

■ `$HOME/Sandpit/touchgui/touchgui.py`

```
#
# create_cache - Pre-condition: None.
#               Post-condition: directory $HOME/.cache/touchgui
#
def _create_cache ():
    d = os.path.join (os.path.join (os.environ["HOME"], ".cache"),
                      os.system ("mkdir -p %s" % (d))
```

■ `$HOME/Sandpit/touchgui/touchgui.py`

```
#
# reset_cache - Pre-condition: None.
#              Post-condition: all contents of $HOME/.cache/touchgui
#
def reset_cache ():
    d = os.path.join (os.path.join (os.environ["HOME"], ".cache"),
                      _safe_system ("rm -r %s" % (d))
                      _create_cache ()
reset_cache ()
```

- you should enable `reset_cache()` which will delete the cache and create an empty cache directory

## Adding double tap to construct walls

- it would be good to add the ability for users to double tap and create walls along an axis
- requires a change to `cell_back` and keeping track of the `last_pos` tapped

## Adding double tap to construct walls

- ```
last_pos = [] # the last saved position
#
# save_wall_pos - saves the coordinate [x, y] to last_pos
#
def save_wall_pos (x, y):
    global last_pos
    last_pos = [x, y]
#
# match_line - return True if [x, y] is the same as the
#
def match_line (x, y):
    return (last_pos != []) and ((last_pos[0] == x) or (last_pos[1] == y))
```

## Adding double tap to construct walls

```
def cellback (param, tap):
    global clicked, cell_array, button_array, last_pos
    clicked = True
    mouse = pygame.mouse.get_pos ()
    x, y = get_cell (mouse)
    old = cell_array.get (x + xoffset, y + yoffset)
    button = button_array.get (x + xoffset, y + yoffset)
    if (old in ["v", " "] and (tap == 2)):
        save_wall_pos (x + xoffset, y + yoffset)
    elif old == " ":
        # blank -> next_tile
        if match_line (x + xoffset, y + yoffset):
            draw_line (x + xoffset, y + yoffset)
        else:
            function_create[next_tile] (button)
        last_pos = [] # forget about last_pos
    ...
```

## Adding double tap to construct walls

```
#
# draw_line - draw a line from the last_pos to, [x, y] providing [
#               lies on the same axis.
#
def draw_line (x, y):
    global cell_array, button_array
    if last_pos != []:
        if last_pos[0] == x:
            for j in range (min (y, last_pos[1]), max (y, last_pos[1])):
                old = cell_array.get (x, j)
                button = button_array.get (x, j)
                if old == " ":
                    button.to_wall ()
                    cell_array.set_contents (x, j, "v")
```

## Adding double tap to construct walls

```
elif last_pos[1] == y:
    for i in range (min (x, last_pos[0]), max (x, last_pos[0])):
        old = cell_array.get (i, y)
        button = button_array.get (i, y)
        if old == " ":
            button.to_wall ()
            cell_array.set_contents (i, y, "v")
```

## Implementing a safe export

- it would be good if the export facility checked to see that the map exported was successfully converted by chisel
  - chisel like all GNU/Linux and Unix programs exits with status 0 on success
  - and non zero on failure
  - we can test this and change the doom3 button (freeze it)
- we need to change: myexport and add try\_export which can also be called from the mydoom3 callback

## Implementing a safe export

```
def myexport (name, tap):
    pygame.display.update ()
    save_map (current_map_name)
    try_export (os.getcwd (), current_map_name)

def try_export (directory, map_name):
    os.chdir (os.path.join (os.getenv ("HOME"), "Sandpit/"))
    r = os.system ("./developer-txt2map " + os.path.join
    os.chdir (directory)
    if r == 0:
        print "all ok"
        doom_button.set_images (private_list ("doom3"))
    else:
        doom_button.set_images (error_list ("doom3"))
```

## Implementing a safe export

```
def mydoom3 (param, tap):
    pygame.display.update ()
    pygame.time.delay (toggle_delay * 2)
    try_export (os.getcwd (), "test.txt")
    pygame.quit ()
    dmap ()
    exec_doom_map ()
    quit ()
```