# Teaching Programming by Gamification

Proof-of-concept game demo

## Table of Contents

User manual ........ 2

Controls ........ 2

On foot ........ 2

In Editor ........ 2

Common nodes and combinations ........ 3

Function calls ........ 3

If-else statements and While loops ........ 3

Game world ........ 5

Computer Terminals ........ 5

Programmables ........ 6

Troubleshooting ........ 7

Bugs, glitches & debug console ........ 7

A note on screen resolutions ........ 7

# User manual

This is a short manual to introduce you to the game's systems, controls and user interface.

## Controls

### On foot

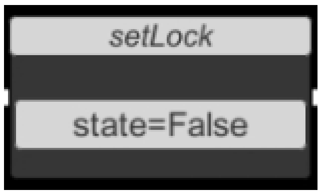
When outside of the code editor, you are able to move your character freely with a first-person perspective. **Use the keys W, A, S, D to move forwards, left, backwards, and right respectively**, and **move your mouse to look around**.

There is no jump ability, which is why you'll have to use some programmable objects to help you navigate the level and tackle any elevations you might encounter.

### In Editor

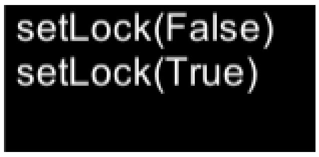
When using the code editor, you are by default placed in "flow chart" mode. This is a visual representation of the program running on this terminal (see: Computer Terminals).

Each program is made of **command nodes** which map to various operations performed by real programming languages.



The figure above shows a function call node. It calls the function named "setLock" with its *state* parameter set to the value of False. Function calls are not the only node type available; other types include creating variables, lists and flow control (if-else statements, loops).

**The programming style and principles used in this editor are largely based on the Python programming language. You can view your code translated into Python syntax by pressing F1.**

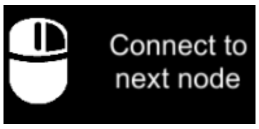


```
setLock(False)
setLock(True)
```

Please be aware that you will not be able to see your code if your Start node isn't connected to anything, or if you do not have any command nodes in your program, as that means there is no code anyway.

You can edit this program in different ways:

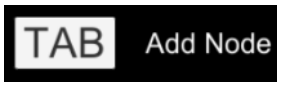
- You can change the order of execution by connecting nodes together
    - Right-click the right side of the node when you see this hint in the bottom-left corner of the screen:

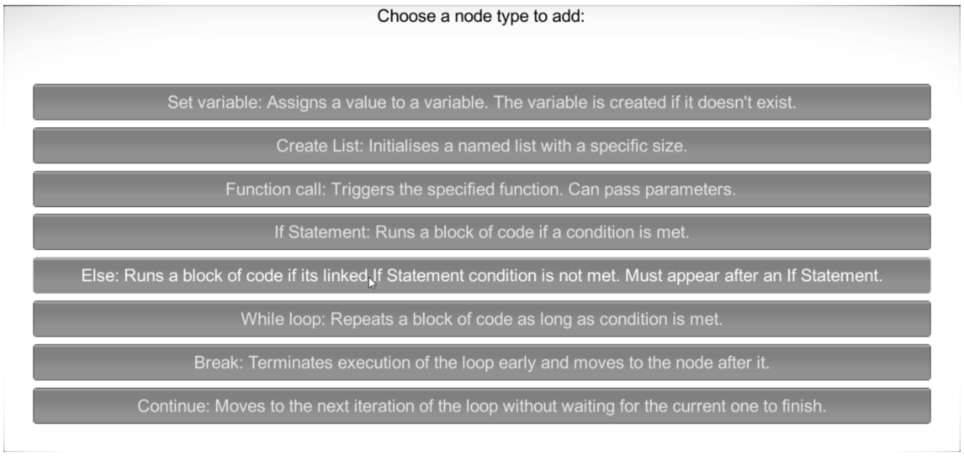
    - Right-click on the node that you want to execute **after** this one.
    - If you want to remove a connection, right-click on the first node, and right-click it again. This will get rid of the link between the two nodes.

