**Spike:** Spike\_08

**Title:** Navigation with Graphs

**Author:** Adnan Zafar, 103169535

**Goals / deliverables:**

The goal of this spike is to modify and extend the existing BoxWorld codebase to implement a simulation where agents plan paths using heuristic search algorithms and then navigate the environment. All steering forces were removed and only velocity was updated for the agents to follow the path properly.

.

**Technologies, Tools, and Resources used:**

* Visual Studio Code
* Python 3.0
* Pyglet 1.5.27
* Lab14 and Lab09 code

**Tasks undertaken:**

1. **Created the Agent class:**

The Agent class is designed to represent moving agents in the simulation. Here's an explanation of the methods added to this class and their purpose:

* **\_\_init\_\_(self, start\_pos, speed, color='GREEN'):** The constructor initializes an agent with a starting position (start\_pos), a speed (speed), and an optional color (default is 'GREEN'). It also sets the agent's path as not set initially (self.path\_set = False).
* **update(self):** This method updates the agent by calling the follow\_path() method, which makes the agent follow its assigned path.
* **set\_new\_target(self, new\_target):** This method sets a new target for the agent by updating its path with the new target points. It also resets the current point index of the agent's path so that the agent starts navigating from the beginning of the new path.
* **render(self, color=None):** This method renders the agent on the screen using the provided color or the agent's default color. It draws a cross and a circle at the agent's current position.
* **follow\_path(self):** This method makes the agent follow its assigned path. It calculates the distance to the current target point and checks if the agent is within a certain threshold. If the agent is close enough to the target point, it increments the current point index to navigate to the next point on the path. It then calculates the direction vector to the target point and updates the agent's position based on its speed and direction.

These methods enable the agent to navigate through the environment by following a path while updating its position based on its speed. This results in a moving agent that can adapt its movement according to the assigned path and target points.

Text

Description automatically generatedText

Description automatically generated

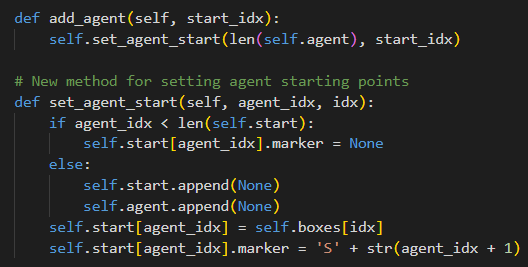
1. **Modified the BoxWorld class:**

The BoxWorld class has been modified to implement moving agents in the simulation. Here's an explanation of the added or modified methods to handle moving agents:

* **add\_agent(self, start\_idx):** This method adds an agent to the world by calling the set\_agent\_start() method, which sets the agent's starting position based on the provided index.
* **set\_agent\_start(self, agent\_idx, idx):** This method sets the starting position for the agent with the given index (agent\_idx). If the agent index is within the current agent list, it updates the agent's starting position. Otherwise, it adds a new agent to the list. The agent's starting position is set using the box index (idx).
* **set\_target(self, idx):** This method sets the target box based on the provided index (idx). It updates the agents' starting positions to the previous target position and sets the new target node. It then updates the agents' paths and targets based on the new target position.
* **plan\_path(self, search, limit, agent\_idx, speed=None):** This method is modified to handle the path planning for a specific agent, given by agent\_idx. It initializes the agent with a start position and speed if it doesn't exist. It then sets the agent's path if it hasn't been set before.

These modifications to the BoxWorld class allow the simulation to handle multiple moving agents, update their starting positions and target positions dynamically, and plan paths for each agent individually.

**Text

Description automatically generated**

**Text

Description automatically generated**

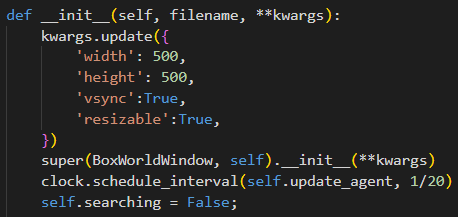
1. **Modified the Main class:**

In the main file, several changes have been made to implement agent movement in the BoxWorld simulation. Here's an explanation of the modifications:

* In the BoxWorldWindow class, the \_\_init\_\_() method has been updated to include a clock.schedule\_interval() function call that schedules the update\_agent() method to be called 20 times per second (1/20).
* The update\_agent(self, dt) method is added to update the agents in the world. It iterates through the agents and calls the update() method for each agent to update their position according to their path.
* In the on\_mouse\_press() event handler, when setting the start node (agent position) or target node, the method now calls self.plan\_path() with the agent's speed as an argument to plan paths for all agents based on their speeds.
* The on\_key\_press() event handler has been updated to handle agent speed changes. When the user presses 'A' or 'D', the method sets the search mode to A\* or Dijkstra respectively, and calls self.plan\_path() with a specified speed for the agent.
* The plan\_path(self, speed=None) method has been modified to plan paths for all agents. It iterates through the self.world.start list and calls the self.world.plan\_path() method with the agent index and speed as arguments. This ensures that each agent's path is planned individually based on their speed.
* The on\_draw() method remains unchanged, as it already takes care of drawing agents through the self.world.draw() method.

