Documentation

Read Me:

- Multiple button presses may be required to move the character on the map screen
- The Start combat button must be pressed before starting combat
- The Start button must be hit when you first load into the map
- JUnit was used in the testing of the project. The program will not run correctly if the user does not have JUnit
- The Entity project is a fully fleshed working version of the game without a GUI
- The GUI project is a slightly stripped down version of the game with a GUI
- The sounds folder is the soundtrack made by Thomas
- The Images folder is the GUI game art made by Thomas

In the project there are 18 java classes this includes: Character, Consumable, Enemy, Entity, EntityMain, Floor, GUI Controller, GUI Driver, Item, Manager, Player, Slime, Stairs, Tile, Trap, Vampire, Weapon, and World. Each class does something specific for the overall game making it run seamlessly.

Manager Class

- **Manager consists of all other classes,** it Manages the Game.
- It loads, generates and stores the Game maps.
- All Characters are stored in an Array called Game Loop
- Run's Game by incrementing turns through all Characters. Each character individually runs its own StartTurn.
- Checks for all collisions of characters and triggers combat when collisions between the player and an enemy occurs
- Manages combat and attack priority.
- Generates enemies and validates initial placement
- Generates consumibles & weapons
- Generates staircases in Wild West world type
- Run GameOver and save after player dies
- Run

Character Class

- Characters is an abstract, it's the parent of Enemy / Player / Stairs.
- Character's decide on their own movement but request it through Manager.
- Consists of at least one weapon.
- Combat requires two characters*

Entity Class

- The parent class of Character. It keeps track of name and location as while as some of the most basic methods like teleport (which moves characters on the map).
- Originally a child class called features would have existed but due to unforeseen complexity it was changed to Stairs and changed to a child of Character.

Item Class

- Originally it was intended you could carry 5 Item's any combination of consumable and weapon, This became a failed Gameplay idea which led to a divide in storage types.

Consumable class

- Consumibles is are one time use items witch destroy themselves on usage.
- Exclusively used by the Player class.

Weapon Class

- Held By Characters
- Int Damages
- Boolean damage type (true = piercing, false = Blunt)
- Piercing damage ignores armor.

Enemy class

- capable of anything the player Could do.
- Moves at Random until it see's player in which it Runs a pathfinding system to follow and track down the player.
- Each enemy type Contains a Single Weapon for gameplay continuity.

Player Class

- Requests inputs from the user // you're the player
- Consists of a array of Consumibles and Weapons
- Can select Consumables thought infinitory
- Can Level up which increase health, and Armor.

World Class

- Consists of an Array Of Floors
- Self Error Checking Tilt's Requesting.

Floor class

- Consists of 2 dimensional Array of tiles

- Generate Rooms based on request parameters.
- Generates a Random Room Layout with tunnels to insure a Fully Accessible Floor

Tile Class

- A boolean value to symbolize a Wall
- A integer Value to which can be used to get the Character on the tile based on their location in Manages Game Loop

Slime & Vampire Class (OverWrites)

- Children of Enemy
- Vampires OverWrite Attack to Heal based on Damage they do
- Slimes OverWrite Attack to dissolve there opponents Armor on Attack

Stairs Class

- Child of Character
- OverWrites interact to teleport other Character on Contect

GUI Controller class

- The setTiles method takes in an array and sets the values of the tiles array to match the array the method has been passed
- The setColour method takes in a Rectangle object and a String, and sets the fill of the Rectangle object to the colour code of the String
- The updateBoard method takes in an array of integers, parses that array using two for loops and, using the setTileImage method, sets the images of the tile Rectangle objects in the grid array to different images depending on the integer values found in the corresponding integer array slots
- The move method takes in an integer, calls the setArray method passing it that integer as a parameter, then branches into an if statement. If the first value in the grid array is null then the grid array slots are set to their corresponding Rectangle tile objects. If not then the program calls the updateBoard method, passing it the tiles array.
- The setArray method takes in an integer and, using two for loops, sets the very last(bottom right) value of the tiles array to the value of the number passed to the method
- The setTileImage method takes in a Rectangle object and an Image object, and sets the fill of the Rectangle to an ImagePattern containing the Image object.
- The loadScene method takes in an ActionEvent and a String

GUI Driver class

- The main method launches the GUI elements of the project
- The start method loads and initializes the initializes the initial scene and stage of the GUI, and displays them with the start menu FXML file loaded
- The handle method is a simple event handler method that sets the value of the keyPress variable to the string value of the last key pressed by the user

Fully Functional Text Version of the Game, Directory

-1	Your Player
0	A Wall
1	Nothing
2	Down Stairs
3	Slime
4	Spider
5	Bat
6	Goblin
7	Ghost
8	Zombie
9	Skeleton
10	Bear
11	Witch
12	Vampire
13	Demon
14	Up Stairs
15	Error Texture