Software Overview

Year: 2024 Semester: Spring Team: __1__ Project: Dungeon crawler board
Creation Date: _____01/25/2024_____ Last Modified: February 21, 2024
Author: Neil Brown Email: brow1950@purdue.edu

Assignment Evaluation: See Rubric on Brightspace Assignment

1.0 Software Overview

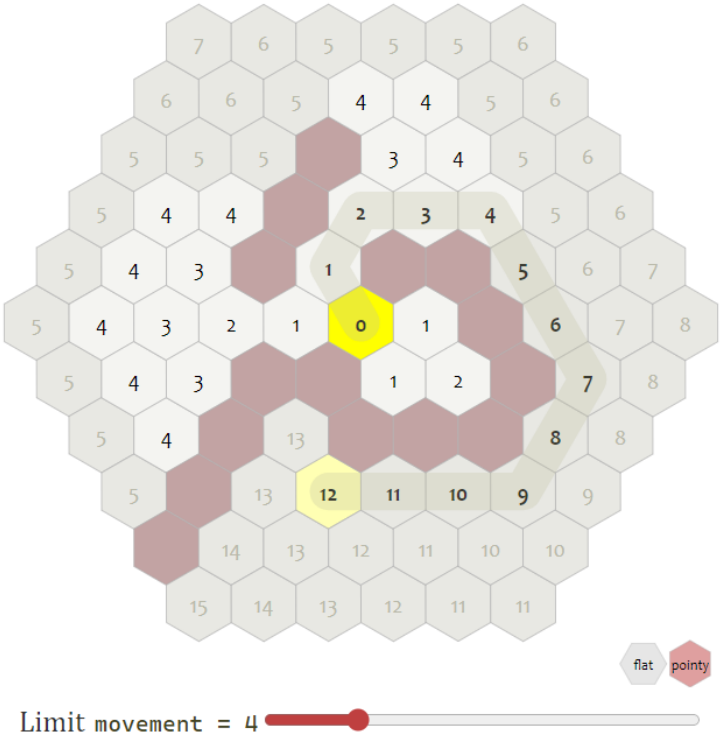
There are two pieces of software that must be created for this project. The first piece of software that needs to be created runs on a PC. This project will implement a custom desktop application, created using Unity, which allows the user to create their own map and characters with a variety of attributes. The game involves the movement of characters across a hexagonal map on the game board. This application will allow the user to customize this 16x16 map by setting the hexes to have different values. Currently there are values for clear space, walls, chests, players and enemies, however in an extension of this project, more hex types could be added. Once the map has been constructed with a variety of hexes, the players and enemies are constructed using the application. Both the players and enemies have a variety of attributes that need to be customized by the user before beginning the game. Once the map and characters have been created, this information is parsed into a JSON file and sent across USB to be stored on the microcontroller.

The second piece of software used for this project is the software that controls the actual gameplay. The two aspects of gameplay that will be implemented are movement and combat. First, players will be moving across the map to interact with the environment including enemies and chests. Movement is determined based on a speed attribute in each character as well as a dice roll that is input through the keypad. This will give the player the maximum distance that they are allowed to travel in the given turn. Using this distance, a version of breadth first search algorithm is used to display to the player which hexes they are allowed to move to. Once a player confirms the hex that they want to move to, the map, stored as an array, will update the hex's value to indicate that the player has moved there. When a player moves and encounters an enemy, combat will be initiated. Combat will be facilitated based upon the attributes that were set by the player and the enemy as well as a dice roll, input by the keypad. The calculations done for combat will be based upon the DnD 5e rules set [1]. After the result of combat, the loser either being the monster, or the player will be removed from the map, and play will continue to the next player.

The firmware used for this project will directly interface with the second piece of software. After all the movement and combat calculations have been completed by the software, firmware must be used to drive the hardware that visualizes the game for the players. The most important firmware is the code that drives the LEDs and Hall effect sensors. The LEDs are controlled using PWM out of a GPIO pin. The firmware needs to control the duty cycle of this PWM signal to drive different colors and intensities to the LEDs on the game board. Next, to interact with Hall effect sensors, an I2C I/O expander will be used to read from all the sensors. The firmware needs to interpret the I2C signals to tell the system where the magnet associated with each player is located. Other firmware includes the logic to take input from the keypad and display text prompts on the LCD screen. The firmware will be used to interpret GPIO matrix signals from the keypad and set the data that will be sent over SPI to the LCD screen.

