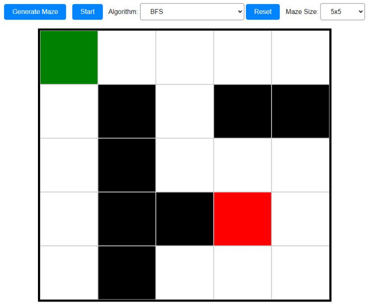
AI Maze Solvers + Map-Terrain-Vehicle Criterion

Tommy Chan, Eric Detjen, Akshay Subramaniam, Austin Shaw, Derek Miller

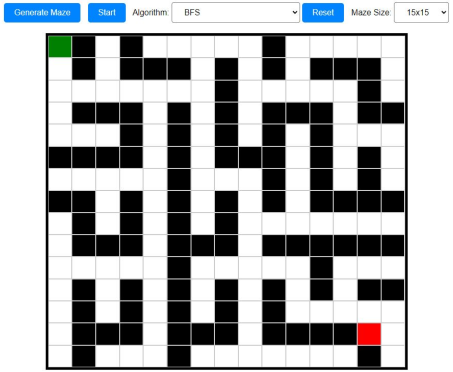
Maze Generation

- Done using Prim's Algorithm
- 2-D array used to represent each cell in the maze
- Wall sequences are randomly generated around maze



Algorithm: BFS

Maze Size: 5x5



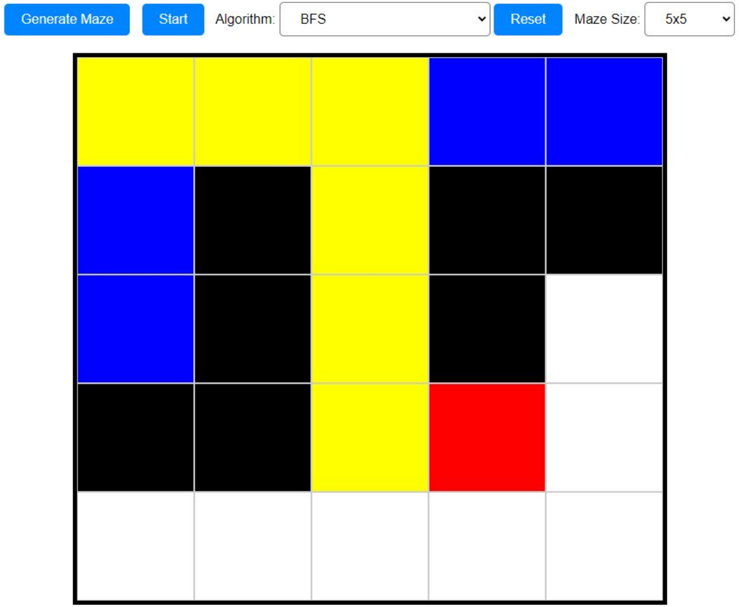
Algorithm: BFS

Maze Size: 15x15

Search Algorithms make use of Priority Queue

- JavaScript Class → defined with enqueue(), dequeue(), isEmpty() methods
- Used to keep track of the next Tile we need to search based on some calculated priority

Breadth First Search



Algorithm: BFS

Maze Size: 5x5

(1) **Define** *Start and Goal Nodes*

(2) **Create** *visited List* → keeps track of tiles visited

(3) **Create** *path Queue*→ keeps track of paths of nodes

(4) ***Recursive Function*** → keep exploring the adjacent tiles to the current tile

(a) **Adjacent** = Top, Down, Left, Right

(b) ***Pop FIRST tile*** *off the queue*

(c) *Add EACH adjacent tile to the queue*

(i) *Continuously check for…*

1) Empty Queues

2) Goal Nodes Reached

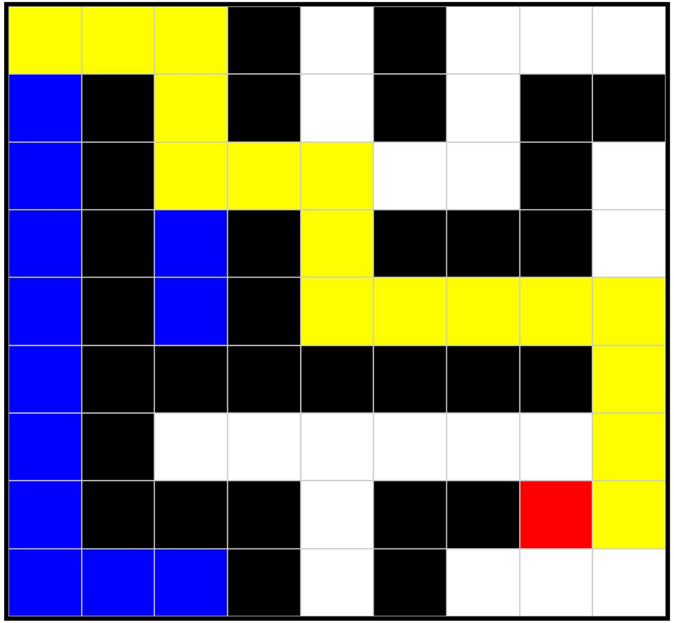
3) Move is Valid (in bounds)

