# Proto Simulator User Manual

Jacob Beal

Last Revision: May 25, 2008

This is the user manual for the 2nd generation Proto simulator. For installation instructions, see the **Proto Installation Guide**. For a reference of commonly used simulator and language commands, see the **Proto Quick Start**. For a tutorial on the Proto language, see the document **Thinking In Proto**. For a reference to the Proto language, see the **Proto Language Reference**. For information on how to extend the functionality of the simulator, see the **Proto Simulator Developer Reference**.

This manual is organized by functional modules within the simulator. For each module, there is a brief description of the purpose and behavior of the module, followed by a list of the command-line arguments, keys, and colors used by that module. Some modules conflict in arguments and keys. Ordinary keys are case sensitive (i.e. **c** and **C** are different); control keys are case insensitive.

When arguments are described, an optional part of an argument is surrounded by square brackets: `[]`. Argument names always begin with a dash.

## 1 Credits for Proto

The Proto language was developed in partnership by Jonathan Bachrach and Jacob Beal. Jonathan Bachrach is the primary programmer for the Proto compiler, kernel, and 1st generation simulator. Jacob Beal is the primary programmer for the 2nd generation simulator.

Additional programming by: Joshua Horowitz, Omari Stephens, Mark Tobenkin, Dan Vickery

## 2 Palette Files

All colors used by the simulator are user-controllable: their default values can be overridden with the use of palette files.

> **Argument:** `-palette FILE`
> Loads the palette in `FILE`. This command can be invoked multiple times, loading multiple palettes. Palettes load in the order they are specified on the command line, with later entries possibly overriding earlier ones.

Palette files are formatted as follows:

- Whitespace is ignored
- '#' as first non-whitespace character in a line indicates a comment
- One color per line, specified `NAME RED GREEN BLUE [ALPHA]`, where `NAME` is the name of the color, and the others are values between zero and one. If unspecified, `ALPHA` defaults to 1.0, which is solid.

Each specified color replaces only the current value assigned to that color, so multiple palette files can be layered, each changing only a portion of the colors.

**Example Palette File:**

```
# Turn the background a horrid pink
BACKGROUND 1 0.8 0.8 1
  # Make the devices green and the times translucent cyan and white
SIMPLE_BODY 0 1 0
TIME_DISPLAY 0 1 1 0.5
FPS_DISPLAY 1 1 1 0.3
```

# 3 Core Simulator

The core functions of the simulator are to create a simulation, control its progress, and manage the user interface.

If there is precisely one unhandled argument, that argument is taken to be the Proto script to run. If there are no unhandled arguments, the script defaults to `(app)`. If there are multiple unhandled arguments, the simulator gives a warning and uses the last one as the script.

**Argument: `-seed N`**
Use `N` as a random seed, defaults to a value set by the current time.

**Key: q**
Quit the simulator.

**Argument: `-mag N`**
Relative magnification of text displays for each device, default 1.

**Argument: `-i`, Key: i**
Show the ID of each device **(Color: `DEVICE_ID` 1, 0, 0, 0.8)**. Toggled by key.

**Starting, Stopping, and Running:**

**Argument: `-T`, Key: T**
Display simulator time in lower left corner **(Color: `TIME_DISPLAY` 1, 0, 1, 1)** and frames-per-second in lower right corner **(Color: `FPS_DISPLAY` 1, 0, 1, 1)**. Toggled by key.

