

CS2340 Computer Architecture Bitmap Project

Name: Che Blankenship

cxb200006

April 25th 2021

(1) Idea

I decided to make an maze game using MIPS assembly with bitmap/keyboard tool.

My purpose of making this game is (1) Review my understanding of basic MIPS programming by implementing functions, loops and syscalls. (2) Learn how 2-dimentional array works in MIPS, and how to access each element dynamically by calling a function.

(2) The Implementation

The code is around 650 – 700 ish. Tried to get shorter line of code by implement loops and functions.

(3) The logic (list of logics I implemented in this program)

(a) 2D array manipulation.

- Generate “outer moat” function
- Generate “path” function
- Display maze function

(b) Game logic

- Set start and goal.
- Verify user next move.

(4) Documentation for the code.

- I would like to explain each part of the code I listed above.

(a) 2D array manipulation

To generate a maze, I used one of the famous/basic algorithm called “Maze-Bar Algorithm”. To efficiently generate a maze and every time changing the maze rout, I had to setup a 2D array with same number of row and columns. In addition, it had to be an odd number array (for example, 13x13, 7x7, 25x25, and row=column>=5). I divided the algorithm into two major steps. First, I generated the outer moat, in other words the wall that's around. Second, I generated the path by using syscall 42 with range from $1 \leq n \leq 12$. Then I divided it to 4 cases to randomly generate walls inside the maze to make no passages. At the end, I called a function to iterate though the whole 2D array, and display the walls on the bitmap.

(b) Game logic

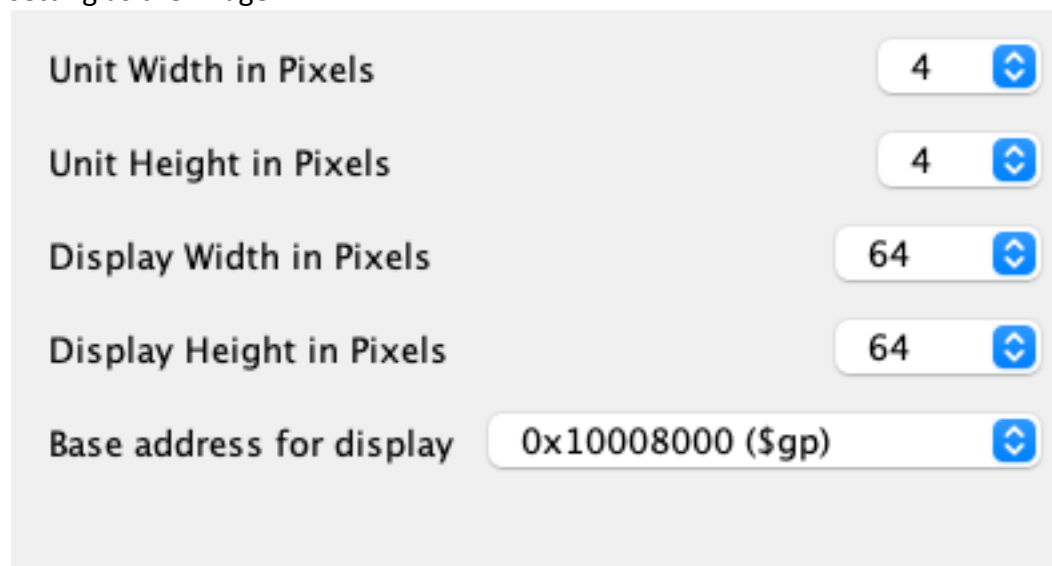
To dynamically handle which direction the user can go next, I used the location on the bitmap and the 2D array x,y coordinates. The user can only go up, down, left, right; hence, I divide the move cases, and see if the next move that the user want to take is valid by checking the 2D array next moving index element. I set “open path” =

0, and "wall" = 1. So if the next move index element in the array is 1, I set the program not to make the user move.
Since the maze generating algorithm is a basic logic, I had to manually set the start/goal, and that location is always at the same point. Start=(1,0) and goal=(13,14). When the user successfully reaches this index, x=13, and y=14, then I display a message saying "Goal!" The message is manually hardcoded.

