Snake2040 

Snake in MicroPython on the Badger2040

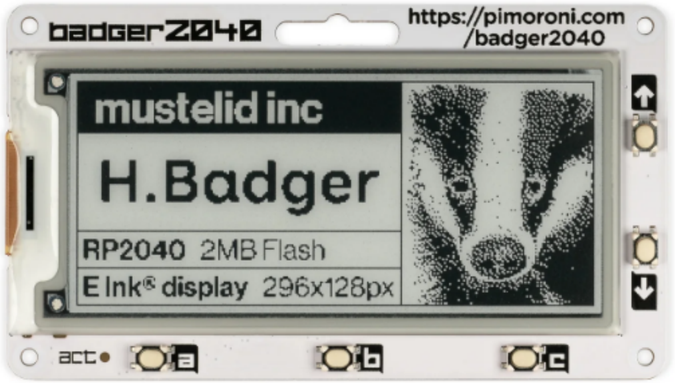
Jordan Duabe

Agenda

1. Hardware
2. Software
3. Game elements
4. Demo

Hardware

Badger2040



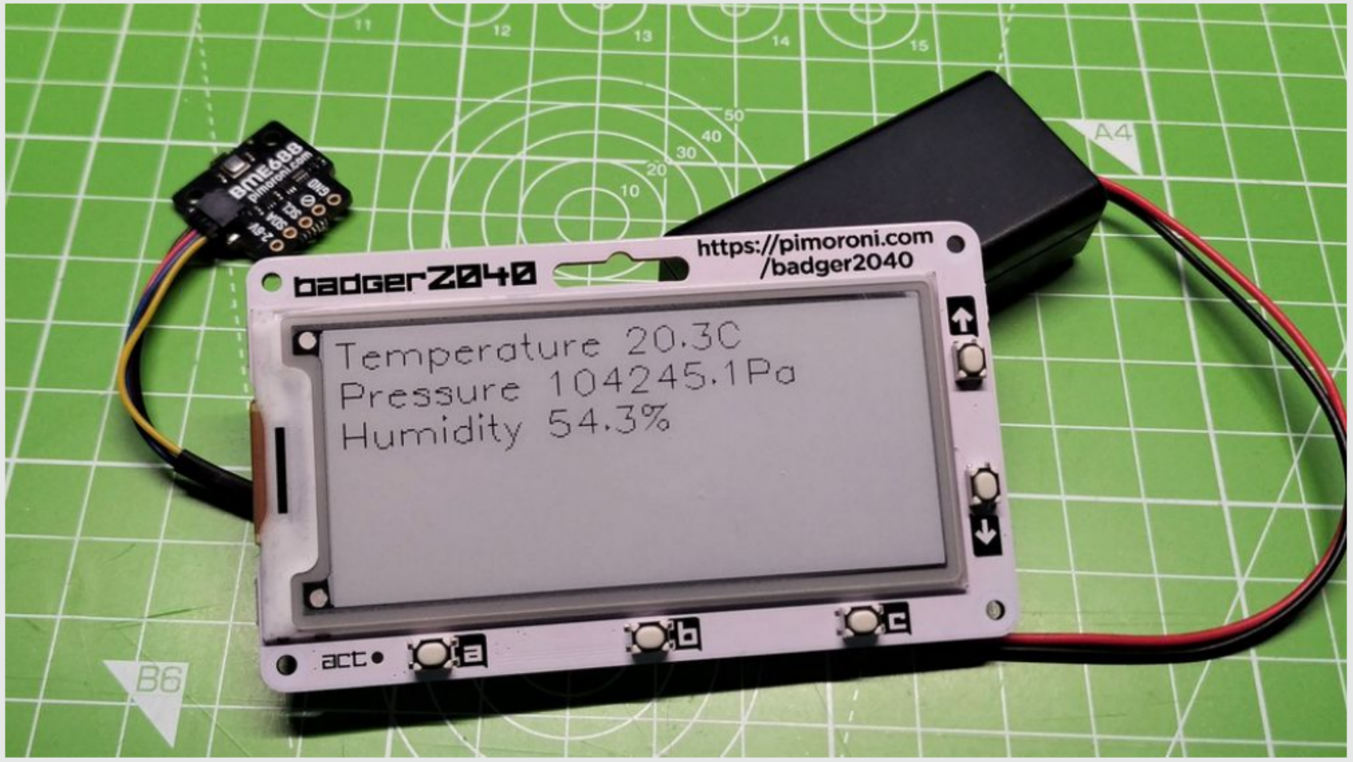
Features

- 2.9" black and white E Ink® display (296 x 128 pixels)
- Powered by RP2040 (Dual Arm Cortex M0+ running at up to 133Mhz with 264kB of SRAM)
- 2MB of QSPI flash
- Five front user buttons
- USB-C connector for power and programming
- JST-PH connector for attaching a battery (input range 2.7V - 6V)