2.0 Description of Algorithms

The most important algorithm used in this project is the breadth first search (BFS) algorithm [2]. BFS will be used to determine the possible hexes that the player can move to which is determined by the number they roll during their turn. The BFS algorithm is a graph traversal algorithm. It works by using a queue to traverse across the entire graph starting with all the nodes closest to the starting point and working outwards. In the dungeon crawler board this will be used to limit player movement across the hexagonal board. A BFS is necessary because of the walls that are built into the maps. With no walls, the distance formula could be used, and a radius could be displayed given the player's available moves. However, movement around the walls must be considered, so a graph approach is taken. The challenge of walls is shown in the picture below from the Hexagonal Grid website [3].

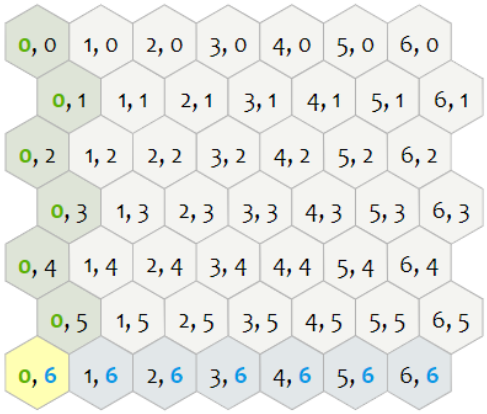


The breadth first search in dungeon crawler board will take in an argument for the maximum distance *d*. Starting at the players location, the breadth first search traverses across all nodes of distance 1,2,3…*d*-1, *d*. Once all nodes of distance *d* have been searched, the algorithm will stop and return a list of all the nodes that have been traversed. These are the nodes that the player can travel to and will be highlighted on the map. This will be run for each players turn so there is no confusion about where the player is able to move.

The only other algorithm needed for this project is one to determine the winner when two characters engage in combat. This is not as complicated as BFS. All that is required is to combine a combination of different player attributes to determine the winner which is based on the DnD ruleset [1].

3.0 Description of Data Structures

The most important data structure implemented in the software for this project is an array. An array is used to represent the map which the game is centered around. The array consists of class data types which are used to describe the different hex types, whether it is empty, a wall, player or enemy. However, this array can be difficult to manage due to how it corresponds to the hexagonal shaped map. There are multiple ways of representing hexagonal coordinate systems. For this project the offset coordinates [3] are being used which simplifies its usability.



"odd-r" horizontal layout
shoves odd rows right

The next data structure used are the classes that describe the characters in the game. Since there are many shared attributes for players and monsters, there is a parent class called character which holds fields that are in common between the two types as well as methods that they have in common. There are then two child classes that can inherit these fields and methods called MonsterFighter and PlayerFighter. These two data structures are important because they store all the information that is used in the combat algorithms. There are also a different group of classes which correspond to hexagon types and make up the array. These are important for distinguishing between the different types of hexagons on the map as well as discerning which hexagons correspond to which characters, especially those of the same type.

The last important data structure is the priority queue. This is used in two situations. First, when conducting the breadth first search, the queue is used to traverse the graph by keeping track of what has been visited and what still needs to be visited. Then, a priority queue will also be used to keep track of the order in which the players take turns in the game by keeping an unchanging order until players are eliminated.

