IB Computer Science IA Report

Battle Map Generator

Criterion C - Development

Criterion C: Development

1.0 Techniques Used to Develop Solution

- File Organisation
  - Application Folder Structure
  - Asset Fetching
    - file_paths.json
    - Extract Files and Assets
    - File Write Helper
  - Map Folder
    - Map Folder to Map Class
    - Map Class to Map Folder
- Algorithm Thinking
  - Map Procedural Generation by Layers
    - Bitmap - 2D Array
  - Structure and Layout
    - Check Cells Connected Expansion
    - Breath-First Search
    - A* Path Finding
    - Recursive Back Tracking
  - Map Rendering
    - Layout Reformating
    - Tile Texture Generation
    - Full Rendering
- Product Extensibility
  - Extendable Cell Types
  - Structure Subclasses
  - Object Subclasses
  - Importable Textures
- Front End
  - PySimpleGui
  - Windows_Setting.json
  - Event Handling
- Form Validation
  - Error Provider

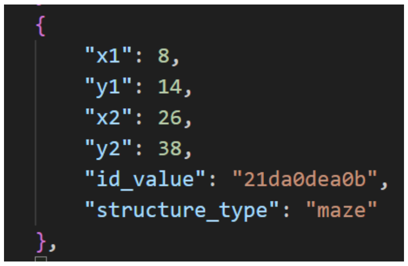
2.0 File Structures

2.1 Map Folder Structure(Crit. B Fig. 6.2)

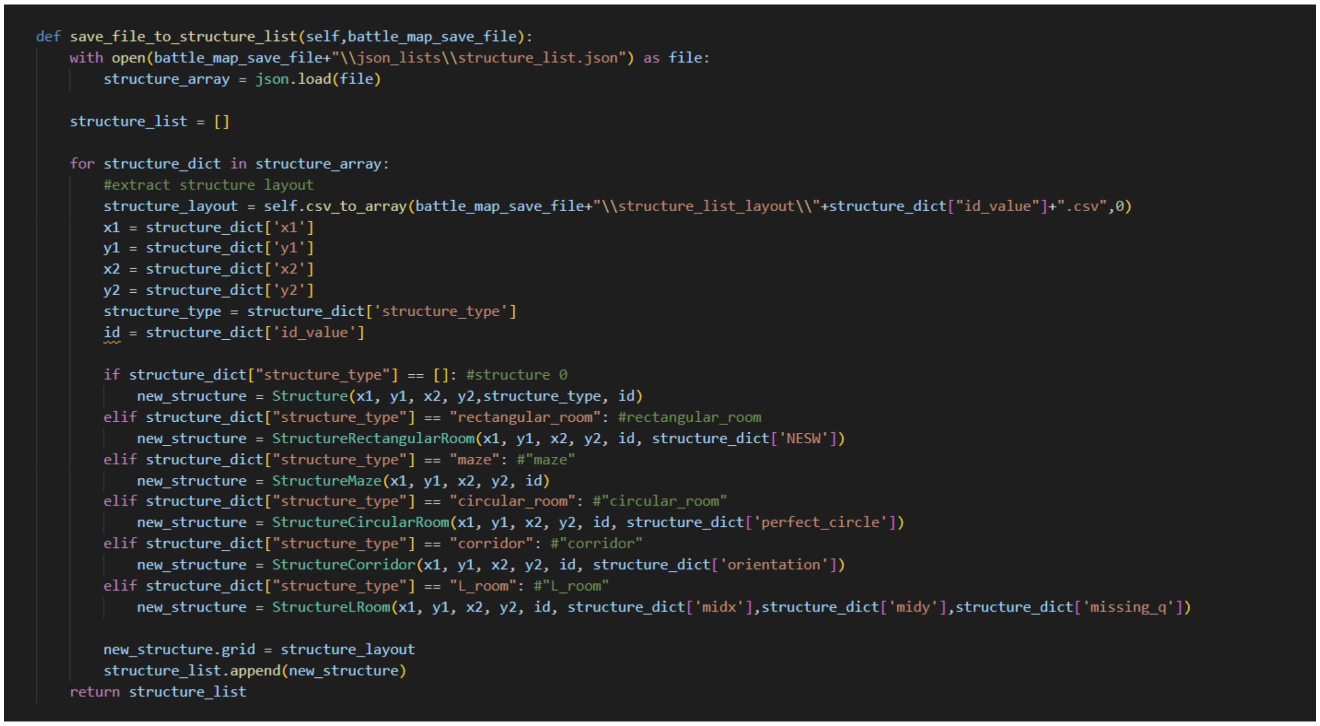
The map folder is organised by the respective data types a Map class would have: CSV for list and 2D bitmaps, png for cache rendering output, dictionaries and aggregated class variables as JSON files. After extraction, each file type would be transformed into its respective data structures in a map class.

Opening a Map Folder - structure_list

The structure class contains a list of variables and a 2D bitmap for its layout, which is stored in structure_list.json and structure_list_layout folder as CSV, where the file names are their ID value for easy data extraction.



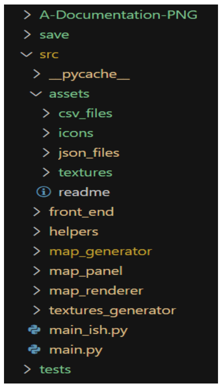
```json
{
    "x1": 8,
    "y1": 14,
    "x2": 26,
    "y2": 38,
    "id_value": "21da0dea0b",
    "structure_type": "maze"
},
```



```python
def save_file_to_structure_list(self,battle_map_save_file):
    with open(battle_map_save_file+"\\json_lists\\structure_list.json") as file:
        structure_array = json.load(file)

    structure_list = []

    for structure_dict in structure_array:
        #extract structure layout
        structure_layout = self.csv_to_array(battle_map_save_file+"\\structure_list_layout\\"+structure_dict["id_value"]+".csv",0)
        x1 = structure_dict['x1']
        y1 = structure_dict['y1']
        x2 = structure_dict['x2']
        y2 = structure_dict['y2']
        structure_type = structure_dict['structure_type']
        id = structure_dict['id_value']

        if structure_dict["structure_type"] == []: #structure 0
            new_structure = Structure(x1, y1, x2, y2,structure_type, id)
        elif structure_dict["structure_type"] == "rectangular_room": #rectangular_room
            new_structure = StructureRectangularRoom(x1, y1, x2, y2, id, structure_dict['NESW'])
        elif structure_dict["structure_type"] == "maze": #"maze"
            new_structure = StructureMaze(x1, y1, x2, y2, id)
        elif structure_dict["structure_type"] == "circular_room": #"circular_room"
            new_structure = StructureCircularRoom(x1, y1, x2, y2, id, structure_dict['perfect_circle'])
        elif structure_dict["structure_type"] == "corridor": #"corridor"
            new_structure = StructureCorridor(x1, y1, x2, y2, id, structure_dict['orientation'])
        elif structure_dict["structure_type"] == "L_room": #"L_room"
            new_structure = StructureLRoom(x1, y1, x2, y2, id, structure_dict['midx'],structure_dict['midy'],structure_dict['missing_q'])

        new_structure.grid = structure_layout
        structure_list.append(new_structure)
    return structure_list
```

2.2 Application File Structure(Crit. B Fig. 6.3)

The application folders are organized in terms of function, where source code is separated from documentation, battle-map save files, and developmental test files. Inside the src, since this project is small, assets are stored separately and categorised by their file types rather than function.