**Argument: `-step`, Key: s**
Use stepping mode, advancing one step on key **s**

**Key: x**
Execute freely (ending stepping mode).

**Argument: `-stop-after N`**
Terminate after `N` simulated seconds, default infinity.

**Argument: `-throttle`, Key: X**
Throttle simulated time to advance relative to real time (toggled by key). When the simulator cannot keep up, a warning appears in the lower center **(Color: `LAG_WARNING` 1, 0, 0, 1)**

**Argument: `-ratio N`**
Ratio between real time and simulated time when throttling is active, default 1

**Argument: `-s N`**
Set simulated seconds per step, default 0.01/`ratio`

**Key: CTRL-s**
Slow throttled simulator. Each keystroke divides speed by $2^{\frac{1}{4}}$

**Key: CTRL-a**
Accelerate throttled simulator. Each keystroke multiplies speed by $2^{\frac{1}{4}}$

**Key: CTRL-d**
Return throttled simulator to a ratio of 1:1 simulated to real time.

**Display:** The display background is colored **(Color:** `BACKGROUND` **0, 0, 0, 0.5)**.

**Argument:** `-window-name NAME`
Title the simulator window `NAME`, default "Proto Simulator"

**Argument:** `-f`**, Key: f**
Full screen display (toggled by key)

**Argument:** `-headless`
Run without a display or user interface; default **FALSE** unless compiled without OpenGL. When headless, `-stop-after` must be specified.

**Key: CTRL-x**
Zoom in. *Not working!* Note: conflicts with the key for transmission backoff.

**Key: CTRL-z**
Zoom out. *Not working!*

**Key: z**
Reset display to initial view.

**Key: ARROW KEYS**
Shift simulation display in the direction of the arrow, in the simulation's coordinate system.

**Mouse: LEFT DRAG**
Rotate display.

**Mouse: RIGHT DRAG**
Zoom display; toward center zooms out, away zooms in.

**Selection:** Selected devices are indicated with a thin ring **(Color:** `DEVICE_SELECTED` **0.5, 0.5, 0.5, 0.8)** four times the device's body diameter,

**Mouse: LEFT CLICK**
Select the devices clicked on.

**Mouse: RIGHT CLICK**
Select, then print the state of the selected device(s) to standard out.

**Mouse: SHIFT LEFT DRAG**
Toggle the selection of all devices in a rectangular display region. The area to be selected is colored **(Color:** `DRAG_SELECTION` **1, 1, 0, 0.5)**.

**Mouse: SHIFT RIGHT DRAG**
Move selected devices.

**Key: U**
Unselect all devices.

**Code**

**Key: l**
Recompile and reload the current script, injecting at the selected devices.

**Argument:** `-proto-directory DIR`
Directory where the simulator runs; defaults to the directory where the command was invoked.

**Argument:** `-path PATH`
Set the order of directories to be searched for Proto programs. The `PATH` is a list of directories separated by semicolons, default is `.;../usr;../lib;../lib/core`

**Argument:** `-k`**, Key: k**
Display the current script across the middle of the window (toggled by key). *Not working!*

**Argument:** `-show-script-version`**, Key: j**
Show what version of the script is loaded at each device **(Color:** `DEVICE_SCRIPT` **1, 0, 0, 0.8)**. Toggled by key.

**Argument:** `-v`**, Key: n**
Show value computed at each device **(Color:** `DEVICE_VALUE` **0.5, 0.5, 1, 0.8)**. Toggled by key.

**Argument:** `-sv`**, Key: v**
When device outputs a 2- or 3-tuple, display it as a vector (toggled by key). The vector is interpreted as meters, drawn as a line **(Color:** `VECTOR_BODY` **0, 0, 1, 0.8)** with a differently colored tip to indicate direction **(Color:** `VECTOR_TIP` **1, 0, 1, 0.8)**.

# 4 Debugging

The debugger logs chosen types of activity to standard out. To prevent complete overload, debugging information flows only when global debug mode is on and then only from devices selected for debugging. In that case, debugging information from designated categories prints to standard out from the devices selected for debugging. Devices currently logging debugging information are shown by a filled disc twice the size of the device **(Color:** `DEVICE_DEBUG` **1, 0.8, 0.8, 0.5)**.

For example, to get a trace of networking activity in the kernel, add the command line arguments `-g -debug-kernel`, then select the desired devices and hit **D**.

