LAB 1 :

Student name: Donfack Tsopfack Yves Dylane

Step 1 - Environment Creation:

The environment in this code operates based on a multi-layered grid system generated through an app that I created, called **EE**. Each layer in this environment serves a specific purpose, such as containing objects, obstacles, the agent, or visual elements for aesthetics. The code for the app can be found on my github repos (https://github.com/yvesdylane/EnviromentEditor). The following details each layer and its purpose:

Running EE to Create Environment:

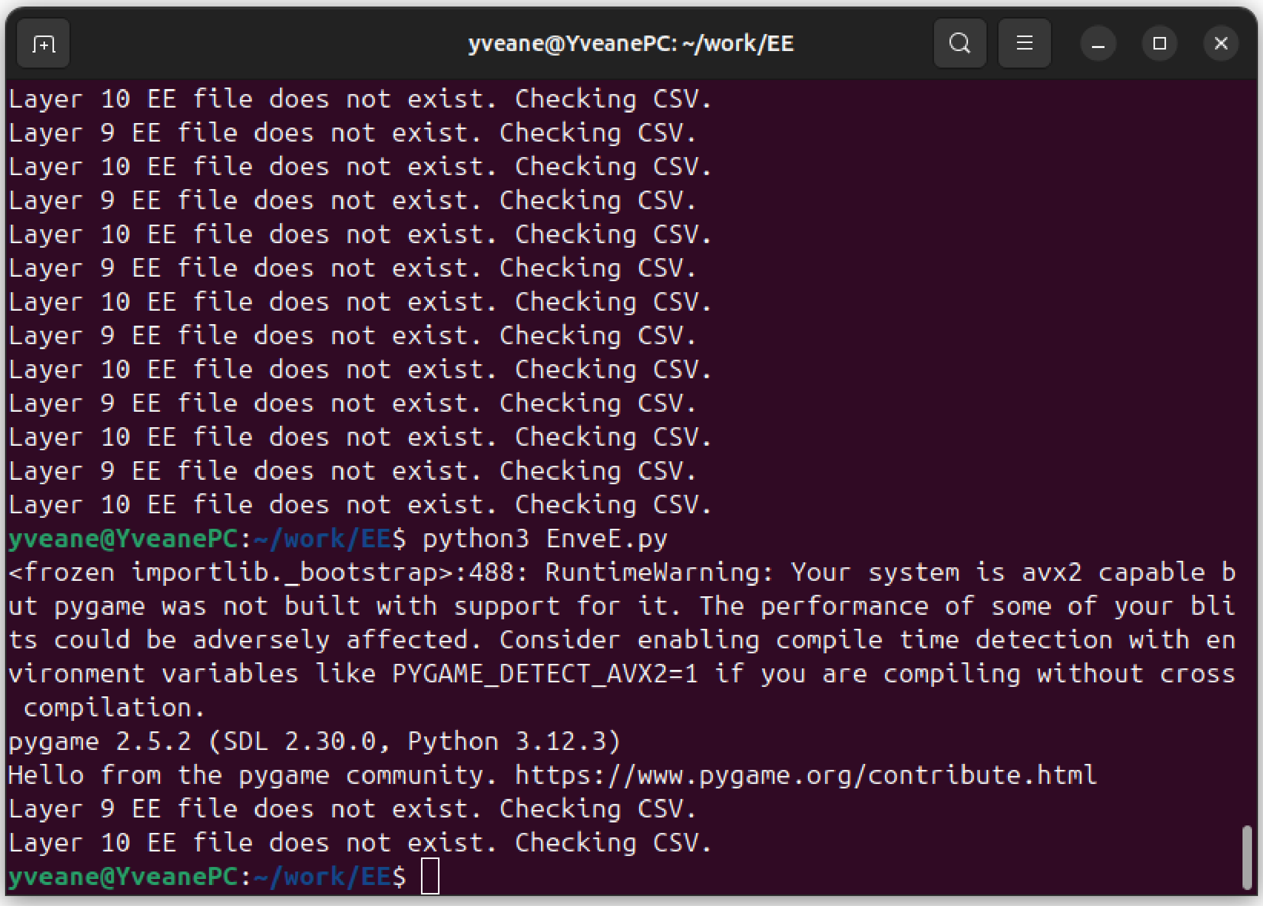

Figure 1: Running and executind EE(EnveE)

