CAB302 Log

Week 1:

|  |  |
| --- | --- |
| Team Member | Work |
| Aparna | * Established the foundation for obstacle representation with the Obstacle interface. * Implemented essential methods for coordinate management in the Location class. * Outlined a clear structure for obstacle hierarchy with the LocatableObstacle abstract class. |
| Harnoor | * Structured the codebase effectively to handle different obstacle types and their locations. * Ensured modularity by introducing abstract classes and interfaces for obstacle management. * Implemented parsing methods in the Location class for seamless integration with command line arguments. |
| Alex | * Laid down the groundwork for obstacle modelling with the Obstacle interface and Location class. * Implemented essential methods for obstacle identification and location management. * Ensured consistency and clarity in code structure, facilitating future expansions and modifications. |
| Eric | * Contributed to the implementation of the Guard class, providing insights into obstacle representation. * Ensured accurate parsing of guard locations from command line arguments. * Reviewed and refined class structure for optimal organization and functionality. |
| James | * Actively participated in the development of the Guard class, contributing to its parsing and implementation. * Identified potential edge cases and provided solutions for robust guard location parsing. * Collaborated with the team to ensure consistency and coherence in class design and functionality. |

Week 2:

|  |  |
| --- | --- |
| Team Member | Work |
| Aparna | * Implemented basic parsing logic for command line arguments. * Ensured correct separation of keys and values from command line input. * Handled edge cases such as null values to prevent exceptions. |
| Harnoor | * Successfully implemented parsing function to capture key details from command line inputs. * Used HashMap effectively for storing arguments, enabling easy access and manipulation. * Ensured clarity and readability of code through appropriate variable names and comments. |
| Alex | * Parsing function efficiently captures and organizes command line arguments for further processing. * Utilized HashMap's flexibility to handle various argument types and values. * Code structure demonstrates understanding of iteration logic and control flow. |
| Eric | * Contributed to the discussion on parsing strategy and approach. * Provided valuable insights into potential optimizations for parsing logic. * Reviewed code for clarity and consistency, ensuring adherence to coding standards. |
| James | * Assisted in testing the parsing functionality with diverse input scenarios. * Identified and reported edge cases for further refinement of parsing logic. * Participated actively in brainstorming sessions to address parsing challenges. |

Week 3:

|  |  |
| --- | --- |
| Team Member | Work |
| Aparna | * Implemented the Path class to represent the path from start to end location. * Implemented methods to check if a location is in the path and to get the symbol for a location on the path. * Implemented logic to determine symbol based on directions. |
| Harnoor | * Implemented BFSPathFinder class to represent breadth-first search pathfinding algorithm. * Implemented findPath method to find shortest path using BFS. * Implemented getNeighbors method to retrieve neighboring locations. |
| Alex | * Parsing function efficiently captures and organizes command line arguments for further processing. * Utilized HashMap's flexibility to handle various argument types and values. * Code structure demonstrates understanding of iteration logic and control flow. |
| Eric | * Conducted code refactoring for improved readability and efficiency. * Optimized variable names and method implementations for clarity. * Enhanced code documentation and added inline comments for better understanding. |
| James | * Implemented unit tests to validate functionality of pathfinding algorithms. * Tested various scenarios including obstacle avoidance and path accuracy. * Ensured pathfinding algorithms produce expected results under different conditions. |

Week 4

|  |  |
| --- | --- |
| Team Member | Work |
| Aparna | * Started by setting up the GridPathFinder interface and the BFSPathFinder class. * Defined the basic structure for pathfinding algorithms. * Next step is to integrate this with map and obstacles. |
| Harnoor | * Ensuring seamless integration with the map class will be essential. * Eager to witness the algorithm's navigation through obstacles. |
| Alex | * Added Path class to represent the discovered path. * Implemented Direction enum for handling path directions. * Next step: integrate pathfinding results with map display. |
| Eric | * Path representation looks solid. * Direction enum will be useful for determining path symbols. * Let's make sure the path updates dynamically as obstacles change. |
| James | * Defined Obstacle interface with methods for symbol and obstruction check. * Implemented Guard class to represent guard obstacles. |

Week 5

|  |  |
| --- | --- |
| Team Member | Work |
| Aparna | * Started by setting up the GridPathFinder interface and the BFSPathFinder class. * Defined the basic structure for pathfinding algorithms. * Next step is to integrate this with map and obstacles. |
| Harnoor | * Established Obstacle interface for handling diverse obstacles. * Implemented Guard class to exemplify obstacle representation. |
| Alex | * Added Path class for managing discovered paths. * Direction enum provides easy handling of path directions. * Next, integrating path representation with map visualization. |
| Eric | * Path representation and direction management implemented smoothly. * Ensured clear structure for representing and manipulating paths. * Dynamic updating of paths will be a crucial aspect to consider. |
| James | * Created Obstacle interface to handle different obstacle types. * Implemented Guard class as an example obstacle. * Now, we need to extend this for other obstacle types like Fence, Sensor, and Camera. |