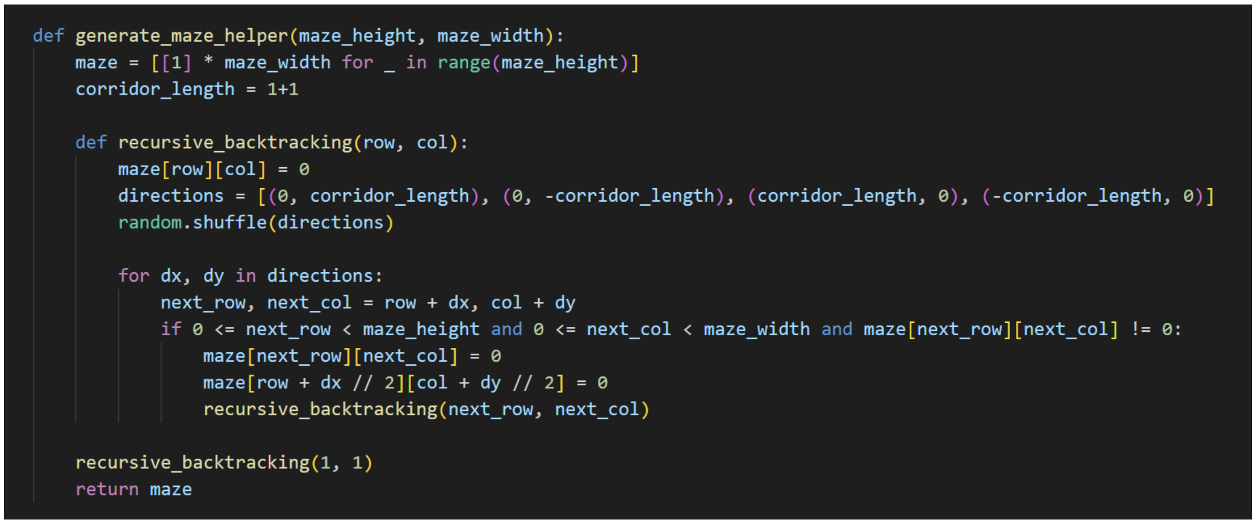


```
> A-Documentation-PNG
> save
v src
  > __pycache__
  v assets
    > csv_files
    > icons
    > json_files
    > textures
    readme
  > front_end
  > helpers
  > map_generator
  > map_panel
  > map_renderer
  > textures_generator
  main_ish.py
  main.py
> tests
```

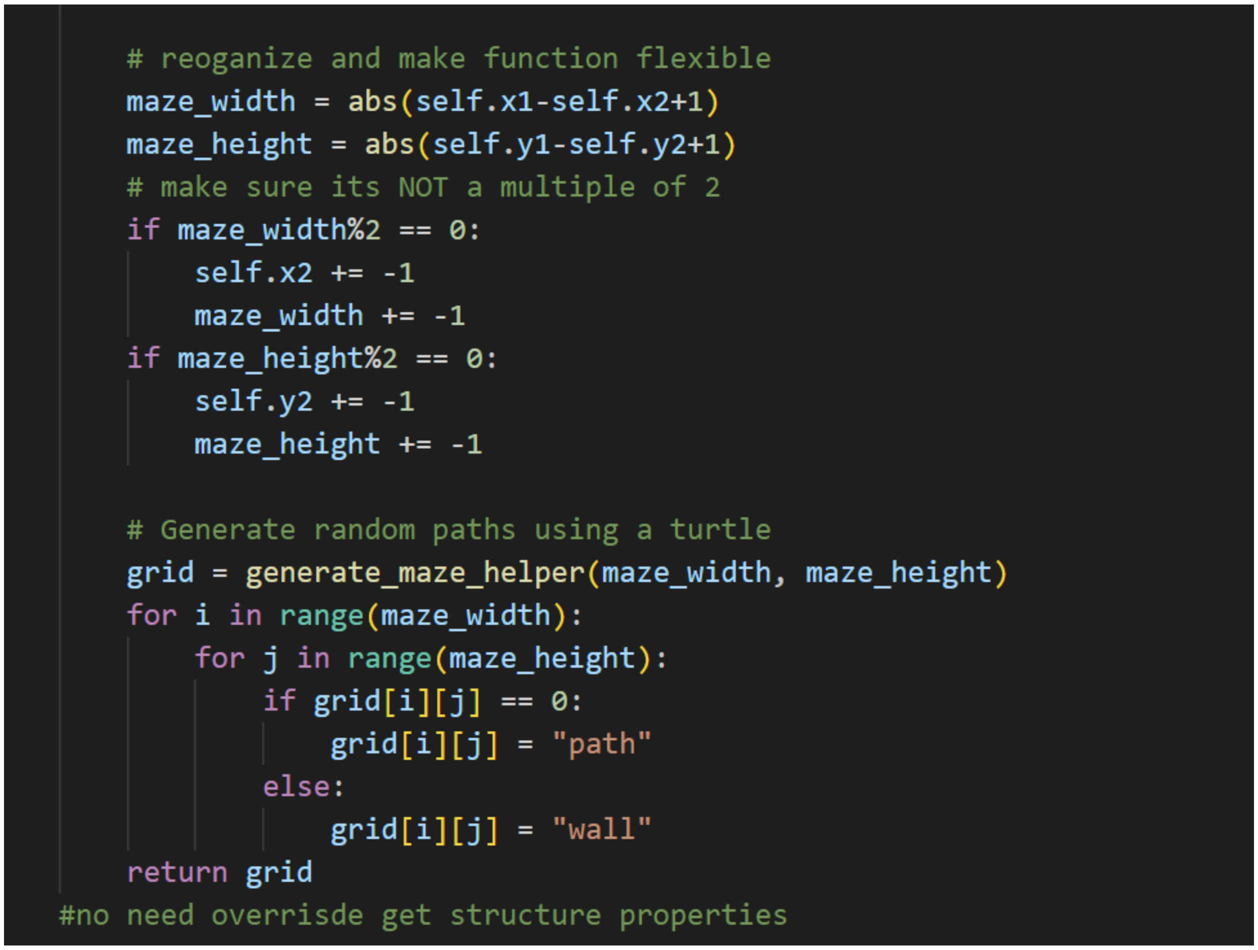
3.0 Algorithm Thinking

3.1 Generate Maze(Crit. B Fig. 5.1)(Abed-Esfahani)

Mazes can be generated using recursive backtracking. It is first generated in 1s and 0s, then converted to path and wall cells.



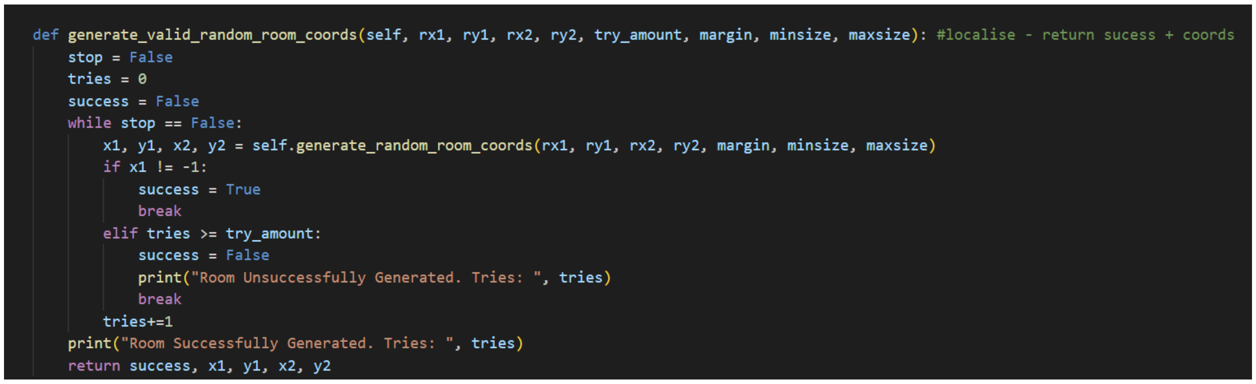
```python
def generate_maze_helper(maze_height, maze_width):
    maze = [[1] * maze_width for _ in range(maze_height)]
    corridor_length = 1+1

    def recursive_backtracking(row, col):
        maze[row][col] = 0
        directions = [(0, corridor_length), (0, -corridor_length), (corridor_length, 0), (-corridor_length, 0)]
        random.shuffle(directions)

        for dx, dy in directions:
            next_row, next_col = row + dx, col + dy
            if 0 <= next_row < maze_height and 0 <= next_col < maze_width and maze[next_row][next_col] != 0:
                maze[next_row][next_col] = 0
                maze[row + dx // 2][col + dy // 2] = 0
                recursive_backtracking(next_row, next_col)

    recursive_backtracking(1, 1)
    return maze
```