4.0 Sources Cited:

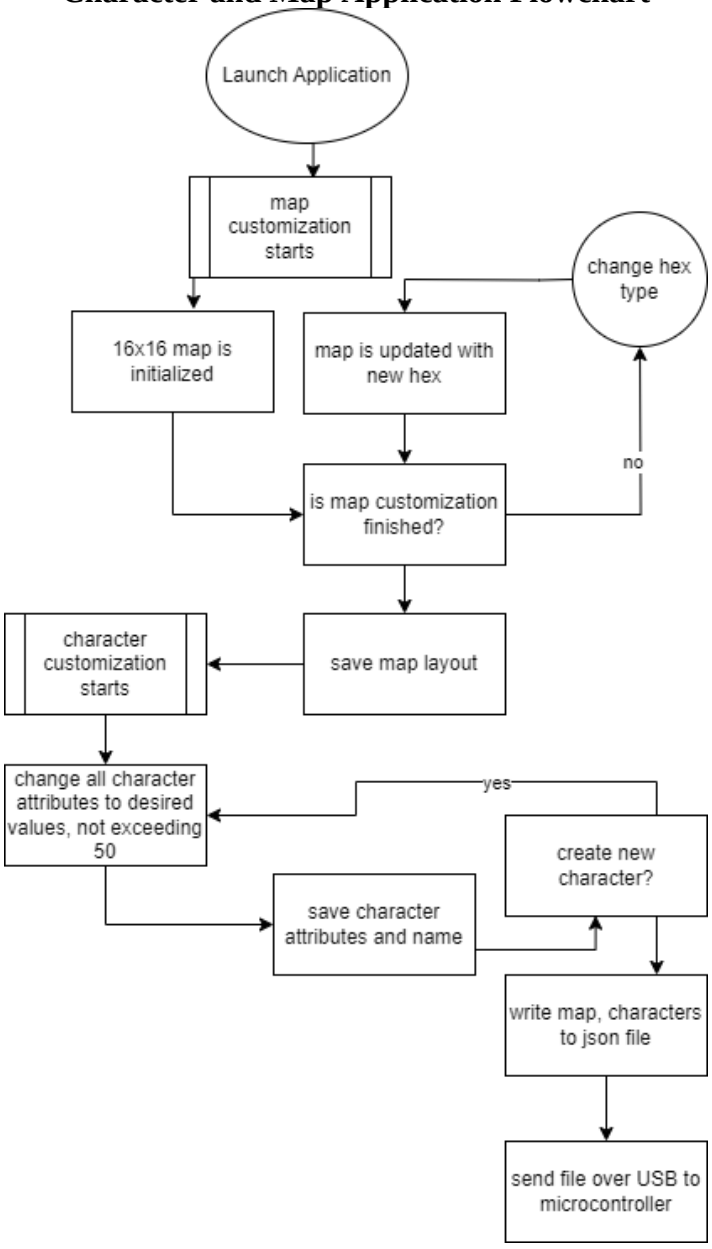
[1] "Legal Information." Accessed: Jan. 25, 2024. [Online]. Available: https://media.wizards.com/2023/downloads/dnd/SRD_CC_v5.1.pdf

[2]GeeksforGeeks, "Breadth First Search or BFS for a Graph - GeeksforGeeks," GeeksforGeeks, Feb. 04, 2019. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[3]A. J. Patel, “Hexagonal Grids,” www.redblobgames.com, 2013. https://www.redblobgames.com/grids/hexagons/#coordinates