1. **Layer 2 (Road):** This layer represents navigable paths where the agent can move freely.
2. **Layer 3 (Target):** The objective or target area that the agent aims to reach.
3. **Layer 5 (Agent):** Contains the agent's current position.
4. **Layer 8 (Obstacles):** Contains obstacles the agent must avoid during pathfinding.
5. **Other Layers (Decorations):** Reserved for visual elements in the environment.

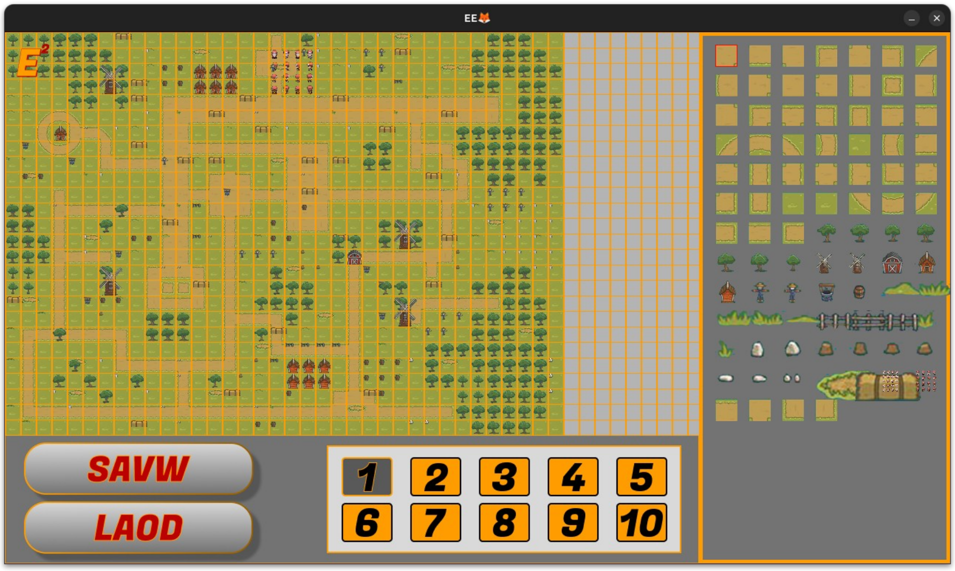

Figure 2: Creation of Environment using EE

Each number in these layers directly corresponds to an image in the image set managed by the EE application, which allows the game to load and display environmental elements correctly based on the agent's actions.

Each layer are now loaded and display by world_loader.py.