**Key: D**
Toggle selection for debugging on selected devices.

**Argument:** `-g`**, Key: d**
Global debug mode (toggled by key)

**Argument:** `-t`**, Key: a**
When debugging, post trace of evaluation in kernel of debug devices (toggled by key).

**Argument:** `-debug-kernel`
When debugging, post trace of networking activity in kernel

**Argument:** `-debug-script`
When debugging, post trace of viral code distribution in in kernel

# 5 State "Dumping"

The simulator can save "dumps" that contain a snapshot of the current state of every device in the simulator. These snapshots produce Matlab-readable files. The first line is a Matlab comment containing the names of all the state fields to be dumped; each subsequent line contains the state of a device, one number per field, with fields separated by white-space.

The first three fields are always `UID TICKS TIME`, giving device ID, number of rounds executed, and current time estimate.

Each simulator module has two arguments, `-D`*`module`* and `-ND`*`module`* that assert that state for the module must or must not be included in dumps, respectively. Some modules also have an argument `-D`*`module`*`-mask` that allows specific elements of its state to be included and excluded. The first element is enabled by the first bit, the second by the second bit, and so on; these masks generally default to -1, meaning that all elements will be included. These dumping arguments also control what information is printed when a device state is printed to standard out, though that is printed more verbosely.

When a snapshot is taken, the screen flashes for 1/10 of a second **(Color: `PHOTO_FLASH` 1, 1, 1, 1)**. Snapshot data is recorded in files named {`STEM`}{`SIM TIME`}`.log` for automatic dumps and {`STEM`}{`REAL TIME`}-{`SIM TIME`}`.log` for dumps triggered by the user.

**Argument:** `-no-dump-snaps`
Don't show a flash on dumps.

**Key: Z**
Dump a snapshot immediately.

**Argument:** `-D`**, Key: 9**
Enable periodic dumping of data. Disabled by default.

**Key: 0**
Disable periodic dumping of data.

**Argument:** `-dump-after N`
Start periodic dumping at time `N`, default 0.

**Argument:** `-dump-period N`
During periodic dumping, snapshot once every `N` seconds, default 1.

**Argument:** `-dump-dir DIR`
Store snapshot files in directory `DIR`, default `dumps/`. If the directory does not exist, it will be created.

**Argument:** `-dump-stem STEM`
Start snapshot file names with `STEM`, default `dump`.

**Positive Argument:** `-Dall`**, Negative Argument:** `-NDall`
By default, are all modules included in dumps? Default **TRUE**.

**Positive Argument:** `-Dhood`**, Negative Argument:** `-NDhood`
Include neighborhood data in printed state, but not snapshot files (due to variability of size).

**Argument:** `-probe-dump-filter`**, Key: 8**
"Probe filtering" an unimplemented feature from the 1st generation simulator. *Not working!*

# 6 Device Distribution

Simulations are created with `n` devices distributed through a bounded 2D or 3D volume according to one of several distribution rules.

**Argument:** `-n N`
Number of devices.

**Argument:** `-3d`
Use a 3D distribution; default is 2D.

**Argument:** `-dim X [Y [Z]]`
Set bounds of initial device distribution, defaults are `X` = 132, `Y` = 100, and `Z` = 0 (`Z` = 40 for 3D). Specifying a `Z` argument implies a 3D distribution.

By default, devices are distributed uniformly randomly through the initial volume.

**Argument:** `-fixedpt X Y [Z]`
Set the position of the first device (Z defaults to 0); all others are distributed uniformly randomly. If used more times, times the argument sets the position of more devices (e.g. the second use sets the position of the second device).

**Argument:** `-grid`
Distribute devices in a grid that evenly fills the initial volume. If the grid does not divide evenly, the rightmost row/plane will be incomplete.

**Argument:** `-grideps EPSILON`
Distribute devices in a grid as for `-grid`, but shift each device randomly in the range $[-\epsilon/2, \epsilon/2]$ for each coordinate.

