TurtleBot

1 The TurtleBot application

At its very core is the TurtleBot application a combined integrated development environment and simulator, enabling children to explore the world of robotics and programming. The application was created with extendability in mind, having two distinct methods for creating programs, the possibility of extending the existing block language, several programming languages, and different visual setups depending on your preference and needs.

1.1 Basic screen layout

The application is divided into three major components; *header*, *body* and *footer*(figure 1).

In order to make the application easy to use by people unfamiliar to programming and robotics it utilizes a block programming interface as its default input method. With this input method the user can drag *codeblocks* from the header, down to the body/programming area. After the blocks has been placed in the programming area they may be moved around to change the program or deleted by dragging them back to the header.

As seen in figure 1 the application starts with four codeblocks the first time. These blocks are the *move-blocks* in the application. Everytime one of these blocks are executed by the simulator it will move the robot around.

In order for you to get more familiar with the layout we have highlighted the different components, and provided a simple description of them below figure 1.

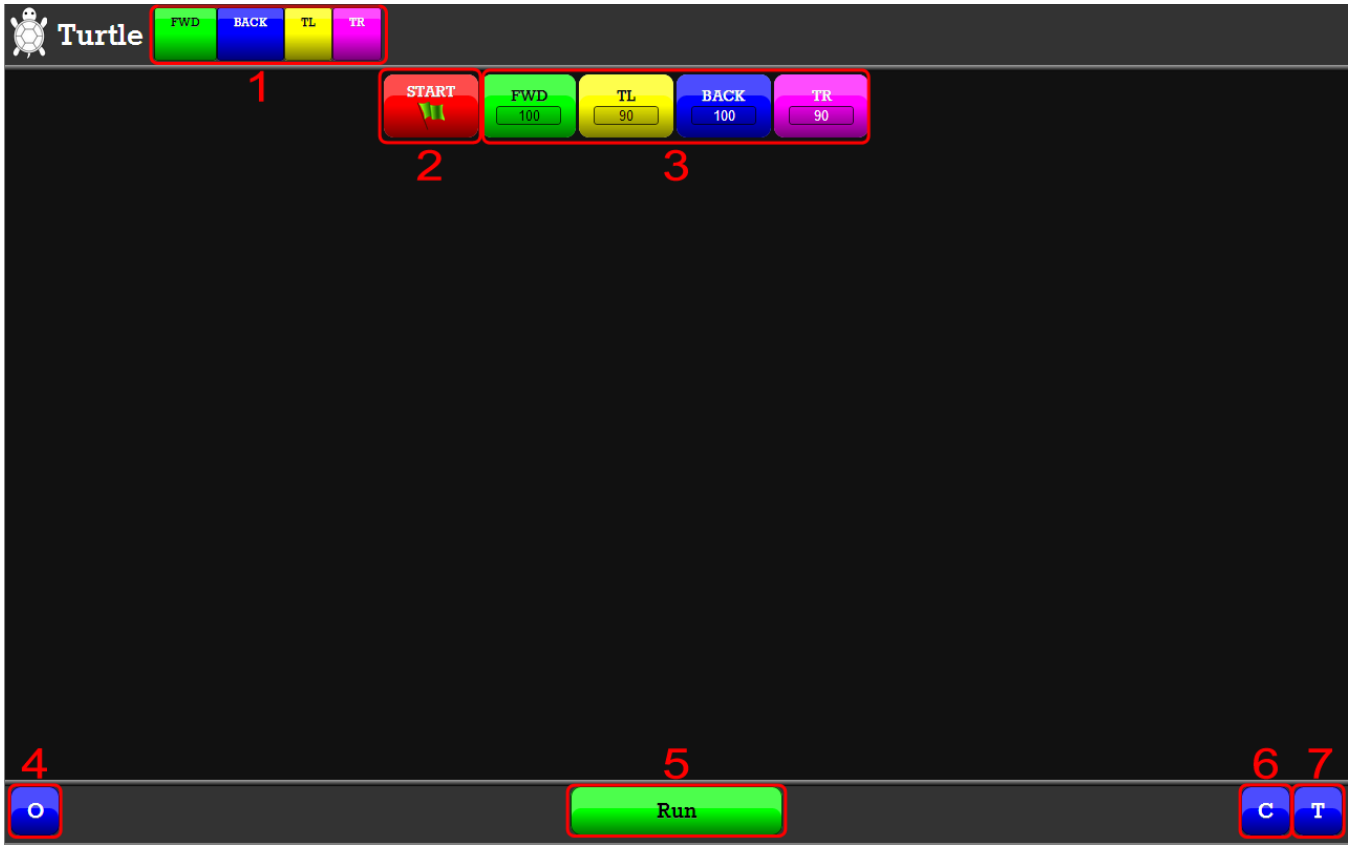

Figure 1: Graphics view.
NB: button 6 has been moved to the left side

1 This is the available *codeblocks* for your *program*.

2 The *startblock*. This is a special kind of *codeblock*. It may not be moved or removed, and must always be the first *codeblock* in the *program*.