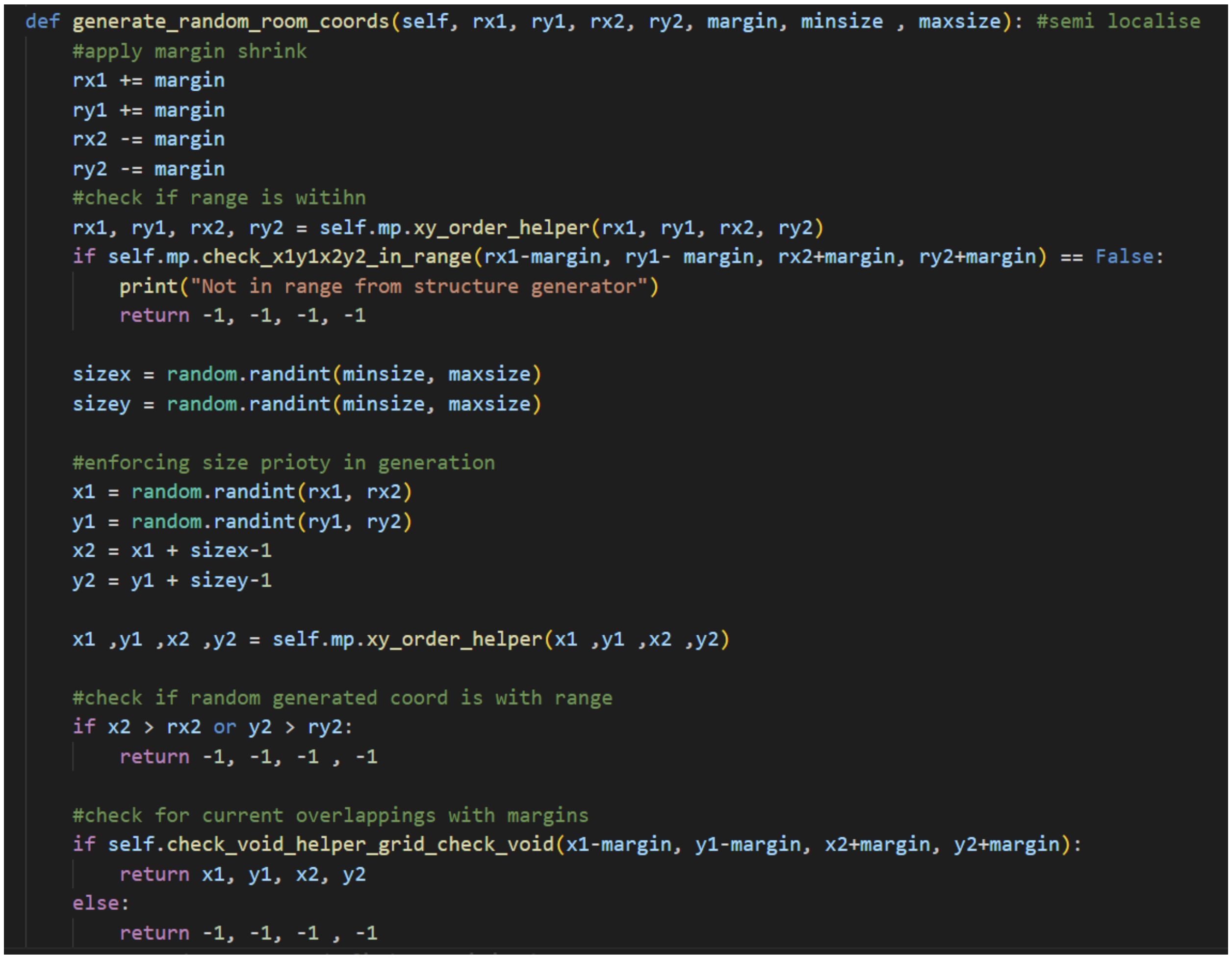
```python
        # reoganize and make function flexible
        maze_width = abs(self.x1-self.x2+1)
        maze_height = abs(self.y1-self.y2+1)
        # make sure its NOT a multiple of 2
        if maze_width%2 == 0:
            self.x2 += -1
            maze_width += -1
        if maze_height%2 == 0:
            self.y2 += -1
            maze_height += -1

        # Generate random paths using a turtle
        grid = generate_maze_helper(maze_width, maze_height)
        for i in range(maze_width):
            for j in range(maze_height):
                if grid[i][j] == 0:
                    grid[i][j] = "path"
                else:
                    grid[i][j] = "wall"
        return grid
    #no need overrisde get structure properties
```

3.2Generate and Place Structures(Crit. B Fig. 1.1)

Using a separate bitmap, these chains of if statements generate the coordinates for a room according to the minimum and maximum room size, before affirming that these coordinates are within the confines of the set range and do not intersect with existing structures.



```python
def generate_valid_random_room_coords(self, rx1, ry1, rx2, ry2, try_amount, margin, minsize, maxsize): #localise - return sucess + coords
    stop = False
    tries = 0
    success = False
    while stop == False:
        x1, y1, x2, y2 = self.generate_random_room_coords(rx1, ry1, rx2, ry2, margin, minsize, maxsize)
        if x1 != -1:
            success = True
            break
        elif tries >= try_amount:
            success = False
            print("Room Unsuccessfully Generated. Tries: ", tries)
            break
        tries+=1
    print("Room Successfully Generated. Tries: ", tries)
    return success, x1, y1, x2, y2
```



```python
def generate_random_room_coords(self, rx1, ry1, rx2, ry2, margin, minsize , maxsize): #semi localise
    #apply margin shrink
    rx1 += margin
    ry1 += margin
    rx2 -= margin
    ry2 -= margin
    #check if range is witihn
    rx1, ry1, rx2, ry2 = self.mp.xy_order_helper(rx1, ry1, rx2, ry2)
    if self.mp.check_x1y1x2y2_in_range(rx1-margin, ry1- margin, rx2+margin, ry2+margin) == False:
        print("Not in range from structure generator")
        return -1, -1, -1, -1

    sizex = random.randint(minsize, maxsize)
    sizey = random.randint(minsize, maxsize)

    #enforcing size prioty in generation
    x1 = random.randint(rx1, rx2)
    y1 = random.randint(ry1, ry2)
    x2 = x1 + sizex-1
    y2 = y1 + sizey-1

    x1 ,y1 ,x2 ,y2 = self.mp.xy_order_helper(x1 ,y1 ,x2 ,y2)

    #check if random generated coord is with range
    if x2 > rx2 or y2 > ry2:
        return -1, -1, -1 , -1

    #check for current overlappings with margins
    if self.check_void_helper_grid_check_void(x1-margin, y1-margin, x2+margin, y2+margin):
        return x1, y1, x2, y2
    else:
        return -1, -1, -1 , -1
```

3.3 Connecting Structures using Paths(Crit. B Fig. 1.3)

To preserve the grid's integrity and offer algorithmic flexibility, the path generation algorithm operates on another grid — check_rooms_connected_grid.

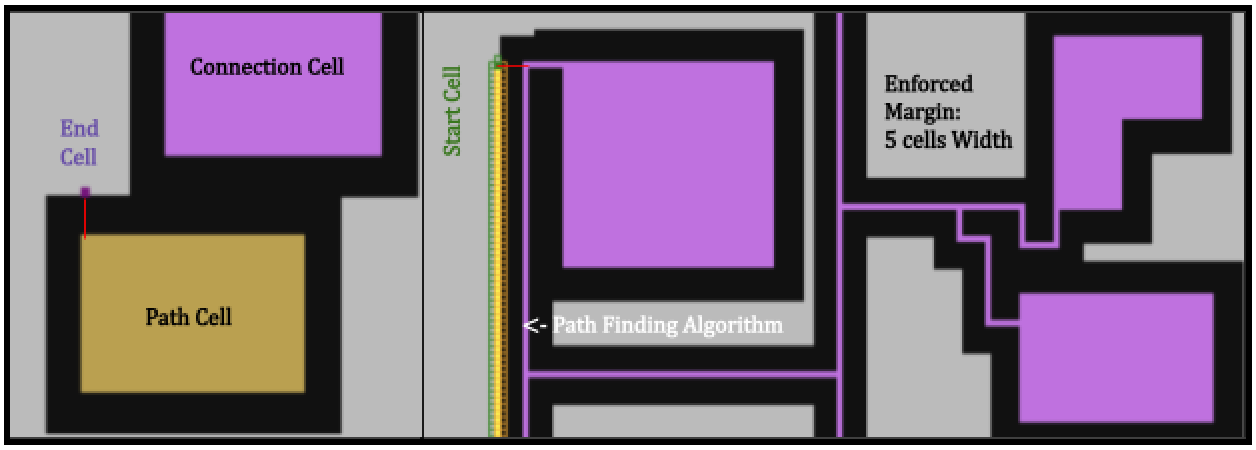
The algorithm first checks if every structure is connected, then establishes margins to ensure maintain aesthetic appeal. It selects a start and end point for each pair of structures, utilizing breadth-first search and the A* path-finding algorithm to create paths between them. This process is repeated until all structures are connected.

check_connected_expansion(Crit. B Fig. 1.5)

To ensure accessibility, paths are created through non-void cells. A connection cell is randomly selected from non-void or non-wall cells. Through a while loop, it expands to neighbouring cells until all relevant cells have been updated. This process determines whether a path needs to be generated.