**Argument:** `-xgrid`
Distribute X on a grid and Y randomly. If 3D, Z is distributed on a grid as well.

# 7 Device Time

Each device tracks its estimated time independently, executing the Proto program at regular intervals. The relationship between device time and simulator time may vary, however.

**Argument:** `-sync`
Synchronized execution of devices: no variation in rate or phase.

**Argument:** `-desired-period P`
Base number of device seconds between executions, default 1.

**Argument:** `-desired-period-variance PV`
Device periods are distributed randomly in the range $[\mathtt{P} - \mathtt{PV}, \mathtt{P} + \mathtt{PV}]$, defaulting $\mathtt{PV} = 0$.

**Argument:** `-desired-ratio R`
Base ratio between device time and simulator time, default 1.

**Argument:** `-desired-ratio-variance RV`
Device/simulator time ratios are distributed randomly in the range $[\mathtt{R} - \mathtt{RV}, \mathtt{R} + \mathtt{RV}]$, defaulting $\mathtt{RV} = 0$.

# 8 Unit Disc Radio Communication

Currently, the only communication model supported by the simulator is unit disc radio communication. In this model, any pair of devices within a fixed radius $r$ can communicate. While unit disc is only a coarse approximation of real radio communication, it is a useful first approximation.

To speed up computation of neighbors, the devices are distributed into a grid of cells $r$ in diameter. In order to find its neighbors, a device thus needs only to search through adjacent cells.

The kernel has the option to decrease frequency of transmissions when nothing is changing in a node's outputs. Ordinarily it transmits every round, but when radio backoff is enabled, it increments its backoff level $b$ each transmission with unchanged values, setting the number of rounds between transmissions $R$ to

$$R = \lceil 1.6^b \rceil$$

The backoff level is capped at 11, which gives $R = 176$. When values change, the backoff level goes back to $b = 0$ immediately.

**Positive Argument: `-Dradio`, Negative Argument: `-NDradio`**
Include radio data in dumps? At present, however, no fields are dumped.

**Argument: `-debug-radio`**
When debugging, post neighbor decision-making process and display cell ID **(Color: `RADIO_CELL_INFO` 0, 1, 1, 0.8)**.

**Argument: `-r N`**
Transmission range for radio, default 15.

**Argument: `-ns N`**
Set transmission range to get an expected neighborhood size of `N`. Overrides `-r`.

**Argument: `-txerr N`**
Probability of failure on message transmit, default 0.

**Argument: `-rxerr N`**
Probability of failure on message receive, default 0.

**Argument: `-no-motion-pruning`**
Ordinarily, moving devices delete neighbors that go out of range; this suppresses that behavior, requiring them to time out instead.

**Argument: `-radio-backoff`, Key: CTRL-x**
Use exponential backoff of transmission frequency (toggled by key).

### Display

**Argument: `-c`, Key: c**
Display network connections (toggled by key).

**Argument: `-sharp-connections`, `-sharp-neighborhood`, Key: C**
When network connections are displayed, the lines are drawn either thin and sharp **(Color: `NET_CONNECTION_SHARP` 0, 1, 0, 1)** or thick and fuzzy **(Color: `NET_CONNECTION_FUZZY` 0, 1, 0, 0.25)**. There are three modes: either all are fuzzy (default), all are sharp, or lines are sharp where they connect to selected devices. The **C** key cycles through the three display modes.

**Argument: `-lc`, Key: CTRL-c**
Display "logical connectivity"—connections that are listed in a device's neighborhood, whether or not they currently exist. **(Color: `NET_CONNECTION_LOGICAL` 0.5, 0.5, 1, 0.8)**

**Argument:** `-show-radio`**, Key: r**
Display a ring around each device at its radio range. **(Color:** `RADIO_RANGE_RING` **0.25, 0.25, 0.25, 0.8)**

**Argument:** `-show-radio-backoff`**, Key: S**
Display the current radio backoff level. **(Color:** `RADIO_BACKOFF` **1, 0, 0, 0.8)**

# 9 Simple Dynamics

The "simple dynamics" physics package evolves bodies based on their velocity, does not handle collisions, and allows instantaneous shifts in velocity. The Proto function `(swim` *vector*`)` is used to set the velocity of devices. Simple dynamics is the default physics package.