3 The *program*. This part consists of one of more *codeblocks*, and are the part which tells the robot what to do. An explaination of the different codeblocks can be seen in section 1.3.1 and section 1.3.2.

4 The *Options* menu.

5 The *Run* button. Pressing this button run the *program*.

6 The *Clear* button. This button will remove all blocks except the *startblock*.

7 The *Input mode* button. This button will change the input view. If you are working in the *graphics view* it will change to the *textual view*, and vise versa.

1.2 Views

The header/programming area in the application has three different views, which all serve a different purpose. The view shown in figure 1 is the *graphics view*, here programming is done by drag'n'drop as explained. The second view is the *textual view*, as seen in figure 2. The textual view

provides the textual counterpart to the graphics view, and when used with block programming you are able to switch between the two without losing your program. The view also supports an alternative programming language for more experienced users. The textual view is considered harder to understand, and it is therefore recommended that most people start with the graphics view. The final view is the *simulator view*. This view takes on many different visual appearances, depending on the state of the application and different options. In its most basic form it will look like figure 5, but depending on what has been done previously in the application it may have changed. A list of all the different simulator appearances can be seen in section 1.4.

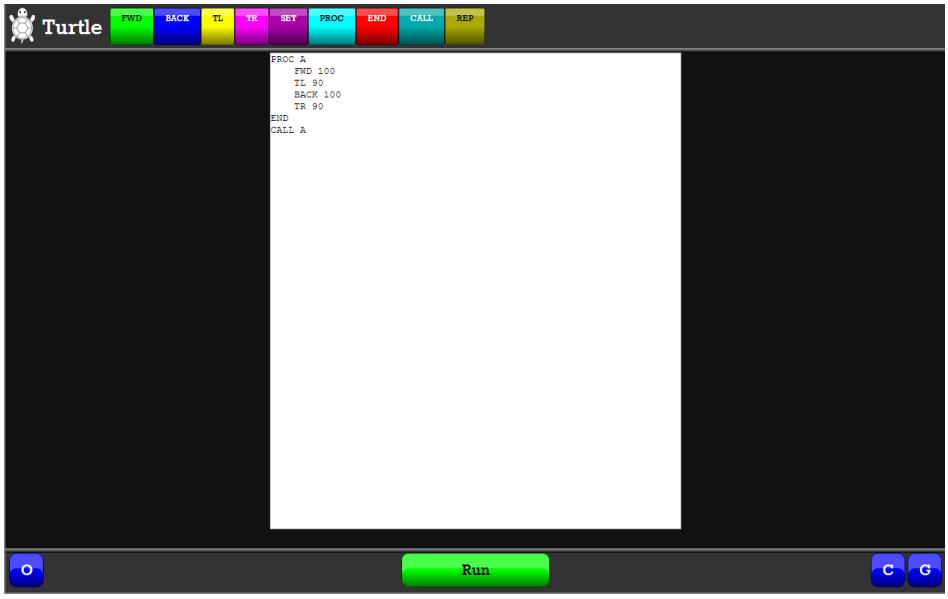

Figure 2: The textual view when extended mode is on.