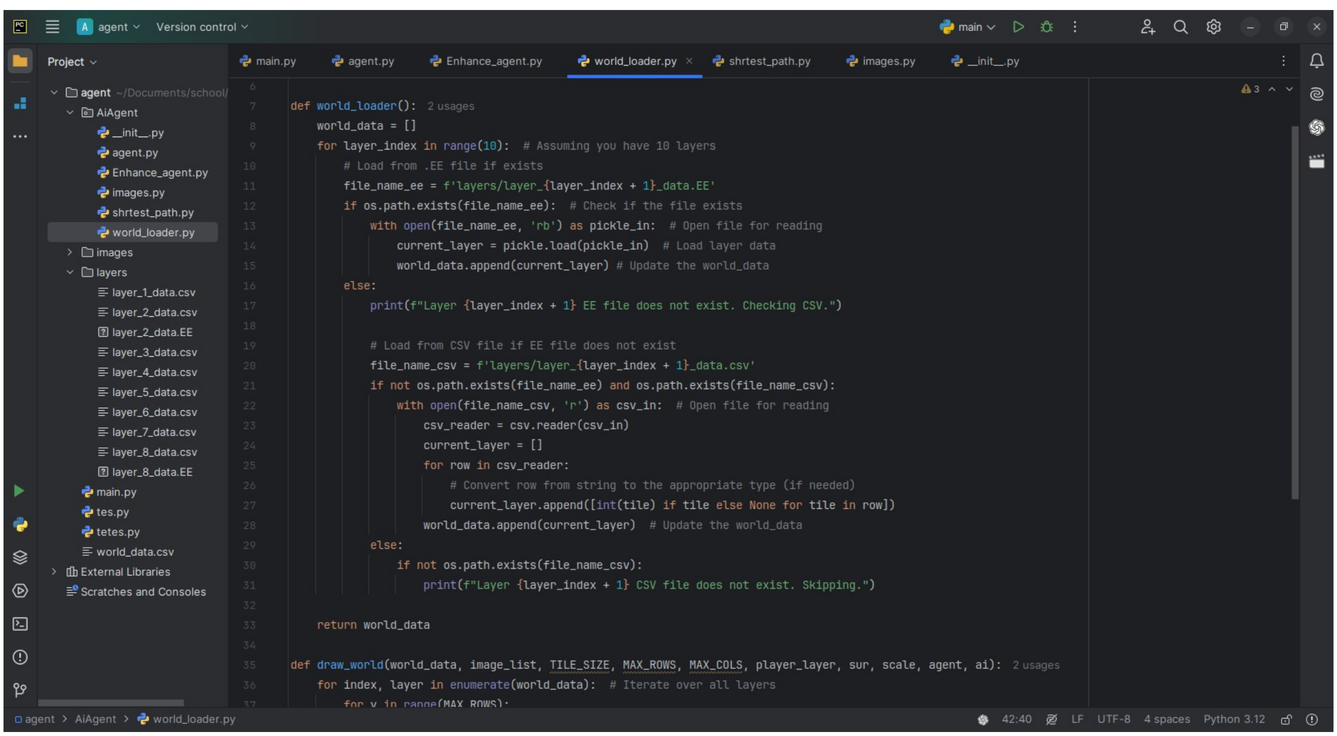

Figure 3: loading Environment from the layers

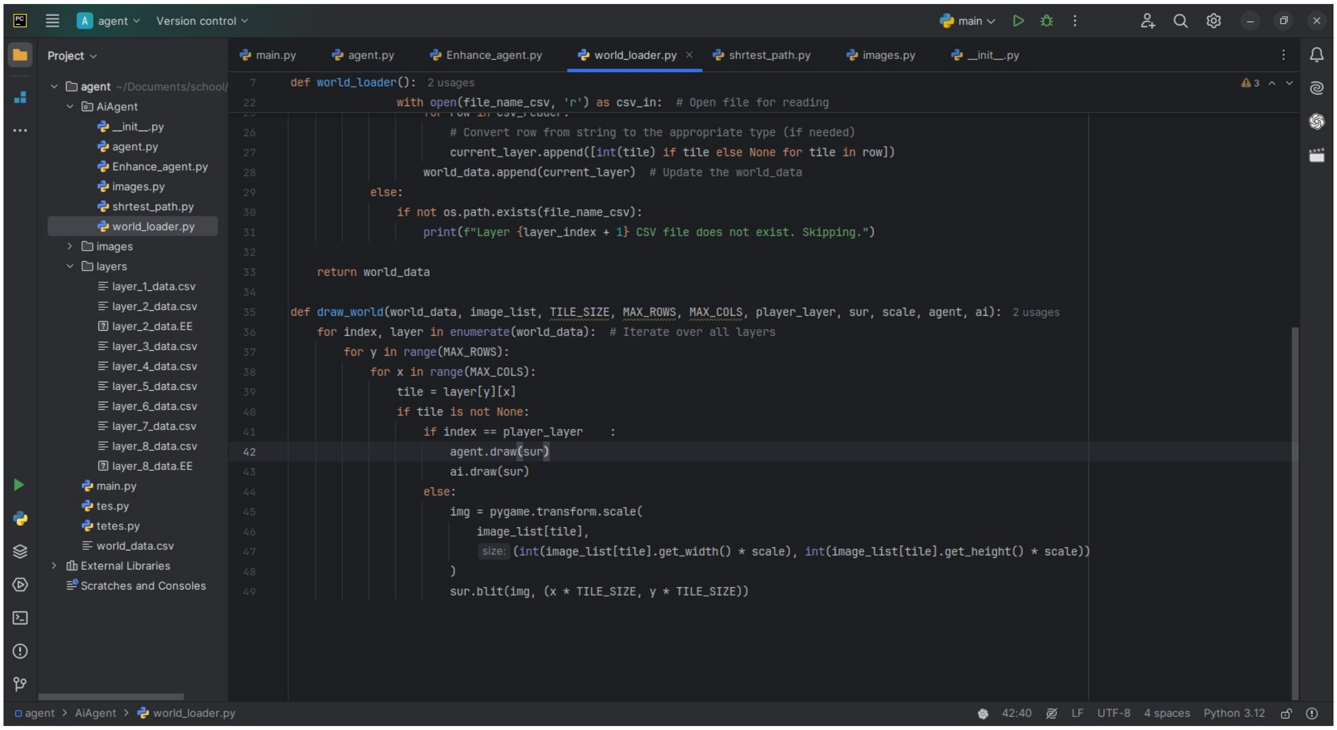

Figure 4: drawing world to the screen

Step2 - Agent Design:

The Agent class have been create with a set of properties and methods. The information need to create an Agent instance are :

- Image : The spring/ image that will be display on the screen for the visual of the Agent

- X and Y: Represent the column and row in the grid world where the agent image will be draw

- Scale : Represent the different/percentage change in size of the image

- Tile Size : Represent the constant that will be use to be multiply the row and col of the image to obtain it image location in pixel

The Agent have a set of method to control how it move, draw, check valid movement and update it position on the display.