Enforce Generation Margin

To ensure that paths aren't generated too close to a structure or each other, at the start and after the generation of every path, void values within a set "margin" value will be updated to "wall".

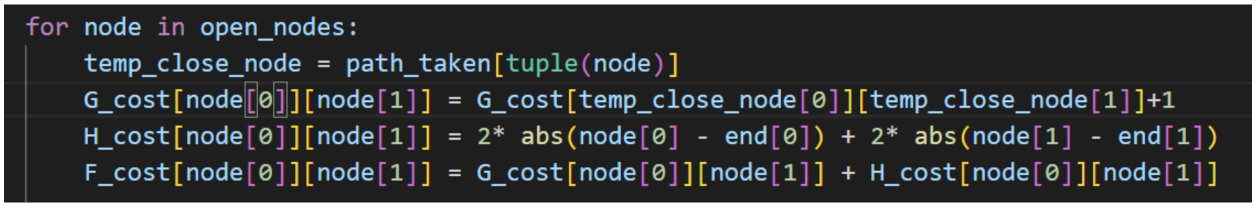


Determine Start and End Point(Crit. B Fig. 1.2)

After setting the enforced margin, start and end points are determined for the path-finding algorithms. Using Breadth-first search and a random void cell, the nearest "connection" and "path" cells are identified as the roots of the start and end points, respectively. Unless there are exposed path cells, these roots are located next to the enforced margins. Hence, a straight path is burrowed through the margin to the nearest void cell, set as the start and end points.

Modified A* Path Finding Algorithm(Crit. B Fig. 1.4, Fig. 5.2)(Belwariar)

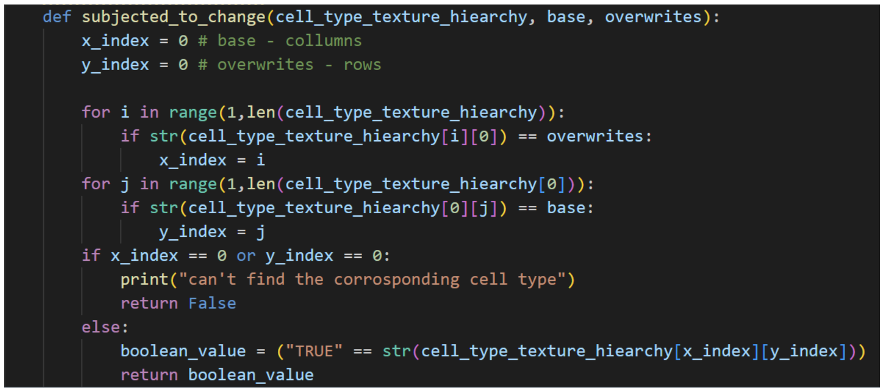
A* is an algorithm that evaluates the heuristic distance from the goal and step distance from the start to determine the shortest path between two paths, where the optimal path is decided by choosing cells that returns the least sum of their two values. In my code, to encourage straight paths with less zig-zag movements, heuristic values of cells diagonal from the endpoint are inflated.



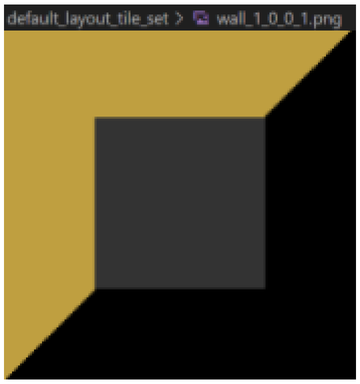
```
for node in open_nodes:
    temp_close_node = path_taken[tuple(node)]
    G_cost[node[0]][node[1]] = G_cost[temp_close_node[0]][temp_close_node[1]]+1
    H_cost[node[0]][node[1]] = 2* abs(node[0] - end[0]) + 2* abs(node[1] - end[1])
    F_cost[node[0]][node[1]] = G_cost[node[0]][node[1]] + H_cost[node[0]][node[1]]
```

3.4 Rendering of the Layout Map

Each cell of the battle-map layout bitmap would contain extra information — its neighbouring cell, in the cardinal direction, which, its texture is subjected to be changed, the number representing the cell type(criterion B fig. 4.3). “Subject to change” represents how water could seep into path cells, or how lava erodes stony walls, which is all represented in the cell_type_texture_hiearchy.csv(criterion B fig. 4.1), which can be tweaked to create unique map rendering.



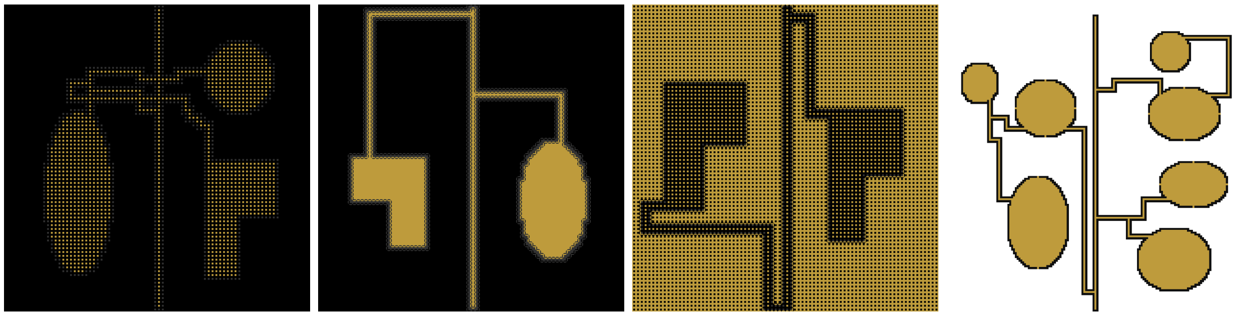
```
def subjected_to_change(cell_type_texture_hiearchy, base, overwrites):
    x_index = 0 # base - collumns
    y_index = 0 # overwrites - rows

    for i in range(1,len(cell_type_texture_hiearchy)):
        if str(cell_type_texture_hiearchy[i][0]) == overwrites:
            x_index = i
    for j in range(1,len(cell_type_texture_hiearchy[0])):
        if str(cell_type_texture_hiearchy[0][j]) == base:
            y_index = j
    if x_index == 0 or y_index == 0:
        print("can't find the corrosponding cell type")
        return False
    else:
        boolean_value = ("TRUE" == str(cell_type_texture_hiearchy[x_index][y_index]))
        return boolean_value
```



There would be a texture for each corresponding cell in “ assets\textures\”, where missing textures would be generated using TileSetGenerator(criterion B fig. 2.2).

3.5 Sample Map Renderings

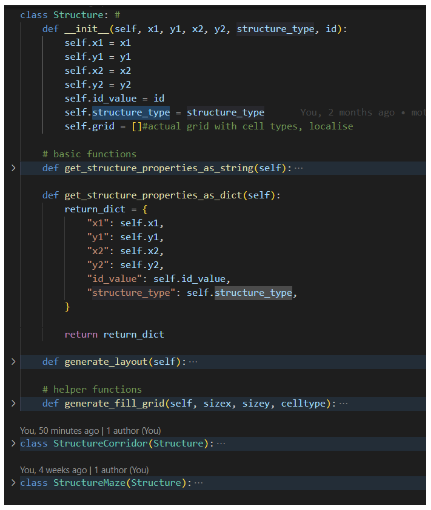
By changing the data in cell_type_texture_hiearchy.csv, here are some unique map renderings.



