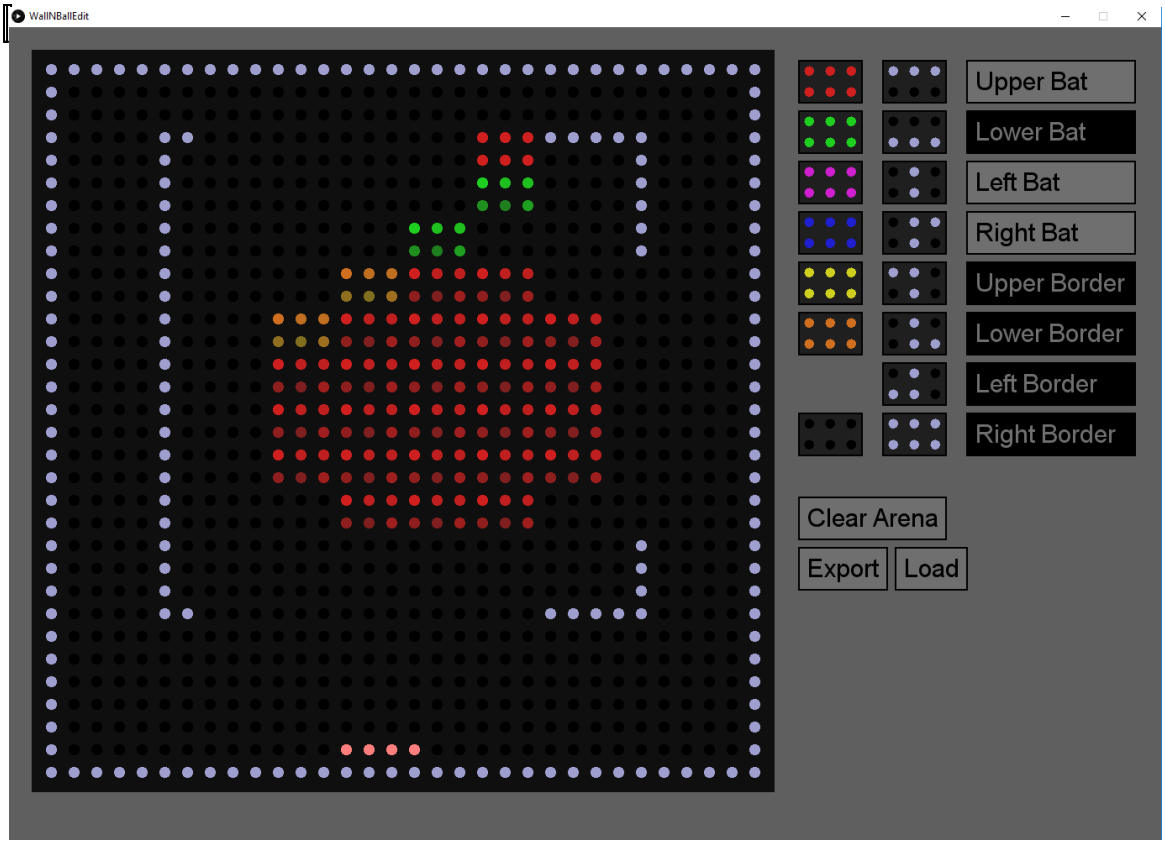
Wall‘n‘Ball Edit

Wall‘n‘Ball Edit is a playfield („arena“) editor for the LEDmePlay breakout-style game *Wall‘n‘Ball*. It was developed using Processing which is an easy-to-learn computer language for designers and artists. It is based on the popular Java programming language. You can download it from https://processing.org/download/.

A ready-made package of **Wall‘n‘Ball Edit** can be executed by clicking its icon on the Windows, Mac OS X or Linux desktop. Alternatively, the source code of the program can be loaded and executed from within the Processing environment. Please read the section on installing **Wall‘n‘Ball Edit** for details.