(5) ***Goal Node Found*** → Mark the FIRST path that found it in the Grid

(a) Relabel tiles in 2D Array maze grid

(b) Update Display Maze

Depth First Search



(1) **Define** *Start and Goal Nodes*

(2) **Create** *visited List* → keeps track of tiles visited

(3) **Create** *path Stack*→ keeps track of paths of nodes

   (a) Initial path consists of

(4) ***Recursive Function*** → keep exploring the adjacent tiles to the current tile

   (a) **Adjacent** = Top, Down, Left, Right

   (b) ***Pop LAST tile** off the queue*

   (c) *Add EACH adjacent tile to the queue*

      (i) *Continuously check for…*

         1) Empty Stack

         2) Goal Nodes Reached

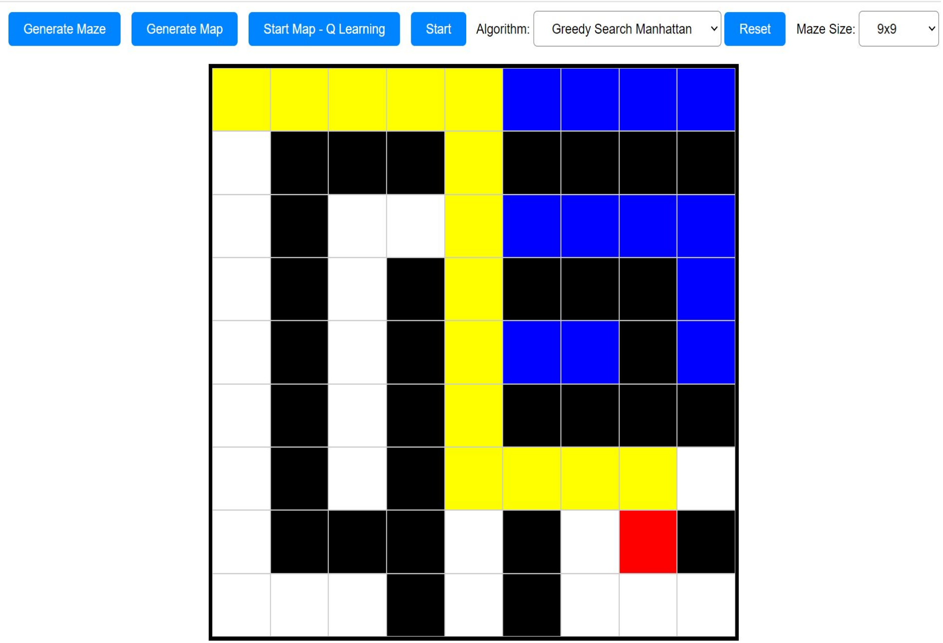
         3) Move is Valid (in bounds)

(5) ***Goal Node Found*** → Mark the FIRST path that found it in the Grid

   (a) Relabel tiles in 2D Array maze grid

   (b) Update Display Maze

Greedy Search - Manhattan Distance



Algorithm: Greedy Search Manhattan

Maze Size: 9x9

(1) **Define** *Start and Goal Nodes*

(2) **Create** *visited List* → keeps track of tiles visited

(3) **Create** *Priority Queue* → paths sorted by order of priority (“Manhattan Distance”)

  (a) Initial path consists of

(4) ***Recursive Function*** → keep exploring the adjacent tiles to the current tile

  (a) **Get tile with *LOWEST heuristic***

  (b) **Adjacent** = Top, Down, Left, Right

  (c) ***Calculate Manhattan Distance*** *for each adjacent tile (|x1 - x2| + |y1 - y2|)*

  (d) Add EACH adjacent tile to the queue

    (i) Continuously check for…

      1) Empty Queues

      2) Goal Nodes Reached

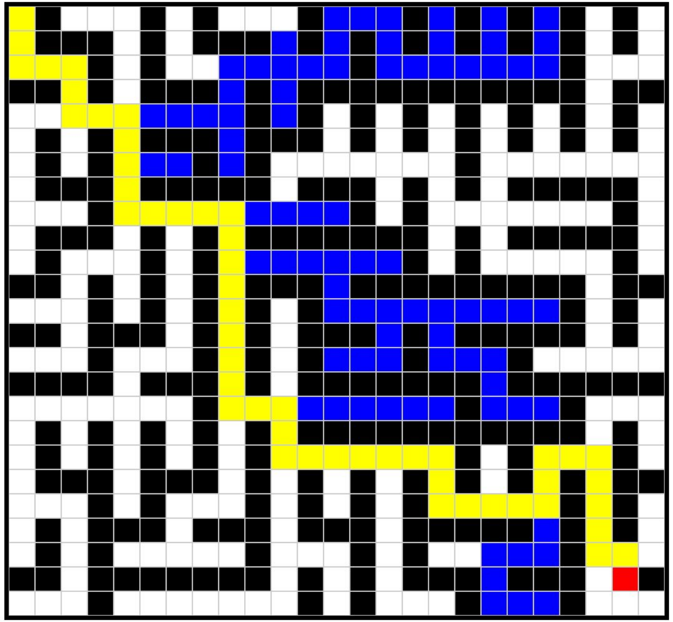
      3) Move is Valid (in bounds)