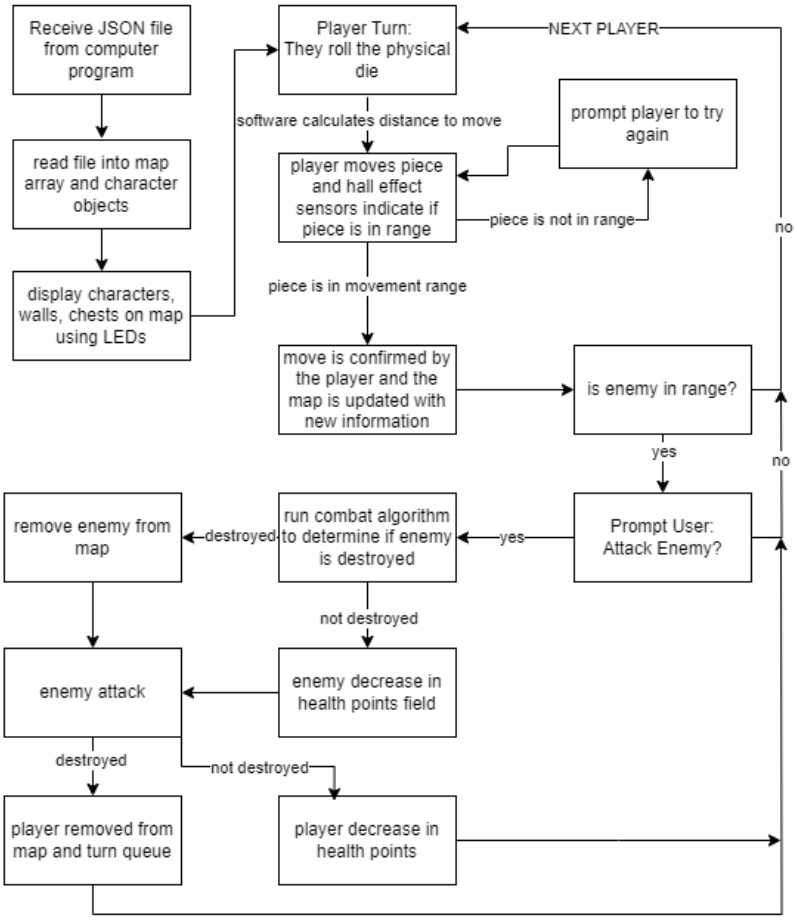
Appendix 1: Program Flowcharts

Character and Map Application Flowchart

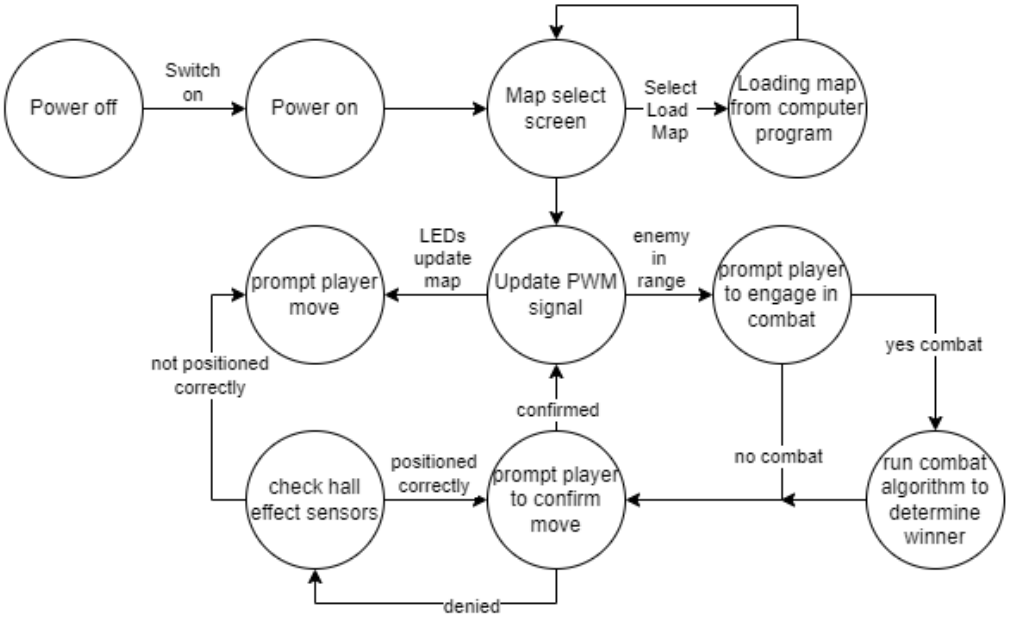


This flowchart shows the flow of the PC application that will be developed to facilitate map and character creation, and then the porting of this information to the microcontroller from the computer.

Gameplay Flowchart



Appendix 2: State Machine Diagrams



This is the state diagram of the microcontroller showing the different states that the microcontroller will be in during the game. Any state that starts with “prompt player to do x” implies the output of the prompt to the LCD screen and the input to the microcontroller through the keypad.