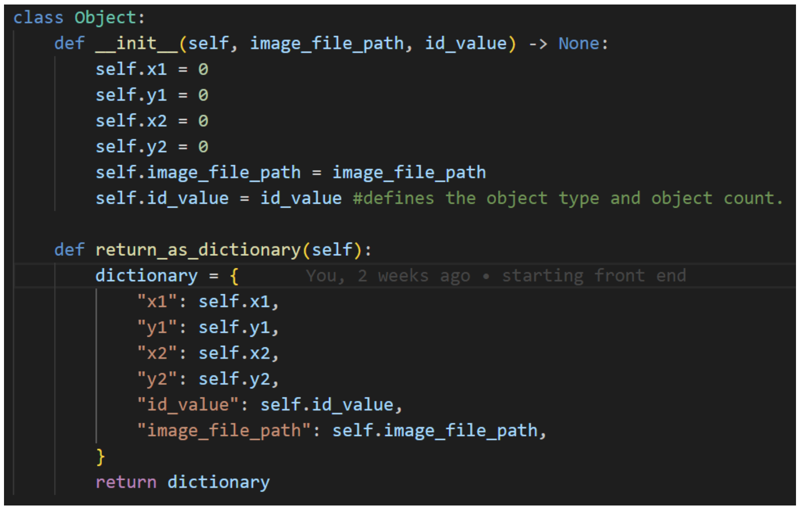
4.0 Product Extensibility

4.1 Structure Classes

Structure subclasses (Crit. B Fig. 2.1, Fig. 4.4) allow numerous structure types and layouts to be added by modders, where overriding allows handling different structures en-masse, whilst Object class requires no subclass as they are simply image overlays.



```
class Structure: #
    def __init__(self, x1, y1, x2, y2, structure_type, id):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.id_value = id
        self.structure_type = structure_type      You, 2 months ago • mot
        self.grid = []#actual grid with cell types, localise

    # basic functions
>   def get_structure_properties_as_string(self):···

    def get_structure_properties_as_dict(self):
        return_dict = {
            "x1": self.x1,
            "y1": self.y1,
            "x2": self.x2,
            "y2": self.y2,
            "id_value": self.id_value,
            "structure_type": self.structure_type,
        }

        return return_dict

>   def generate_layout(self):···

    # helper functions
>   def generate_fill_grid(self, sizex, sizey, celltype):···

You, 50 minutes ago | 1 author (You)
> class StructureCorridor(Structure):···

You, 4 weeks ago | 1 author (You)
> class StructureMaze(Structure):···
```

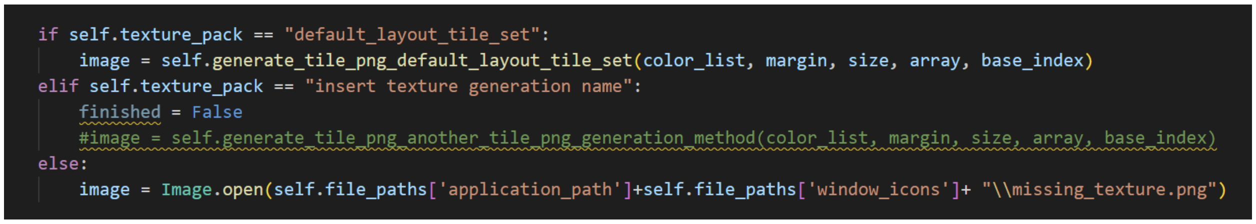


```
class Object:
    def __init__(self, image_file_path, id_value) -> None:
        self.x1 = 0
        self.y1 = 0
        self.x2 = 0
        self.y2 = 0
        self.image_file_path = image_file_path
        self.id_value = id_value #defines the object type and object count.

    def return_as_dictionary(self):
        dictionary = {       You, 2 weeks ago • starting front end
            "x1": self.x1,
            "y1": self.y1,
            "x2": self.x2,
            "y2": self.y2,
            "id_value": self.id_value,
            "image_file_path": self.image_file_path,
        }
        return dictionary
```

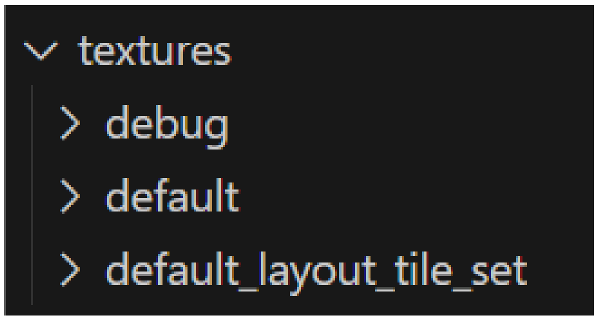
Likewise with cell types, though they need not be classes.(Crit. B Fig. 4.3). 3 types of water and hazards have not been properly implemented as they rely on the texture, though can be added in later on.

4.2 Individual Texture Packs

Texture packs can be generated using a pre-made method, as the current one or other modders can easily add, otherwise, missing textures will be displayed with a missing texture icon.



```
if self.texture_pack == "default_layout_tile_set":
    image = self.generate_tile_png_default_layout_tile_set(color_list, margin, size, array, base_index)
elif self.texture_pack == "insert texture generation name":
    finished = False
    #image = self.generate_tile_png_another_tile_png_generation_method(color_list, margin, size, array, base_index)
else:
    image = Image.open(self.file_paths['application_path']+self.file_paths['window_icons']+ "\\missing_texture.png")
```



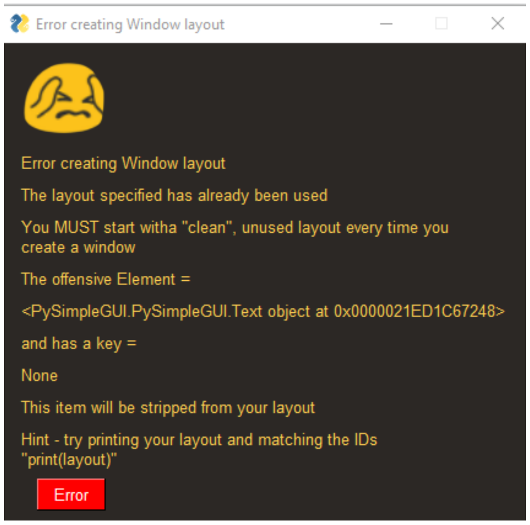
Missing
Texture

5.0 Front End

5.1 Front End Handling

Bypass Layout Reuse Error(The CS Classroom)

A new layout list element must be created every time it is called. A central function controls which window is opened.(image from Eli)



Error creating Window layout

Error creating Window layout

The layout specified has already been used

You MUST start witha "clean", unused layout every time you create a window

The offensive Element =

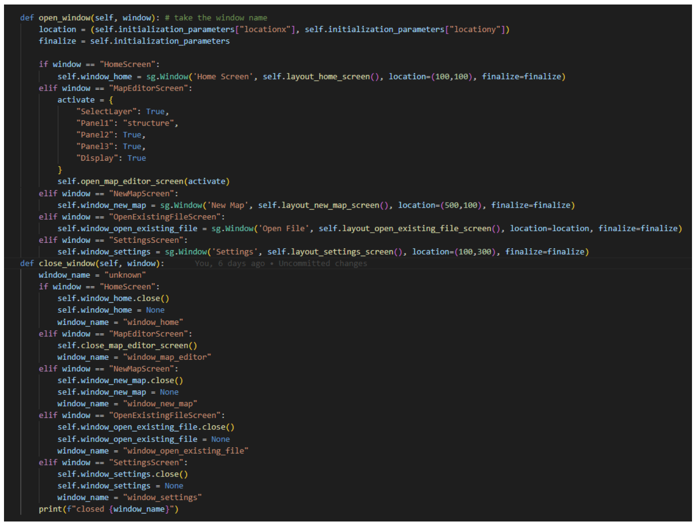
<PySimpleGUI.PySimpleGUI.Text object at 0x0000021ED1C67248>