- You can edit parameters being passed into a function
  - Double-click with the left mouse button on the parameter bar when you see this hint in the bottom-right:

  - Type in your changes in the parameter text field.
  - When you want to **confirm your changes, hit the Enter key**.
  - If you want to **discard your changes to this parameter, left-click anywhere** outside of the edited text field. **Note that this will revert the parameter's value back to what it was before you started editing**.
- You can add a new command node into your program.
  - Press the TAB key.



  - You will see a list of node types that you can add. Click the one you wish to add.



Choose a node type to add:

Set variable: Assigns a value to a variable. The variable is created if it doesn't exist.

Create List: Initialises a named list with a specific size.

Function call: Triggers the specified function. Can pass parameters.

If Statement: Runs a block of code if a condition is met.

Else: Runs a block of code if its linked If Statement condition is not met. Must appear after an If Statement.

While loop: Repeats a block of code as long as condition is met.

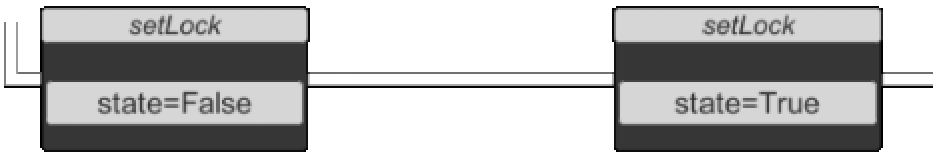
Break: Terminates execution of the loop early and moves to the node after it.

Continue: Moves to the next iteration of the loop without waiting for the current one to finish.

  - If you chose a Function Call node, you'll be asked to choose which function you wish to call. Some of them, like print(text), are shared, while other functions are unique to specific terminals (see: Computer Terminals, Common functions).
    - **Note: There is a minor glitch when you go to add a new node, choose "Function Call" but don't choose which function to call. A blank, invalid node will be added to your editor. You can safely delete it.**

## Common nodes and combinations

### Function calls

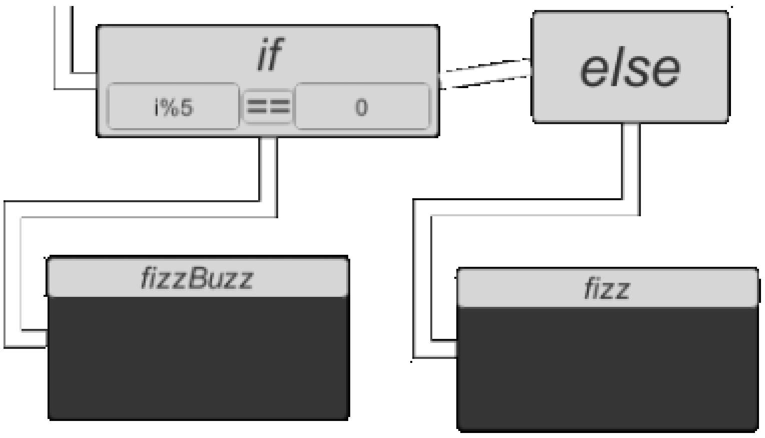


The figure above shows node A, which calls setLock(False), connect to node B, which calls setLock(True). This means that **the program will first run node A, then node B**. The white line represents that order of execution: **the node connected from the right side of the node is the next node**.