Source: https://shop.pimoroni.com/products/badger-2040

Badger2040 digital badge

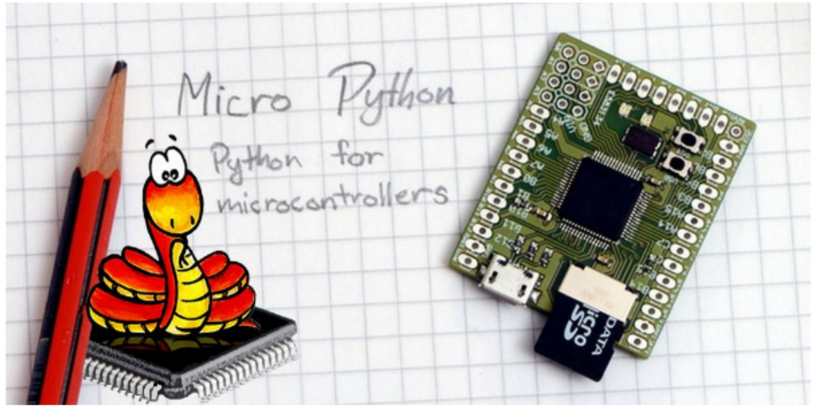
Badger2040 with peripherals



Source: https://www.tomshardware.com/reviews/pimoroni-badger-2040

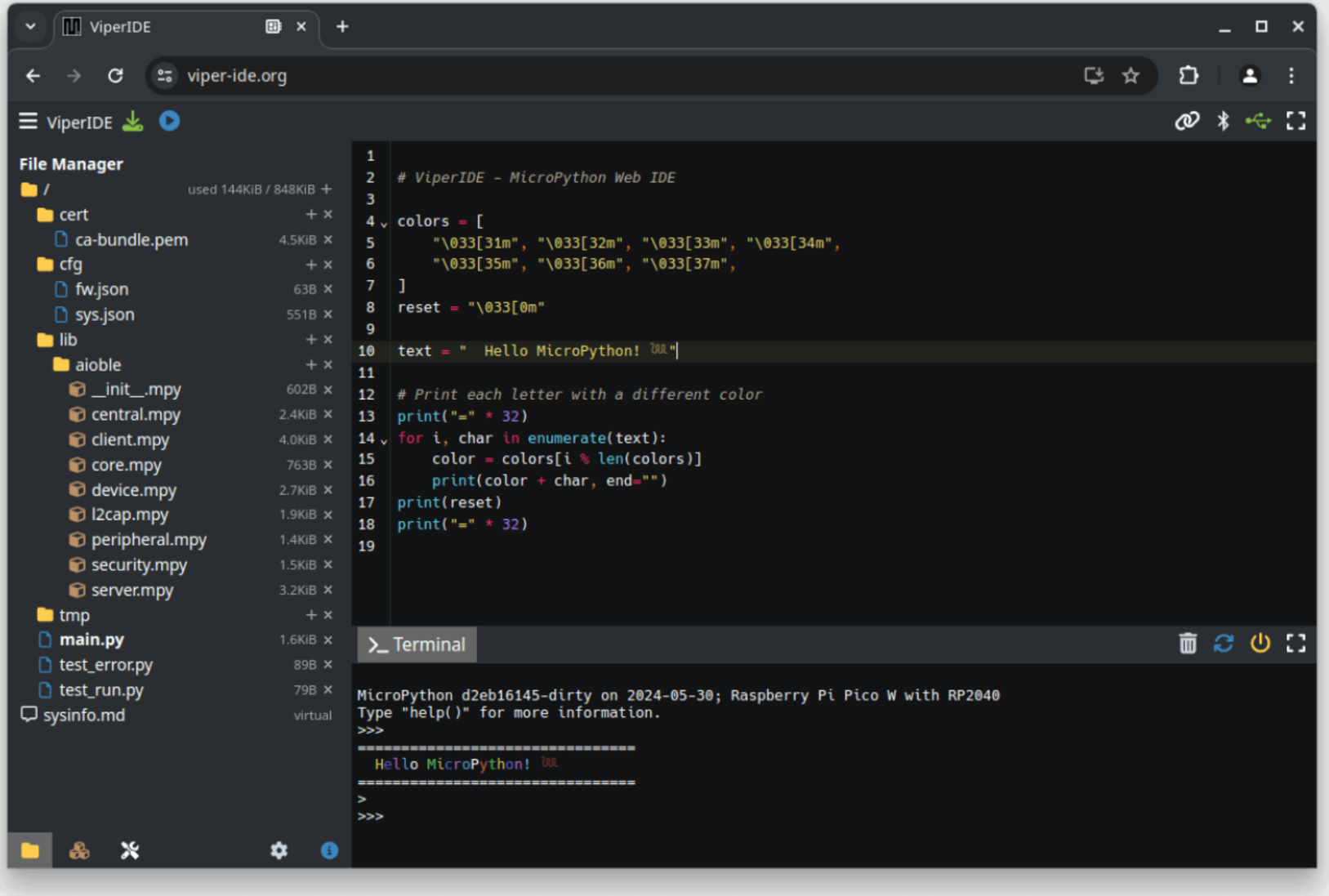
Software

MicroPython



- A lean implementation of Python 3 designed for microcontrollers and embedded systems
- Includes a subset of Python standard libraries
- Requires minimal RAM (as little as 16KB) and storage space (256KB flash)
- Ships with an interactive REPL (Read-Evaluate-Print Loop)
- Provides hardware-specific modules for direct control of microcontroller peripherals

Tooling



IDEs

- Thonny
- PyCharm (with MicroPython extension)
- VS Code (with MicroPython extension)
- ViperIDE (web-based IDE using the WebSerial and WebUSB APIs)

(Flashing) Firmware

The Badger2040 firmware is hosted on GitHub with two variants.

- pimoroni-badger2040-vX.X.X-**micropython-with-badger-os**.uf2
- pimoroni-badger2040-vX.X.X-**micropython**.uf2

To install (or upgrade to a newer version):

1. Connect the device your computer.
2. Enter into bootloader mode by holding the BOOT/USR button and pressing the RST button.
3. Drag and drop the *.uf2 file to the RPI-RP2 drive.