and has a key =

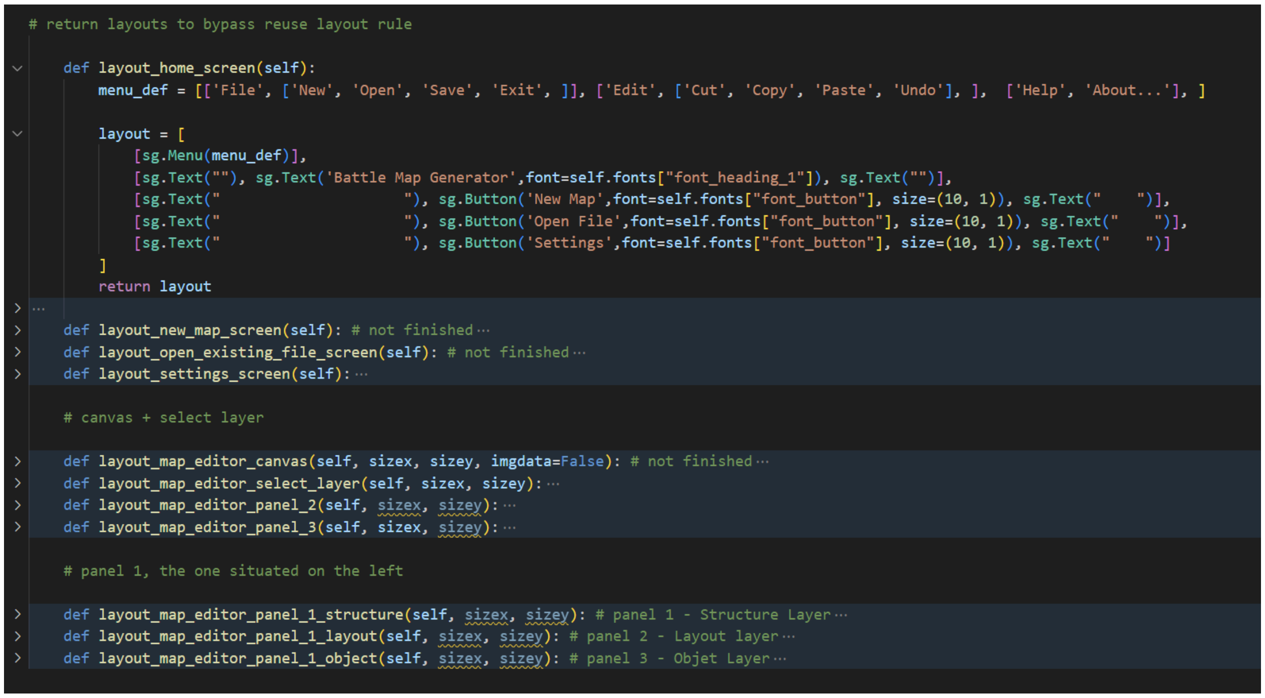
None

This item will be stripped from your layout

Hint - try printing your layout and matching the IDs "print(layout)"



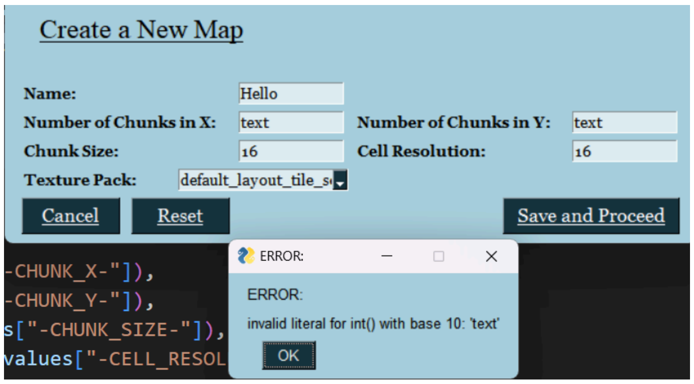
```python
def open_window(self, window): # take the window name
    location = (self.initialization_parameters["locationx"], self.initialization_parameters["locationy"])
    finalize = self.initialization_parameters

    if window == "HomeScreen":
        self.window_home = sg.Window('Home Screen', self.layout_home_screen(), location=(100,100), finalize=finalize)
    elif window == "MapEditorScreen":
        activate = {
            "SelectLayer": True,
            "Panel1": "structure",
            "Panel2": True,
            "Panel3": True,
            "Display": True
        }
        self.open_map_editor_screen(activate)
    elif window == "NewMapScreen":
        self.window_new_map = sg.Window('New Map', self.layout_new_map_screen(), location=(500,100), finalize=finalize)
    elif window == "OpenExistingFileScreen":
        self.window_open_existing_file = sg.Window('Open File', self.layout_open_existing_file_screen(), location=location, finalize=finalize)
    elif window == "SettingsScreen":
        self.window_settings = sg.Window('Settings', self.layout_settings_screen(), location=(100,300), finalize=finalize)
def close_window(self, window):      You, 6 days ago • Uncommitted changes
    window_name = "unknown"
    if window == "HomeScreen":
        self.window_home.close()
        self.window_home = None
        window_name = "window_home"
    elif window == "MapEditorScreen":
        self.close_map_editor_screen()
        window_name = "window_map_editor"
    elif window == "NewMapScreen":
        self.window_new_map.close()
        self.window_new_map = None
        window_name = "window_new_map"
    elif window == "OpenExistingFileScreen":
        self.window_open_existing_file.close()
        self.window_open_existing_file = None
        window_name = "window_open_existing_file"
    elif window == "SettingsScreen":
        self.window_settings.close()
        self.window_settings = None
        window_name = "window_settings"
    print(f"closed {window_name}")
```



```python
# return layouts to bypass reuse layout rule

    def layout_home_screen(self):
        menu_def = [['File', ['New', 'Open', 'Save', 'Exit', ]], ['Edit', ['Cut', 'Copy', 'Paste', 'Undo'], ],  ['Help', 'About...'], ]

        layout = [
            [sg.Menu(menu_def)],
            [sg.Text(""), sg.Text('Battle Map Generator',font=self.fonts["font_heading_1"]), sg.Text("")],
            [sg.Text("                    "), sg.Button('New Map',font=self.fonts["font_button"], size=(10, 1)), sg.Text("    ")],
            [sg.Text("                    "), sg.Button('Open File',font=self.fonts["font_button"], size=(10, 1)), sg.Text("    ")],
            [sg.Text("                    "), sg.Button('Settings',font=self.fonts["font_button"], size=(10, 1)), sg.Text("    ")]
        ]
        return layout
...
    def layout_new_map_screen(self): # not finished...
    def layout_open_existing_file_screen(self): # not finished...
    def layout_settings_screen(self):...

    # canvas + select layer

    def layout_map_editor_canvas(self, sizex, sizey, imgdata=False): # not finished...
    def layout_map_editor_select_layer(self, sizex, sizey):...
    def layout_map_editor_panel_2(self, sizex, sizey):...
    def layout_map_editor_panel_3(self, sizex, sizey):...

    # panel 1, the one situated on the left

    def layout_map_editor_panel_1_structure(self, sizex, sizey): # panel 1 - Structure Layer...
    def layout_map_editor_panel_1_layout(self, sizex, sizey): # panel 2 - Layout layer...
    def layout_map_editor_panel_1_object(self, sizex, sizey): # panel 3 - Objet Layer...
```

5.2 Form Validation(w3school)

Form validation is often used to ensure user inputs are valid, such as for strictly numerical values, required fields or naming conventions.

