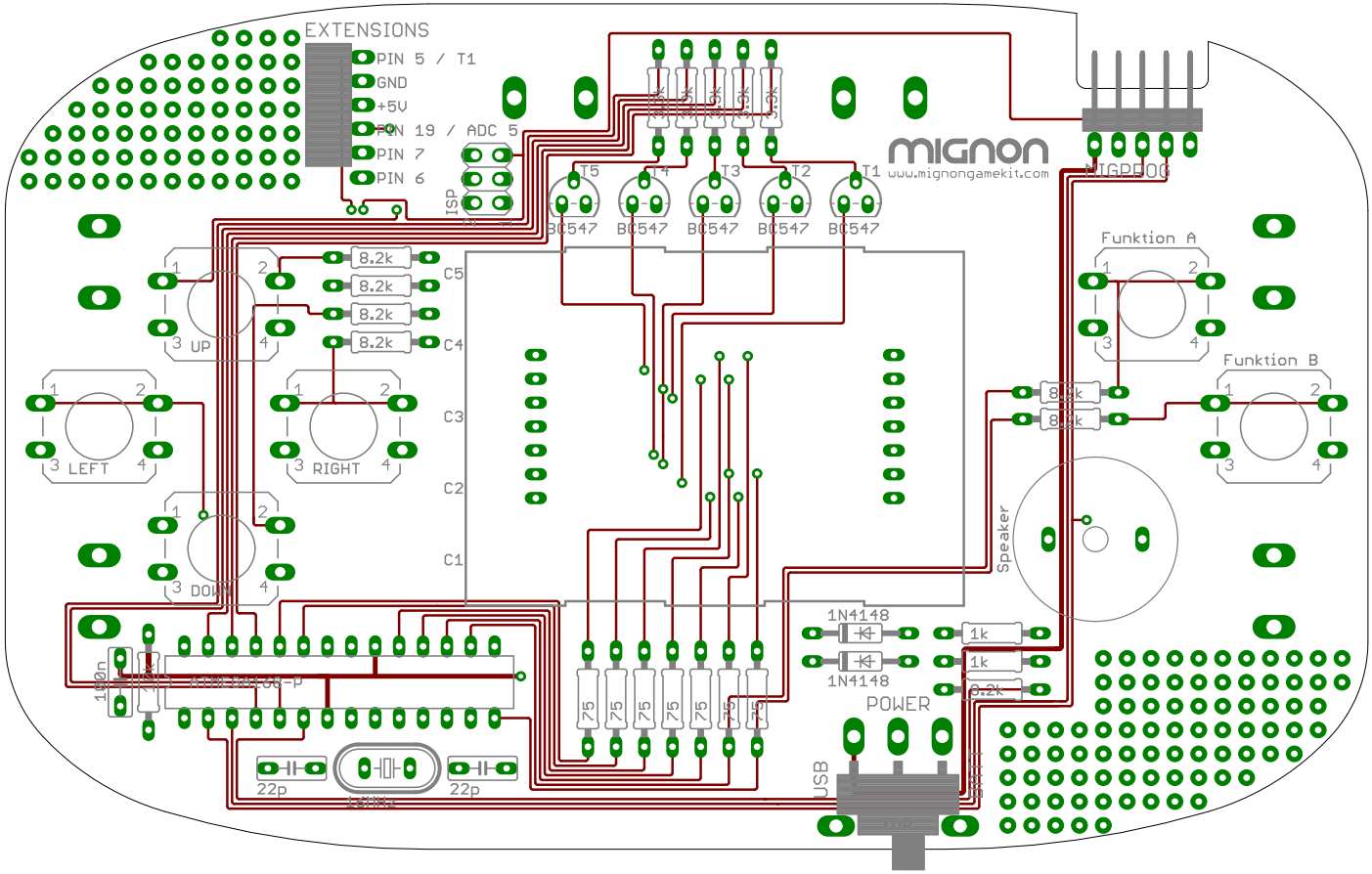
Circuit Diagrams



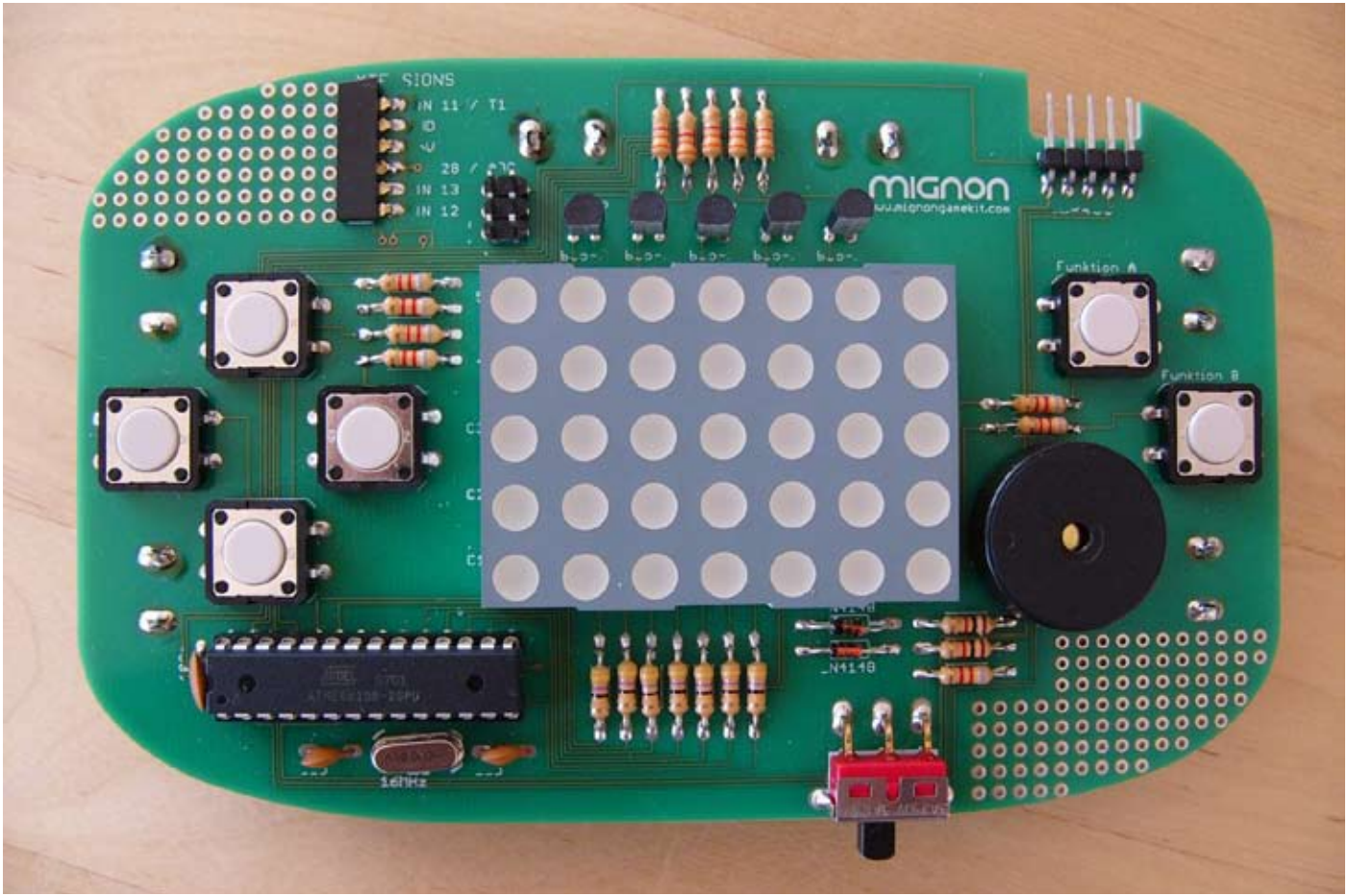

M10PROG
5V
GND
Tx
Rx
RESET#
SV1
GND
5V
100n
C6
IC2
TXD
RXD
RTS#
CTS#
DTR#
DSR#
DCD#
RI#
VCC
VCCIO
RESET#
OSCI
OSCO
CBUS0
CBUS1
CBUS2
CBUS3
CBUS4
3V3OUT
USBDP
USBDM
TEST
GND
FT232RL
T6
BC547
R24
3.3k
GND
5V
R21
1k
LED1
RX
R22
1k
LED2
TX
R23
1k
LED3
POWER
GND
GND
GND
GND
5V
C4
100n
C5
100n
F2
0.2A
F1
0.2A
GND
X1
X2
SV2
GND


AT732RL
TX
RX
POWER
MIGPROG




EXTENSIONS
PIN 5 / T1
GND
+5V
PIN 19 / ADC 5
PIN 7
PIN 6
ISP
MIGNON
www.mignongamekit.com
DEBUG
T5
T4
T3
T2
T1
BC547
BC547
BC547
BC547
BC547
8.2k
8.2k
8.2k
8.2k
UP
LEFT
RIGHT
DOWN
C5
C4
C3
C2
C1
Funktion A
Funktion B
Speaker
1N4148
1N4148
1k
1k
0.2k
POWER
USB
100n
22p
22p
75
75
75
75
75
75
75

www.widgetgamekit.com
widget game kit 2.0
breadboard


IN 11 / T1
IN 13
IN 12
MIGNON
Funktion A
Funktion B
16MHz

Credites

Thomas Wappler / technic
Catrine Val / watercolour paintings
Franz Michael Schmid / illustrations
Henrike Rodegro / English translation

Copyright

www.mignongamekit.com