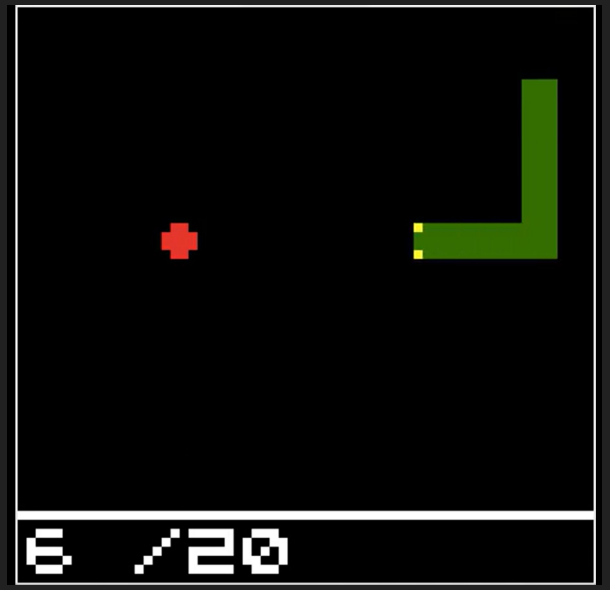
Week 4 Recitation

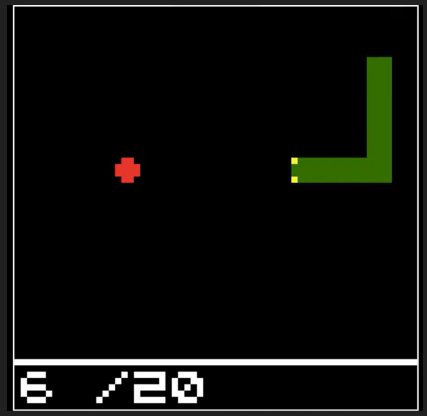
February 1st, 2024

Project 1: Snake



Project 1: Introduction

- We will be using MIPS to build the game snake!
- Remember, you can lose points for poorly written/styled code, so like, Don’t Do That :)
- Rubric (more detailed on the assignment page):
  - [20 Points] Drawing the snake & apple
  - [30 Points] Snake movement w/o keyboard
  - [16 Points] Snake movement w/ keyboard
  - [24 Points] Apple logic
  - [10 Points] Snake cannot hit itself
- Due: Sunday, March 17th

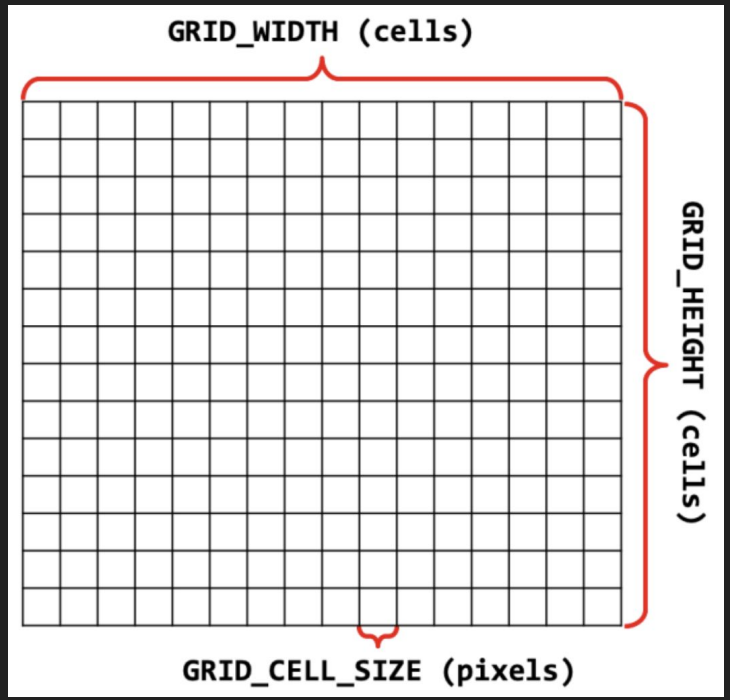


Project 1: Starter Files

- abc123_proj1.asm
  - This is the file we will be editing
  - Contains much of the game logic code
- constants.asm
  - Holds constants for interacting w/ display
- display_2211_0822.asm
  - Library of display functions
- macros.asm
  - Some useful macros!
- textures.asm
  - Contains the graphics