(5) Details

(a) How to run the program.

- (1) Open maze.asm on Mars.
- (2) Open "Bitmap Display" from "Tools" and click "Connect to MIPS". Setup the setting as the image.



- (3) Open "Keyboard and Display MMIO" from "Tools" and click "Connect to MIPS".
- (4) Compile the program and start running the program.
- (5) You will see a random generated maze.

You can see that at the top-left, there is a red pixel that shows your current location. At the right-bottom of the maze, you will see the goal.



- (6) Use the keyboard w="up", s="down", a="left", and d="right" to reach the goal. Whenever you successfully move to next point, the maze will track your path history with black. The image below shows right before it reaches the goal.



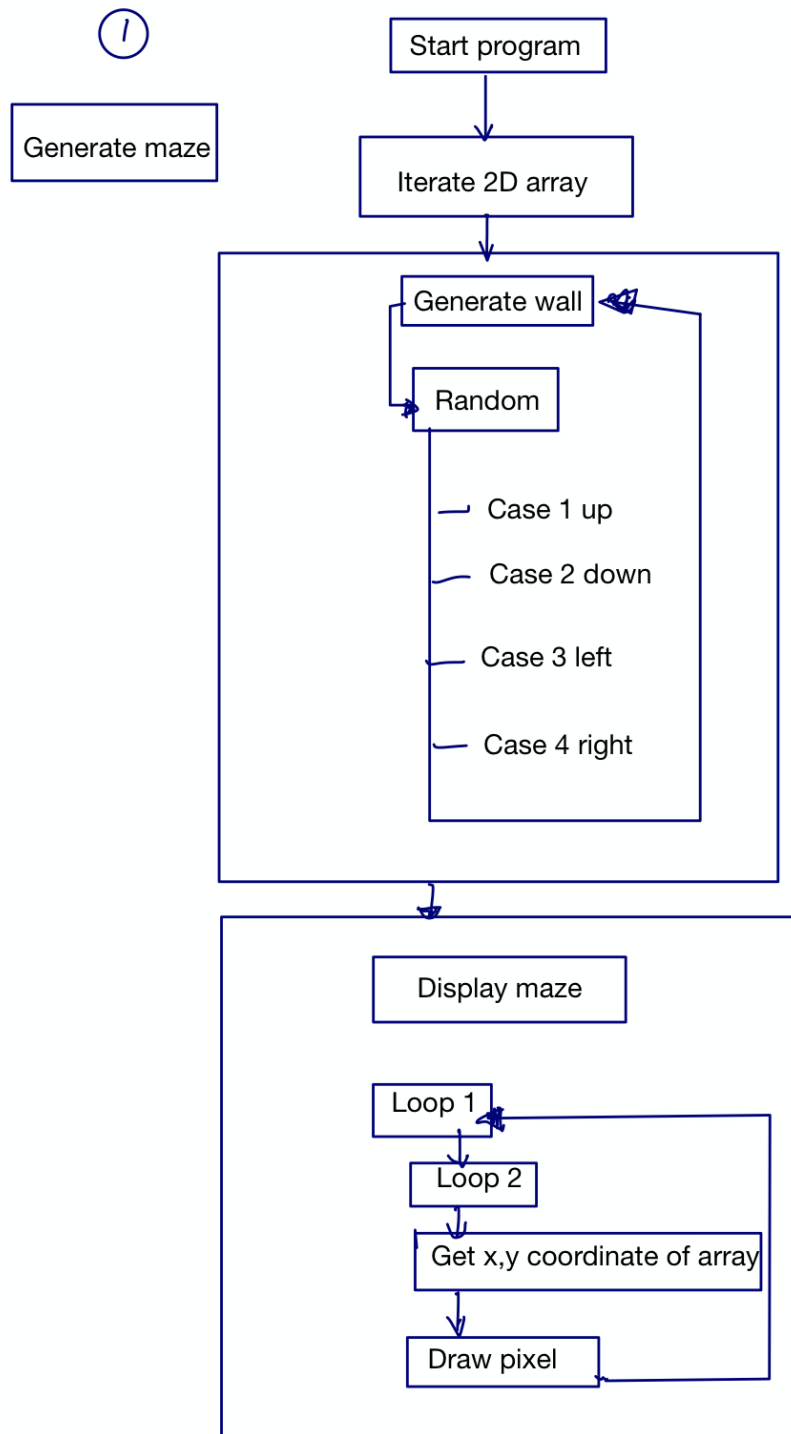
- (7) When you exit the maze successfully, you will see this image and program will finish.



