

John Romero Programming Proverbs

- 4. “Great tools help make great games. Spend as much time on tools as possible.”
- John Romero, “The Early Days of Id Software - John Romero @ WeAreDevelopers Conference 2017”

Implementation of Touchmap

- these notes will show the structure of `touchmap.py`
- they also will describe `touchgui.select`
- they will show you how extend `touchmap`
 - creating an `export` function
 - create a `worldspawn` entity
- also show you how to add your own graphics into the library

Implementation of Touchmap

- touchmap is implemented in a single file
- uses a similar structure to the demo programs in touchgui

Implementation of Touchmap

touchmap-0.1/touchmap.py

```
def main ():
    global players, grid, cell_size

    pygame.init ()
    if full_screen:
        gameDisplay = pygame.display.set_mode ((display_width, \
                                                    display_height), FULLSCREEN)
    else:
        gameDisplay = pygame.display.set_mode ((display_width, display_height))

    touchgui.set_display (gameDisplay, display_width, display_height)
    controls = buttons ()

    gameDisplay.fill (touchguipalate.black)
    while True:
        grid = button_grid (cell_size)
        forms = controls + grid
        touchgui.select (forms, event_test, finished)

main ()
```

touchgui.select

touchgui/touchgui.py

```
def select (forms, event_test, finished = None, timeout = -1):  
    if timeout == -1:  
        _blocking_select (forms, event_test, finished)  
    else:  
        _nonblocking_select (forms, event_test, finished, timeout)
```

- two optional parameters: `finished` and `timeout`
- if `timeout` is absent then it calls a blocking version of `select`
 - in which the process will block until an event occurs
 - this is efficient, but forces the main program to be entirely event based
 - furthermore all events must go through the `touchgui/pygame` event queue

`touchgui.select`

- sometimes you might want to write programs which use a mixture of event based and some polling
- for example the cluedo server program
 - tests the gui briefly and then checks the network stack for input and rotates icons
 - ideally it would be good to join the network stack to the pygame input event queue and timers
 - in practice this is hard to configure, and `touchgui.select` allows a pragmatic (less efficient) solution
 - can *poll* both

Cluedo server example code

```
offset = 0
while not selection_complete:
    s, rpc = getRPC (s)
    processRPC (s, rpc)
    playerIcons = positionIcons (players, [.5, .5], .2, offset)
    forms = playerIcons + playerIconsStatic
    gameDisplay.fill (palate.black)
    touchgui.select (forms, event_test, selected, 10)
    offset = (offset + 1) % 360
return players
```

`touchgui.select`

- redraws all tiles in forms.
- `finished` is polled to see if the function should return
 - `finished` is a parameterless function which returns `True` or `False`
- `timeout` is the maximum no. of milliseconds the function can poll.
 - `timeout` is optional and defaults to -1 if absent
- `finished` is also optional

touchgui.select

- Pre-condition
 - forms is a list of tiles.
 - event_test is which has a single parameter (event)
 - event_test does not return a value
- Post-condition: None.

Extending touchgui: adding a worldspawn button

```
def main ():
    global players, grid, cell_size

    pygame.init ()
    if full_screen:
        gameDisplay = pygame.display.set_mode ((display_width, display_height), \
                                                FULLSCREEN)
    else:
        gameDisplay = pygame.display.set_mode ((display_width, display_height))

    touchgui.set_display (gameDisplay, display_width, display_height)
    controls = buttons () + glyphs ()

    gameDisplay.fill (touchguipalate.black)
    while True:
        grid = button_grid (cell_size)
        forms = controls + grid
        touchgui.select (forms, event_test, finished)
```

Extending touchgui: adding a worldspawn button

- at this point the call back `worldspawn` is in place
 - `worldspawn` can be made turn on worldspawn mode
- then `callback` can be modified to detect this mode and add the appropriate tile
- hint it might be sensible to use an indirect function
 - `empty_cell_click`
 - which is initially set to `empty_to_wall`
 - and can be changed to `empty_to_worldspawn`
- this technique allows `touchmap` to be extended to place hellknights, imps, ticks and pickups

Local images

- local images should be placed inside the `touchmap` directory
 - for example: `touchmap-0.1/images`
 - these images are kept in the source directory `touchmap-0.1`
- our build occurs in: `build-touchmap`
 - therefore the `Makefile.am` needs to have rules inside it to copy the images from the into the build directory

Build directory

- should be treated as volatile
 - can be destroyed and created
- therefore all assets and source files **must** be kept in the `touchmap-0.1` directory

autotools

- are used to configure and maintain the building rules
 - see `Makefile.am` and `configure.ac`
- the file `Makefile.am` contains the building rules
- in our case we just need extend `Makefile.am` to copy the image from the source directory into the build directory

autotools

touchmap-0.1/Makefile.am

```
all: doorh.png doorv.png doorh-bw.png doorv-bw.png hingeh.png \  
    wallh.png wallv.png wallh-bw.png wallv-bw.png \  
    newname.png  
  
newimage.png: $(srcdir)/images/newimage.png  
    °convert -resize 100x100 $< $@
```

- notice that ° needs to be replaced by a single tab character
 - you might need to alter preferences in gedit to allow you to add a tab character

autotools

■ `touchmap-0.1/Makefile.am`

```
all: doorh.png doorv.png doorh-bw.png doorv-bw.png hingeh.png \  
    wallh.png wallv.png wallh-bw.png wallv-bw.png \  
    newname.png  
  
newimage.png: $(srcdir)/images/newimage.png  
    cp -p $< $@ # this line must start with a tab character
```

Tutorial

- attempt to modify your `touchmap.py` file and add a `worldspawn` button
- change `cellback` too call an indirect function `empty_cell_click`
 - this should be a global variable which is initialised at the beginning of the module
 - it should default to creating a wall from an empty space
 - it should be changed by the `worldspawn` button to call a `worldspawn` character into the `cell_array`
 - see if you can make this new function generate a `worldspawn` tile