Bodies are shown as circles **(Color:** `SIMPLE_BODY` **1, 0.25, 0, 0.8)** when movement is enabled, points when it is not.

**Positive Argument:** `-Ddynamics`**, Negative Argument:** `-NDdynamics`
Include dynamics data in dumps? Fields are: `X Y Z V_X V_Y V_Z RADIUS`.

**Argument:** `-Ddynamics-mask MASK`
Set which fields are included in dumps, default all (`MASK` $= -1$).

**Argument:** `-rad N`
Radius of a device body, defaults from the initial distribution bounds: $\sqrt{0.087 * (width * height/n)}$

**Argument:** `-h`**, Key: h**
Show heading of a device with a tick on the body.

**Argument:** `-S N`
Maximum speed of devices, default 100.

**Positive Argument:** `-w`**, Negative Argument:** `-nw`**, Key: w**
Are there walls that repel devices from the edge of the initial distribution bounds? Default is **FALSE**. The repulsive velocity is proportional to the distance into the wall zone, starting 80% of the distance from the center of the bounds.

**Argument:** `-floor`
Include a "floor" at $Z = 0$ that 3D devices cannot move below.

**Argument:** `-m`**, Key: m**
Enable movement (toggled by key).

**Argument:** `-hide-body`**, Key: b**
Do not draw bodies (toggled by key).

# 10 ODE Dynamics

The "ODE dynamics" physics package uses the Open Dynamics Engine[1], a free open-source Newtonian physics simulator. ODE provides a high-quality simulation including collisions and joint physics. The Proto function `(swim` *vector*`)` is used to set the desired velocity of devices, which then accelerate to try to maintain that velocity.

Physics is always run in three dimensions, so "2D" means gravity and a floor, while "3D" means no gravity and no floor. When there is a floor, it is set such that the center of a device is at $Z = 0$ when the device is resting on the floor.

Device bodies are cubes **(Color: `ODE_BOT` 1, 0, 0, 0.7)**, with wire-box edges **(Color: `ODE_EDGES` 0, 0, 1, 1)**. When a device is selected, its color changes **(Color: `ODE_SELECTED` 0, 0.7, 1, 1)**. It is possible for bodies in ODE to disable evolution, typically when a body is at rest and not interacting with other bodies. If this happens, the color of the body changes **(Color: `ODE_DISABLED` 0.8, 0, 0, 0.7)**.

Walls are boxes; when visible they are **(Color: `ODE_WALL` 1, 0, 0, 0.1)** with wire-box edges the same color as device bodies. Walls, when present, are 10 meters thick, centered on the X-Y edges of the initial distribution bounds, and extend in Z ten times the Y dimension of the pen. The corners have a 45-degree wall 20 meters thick that smooths the corners of the pen.

**Argument: `-ODE`**
Use ODE dynamics for physics.

**Positive Argument: `-Ddynamics`, Negative Argument: `-NDdynamics`**
Include dynamics data in dumps? Fields are: `X Y Z V_X V_Y V_Z Q_0 Q_1 Q_2 Q_3 W_X W_Y W_Z`, where `Q_`$i$ is the quaternion specifying orientation and `W_`$i$ is the angular velocity.