1.3 Programming

1.3.1 Normal mode

When starting the application for the first time, it will start in what is called *normal mode* showing the *graphics view*. In *normal mode* you have four *codeblocks* available to you:

FWD - Forward **BACK - Backward**
TL - Turn Left **TR - Turn Right**

FWD and **BACK** makes the robot move forward and backward respectively. The input value determines how far the robot will go. **TL** and **TR** makes the robot turn either left of right. The input value determines how many degrees it will turn.

In figure 1 you can see an example where all these four *codeblocks* are used. The resulting picture from this *program* can be seen in figure 3. For examples of textual program please see section 1.6.

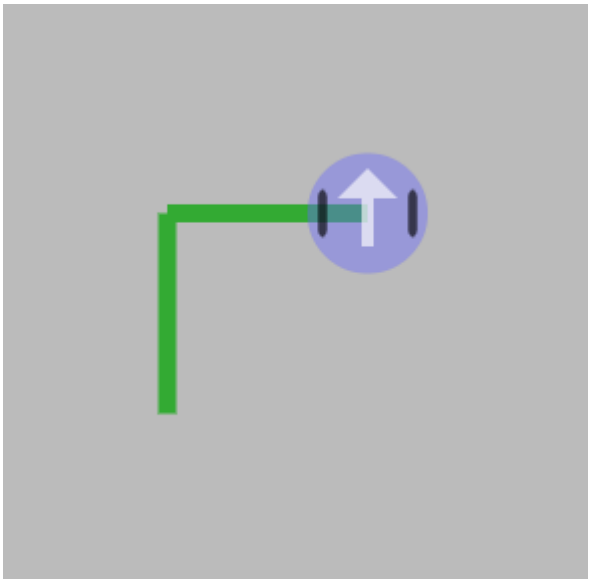

Figure 3: The result from running the program in figure 1

1.3.2 Extended mode

In *extended mode* you will see five new *codeblocks* appear as available (figure 4). These blocks are related to programs that need *variables*, *procedures* and *loops*. The usage of these blocks and how they work is beyond the scope of this introduction, but it should be easy to figure out for anyone with previous experience with programming.

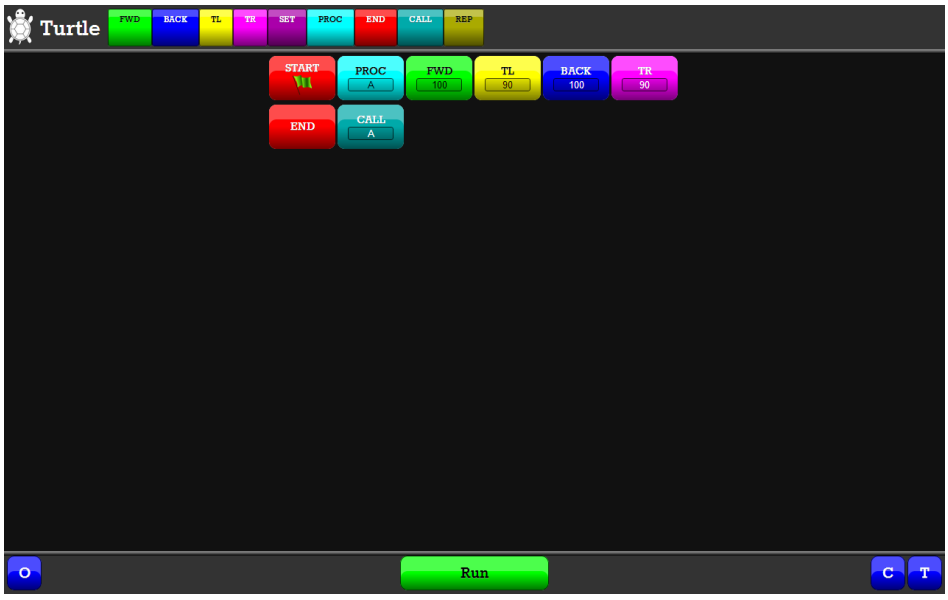

Figure 4: The main view when extended mode is on.