Badger2040-flavored MicroPython

badger2040.py

```python
import machine
from picographics import PicoGraphics, DISPLAY_INKY_PACK
import time

LED = 25
ENABLE_3V3 = 10
BUSY = 26

WIDTH = 296
HEIGHT = 128

BUTTONS = {
    BUTTON_DOWN: machine.Pin(BUTTON_DOWN, machine.Pin.IN, machine.Pin.PULL_DOWN),
    BUTTON_A: machine.Pin(BUTTON_A, machine.Pin.IN, machine.Pin.PULL_DOWN),
    BUTTON_B: machine.Pin(BUTTON_B, machine.Pin.IN, machine.Pin.PULL_DOWN),
    BUTTON_C: machine.Pin(BUTTON_C, machine.Pin.IN, machine.Pin.PULL_DOWN),
    BUTTON_UP: machine.Pin(BUTTON_UP, machine.Pin.IN, machine.Pin.PULL_DOWN),
    BUTTON_USER: machine.Pin(BUTTON_USER, machine.Pin.IN, machine.Pin.PULL_UP),
}

class Badger2040():
    def __init__(self):
        self.display = PicoGraphics(DISPLAY_INKY_PACK)
        self._led = machine.PWM(machine.Pin(LED))
        self._led.freq(1000)
        self._led.duty_u16(0)
        self._update_speed = 0

    def __getattr__(self, item):
        # Glue to redirect calls to PicoGraphics
        return getattr(self.display, item)

    def update(self):
        t_start = time.ticks_ms()
        self.display.update()
        t_elapsed = time.ticks_ms() - t_start

        delay_ms = [4700, 2600, 900, 250][self._update_speed]

        if t_elapsed < delay_ms:
            time.sleep((delay_ms - t_elapsed) / 1000)
```

Source: https://github.com/pimoroni/badger2040/blob/main/firmware/PIMORONI_BADGER2040/lib/badger2040.py

PicoGraphics

Text - Code

```python
import badger2040

display = badger2040.Badger2040()

# Pen Color
# There are 16 pen colours - or "shades of grey" - to choose, from 0 (black) to 15 (white).
display.set_pen(
    colour  # int: color from 0 to 15
)

# Pen Thickness
display.set_thickness(
    value  # int: thickness in pixels
)

# Available fonts
# Bitmap: bitmap6, bitmap8, bitmap14_outline
# Vector: sans, gothic, cursive, serif_italic, serif
display.set_font(font)

display.text(
  text,       # the text string to draw
  x,          # the destination X coordinate
  y,          # the destination Y coordinate
  wordwrap,   # number of pixels width before trying to break text into multiple lines
  scale,      # size
  angle,      # rotation angle (Vector only!)
  spacing     # letter spacing
)
```