### If-else statements and While loops

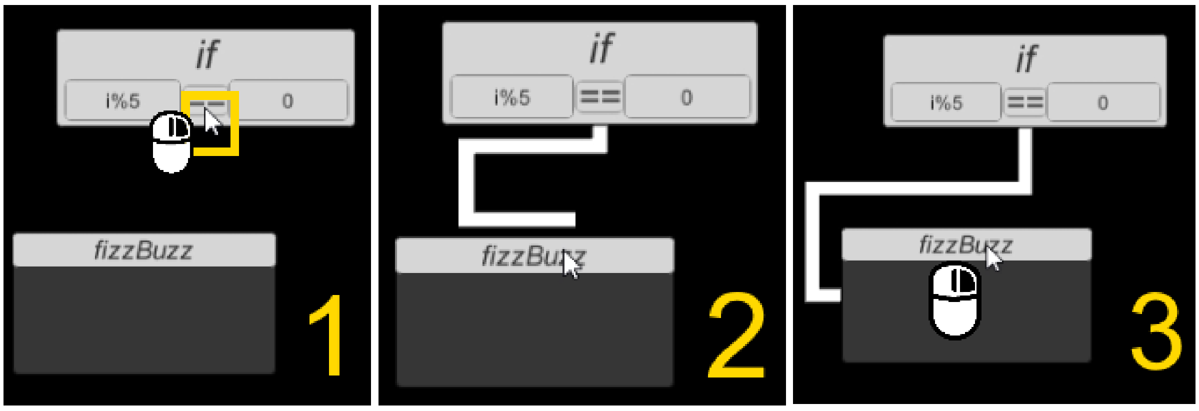
If-else statements let you branch out of the code's original path given a specific condition is true.

Consider these if-else blocks:



Firstly, the if statement checks if the variable named i is divisible by the number 5 (this is done by checking if its modulo 5, ie. the remainder of its division by 5, is equal to 0). **If that's true, it calls fizzBuzz**. **Otherwise, it will jump to the else block, which in turn calls the fizz function**.

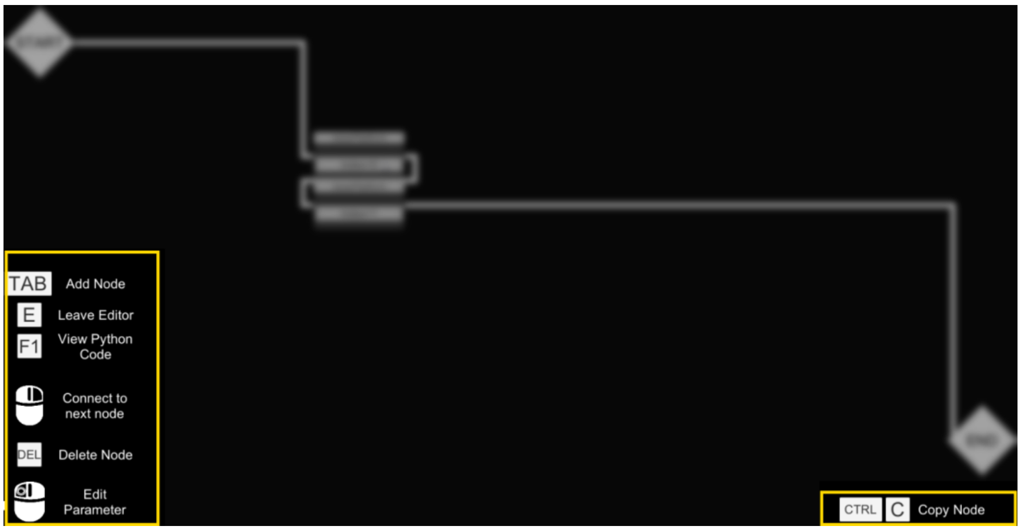
**Note that, in order to start an if-statement or its associated else block, you need to Right-Click the bottom-middle part of the if or else node, and connect it to the first node to be executed in that block.**



**You can have multiple nodes inside an If-Statement or an Else Block. Just connect a next node to this first node, a next node to that one, and so on.**

**<u>Same instructions apply to While loops</u>.**

These are just examples. For a full list of controls in the editor, keep an eye on the bottom left and bottom right side of your screen. It will contain control hints depending on context, showing your available actions.