1.3.3 Javascript mode

The javascript mode is a purely textual mode, and only recommended for those who have mastered using the extended mode in textual form, or have previous knowledge about programming. This mode utilized an architectural artifact within the application to allow for writing pure and real

javascript code directly into the application. To use the javascript mode; change your view to the textual view and add `//#javascript` as the first line and you are good to go.

1.4 The simulator

As mentioned previously the simulator has many different appearances which depends on the current state of the application. Here is a list of the current simulator modes:

1.4.1 Always simulator

With the *always simulator* mode the simulator is shown regardless of the state of your application.

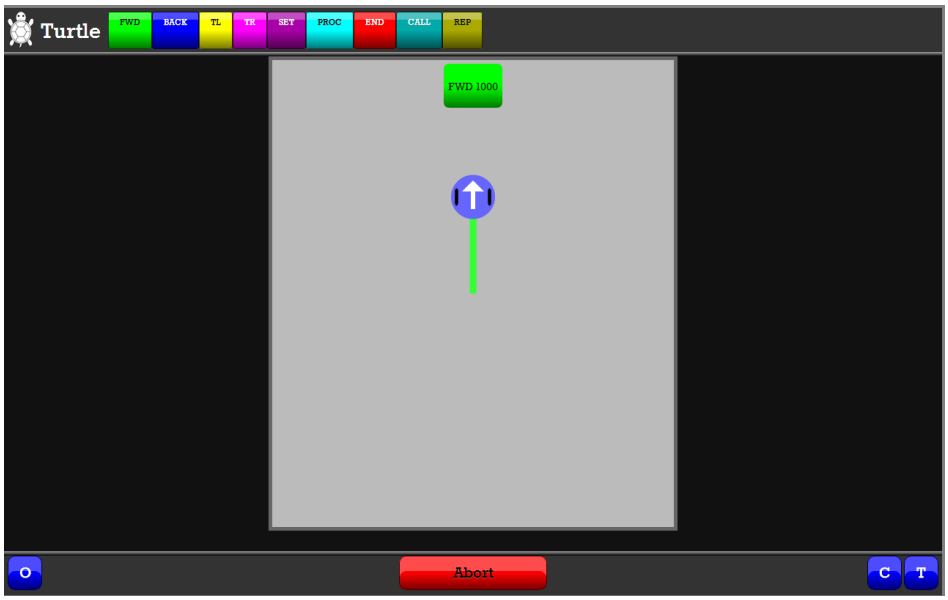

Figure 5: The simulator view.

1.4.2 Bluetooth adaptive headsup display

The *bluetooth adaptive headsup display* changes its appearances depending on wether the application is connected to a robot via bluetooth or not. In the case where it is not connected to a robot it will show the default simulator view, as seen in figure 5. When the application is connected it will just show the current task of the robot, as seen in figure 6

Figure 6: The bluetooth headsup display.

1.4.3 Bluetooth adaptive blackout

The *bluetooth adaptive blackout display* has similar characteristics to the bluetooth adaptive heasdup display, but even removes the heasdup display. While the program is runnin all focus will be at the robot as the screen will at this point not show any useful information.