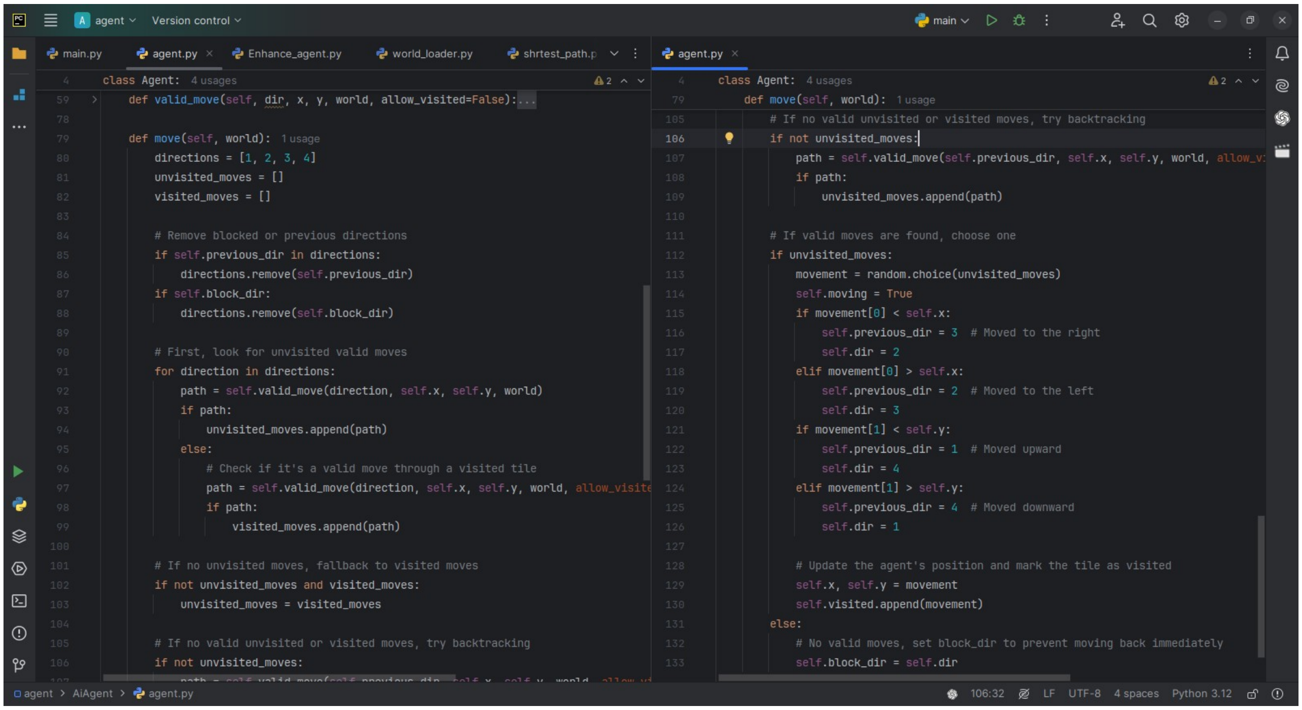

Figure 5: method for the movement of the Agent

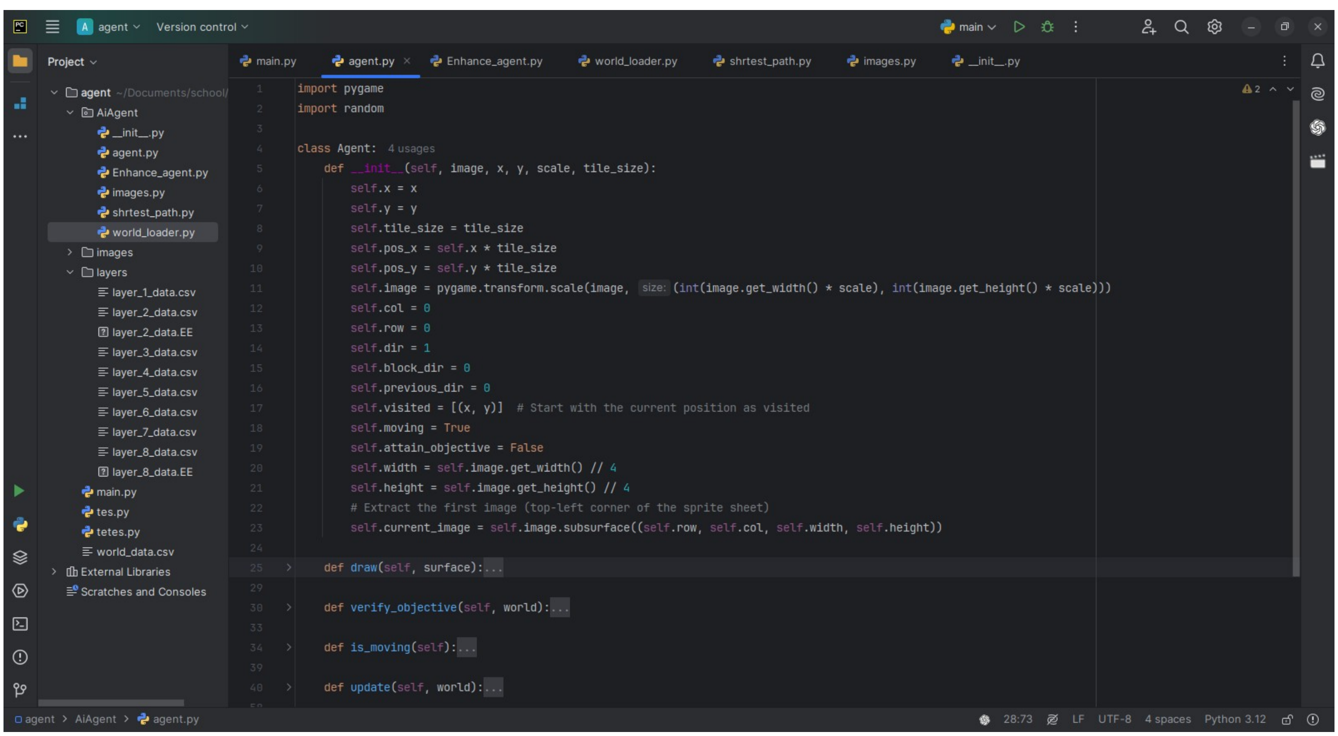

Figure 6: design of the Agent class

Step 3 - Agent Simulation Loop