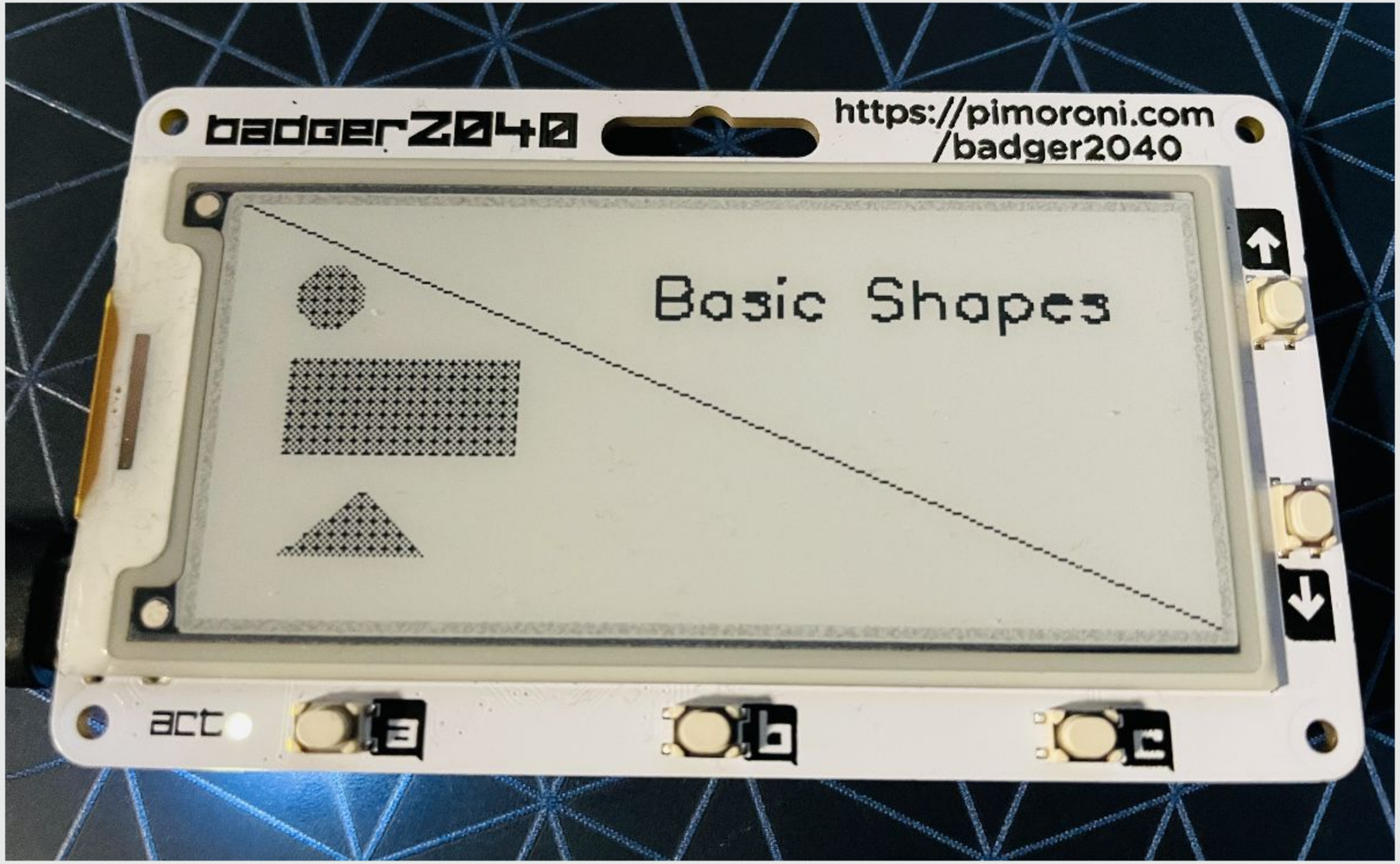
Text - Output


badger2040
https://pimoroni.com
/badger2040
sans
gothic
cursive
serif
serif_italic
act
a
b
c

Shapes - Code

```python
import badger2040

display = badger2040.Badger2040()

# Line
display.line(x1, y1, x2, y2)
display.line(x1, y1, x2, y2, thickness) # you can also specify the line's thickness

# Circle
display.circle(x, y, r)

# Rectangle
display.rectangle(x, y, w, h)

# Triangle
display.triangle(x1, y1, x2, y2, x3, y3)

# Polygon
# To draw other shapes, you can provide a list of points to polygon
display.polygon([
  (0, 10),
  (20, 10),
  (20, 0),
  (30, 20),
  (20, 30),
  (20, 20),
  (0, 20),
])

# Pixels
display.pixel(x, y) # setting individual pixels is slow

# drawing a horizontal span of pixels is a little faster
display.pixel_span(x, y, length)
```

Shapes - Output



Images - Code

```
import badger2040
import jpegdec # BitBank's JPEGDEC library https://github.com/bitbank2/JPEGDEC
from png import PNG # BitBank's PNGDEC library https://github.com/bitbank2/PNGdec
import qrcode

badger = badger2040.Badger2040()

# JPEG Files
jpeg = jpegdec.JPEG(badger.display)
jpeg.open_file("/image.jpg")
jpeg.decode(x, y)

# PNG Files
png = PNG(badger.display)
png.open_file("/image.png")
png.decode(x, y)

# QR Code
code = qrcode.QRCode()
code.set_text("your text here")
```