**Argument: `-rad N`**
Radius of a device body, defaults from the initial distribution bounds: $\sqrt{0.087 * (width * height/n)}$

**Argument: `-density N`**
Density of device bodies, which determines mass. Default is 0.0625.

**Argument: `-S N`**
Maximum speed of devices, default 100.

**Argument: `-m`, Key: m**
Enable movement (toggled by key).

**Argument: `-hide-body`, Key: b**
Do not draw bodies (toggled by key).

**Positive Argument: `-w`, Negative Argument: `-nw`, Key: w**
Are there walls that pen devices into the initial distribution bounds? Default is **FALSE**.

**Argument: `-substep`**
Size of physics "microsteps" used by ODE, default 0.001. If too large, the simulation becomes unstable and devices will fly off the screen.

**Argument: `-draw-walls`**
Display the walls, when there are walls.

**Argument: `-inescapable`**
If any device escapes the walls (when there are walls), put it back inside using the starting distribution. If the starting distribution cannot supply another location, kill the device instead.

**Argument: `-rainbow-bots`**
Set device color by map device IDs onto hue, full saturation and value, alpha=0.7.

# 11 Debugging I/O

This module gives simple sensors and actuators useful for debugging in the simulator. The sensors are three boolean "user sensors" read by the Proto function `(sense` $n$`)`, where $n$ is 1, 2, or 3. When **TRUE**, each sensor displays a disc four times the device size, colored **(Color: `USER_SENSOR_1` 1.0, 0.5, 0, 0.8)**, **(Color: `USER_SENSOR_2` 0.5, 0, 1, 0.8)**, or **(Color: `USER_SENSOR_3` 1.0, 0, 0.5, 0.8)** respectively.

The actuators are red, green, and blue LEDs, set by Proto functions (`red` *value*), (`green` *value*), and (`blue` *value*), respectively. The LEDs may be shown independently as discs the size of the device body, with colors **(Color: `RED_LED` 1, 0, 0, 0.8)**, **(Color: `GREEN_LED` 0, 1, 0, 0.8)**, **(Color: `BLUE_LED` 0, 0, 1, 0.8)**. Independent LEDs normally show their value both in intensity of color and $Z$ displacement of the LED disc above the device. LEDs may also be displayed mixed together into one double-sized disc with base color **(Color: `RGB_LED` 1, 1, 1, 0.8)**, where the base color is scaled by LED values.

There are also three probes, which can expose intermediate values from evaluation. They display the values as a tuple above each device in the positive $Y$ direction **(Color: `DEVICE_PROBES` 0, 1, 0, 0.8)**.

**Positive Argument: `-Ddebug`, Negative Argument: `-NDdebug`**
Include debug module data in dumps? Fields are: `USER1 USER2 USER3 RED_LED GREEN_LED BLUE_LED`.

**Argument: `-Ddebug-mask MASK`**
Set which fields are included in dumps, default all (`MASK` $= -1$). Fields start at `0x2`, not `0x1` (e.g. `RED_LED` is controlled by `0x10`) due to a removed field.

**Argument: `-probes N`, Key: p**
Display the first `N` of the three probes. The `p` key cycles through how many are shown.

**Argument: `-l`, Key: L**
Display LEDs (toggled by key).

**Argument: `-led-ghost`, Key: 1**
LED value controls transparency too, with 0 completely transparent and 1 completely opaque. Toggled by key.

**Argument: `-led-flat`, Key: 2**
Do not use displacement in $Z$ to show LED value. Toggled by key.

**Argument: `-led-stacking N`, Key: 3**
Set LED stacking to one of three modes. Mode 0 stacks LEDs directly, in order red, green, blue bottom to top. Mode 1 sets heights independently, plus an offset of 1 meter for green and 2 meters for blue. Mode 2 sets heights independently. The `3` key cycles through the modes.

**Argument: `-led-blend`, Key: 4**
Show the LEDs as an RGB blend rather than independently. Toggled by key.

**Key: t**
Toggle user sensor 1 on selected devices.

**Key: y**
Toggle user sensor 2 on selected devices.

**Key: u**
Toggle user sensor 3 on selected devices.

# 12 Simple Life Cycle

This module allows devices to duplicate themselves or suicide. Duplication rate is limited by a cloning timer: devices can only duplicate when the timer expires.