- (b) Overview of what the program does.
- (1) Start program
 - (2) Generate 2D array with all elements to zero.
 - (3) Access the 2D array and iterate through each element by having nested loop to access each array[x][y].
 - (4) Call "draw_pixel" function to draw the whole page as white.
 - (5) Call "maze_process"
 - (6) "Maze_process" will call three functions.
 - (7) First, it calls "generate_maze_outer_moat" to set elements to "1" as walls.
 - (8) Second, call "generate_maze_street" to set walls in the maze by random syscall 42.
 - (9) To always have a start – goal path, I implemented a if statement in "generate_maze_street" to check if it does not block the entire path.
 - (10) Call "draw_maze" with an argument of array base address to iterate each element and draw it on the bitmap (if element=1, draw blue wall).
 - (11) User will be able to move the red pixel now.
 - (12) When the user press the keyboard to move to the next location, it will call the direction function to check if the next move is valid by checking the 2D array element.
 - (13) If it moves successfully, then the current location will be updated and you will see the previous location being blackout as "move history".
 - (14) When user reaches the goal and exits the maze, it will display the "Goal!" message.

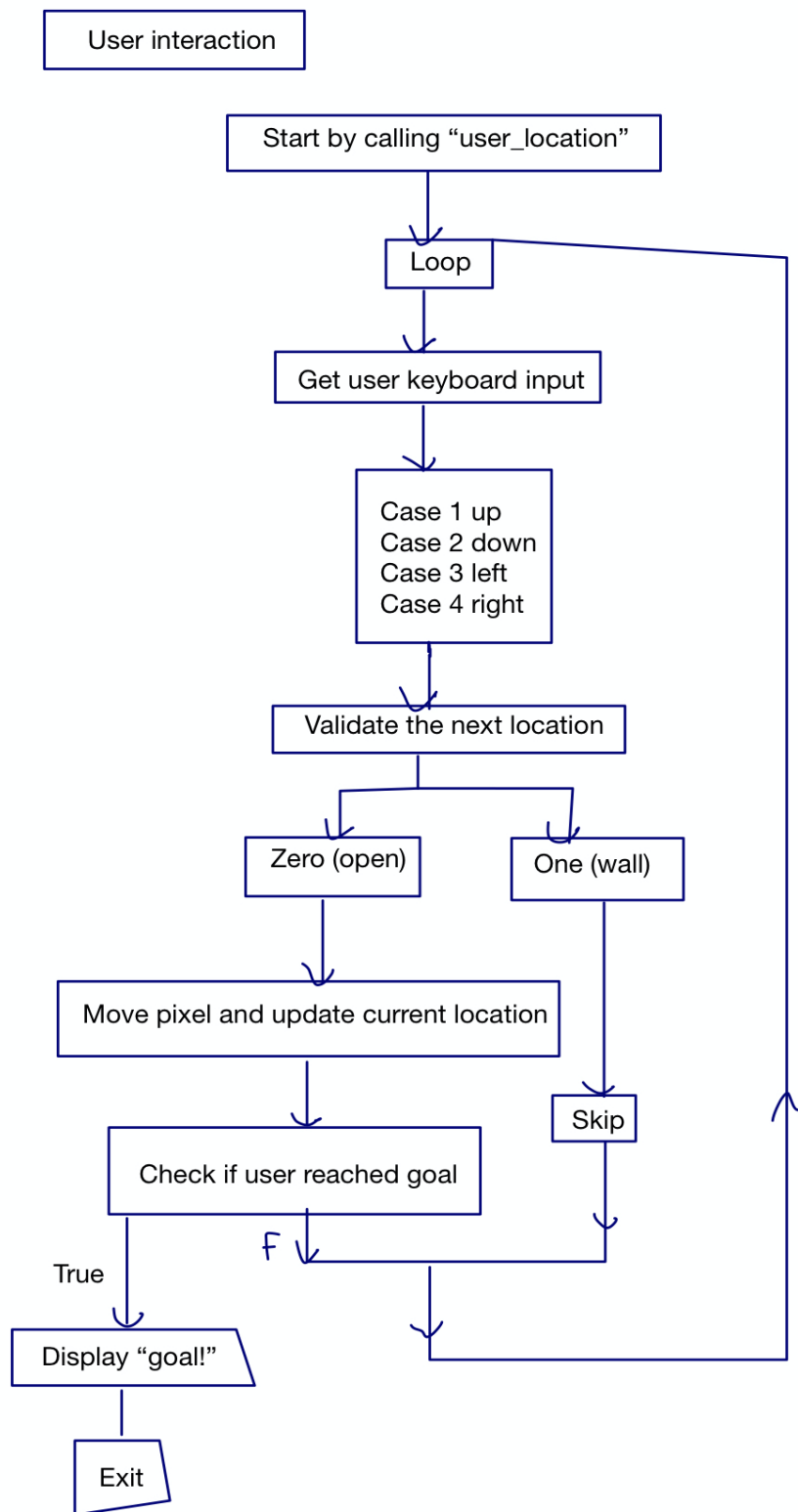
(c) Flowchart

Part (a): Generating maze using a 2D array



Part (b): Use a loop to keep track of user input and moves in the maze.

②



[Sudo code]

```
arr[15][15] = []
```

```
x = 0
```

```
y = 0
```

```
random = 0
```

```
# set the walls up/down
```

```
for (x = 0 to 15):
```

```
    arr[0][x] = 1
```

```
    arr[14][x] = 1
```

```