This loop run the update function every 30 Frame For Per Second.

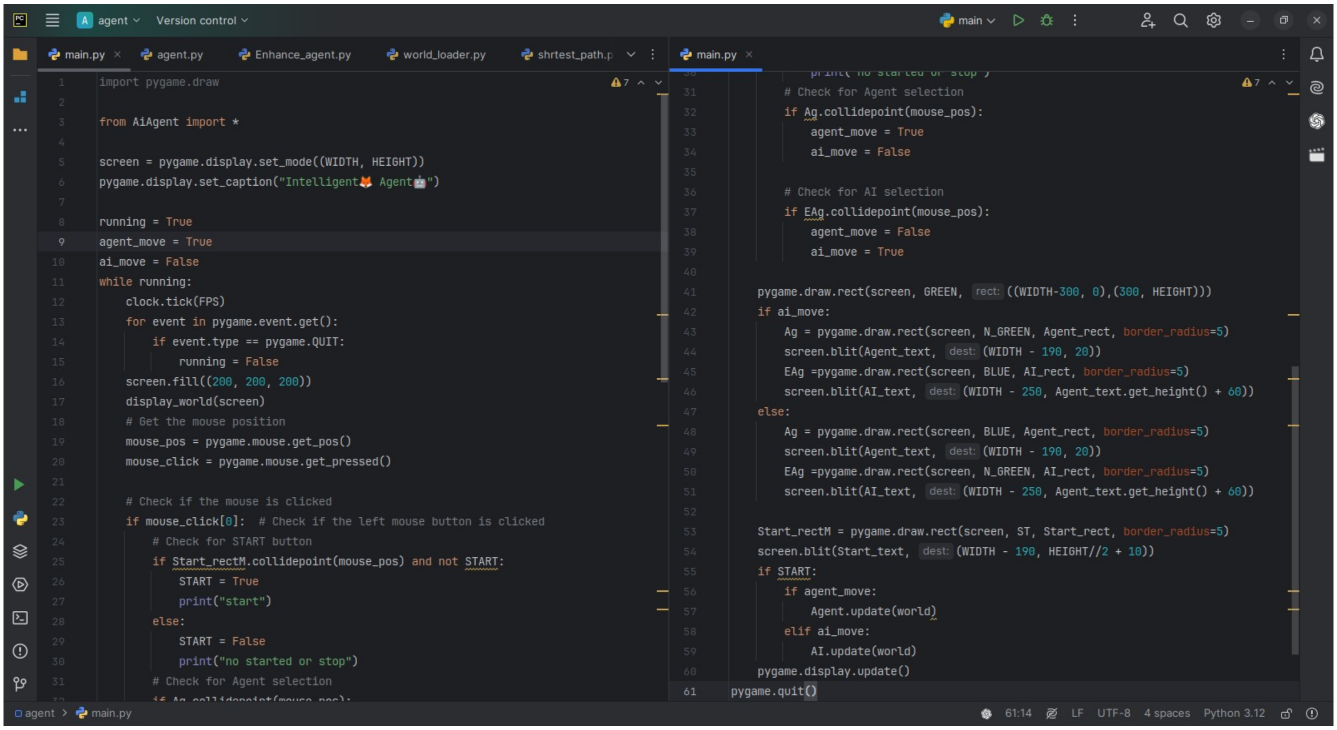

Figure 7: Main loop simulating the agent

Step 4 - Path-finding Enhancement:

Using BFS to enhanced to find the short-es path and move in and efficient moaner:

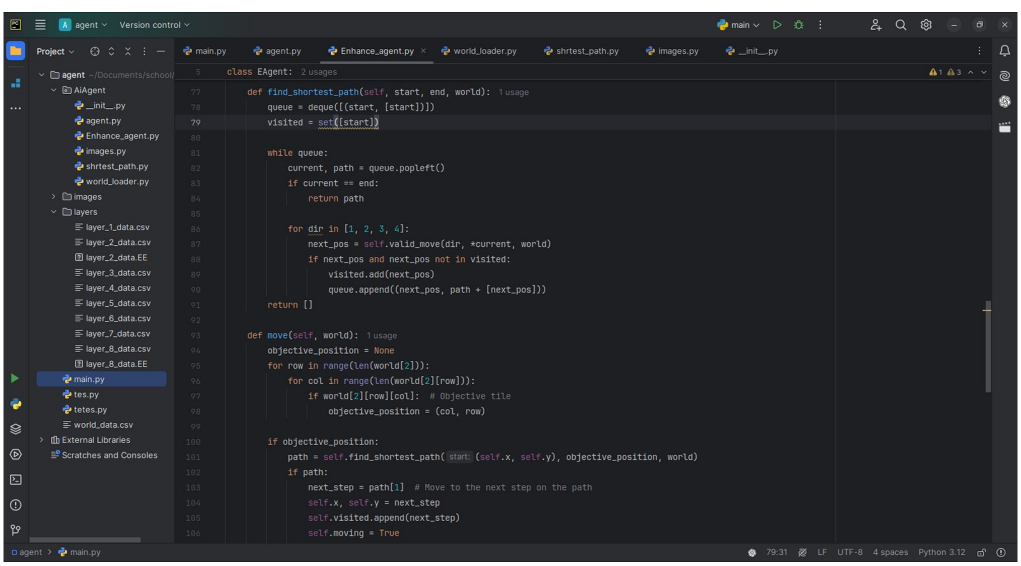

Figure 8: shortes path module using BFS

Comparison of Original and Enhanced Pathfinding Methods(BFS)

The original and enhanced methods each approach pathfinding with specific goals in mind. Here's a breakdown of their differences:

1. **Original Pathfinding:**
   - **Randomized Selection of Movement Directions**: It uses random choices within valid movement directions, only verifying whether a path is visited or unvisited.
   - **Efficiency**: This method is simpler but less efficient because it may backtrack or revisit areas without a strong preference for optimal paths.
   - **Usage**: Works well in smaller, less complex environments where an exhaustive search is feasible.
2. **Enhanced Pathfinding:**
   - **Layer-Dependent Optimization**: This version takes full advantage of the multi-layered EE environment. It checks each direction based on obstacles, paths, and previously visited points and uses a fallback to expand the search.
   - **Breadth-First Search (BFS)**: The enhancement includes BFS-based algorithms to find the shortest path. BFS is ideal for finding the shortest path in grid-like environments, and it reduces redundant checks.
   - **Backtracking with Fallbacks**: If no direct path is found, it attempts to backtrack while considering all available paths in unvisited and visited lists to ensure all options are exhausted.
   - **Use Case**: Ideal for larger environments with multiple obstacles and layers, making it more adaptable to complex grid designs.