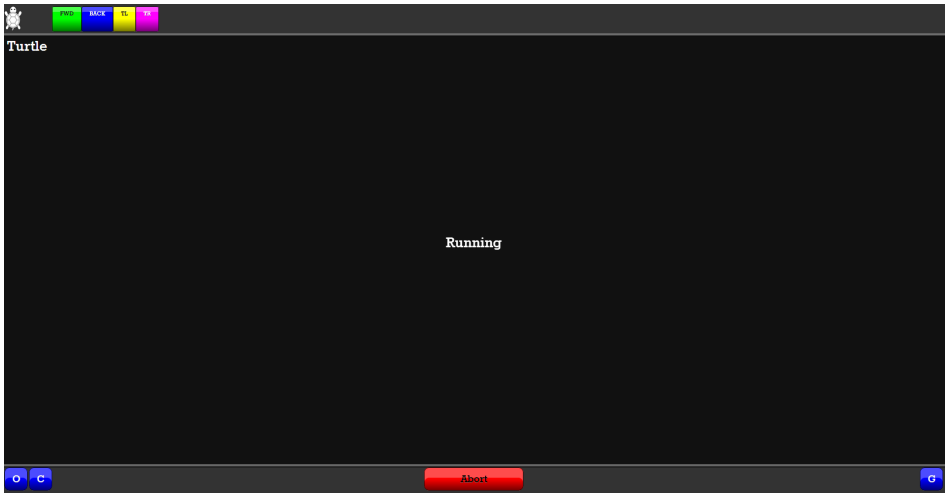

Figure 7: The bluetooth blackout display.

1.4.4 Just trace simulator

This mode is does not send any information to the robot even if it is connected. Instead this mode can be used for rapid debugging of your program. This mode skips all fancy animations that other modes may have, and draws the a trace of the robots movement as fast as it can. The end image will be very similar to those of the normal simulator.

1.5 Options

The *options panel* allows you to change the behaviour of the application at runtime, change the inital setup, and connect to an external robot. If the yellow *Connect* button does not appear then

check the bluetooth settings on your device, as this button only appears if the application detects a device with bluetooth.

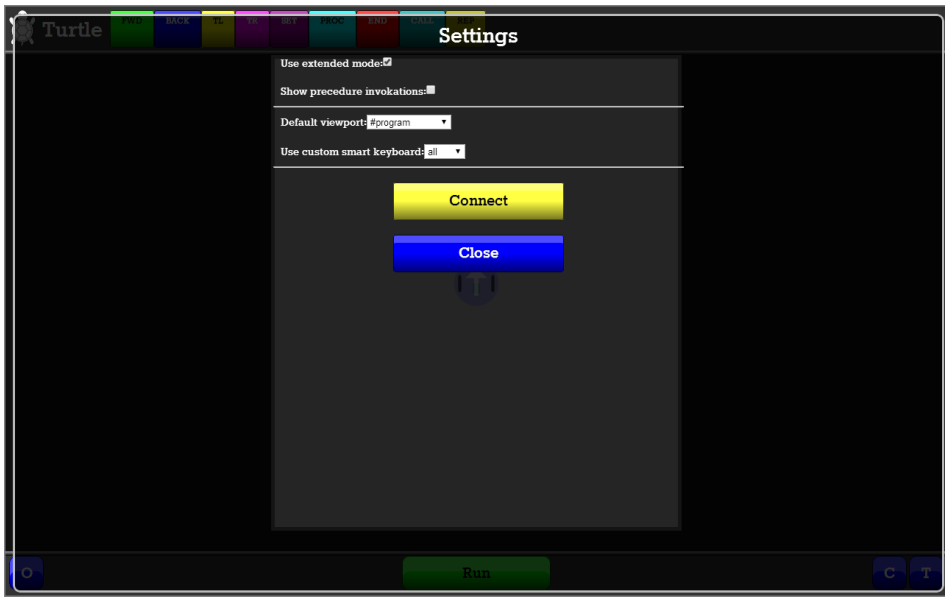

Figure 8: The options view.

The following attributes are exposed through the options menu:

Attribute	Description	Default
Use extended mode	Allows the user to toggle between *normal* and *extended* mode.	unchecked(normal mode)
Show procedure invokations	Determines wether or not to show the blocks from the extended mode during simulation.	unchecked(dont show)
Default view	Allows the user to set the *textual view* as default.	#program(graphics view)
Smart keyboard	Turns on different modes for the on screen keyboard.	none(dont show)
Simulator selection	Allows the user to change which simulator mode that are used.	bluetooth adaptive headsup

1.5.1 Connecting to a robot

To connect to an external robot; click on the *Connect* button and wait for a list of robots to show up on the screen(figure 9), click on the desired robot and wait for the connection to happen. If everything works fine the *Connect* button should turn green, this is an indication that the application is in contact with the robot(figure 10).