Project 1: The Grid

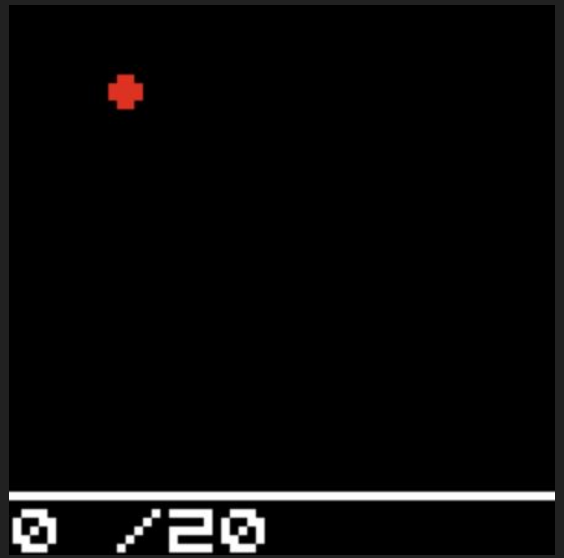
- The game is based on the idea of a grid
- Constants GRID_WIDTH, GRID_HEIGHT, & GRID_CELL_SIZE define the sizing of our grid
- We will be using the grid to place all of our objects! (snake, apple)



Implementation Steps

1. Drawing the Apple

- This function will draw the apple onto the screen
- We use the grid to place it in a location
- Make sure to use la and not lw here!



2. Drawing the Snake

- snake_len tracks the length of the snack (in grid cells)
- snake_x and snake_y are arrays of bytes, trake the coordinates of snake segment
  - These arrays are parallel, so snake_x[0] and snake_y[0] are the coordinates of the snake's head!
- Will need to use a for loop!
  - Check out this example from last week

The rest of the steps

3. Moving the snake
4. Controlling the snake
5. “Eating that dang apple”
6. Moving the apple
7. The snake eating itself

Note: some of this stuff in these steps we will go over next reciation

enter and leave Macros

- These macros prevent us from needing to do:
  - push ra
  - pop ra
  - jr ra
- But, what the flip even are these?????

Functions

Functions

- When we call a function, we have to know 2 things:
  - Where we are going
  - Where to come back to
- To follow this, we use a calling convention: set of rules used to write & call functions
  - These are not automatic, however

Functions: The Program Counter

- Machine code instructions are in memory, so they have addresses
- The program counter (PC) is the address of the current instruction

```
_top:
        lw t0, (s0)         # PC: 0x8000
        add t0, t0, 1       # PC: 0x8004
        sw t0, (s0)         # PC: 0x8008
        add s0, s0, 4       # PC: 0x800C
        blt s0, s1, _top    # PC: 0x8010
```

Functions: Return Address Register

- We have a register to store an address, ra
- So, if we need to jump to a function, we can first store the old address before we jump and then return after

Functions: Calling and Returning

- To call a function, we use jal (jump and link!)

```
jal some_func
```

- To return from a function, we use jr (jump to register)

```
jr ra
```

Functions: Calling a Function

- To call a function, we use jal (jump and link!)
- This sets the pc to the address of the label (some_func) ***AND*** sets ra to the instruction immediately after the jal

```
jal some_func
add s0, s0, 1
```

Functions: Example

- The jal in main sets ra to the address of the line directly below it
- Once we are inside some_func, jr ra jumps is back to ra, the line below the jal

Project 1 > some_funcs.asm

```
1   .macro print_str %str
2       .data
3       print_str_message: .asciiz %str
4       .text
5       push a0
6       push v0
7       la a0, print_str_message
8       li v0, 4
9       syscall
10      pop v0
11      pop a0
12  .end_macro
13
14  .global main
15  main:
16      jal some_func
17      print_str "Yup, we back here"
18
19
20  some_func:
21      print_str "You're not gonna believe it, but we down here"
22      jr ra
23
24  exit:
25      li v0, 10
26      syscall
```

More on functions next week!

Lmk if you have any questions!