# set the walls on both left/right
```

```
for (x = 0 to 15):
```

```
    arr[y][0] = 1
```

```
    arr[y][14] = 1
```

```
# make path in the maze by using maze-bar algorithm
```

```
y = 2;
```

```
For (x=2, x=13, x+2):
```

```
    Random = random int between 1 ~ 12
```

```
    If(random_int <= 3):
```

```
        If (arr[y-1][x] == 0):
```

```
            Arr[y-1][x] = 1 // set bar/wall
```

```
        Else:
```

```
            X = x -2 // no update
```

```
    Else If(random_int <= 6):
```

```
        If (arr[y-1][x] == 0):
```

```
            Arr[y-1][x] = 1 // set bar/wall
```

```
            X = x -2 // no update
```

```
    Else If(random_int <= 9):
```

```
        If (arr[y-1][x] == 0):
```

```
            Arr[y-1][x] = 1 // set bar/wall
```

```
            X = x -2 // no update
```

```
    Else If(random_int <= 12):
```

```
        If (arr[y-1][x] == 0):
```

```
            Arr[y-1][x] = 1 // set bar/wall
```

```
            X = x -2 // no update
```

```
Arr[0][1] = 0 // make start point
```

```
Arr[13][14] = 0 // make goal
```

```

// user interaction
Current_x = 1
Current_y = 0
While Loop:
    If keyboard == up:
        If Arr[current_y-1][current_x] != 1:
            Move / update bitmap location
    If keyboard == down:
        If Arr[current_y+1][current_x] != 1:
            Move / update bitmap location
            If (x == goal_x) && (y+1 == goal_y):
                Display "Goal!"
                Exit();
    If keyboard == left:
        If Arr[current_y][current_x-1] != 1:
            Move / update bitmap location
    If keyboard == right:
        If Arr[current_y][current_x+1] != 1:
            Move / update bitmap location

```