Comparison of Methods

Feature	Original Code	Enhanced Code (BFS)
Movement Strategy	Random, favors unvisited tiles	Prioritizes shortest path to objective using BFS
Efficiency	Inefficient with increased backtracking	More efficient, minimizes revisits and random moves
Pathfinding	Random exploration with fallbacks	BFS for shortest path to objective or fallback
Best Use Case	Small grid, no specific target	Large grid, target objective within agent's reach

The enhanced code improves pathfinding efficiency by eliminating random moves when a path to the target is available, thus making the agent faster and more effective in reaching its objective. The original method's fallback to random movement still remains, providing flexibility if the agent cannot directly reach the target.

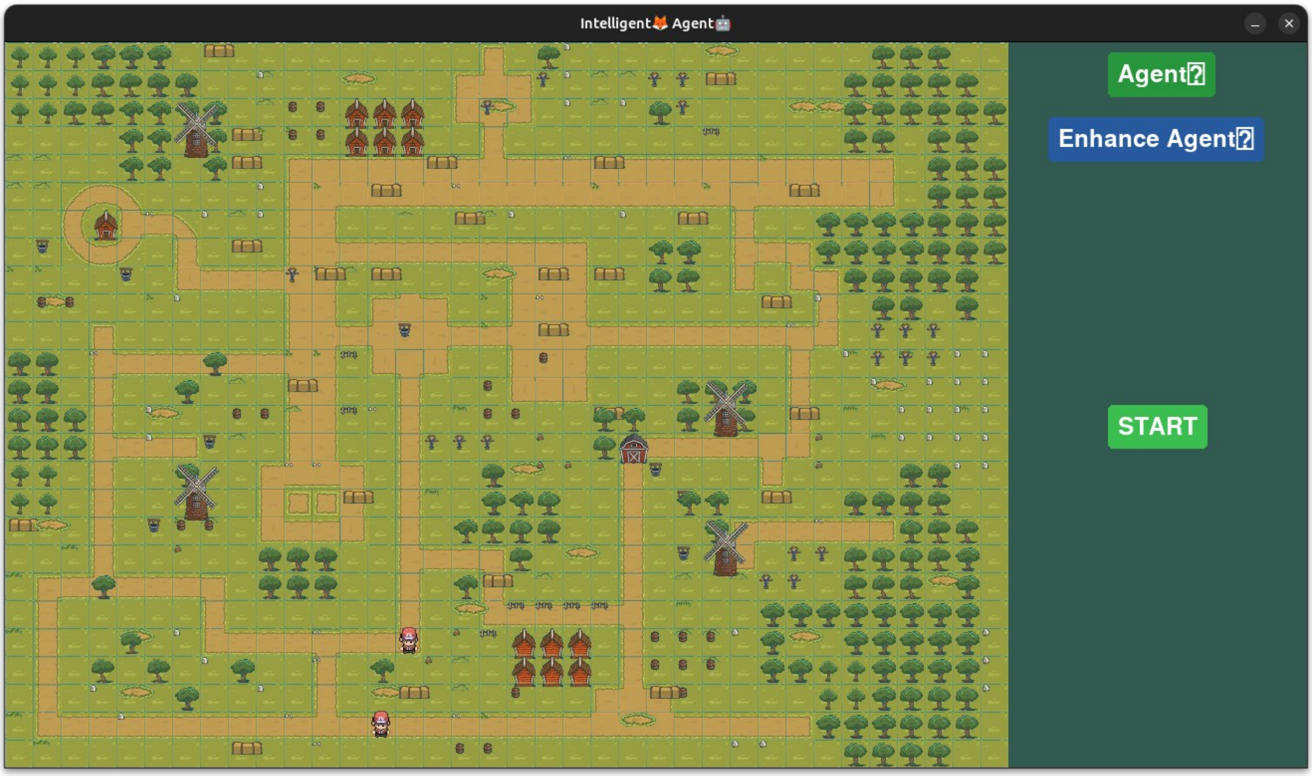

Figure 9: Agent And EnhanceAgent moving in the world