**Positive Argument: `-Dclone`, Negative Argument: `-NDclone`**
Include life cycle data in dumps? Fields are: `CLONETIME` (the time remaining before a clone can be made).

**Argument:** `-clone-delay N`
Minimum number of seconds between duplications, default 1.

**Key: B**
Clone selected devices.

**Key: K**
Kill selected devices.

# 13 Mote I/O

This module represents the collection of sensor and actuator hardware that we have used on Mica2 Motes at various times. There is one actuator (a speaker), four sensors (a light sensor, microphone, temperature sensor, and short sensor), and two controls (a button and a slider). Of these, the 2nd generation simulator only implements the button, which is drawn as a four-times radius disc **(Color:** `BUTTON_COLOR` **0, 1, 0.5, 0.8)**.

**Argument:** `-mote-io`
Use the Mode I/O module.

**Positive Argument:** `-Dmoteio`**, Negative Argument:** `-NDmoteio`
Include Mote I/O data in dumps? Fields are: `SOUND TEMP BUTTON`.

**Key: N**
Toggle button on selected devices.

# 14 Other Modules

There are a number of other modules in the 2nd generation simulator which either have no controls or are not yet properly implemented.

**Perfect Localizer** The only localizer implemented is a perfect localizer that supplies error-free device coordinates and velocity in global coordinates. It has no controls.

**Mote-link** The mote-link allows the simulator to interface with a network of Mica2 Motes, creating a mixed network of real and simulated devices. It is not yet implemented for the 2nd generation simulator. When it is, it will be invoked with the argument `-motelink`, and real devices will be drawn with **(Color:** `MOTE_BODY` **1, 0, 1, 0.8)**.

**Platforms** There are no 2nd generation versions of kernels for the mote, swarm, or topobo platforms.

# References

[1] R. Smith, G. Carlton, F. Condello, N. Lin, M. C. Martin, T. Schmidt, K. Slipchenko, J. Smith, V. Macagon, A. D. Moss, E. de Vries, N. Waddoups, and D. Whittaker. Open Dynamics Engine (version 0.7). `http://www.ode.org`, 2001 to 2006.

# A Keyboard Assignments:

Gray indicates reserved or unusable keys. Many keys are rendered unusable by the peculiarities of GLUT, including its inconsistent implementation across platforms. Blue is allocated keys, pink is non-functional or conflicting keys, and green is keys reserved for import of more 1st generation simulator functions.