Scope

Wall‘nBall Edit was written in order to be able to conveniently create playfields, „arenas“, for the LEDmePlay game *Wall‘n‘Ball* on a personal computer that runs a *Processing* compatible operating system, i.e., one of the more recent incarnations of Windows, Mac OS X and Linux. With **Wall‘n‘Ball Edit**, you can load and modify arenas from Wall‘n‘Ball or create your own from scratch. Once finished, you can save them to your PC's harddisk . You may then open one of these files in a text editor and copy and paste its content right into the program Wall‘n‘Ball edit. It is described later on how this can be accomplished.

As *Processing* does not provide for user interface elements such as pull-down menus, sliders, buttons and so on, one needs to invent one's own interface elements, as can be seen in the screenshot above. The program is mouse-controlled.

Installation

You should have downloaded a ZIP file with a name similar to „WallNBallEdit.zip“ from this website, www.mithotronic.de. Copy it to a location you want it to be, then unzip it. Within the newly deflated directory are the two folders WallNBallEdit.windows32, WallNBallEdit.windows64, the font file 'ArialMT-30.vlw', the two example files 'apple' and 'helicopter' and WallNBallEdit's source code for Processing, 'WallNBallEdit.pde'. The suffix '.pde' might be invisible, dependent on your operating system's current parameter settings.

There are three ways how to execute **Wall‘n‘Ball Edit**. All might work for you. Either download and install the Processing Environment and start the source code from there, or run the executable file for Windows. The third way is to export the source code from within the Processing IDE. More about this will follow later on. In any case the font file 'ArialMT-30.vlw' needs to be present in the directory with the *Wall‘n‘Ball Edit* source code or the *Wall‘n‘ Ball Edit* executable, otherwise the application will not work. Please try one of the following alternatives:

1. Run the *Wall‘n‘ Ball Edit* source code

Download the **Processing** IDE ("Integrated Development Environment") for your operating system. Get it from https://processing.org/ and install it on your harddisk as described on their website.

Locate the file 'WallNBallEdit.pde' in the newly created directory obtained by deflating the downloaded ZIP file. Please note that the date could be different, and the suffix may be invisible. Check that the font file 'ArialMT-30.vlw' is present in the same directory. Also note that the directory bears the same name as the source code file. This is mandatory for the Processing Environment. If you double click the source code file, Processing 3 will be loaded and run and then show you the 'LEDmePlay Draw' source code in its window. From there you may click on the Play button on top left which will compile and run the 'LEDmePlay Draw' application. Alternatively, you can start the Processing IDE yourself and load the source code from there.

2. If your operating system is Windows 7/8/10, run the 'WallNBallEdit' executable in one of the subfolders 'WallNBallEdit_windows32' or 'WallNBallEdit.windows64' in the newly created directory that you obtained by deflating the ZIP file you downloaded from this website, www.mithotronic.de. The files 'ArialMT-30.vlw' should also be in this folder. Double click on the executable file 'WallNBallEdit.pde'. This should start the application. Please note that the date could be different, and the suffix may be invisible. If the file in the directory with suffix 'windows32' does not work for you, try the 'windows64' version. If neither works for you, please see alternative 3 which follows. If your operating system is Mac OS X or a Linux OS, please see alternative 3.

3. Export the Processing project from within the Processing IDE.

Download the **Processing** IDE ("Integrated Development Environment") for your operating system. Get it from https://processing.org/ and install it on your harddisk as described on their website.

Start the Processing IDE via double clicking on its icon. Locate the 'WallNBallEdit' source code as described above and load it from within Processing. You should now see the source code. Select 'Export' from the File menu. Select your platform - either Windows, Mac OS X or Linux. If you like the application to run in fullscreen mode, select 'Presentation Mode'. However, I do not recommend this as you may want to use other applications besides 'WallNBallEdit'.