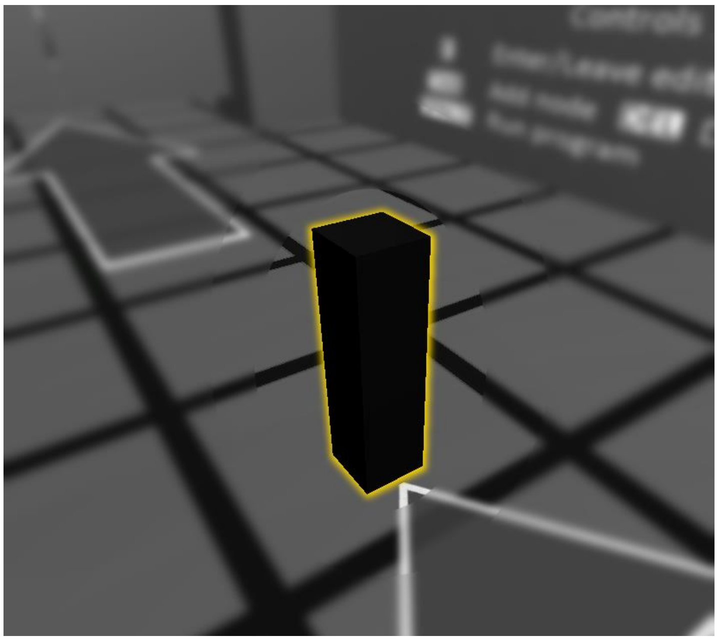


## Game world

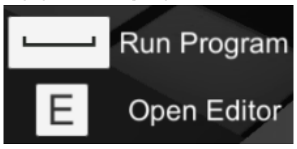
There are multiple objects in the game world that you can interact with.

### Computer Terminals

The small black towers in the level represent computer terminals. Each terminal controls some programmable object(s) and provides some function(s) specific to that (see: Programmables).



When you're within the interaction radius of a terminal, you can choose to **run or edit its program by pressing Spacebar or E, respectively**.



Each terminal has a set tick time. **The tick time is how long it takes to move from one command node to the next**.

### *Common functions*

These are the functions available to every terminal:

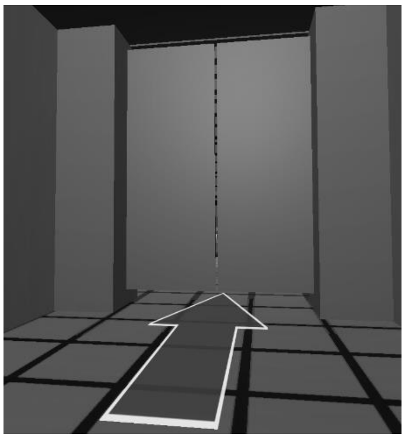
| Function name | Parameters | Description |
|---|---|---|
| print | *text* | Prints *text*, which is either a string literal – ie. text wrapped in double-quotation marks (") – or a variable that holds a string literal as its value. The printed text is stored in the output buffer, which can only be viewed if the terminal is attached to a display. |
| printNewline | | Prints "\n" to the output buffer, effectively adding a new line to it. It is thus equivalent to print("\n") |
| sleep | | Does nothing for one tick. Can be used to make the program wait for some time. |

These are mostly provided for utility and you don't have to use them to progress through the level, as they are likely to have no effect whatsoever (except for the FizzBuzz puzzle, where the print function can be used instead of the fizz, buzz and fizzbuzz functions that issue print calls with the right values).

## Programmables

This section describes the Programmable object types you will encounter in the demo level. Each Programmable type has different ways you can manipulate it and use to your advantage.