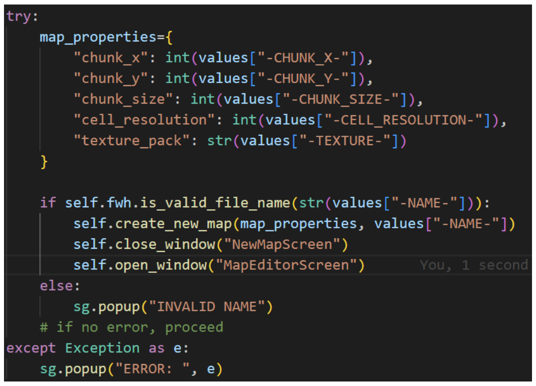


Create a New Map

Name:	Hello		
Number of Chunks in X:	text	Number of Chunks in Y:	text
Chunk Size:	16	Cell Resolution:	16
Texture Pack:	default_layout_tile_s		

```
-CHUNK_X-"]),
-CHUNK_Y-"]),
s["-CHUNK_SIZE-"]),
values["-CELL_RESOL
```

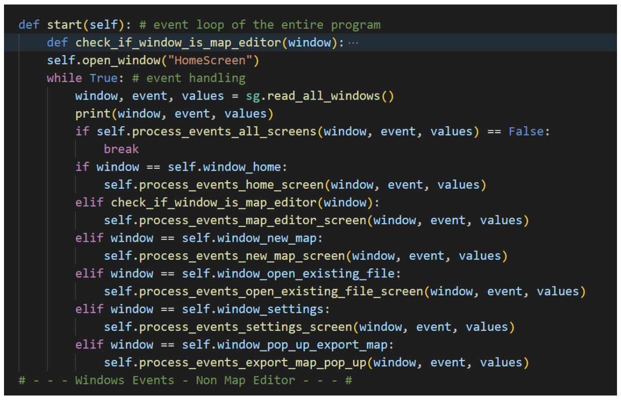
```
ERROR:

ERROR:
invalid literal for int() with base 10: 'text'
OK
```



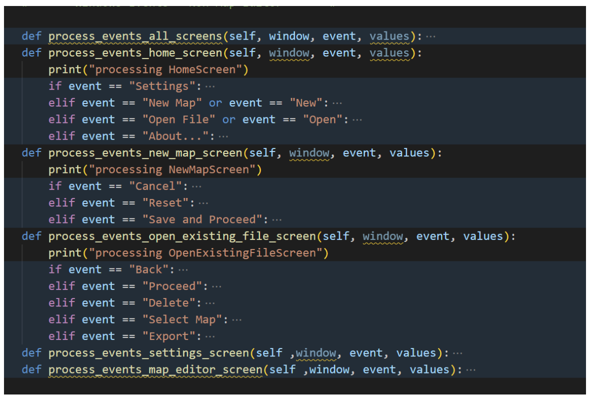
```
try:
    map_properties={
        "chunk_x": int(values["-CHUNK_X-"]),
        "chunk_y": int(values["-CHUNK_Y-"]),
        "chunk_size": int(values["-CHUNK_SIZE-"]),
        "cell_resolution": int(values["-CELL_RESOLUTION-"]),
        "texture_pack": str(values["-TEXTURE-"])
    }

    if self.fwh.is_valid_file_name(str(values["-NAME-"])):
        self.create_new_map(map_properties, values["-NAME-"])
        self.close_window("NewMapScreen")
        self.open_window("MapEditorScreen")        You, 1 second
    else:
        sg.popup("INVALID NAME")
    # if no error, proceed
except Exception as e:
    sg.popup("ERROR: ", e)
```

Central Event Handling - Laddering

“If chains” are used to differentiate window events and handle their output.



```
def start(self): # event loop of the entire program
    def check_if_window_is_map_editor(window):···
    self.open_window("HomeScreen")
    while True: # event handling
        window, event, values = sg.read_all_windows()
        print(window, event, values)
        if self.process_events_all_screens(window, event, values) == False:
            break
        if window == self.window_home:
            self.process_events_home_screen(window, event, values)
        elif check_if_window_is_map_editor(window):
            self.process_events_map_editor_screen(window, event, values)
        elif window == self.window_new_map:
            self.process_events_new_map_screen(window, event, values)
        elif window == self.window_open_existing_file:
            self.process_events_open_existing_file_screen(window, event, values)
        elif window == self.window_settings:
            self.process_events_settings_screen(window, event, values)
        elif window == self.window_pop_up_export_map:
            self.process_events_export_map_pop_up(window, event, values)
# - - - Windows Events - Non Map Editor - - - #
```

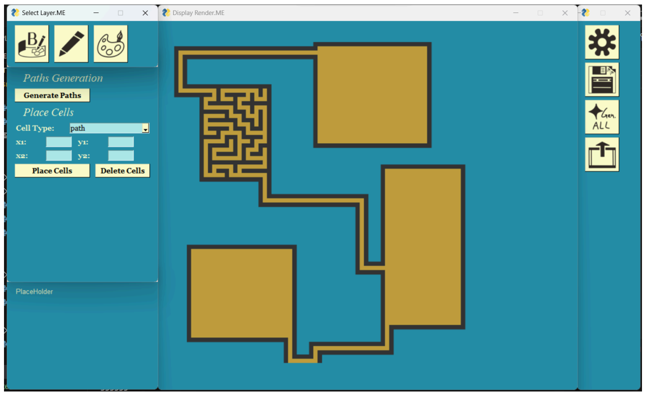


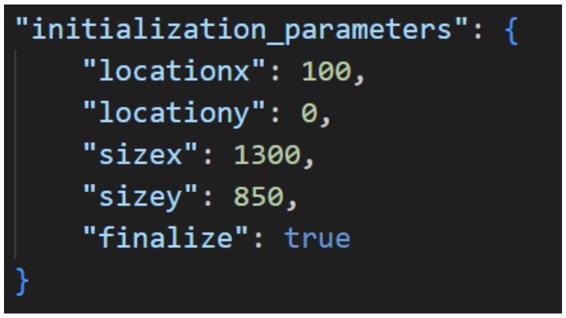
```
def process_events_all_screens(self, window, event, values):···
def process_events_home_screen(self, window, event, values):
    print("processing HomeScreen")
    if event == "Settings":···
    elif event == "New Map" or event == "New":···
    elif event == "Open File" or event == "Open":···
    elif event == "About...":···
def process_events_new_map_screen(self, window, event, values):
    print("processing NewMapScreen")
    if event == "Cancel":···
    elif event == "Reset":···
    elif event == "Save and Proceed":···
def process_events_open_existing_file_screen(self, window, event, values):
    print("processing OpenExistingFileScreen")
    if event == "Back":···
    elif event == "Proceed":···
    elif event == "Delete":···
    elif event == "Select Map":···
    elif event == "Export":···
def process_events_settings_screen(self ,window, event, values):···
def process_events_map_editor_screen(self ,window, event, values):···
```

5.3 Front End Functions

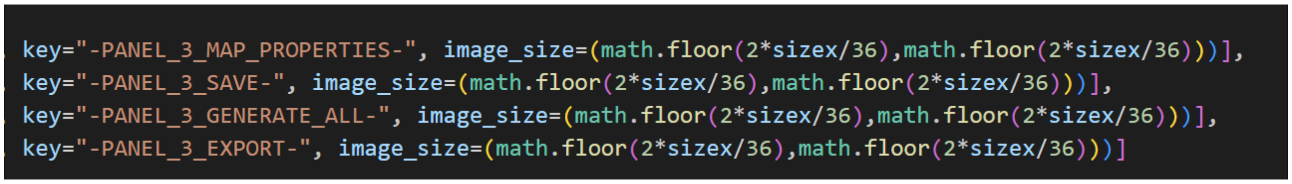
Modular Map Editor Screen(Crit. B Fig. 3.0)

To create panels on the map editor, several windows are formatted like so. This window handling is scalable since their formatting is based on the ratio of the available window size.





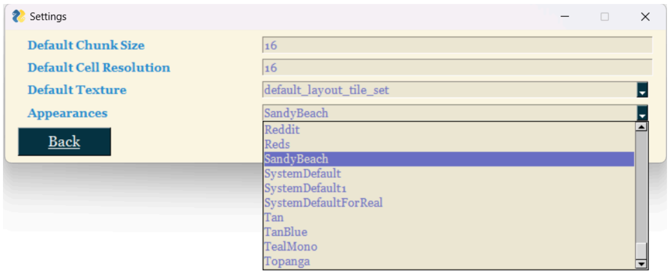
```
"initialization_parameters": {
    "locationx": 100,
    "locationy": 0,
    "sizex": 1300,
    "sizey": 850,
    "finalize": true
}
```



```
key="-PANEL_3_MAP_PROPERTIES-", image_size=(math.floor(2*sizex/36),math.floor(2*sizex/36)))],
key="-PANEL_3_SAVE-", image_size=(math.floor(2*sizex/36),math.floor(2*sizex/36)))],
key="-PANEL_3_GENERATE_ALL-", image_size=(math.floor(2*sizex/36),math.floor(2*sizex/36)))],
key="-PANEL_3_EXPORT-", image_size=(math.floor(2*sizex/36),math.floor(2*sizex/36)))]
```

Adjustable Aesthetics(Crit. B Fig. 3.0)

Aesthetics can be altered, which is saved in the settings file under src\assets\json_files.