You may want to embed Java 8 right into the exported application. If you want to do this, you need to install Java 8 beforehand from http://www.java.com/download. See the documentation there. I suggest to make a first try without embedding Java 8 into the application. If it does not work then, you can repeat the procedure and embed Java 8.

Anyway, if you have made the necessary choices and then clicked the 'Export' button, Processing will export the 'WallNBallEdit' source code (called a 'sketch' in Processing lingo) into an executable for your platform. Processing creates a new directory in the folder with the 'WallNBallEdit' sketch, 'WallNBallEdit.windows32' and 'WallNBallEdit.windows64' if you are on Windows. You can start the executable as usual by double-clicking on its icon. For further explanations, see alternative 2. For Linux and Mac OS X, the executables may look similar. However, I did not try it.

Elements in the Application Window

At the left side within its window, **Wall‘n‘Ball Edit** shows an abstract view of the LEDmePlay's LED matrix panel with stylized 1024 LEDs. This is the area in which all drawing operations take place. When you move the mouse over it, the drawing position under the mouse is highlighted. As all game elements in Wall‘n‘Ball are 2x3 „pixels“ wide, you cannot use elements smaller than that in your arena design, for instance single dots. With the drawing tools supplied, you may draw bricks in different colors for the player to hit and demolish and indestructible walls in different shapes. You may also clear the panel and start anew.

Situated at the right side are two columns of arena elements, bricks in six colors and walls in eight shapes. There is also an empty brick that represents empty space in the playfield. If you click on any of these elements, the shape of the mouse cursor changes into this shape. Left clicking the mouse within the LED matrix places the selected element around the mouse position.

On the right edge of the screen are eight lock knobs. The four upper knobs let you add or remove one of the bats, the four lower knobs let you add or remove one of the border pieces.

On the lower right part of the screen are three buttons, „Clear Arena“, „Export“ and „Load“. The „Clear Arena“ purges all arena elements from the LED matrix, „Export“ allows you to select a name for the current arena and then save it to disk, and „Load“ enables you to retrieve an arena definition from a file on the hard disk.

You may terminate the application by either pressing the Escape key or by clicking the cross in the upper right corner of the application window. Please note that this does not save the arena data. If you have not saved it beforehand, it is being deleted.

Using Wall‘n‘Ball Edit

You start a editing session either by loading a file with an arena definition, or by drawing straight-on on a blank LED matrix. The latter is the default state once the program has been started. If you load an arena definition, it is displayed on-screen immediately, and, when over the LED matrix illustration, the mouse cursor changes into a red brick as default.

Place the mouse somewhere over the LED matrix and click the left mouse button: the currently selected brick is placed within the arena. If you want to change the brick later on, replace it by another brick, for instance the empty brick. Place bricks on the LED matrix until you are done. You may then want to decide how many open border parts and bats you place in the arena. If you place an open border part it usually makes sense to also place a bat there. Use the toggle buttons marked „Upper Bat“ and so on to place the bats and the toggle buttons „Upper Border“ etc. to add or remove the respective borders. The program does not allow to place no bats at all into the arena. Should you deselect the only bat left, it activates the lower bat.

If you are done designing your arena, press the „Export“ button and save your creation to disk. You might also want to load an arena from disk with the „Load“ button. When you need to start from scratch, clear the LED matrix with the „Clear“ button.

Embedding a playfield („arena“) in Wall‘n‘Ball

In the following you will learn how to embed your playfields into the Wall‘n‘Ball main Arduino sketch. Wall‘n‘Ball does not allow to load a file with playfield („arena“) definitions, as there is no drive etc. attached to the Arduino Mega 2560 in the LEDmePlay. For the time being you need to manually place your playfield definitions in the Wall‘n‘Ball source code (or „sketch“ in Arduino lingo). You should make a copy of the original Wall‘n‘Ball source code before you modify it.

Please note that you may not delete any of the playfield definitions in the source code. Instead you should replace them with your own playfield definitions. This is because the count of playfield definitions needs to remain constant. With a more profound knowledge of the program you can circumvent this limitation.

