TASKS

Next (MVP vertical slice)

- [x] Create SDL2 init + window + event loop (mouse/keyboard)

- This logic goes into main.rs.

- The window should be 640 x 480 pixels, but I also want there to be a global integer scaling constant so that we can create the window at twice, thrice, etc. the size.

- [x] Create a simple renderer wrapper (clear + present).

- This lives in main.rs.

- Clear the screen with clear blue (0x00FF00)

- Wait for vertical sync in each iteration of the game loop

- [x] Create Game class

- Create a game.rs class which will hold our game logic

- Create an instance of this game class

- Add a step function to the Game class which will take the input and step the simulation. For now it shouldn't take any input, but make a note about it.

- Add a render function to the Game class which will render the game. This should take the renderer as an input

- The render function will need the renderer, so move that that to a separate file

- [x] Create the Map class

- Create a new Map class which will be used to hold our map in map.rs

- The constructor of the class should take the width and height of the map

- The constructor should take an enum specifying which biome the map is using. The available values are

    * Winter - Load the file Data/tiles-winter.png as tileset

    * Wasteland - Load the file Data/tiles-wasteland.png as tileset

    * Swamp - Load the file Data/tiles-swamp.png as tileset

    * Summer - Load the file Data/tiles-summer.png as tileset

- The tiles in each tileset image is 32 x 32 pixels in size, but they are separated by a pixel border on the interior.

- I.e. if we imagine we had a tileset containing 2 x 2 tiles it would be (32 x 2 + 1) x (32 x 2 + 1) = 65 x 65 pixels in size

- When a map is default constructed divide it into 8 x 8 macro tiles in a checker board pattern and use the index 133 for 'white' tiles and 266 for 'black'

- Instantiate an instance of the Map class of size 128 x 128 using the summer biome and pass it as an argument to the constructor of Game

- [x] Initial rendering

- We are going to start building out the render function in Game

- The Game class should do all rendering to a temporary surface of size 640 x 480 pixels.

- Update the Game render function so it takes our scale value as a parameter

- The temporary surface should be blitted and scaled to the target surface provided by the renderer using the scale value

- Update the Map and Tileset classes so that we load the tileset bitmaps from disk

- Add a render function to the Map class which takes all necessary parameters to render to the temporary surface

- This render function should also take an X and Y offset value which we'll use to indicate which part of the map we want to render

- This render function will render 14 x 14 tiles of the map offset by the values provided.

- The top left corner of the "window" we render the map into is at the coordinate 176, 16.

- Add a call to the Map render function with initially hard coded offset values of 0, 0

- [x] Mini map rendering

- In the Map render function we will render a minimap version of the map as well

- The minimap is a "window" of size 128 x 128 pixels with the top left corner at 24, 26

- Render a scaled version of the full map into this window so that it fits and fills the minimap "window"

- Render a gray line rectangle indicating where the current viewport of the non scaled map is

- This rectangle needs to be scaled in the same way as the rest of the minimap

- [x] Fix input

- Collect all input events that are pumped in the event loop into some struct (your choice)

- Pass the struct containing the list of events to the Game step function

- Update the step function so that it begins with doing a dummy parsing of these events

- The only input we care about for now is the arrow keys.

- Add a X and Y view offset member to Game which is modified by the arrow keys

- Clamp the offset so that it can't be less than zero and not larger than map size - view window size

- [x] More input

- We want to be able to scroll the map by holding down the arrow keys

- Add an x velocity member and y velocity member to Game

- When left is pressed the x velocity is decreased and when it is released the value is increased

- The opposite for the right button

- Similar logic for up and down

- Before passing mouse events to the Game class, convert them to unscaled space using correct rounding

- In the Game event loop if the player clicks inside the minimap area move the view port to the correct position

- To do this you will need to figure out what kind of scaling factor is currently applied when fitting the map into the mini map window

- [x] Load an actual PUD file

- We are going to load a PUD file now

- Instead of the default map we're currently using, use the file Data/death-in-the-middle.pud

- Add a constructor to the Map which takes a filename

- Add a parser for the PUD file based on the document in Docs/map-format.md

- Process this map as detailed as you can but discard everything but the following for now (make a comment about this in the relevant documents):

    * The biome (referred to as tileset / terrain theme in the document)

    * The map dimensions

    * The tile map (the 'MTXM' identified)