Images - Output

Updating the display

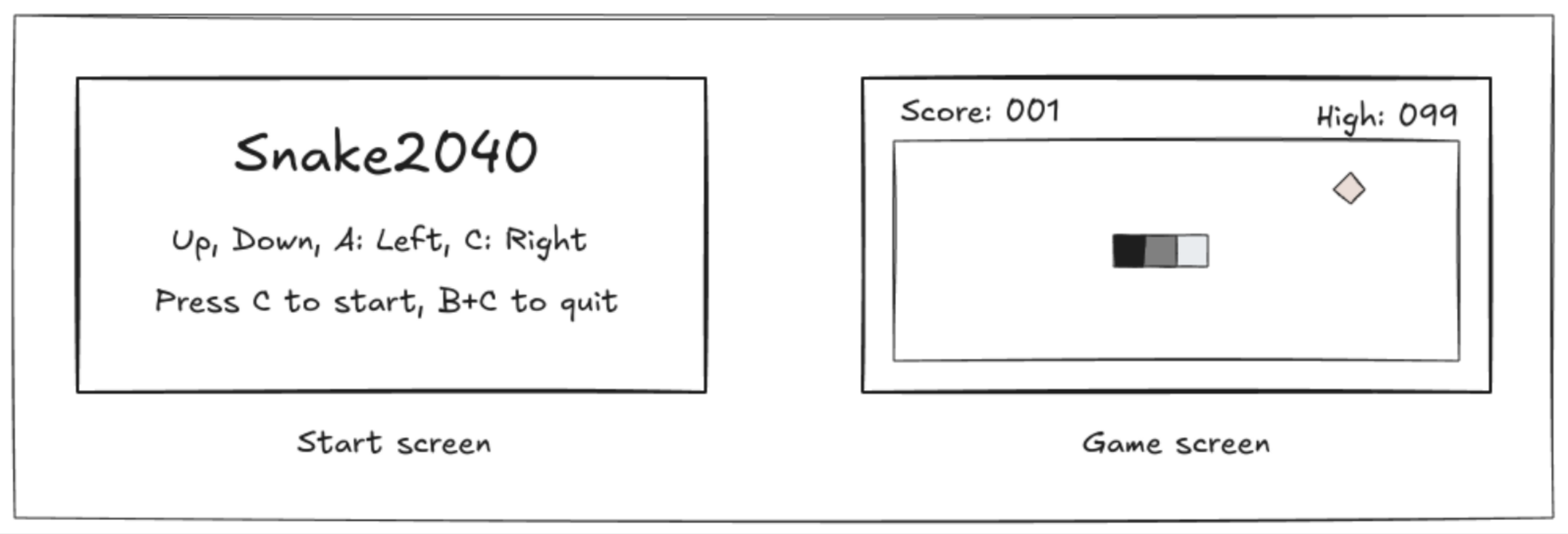
```python
import badger2040

display = badger2040.Badger2040()

# UPDATE_NORMAL - great for display of the first screen, good contrast and no ghosting
# UPDATE_MEDIUM - a good balance of speed and clarity
# UPDATE_FAST   - good for stepping through screens such as the pages of a book or the launcher
# UPDATE_TURBO  - prone to ghosting, great for making minor changes
display.set_update_speed(
    speed  # int: one of the update constants
)
# Starts a full update of the screen. Will block until the update has finished.
display.update()
```

Buttons

```
import badger2040

display = badger2040.Badger2040()

# BUTTON_A = 12
# BUTTON_B = 13
# BUTTON_C = 14
# BUTTON_UP = 15
# BUTTON_DOWN = 11
if display.pressed(badger2040.BUTTON_C):
  # handle button press
```