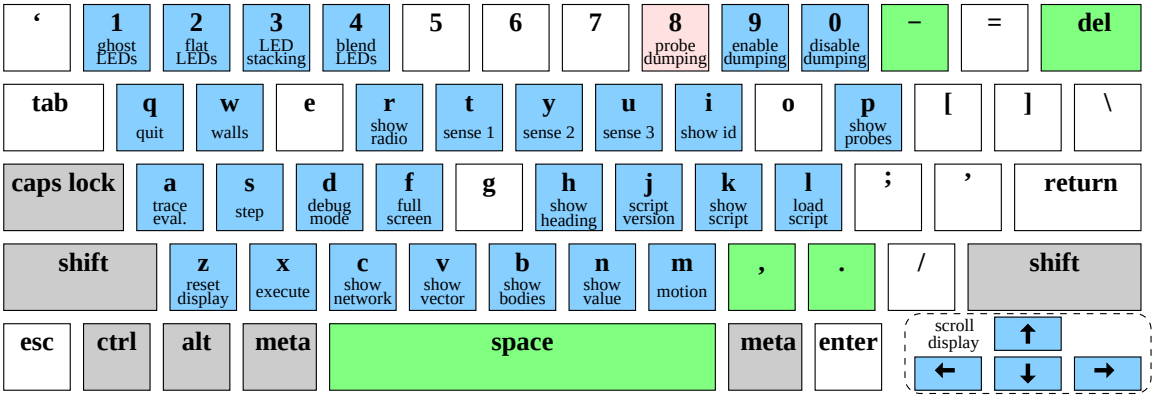


Figure 1: Lower case keyboard assignments

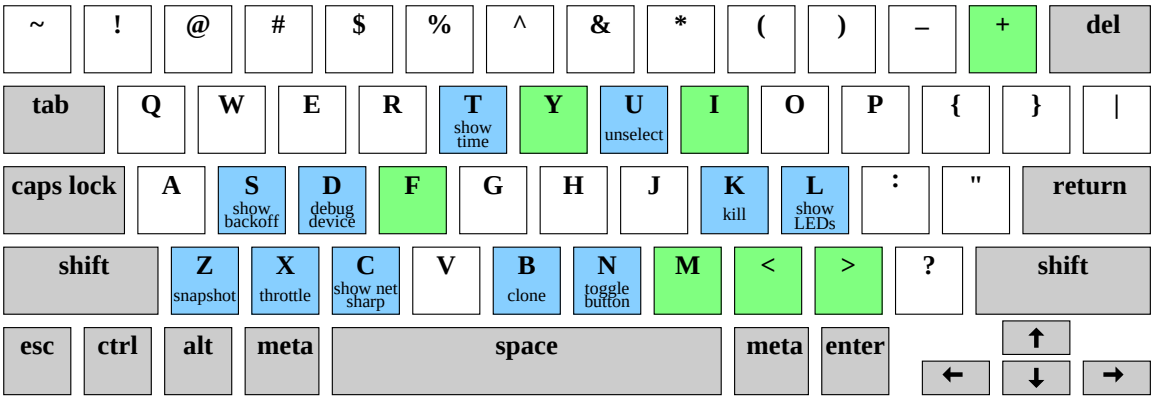


Figure 2: Upper case keyboard assignments

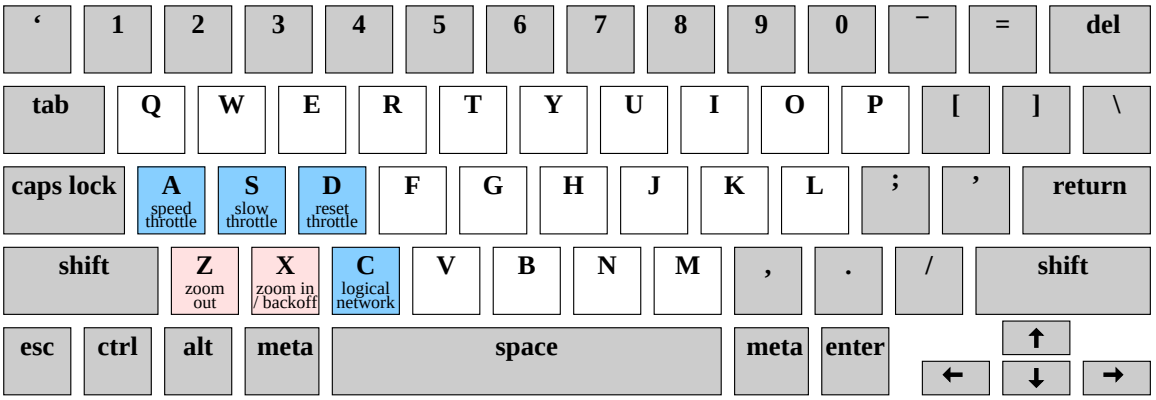


Figure 3: Control-key keyboard assignments

# B Mouse Assignments:

| BUTTON | CLICK | DRAG | SHIFT-CLICK | SHIFT-DRAG |
|---|---|---|---|---|
| LEFT | select | rotate | — | select region |
| RIGHT | print | zoom | — | move devices |
| MIDDLE | — | — | — | — |