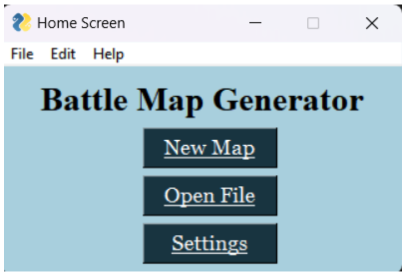


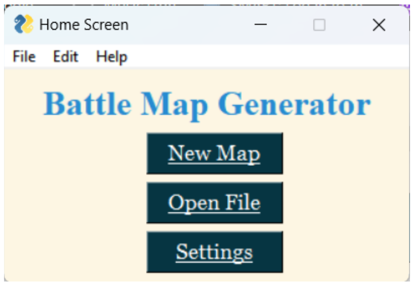
Settings

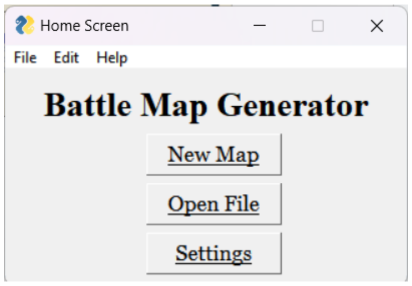
Default Chunk Size	16
Default Cell Resolution	16
Default Texture	default_layout_tile_set
Appearances	SandyBeach

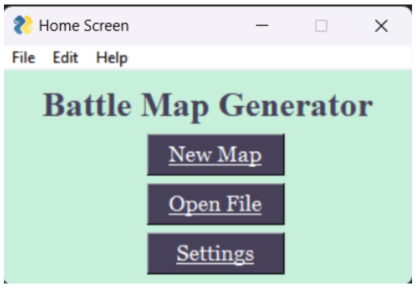
Back

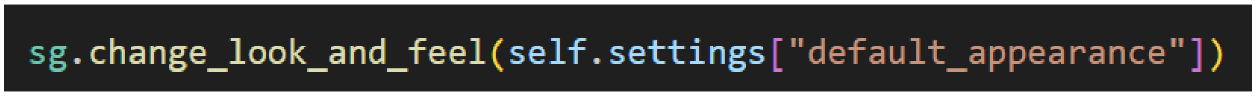
Reddit
Reds
SandyBeach
SystemDefault
SystemDefault1
SystemDefaultForReal
Tan
TanBlue
TealMono
Topanga







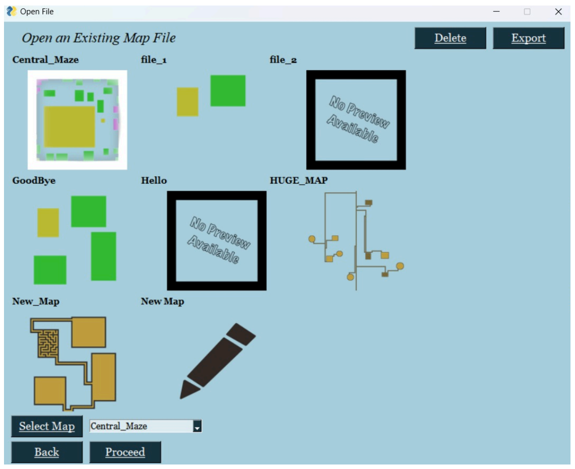


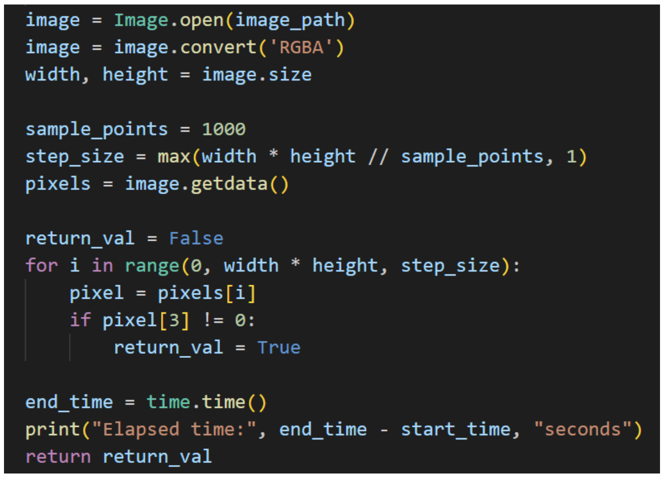


```
sg.change_look_and_feel(self.settings["default_appearance"])
```

Map Previews(Crit. B Fig. 3.0)

Cache image files of maps are fetched to create previews, which are placed in a 3 by X grid. Efficient sampling is used to fetch image files which are not empty.

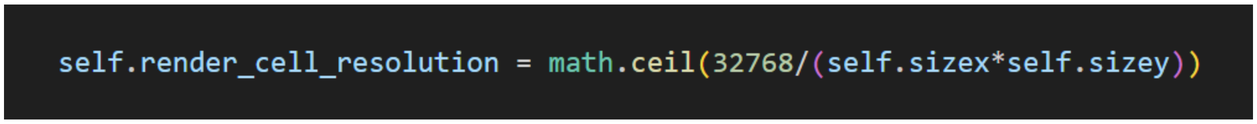




```
image = Image.open(image_path)
image = image.convert('RGBA')
width, height = image.size

sample_points = 1000
step_size = max(width * height // sample_points, 1)
pixels = image.getdata()

return_val = False
for i in range(0, width * height, step_size):
    pixel = pixels[i]
    if pixel[3] != 0:
        return_val = True

end_time = time.time()
print("Elapsed time:", end_time - start_time, "seconds")
return return_val
```

Streamline Renderings

When rendering for the editor, a separate resolution would be used based on this equation.



```
self.render_cell_resolution = math.ceil(32768/(self.sizex*self.sizey))
```

Bibliography

Abed-Esfahani, Aryan. "Maze Generation — Recursive Backtracking." Medium, 5 Jan. 2021, ary-anab.medium.com/maze-generation-recursive-backtracking-5981bc5cc766.

Amangeldi, Alimzhan. "Graph Traversal. Breadth First Search (BFS) and Depth First Search (DFS)." Medium, 9 May 2022, https://medium.com/@211107001/graph-traversal-breadth-first-search-bfs-and-depth-first-search-dfs-d775d0610917.

Belwariar, Rachit. "A* Search Algorithm." GeeksforGeeks, https://www.geeksforgeeks.org/a-search-algorithm/. Accessed 17 February 2024.

Eli. "Issue when re-using layout in PySimpleGUI." Stack Overflow, 21 Nov. 2020, https://stackoverflow.com/questions/64944805/issue-when-re-using-layout-in-pysimplegui. Accessed 21 Nov. 2020.

“PySimpleGUI Documentation." PySimpleGUI, 2024, docs.pysimplegui.com/en/latest/.

Python Software Foundation. "os — Miscellaneous operating system interfaces — Python 3.12.2 documentation." Python 3.12.2 Documentation. Accessed March 12, 2024. https://docs.python.org/3/library/os.html.

"Python Try Except." W3Schools, www.w3schools.com/python/python_try_except.asp.

"secrets — Generate secure random numbers for managing secrets" Website: Python 3.12.2 documentation URL: https://docs.python.org/3/library/secrets.html Accessed: March 11, 2024

"Python | Using 2D arrays/lists the right way" Website: GeeksforGeeks URL: https://www.geeksforgeeks.org/python-using-2d-arrays-lists-the-right-way/, Accessed: December 20th, 2023

This Candidate, “Equation for Ovals” https://www.desmos.com/calculator/tkvm5buuoh, accessed 30th December.

The CS Classroom, "PySimpleGUI - Multiple Window Applications." YouTube, uploaded 18 Nov 2021, https://youtu.be/Zv4o6CGlhbE?si=tPHH0NeTsB5RTHLx