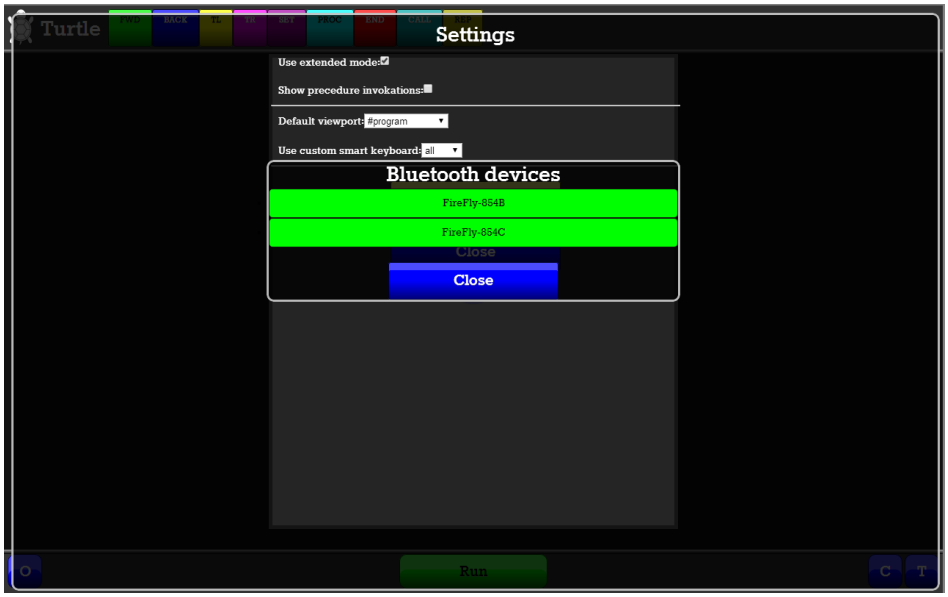

Figure 9: The list of available robots.

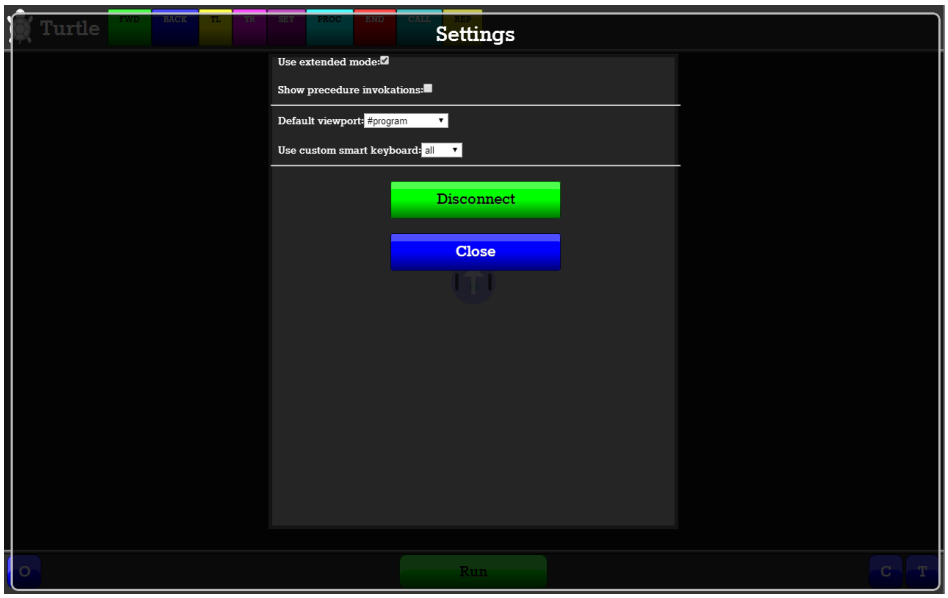

Figure 10: The options view when connected to a robot.