These modifications in the main file allow the simulation to handle multiple moving agents, update their positions based on their paths, and plan paths individually based on each agent's speed.

**Text

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Text

Description automatically generated**

**Text

Description automatically generated**

1. **Reused the existing codes to build the box world and path planning:**

The original BoxWorld codebase provides a solid foundation for path planning and navigation, with several essential components already in place. We have reused the following files from the codebase for this spike:

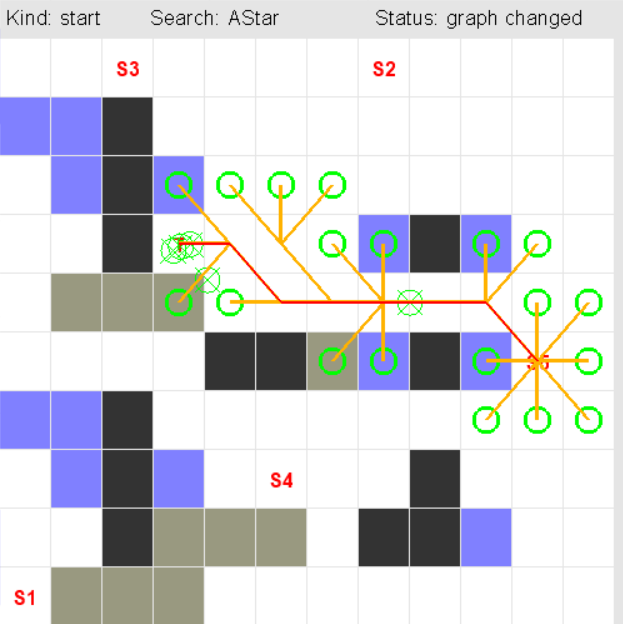
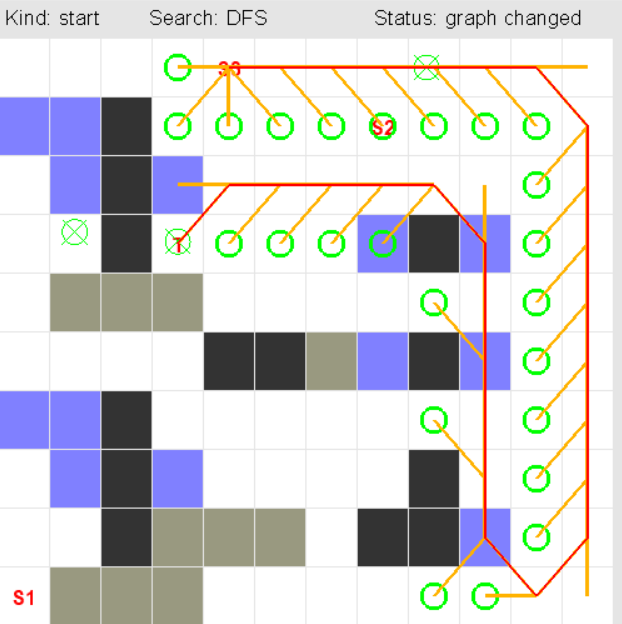
1. **graph.py:** Contains the Graph class and associated methods for creating and managing the navigation graph.
2. **graphics.py:** Provides the EGI (Extended Graphics Interface) for drawing primitives and text on the screen.
3. **path.py:** Contains the Path class and associated methods for managing agent paths.
4. **matrix33.py:** Provides Matrix33 class for 3x3 matrix operations (used for calculations within the path and graph classes).
5. **point2d.py**: Contains the Point2D class for handling 2D points and their operations.
6. **searches.py:** Contains different search algorithms (BFS, DFS, Dijkstra, A\*) for path planning.
7. **vector2d.py:** Provides the Vector2D class for 2D vector operations (used for agent movement and calculations).

**What we found out:**

As a result of these modifications and extensions, the BoxWorld simulation now supports multiple agents that can plan paths using heuristic search algorithms and navigate the environment using steering-based movement. The simulation can handle different agent speeds and allows users to interact with the agents and the environment for an engaging experience.

Flexibility of Heuristic Search Algorithms: The use of heuristic search algorithms, such as A\* and Dijkstra, proved to be highly adaptable for path planning in a dynamic environment. These algorithms were able to generate efficient paths for the agents despite changes in the environment, such as the addition or removal of obstacles.

Importance of Agent Speed Variation: Allowing agents to have different speeds added an interesting layer of complexity to the simulation. Agents with different speeds could reach their targets at varying times, which could be useful for modelling real-world scenarios where agents have different capabilities or priorities.



**Open Issues and Recommendations:**The current implementation directly updates the agent's velocity to follow the path. Although this approach works for the given simulation, utilizing more advanced steering behaviors like the "seek" function could have potentially led to smoother agent movement and better overall performance.