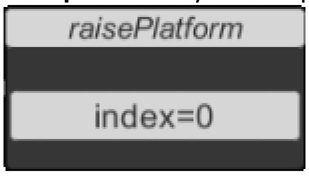
### *Door*

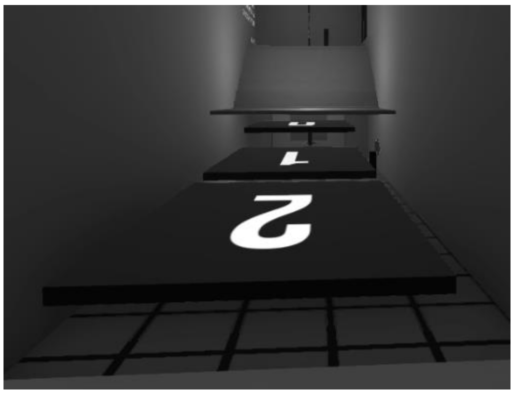


The door is the most basic Programmable type. At its default state, it prevents you from moving past it by remaining closed until you solve its associated programming puzzle. **The condition for unlocking the door varies**; it might be as simple as just issuing a "setLock" function call with the right Boolean value, or it could require you to guide a robot to a button that activates the door.

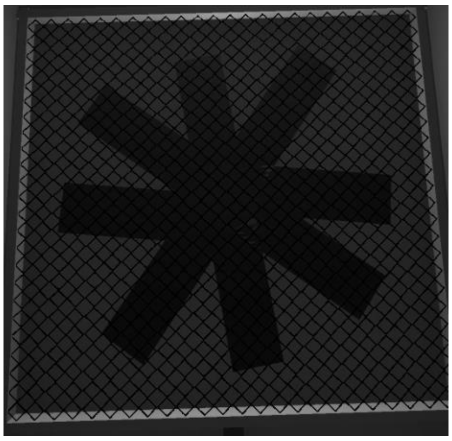
### *Platform*

The platform is a plate that can be **raised or lowered** from inside the program. Since the terminal usually controls a set of platforms, each platform will be assigned an index number. This number is how you refer to them in your program. Calling **raisePlatform** with ***index*=0** will raise the **first platform** by one step, ***index=1*** will raise the **second platform** by one step, and so on.

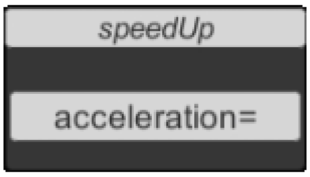




### *Fan*



The fan is able to generate wind that can propel you upwards. This can be helpful when trying to get on top of tall walls. It provides two functions to help you achieve that: speedUp and speedDown. Each takes an *acceleration* or *deceleration* parameter, respectively. Both are numerical values. There are two main approaches to using these: either find the right value to accelerate/decelerate by once, or use a variable to keep accelerating more and more with the use of a loop.



## Troubleshooting

### Bugs, glitches & debug console

As this is a proof-of-concept demo version of a rather complex program, it is inevitable that there may be minor errors & bugs still present in this build. To alleviate some issues they may potentially cause, the game includes a simple debug console. **You can enable it by pressing the Backquote (`) key**. Below are the available commands:

| Command | Description |
|---|---|
| quit | Exits the game to desktop. |
| restart | Reloads the level and starts from the first spawn point. |
| stopall | Stops execution of all programs in the level. Useful if you're unable to exit an infinite loop or the execution model runs into a glitch. |
| section *number* | Teleports to puzzle marked by *number*, where *number* is in the range 0 to 5.<br>0: Door 1<br>1: Platforms 1<br>2: Platforms 2<br>3: FizzBuzz<br>4: Fan Launch<br>5: Door 2 – Robot<br><br>This is useful if you had to restart the level due to a bug and don't wish to go through the previous puzzles again. **Unless that is the case, please do not use this to skip puzzles. Try solving them yourself, or indicate that you did not finish the level in the survey. Otherwise, the results of the study will not be accurate.** |

### A note on screen resolutions

Depending on your screen resolution, the game's user interface might not scale up well. The game was tested on resolutions 1280x720, 1600x900 and 1920x1080 during development. If your computer display is above that and is causing issues, try using one of those resolutions. In theory, the issues should not be game-breaking, but text might appear either too blurry or oversized, etc.