1.5.2 On screen keyboard

This is fallback solution, if the native keyboard doesn't work correctly.

Figure 11: The on screen keyboard for numeric input.

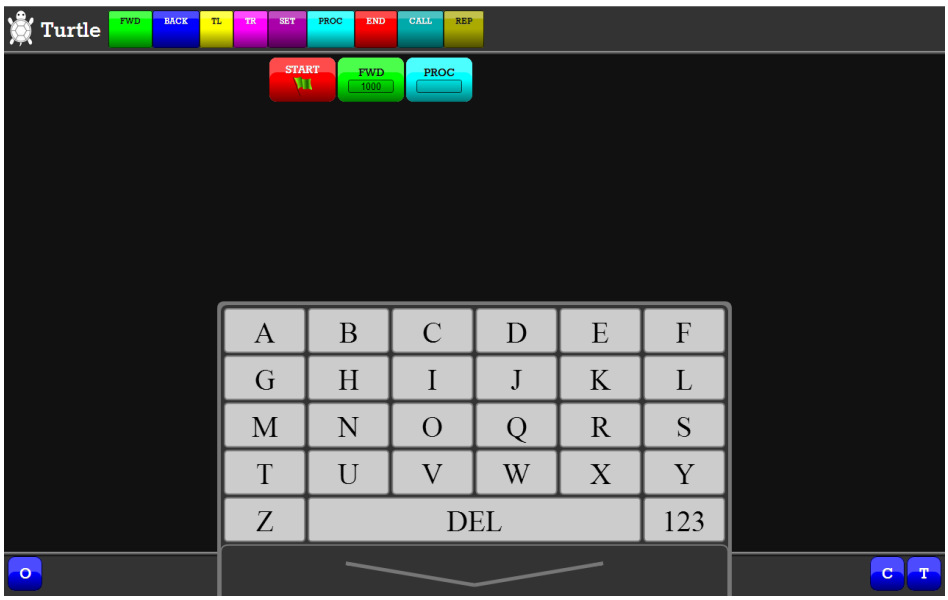

Figure 12: The on screen keyboard for alphanumeric input.

1.6 Textual code examples

All the example below will draw a square with side length of 100, and return to the original starting position and facing the same way as when it started.

1.6.1 Normal mode

```
FWD 100
TL 90
FWD 100
TL 90
FWD 100
TL 90
FWD 100
TL 90
```

Listing 1: Normal mode.

1.6.2 Extended mode

```
REP 4
  FWD 100
  TL 90
END
```

Listing 2: Extended mode using loops.

```
PROC A
  FWD 100
  TL 90
END
REP 4
  CALL A
END
```

Listing 3: Extended mode using procedures and loops.

```
SET NUM 4
SET LENGTH 100
SET ANGLE 90
PROC SEGMENT
    FWD LENGTH
    TL ANGLE
END
REP NUM
    CALL SEGMENT
END
```

Listing 4: Extended mode using procedures loops and variables.

```
var icount = 0;

SET("NUM", 4);var NUM = 4;
SET("LENGTH", 100);var LENGTH = 100;
SET("ANGLE", 90);var ANGLE = 90;

function SEGMENT() {
  icount++;
  if (icount > 30000){return;}
  FWD(LENGTH);

  icount++;
  if (icount > 30000){return;}
  TL(ANGLE);
  END();
}

REP(NUM);
for (var generated_55033 = 0; generated_55033 < NUM;generated_55033++) {
  CALL("SEGMENT");
  SEGMENT();
  END();
}
```

Listing 5: The internal representation of the previous program.

1.6.3 Javascript mode

```
//#javascript
var num = 4;
var length = 100;
var angle = 90;
function segment(){
  FWD(length);
  TL(angle);
}
for (var i = 0; i < num; i++){
  segment();
}
```

Listing 6: Javascript mode