The playfield definition from the file „apple“ that comes with **WallNBall Edit** looks like this:

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 11, 0, 0, 0, 0, 0, 9, 12, 0,
0, 10, 0, 0, 0, 0, 3, 0, 10, 0,
0, 10, 0, 0, 0, 3, 0, 0, 10, 0,
0, 10, 0, 0, 8, 2, 2, 0, 0, 0,
0, 10, 0, 8, 2, 2, 2, 2, 0, 0,
0, 10, 0, 2, 2, 2, 2, 2, 0, 0,
0, 10, 0, 2, 2, 2, 2, 2, 0, 0,
0, 10, 0, 2, 2, 2, 2, 2, 0, 0,
0, 10, 0, 0, 2, 2, 2, 0, 0, 0,
0, 10, 0, 0, 0, 0, 0, 0, 10, 0,
0, 14, 0, 0, 0, 0, 0, 15, 13, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, // Upper, lower, left, right bat is 1-active/0 - inactive
0, 0, 0, 0,  // Upper, lower, left, right border is 0-closed/1-open
```

The first 15 lines define the 15 rows of bricks in the playfield. A line in the arena contains ten bricks side-by-side. Each individual number represents a certain type of brick.

The second to last line determines which bats are being used in the playfield. A zero means the respective bat is not present, a one means the bat is there. The first number stands for the upper bat, the second for the lower bat, the third for the left and the fourth for the right bat.

The last line defines which of the wall elements in the border are open and closed. A zero codes for a closed wall and a one for an open wall. Please notice that by „wall“ we refer to an entire side of the playing area, not merely one brick.

Next comes the Arduino code from the sketch *WallNBall.ino* (or a similar name with or without the suffix .ino) from ca. line 574:

```
// Playfield encoding

#define EMPTY 0

#define UNUSED_BRICK 1 // currently unused

#define RED_BRICK 2

#define GREEN_BRICK 3

#define BLUE_BRICK 4

#define VIOLET_BRICK 5

#define YELLOW_BRICK 6

#define SOLID_BRICK 7
```

```
#define ORANGE_BRICK 8

#define HORIZ_WALL_HIGH 9

#define VERT_WALL 10

#define CORNER_NW 11

#define CORNER_NE 12

#define CORNER_SW 13

#define CORNER_SE 14

#define HORIZ_WALL_LOW 15
```

Compare these definitions with the playfield definitions either from the file „apple“ or with those from within the Wall‘n‘Ball sketch. You will find them at approximately line 619. Have a look at the first fifteen lines in the file „apple“ (or the corresponding section in the sketch). The first two rows of bricks are empty. The ball(s) can rebound there from the upper wall. In the third row the first brick is empty, too, and the second brick is „CORNER_NW“, a wall corner element with its bend in northeast direction.

Notice how the entire playfield is surrounded by 0‘s. The last row in the playfield is empty because the lower bat needs to hover around there. There is no other bat. All border parts are closed.

When you are going to replace one of the playfield definitions in the *Wall‘n‘Ball* Arduino sketch with your own, you can mark the particular code section using the mouse, delete it with the Del key, then copy the new playfield definition from the file you saved in *Wall‘n‘Ball Edit* to the clipboard and finally paste it right in-place in the empty section in the *Wall‘n‘Ball* sketch.

Room for Improvement

There are plenty of possibilities for improving the program. For a better user interface, one could try Peter Lager's G4P GUI controls library from http://www.lagers.org.uk/g4p/index.html

You cannot currently enlarge or shrink the application's window. This is bad practice. If you want this, however, the GUI control elements need to change their size accordingly which might be hard to achieve.

The logic of the buttons for the bats is reversed with respect to that of the buttons for the wall borders. The program crashes occasionally while the user is load a new playfield definition from a file. Both will be fixed in an upcoming version of the program.

License