Game elements

Game screens



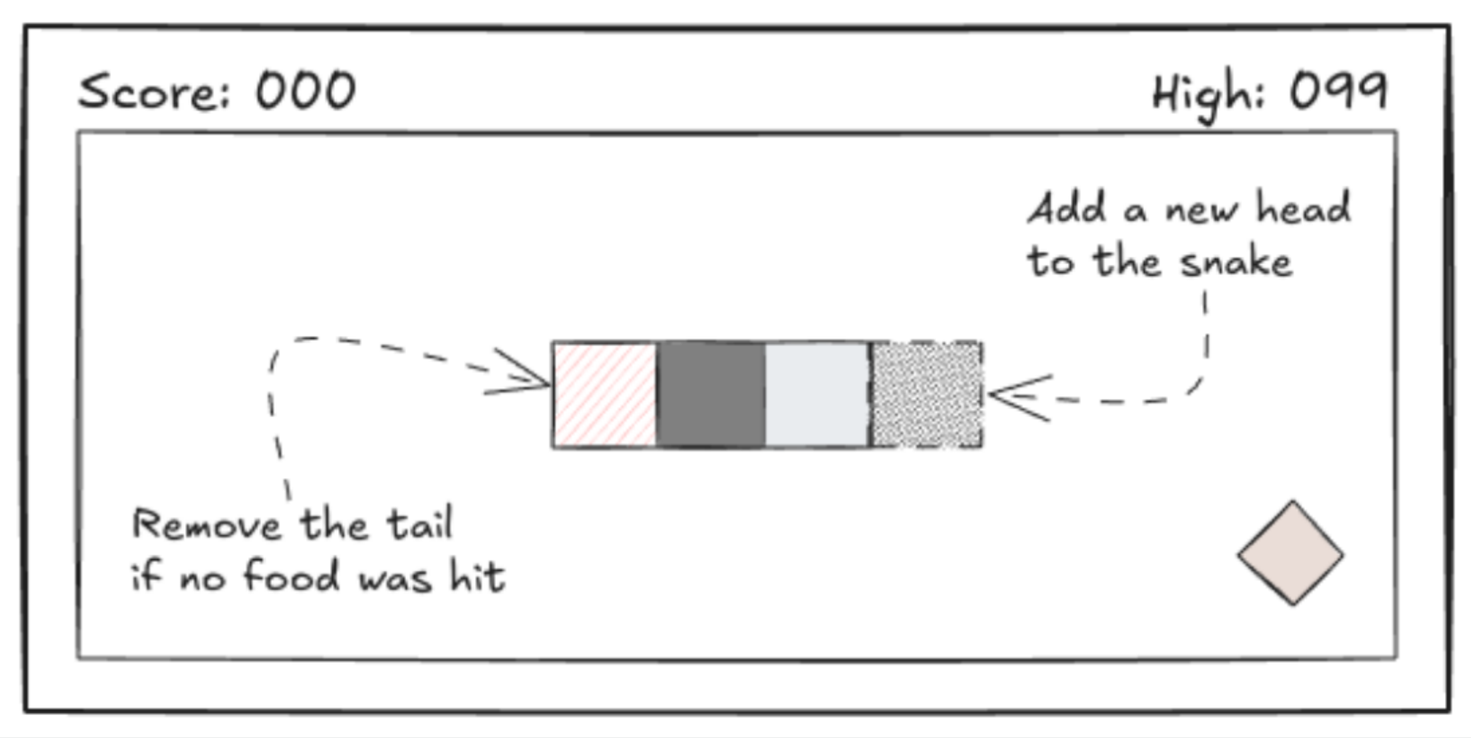
Play area and spawn point

```python
import badger2040

GRID_SIZE = 8  # Size of each cell in pixels
GRID_WIDTH = badger2040.WIDTH // GRID_SIZE
GRID_HEIGHT = badger2040.HEIGHT // GRID_SIZE

display = badger2040.Badger2040()

# List of points
snake = []

# Start with 3 segments in the middle of the screen
snake = [(GRID_WIDTH // 2, GRID_HEIGHT // 2)]
for i in range(1, 3):
    snake.append((snake[0][0] - i, snake[0][1]))

# Generate food at a random position that is not occupied by the snake
while True:
    x = random.randint(0, GRID_WIDTH - 1)
    y = random.randint(0, GRID_HEIGHT - 1)
    if (x, y) not in snake:
        display.rectangle(x * GRID_SIZE, y * GRID_SIZE, GRID_SIZE, GRID_SIZE)
        break
```

Input and controls

```
import badger2040

display = badger2040.Badger2040()

# Directions
UP = (0, -1)
DOWN = (0, 1)
LEFT = (-1, 0)
RIGHT = (1, 0)

# Handle button inputs
if display.pressed(badger2040.BUTTON_UP) and direction != DOWN:
    next_direction = UP
elif display.pressed(badger2040.BUTTON_DOWN) and direction != UP:
    next_direction = DOWN
elif display.pressed(badger2040.BUTTON_A) and direction != RIGHT:
    next_direction = LEFT
elif display.pressed(badger2040.BUTTON_C) and direction != LEFT:
    next_direction = RIGHT

# Check for quit (B + C together)
if display.pressed(badger2040.BUTTON_B) and display.pressed(badger2040.BUTTON_C):
    game_over = True
    break
```

