Inf-2101: Algoritmer
You are likely to be eaten by a Grue!

Sjøli Stian
Michael Kampffmeyer

stian.sjoli@uit.no mka050@post.uit.no
University of Tromsø
Tromsø, Norway

***Abstract*—An algorithm was implemented for the task of finding a path between two random points (source and destination point) in a labyrinth design. The labyrinth terrain is not a priori known, and therefore awareness of the labyrinth has to be found through exploration. Two important aspects of the exploration had to be addressed, the sensor and the map model generation. The sensor system consist of vision, lookups, which reveal near nabour-positions to the current position. The search implemented explored unknown areas, or tiles, using the sensor information to move to the nabour boardering to the most unknown nabours at each action point. The sensor information was also used to update the map. The map gained importance in resolving problems with local minimas by finding a shortest path from the local minima to an already disclosed tile,within the map, which would again allow the search to encounter tiles with unknown nabours. The algorithm was unaware of the difference between a larger open area of tiles, rooms, and the hallways connecting them.**

I. Introduction

Finding Toby in the least amount of steps in an unknown labyrinth is describes as a shortest path with unknown map-problem. The map in this case is a maze consisting of a tiles, which is a rectangular object connected to four other objects in 4 directions (north, south, west, east). The maze then can be represented as a graph where the nodes are equivalent to tiles, while edges exist between near nabouring tiles in those four directions. Each node then have always the same sum of neigbouring tiles and walls, and a position, given as x and y coordinates, within the graph. The search itself is online, meaning that the agent, or acting component, does not know which states exist nor what its actions does [1] [2]. The online search must be solved by the agent through actions rather than pure computation.

II. Implementation

At the core of the implementation lies the fact that we need to store the sensor information which is gathered over time. Once information is collected and used during the decision process the search converts from a completely random search to a more focused search. Repeated visits to a Tile will be avoided if possible, thus making the search more efficient. In the implementation a dictionary of Tiles is used to store this information. The Tiles in the dictionary are indexed by a tuple of their coordinate respectively to the start position (origo). Tiles also contain knowledge about their four neighbours. Encountering a new tile in the maze will lead to an updating process of all neighbouring nodes in the dictionary. Previously unknown tiles will be instantiated from a Tile class, and added to the dictionary. Whenever a move is to be made a decision process has to determine the next optimal move. This decision process is based on the previous collected information and involves a graph based search. In the simplest case all four neighbour Tiles are checked for unexplored adjacent Tiles, if one or more are found the search is concluded and the move can be made. When multiple options arise the Tile with the most unexplored adjacent Tiles will be chosen for reasons that will be discussed later in the Discussion. Multiple neighbouring Tiles can share the same amount of unknowns, which causes the direction is chosen at random. In the case

of a local minimum, meaning that all neighbours of the adjacent Tiles are explored, the decision is made by iteratively building up a shortest path tree. The idea of the shortest path tree is based on Dijkstra's algorithm.[3] Here, initially all nodes in the dictionary are initialised as being at an infinite distance away from the root node. As the algorithm progresses the distances are updated according to the proximity to the root node, thus facilitating finding the shortest path between the root node and potentially any other explored node. Once the nearest Tile with unexplored neighbours is found the algorithm continues until all the siblings at the level are checked and chooses the next target Tile. A backpropagation step is then used to create a list of actions that need to be undertaken to get to the desired target Tile. The process of exploring, storing the new information in the dictionary, making a decision based on collected information and moving to a target Tile is repeated until the server sends a signal indicating that Toby is found (success) or the maximum number of moves was reached (failure).

III. Discussion

The depth first search (DFS) or the breath first search (BFS) can solve the maze problem, compared with the graph traversal of assignment 1. This is rooted in the step limitation. A DFS, introduced in assignment 1, would involve exploring one path until either Toby or a dead-end is discovered. In those cases, a backtracking (or several) could claim many steps that would be necessary for completing the task. Exploring the maze tiles one level at the time, as in a BFS search, would involve moving back and fourth between level i and level i+1. These two types of searches are therefore not suited for the task at hand. The agent move at the frontier depending on how many unknown tiles are found in level i+2. The agent, reaching a local minima, now has to find a path to the closest tile neighbouring with unknown tiles. Here, it is apparent that the exploration deviates from BFS and DFS. At each position the decision is based on a BFS search on all the surrounding nodes of the current position, to find the shortest path to the next target node. To increase the performance of the implementation each tile contains a reference to it's neighbouring tiles, which can be used to reduce the number of dictionary lookups. In the case that multiple Tiles have unknown neighbours the Tile with the most unknown neighbours in the layer is chosen. This was implemented in this way since the probability of entering a local minima will be reduced and will also likely lead to the exploration of larger unexplored areas. The *trap* and *Grue* version of the labyrinth has not been implemented. In theory the trap and Grue mode could have been implemented by using a different (or modified) weighting system when deciding where to move next.

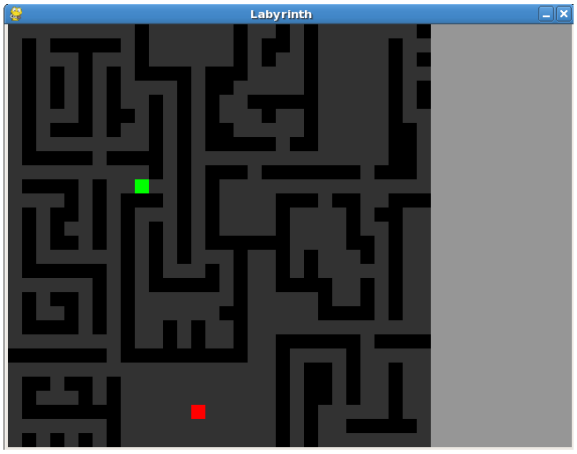

Fig. 1. Labyrinth

IV. Conclusion

The implemented solution to the given problem seems to perform well and manages to find Toby on both the *empty* map as well the *lab* map on all test runs. The success rate on the *awesome* map is lower, but Toby is still found during some of the testruns. The implementation could certainly be improved in regards to performance aspects, such as number of dictionary interactions and steps taken to find Toby.

References

[1] P. Norvig and S. Russel, *Artificial Intelligence: A modern Approach Chapter 2*, Prentice Hall, 2009.
[2] P. Norvig and S. Russel, *Artificial Intelligence: A modern Approach Chapter 4*, Prentice Hall, 2009.
[3] R. Sedgewick, *Algorithms in C Third Edition Part 5 Chapter 21*, Addison-Wesley Professional, United States, 2009.