Screen updates: Visualization



Collision detection

```
# Get the position for the new head
head_x, head_y = snake[0]
new_head = (head_x + direction[0], head_y + direction[1])

# Check if the new head is out of bounds
if (new_head[0] < 0 or new_head[0] >= GRID_WIDTH or
    new_head[1] < 0 or new_head[1] >= GRID_HEIGHT):
    game_over = True
    return

# Check if the snake hits itself
if new_head in snake:
    game_over = True
    return

# Add the new head to the snake
snake.insert(0, new_head)

# Check if the snake eats the food
if new_head == food:
    score += 1
    generate_food()
else:
    # Remove the tail if no food was eaten
    snake.pop()
```

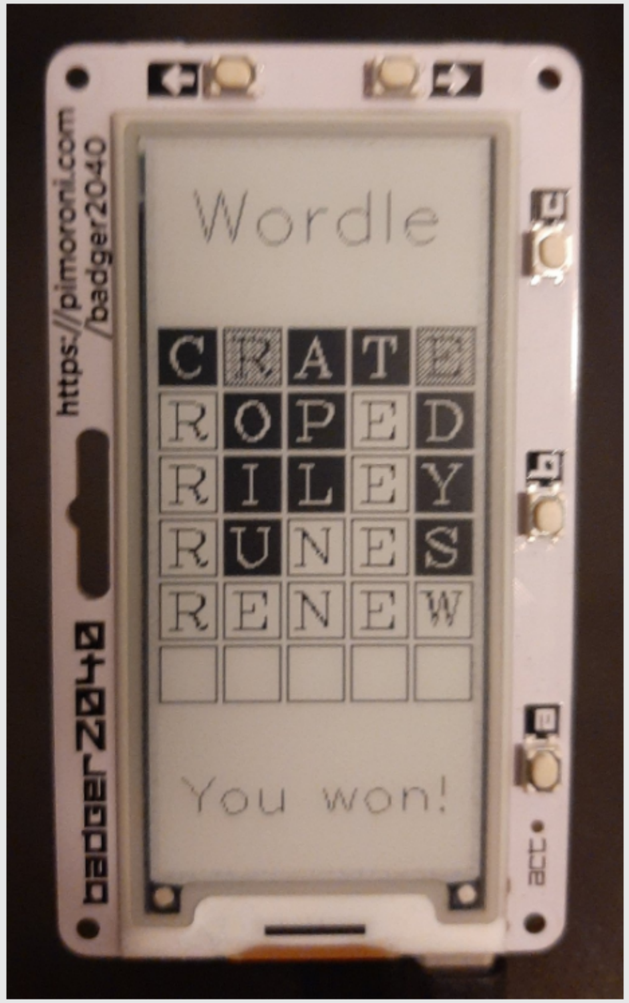
Screen updates

```
import badger2040
import time

display = badger2040.Badger2040()

last_update = time.ticks_ms()
move_delay = 200  # Start with a 200 ms delay between moves (adjust for difficulty)

while not game_over:
    now = time.ticks_ms()
    if time.ticks_diff(now, last_update) > move_delay:
        move_snake()
        last_update = now
        # Make the game faster as the score increases
        move_delay = max(100, 200 - (score * 5))

    # Draw everything
    draw_snake()
    draw_food()

    display.update()
```

Demo

Demo: Snake2040

Cool stuff: Wordle



Source: https://github.com/makew0rld/wordle-badger2040

Cool stuff: Dino game

Source: https://github.com/niutech/dino-badger2040

Show me the code

snake2040 GitHub repo

Slides

Thank You

- **Web**: *jduabe.dev*
- **Mastodon**: *@jordan@jduabe.dev*
- **GitHub**: *@j4ckofalltrades*