(5) ***Goal Node Found*** → Mark the FIRST path that found it in the Grid

  (a) Relabel tiles in 2D Array maze grid

  (b) Update Display Maze

Greedy Search - Euclidean Distance



(1) **Define** *Start and Goal Nodes*

(2) **Create** *visited List* → keeps track of tiles visited

(3) **Create** *Priority Queue* → paths sorted by order of priority (“Euclidean Distance”)

(a) Initial path consists of

(4) ***Recursive Function*** → keep exploring the adjacent tiles to the current tile

(a) **Get tile with *LOWEST heuristic***

(b) **Adjacent** = Top, Down, Left, Right

(c) ***Calculate Euclidean Distance*** *for each adjacent tile (Length of Line Segment)*

(d) Add EACH adjacent tile to the queue

(i) Continuously check for…

1) Empty Queues

2) Goal Nodes Reached

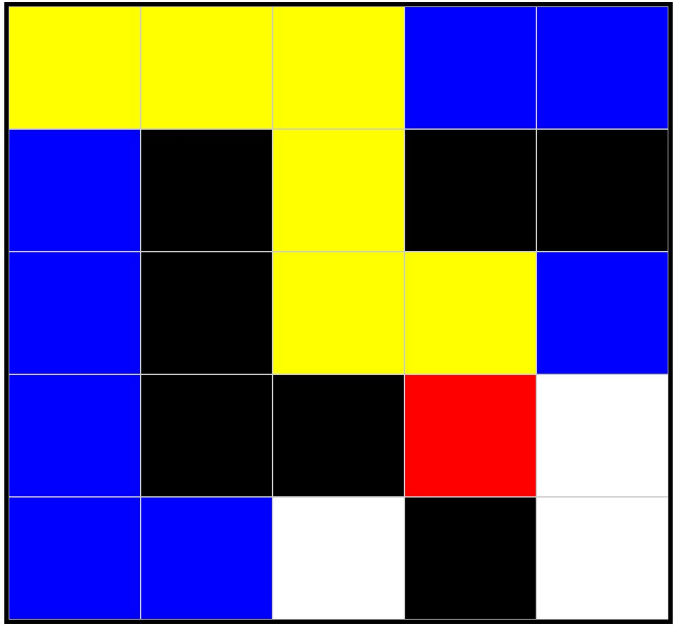
3) Move is Valid (in bounds)

(5) ***Goal Node Found*** → Mark the FIRST path that found it in the Grid

(a) Relabel tiles in 2D Array maze grid

(b) Update Display Maze

Uniform Cost Search



(1) **Define** *Start and Goal Nodes*

(2) **Create** *visited List* → keeps track of tiles visited

(3) **Create** *Priority Queue* → paths sorted by order of priority (“UCS”)

- (a) Initial path consists of

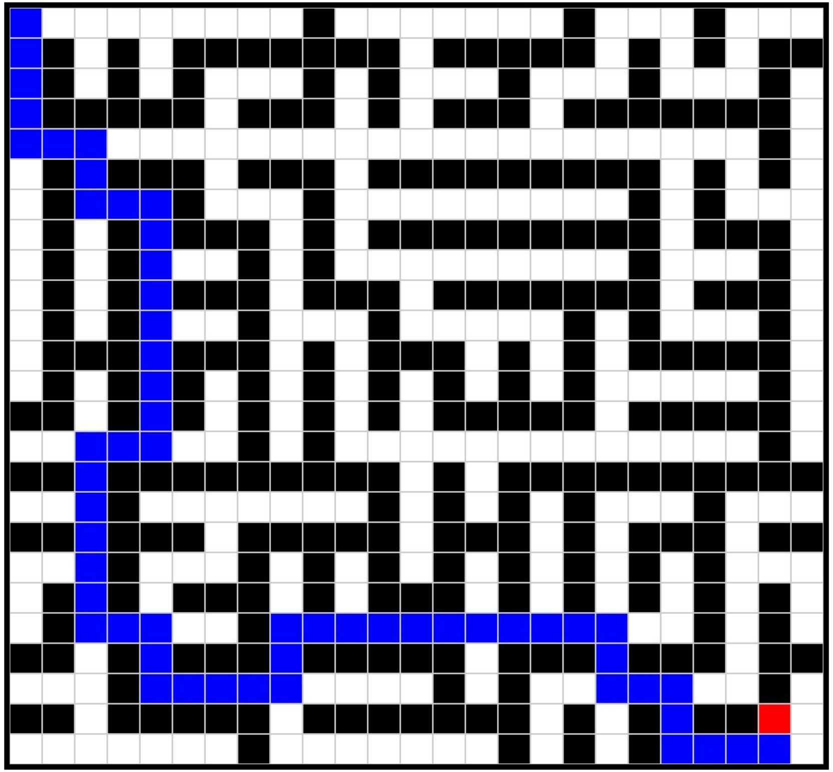
(4) ***Recursive Function*** → keep exploring the adjacent tiles to the current tile

- (a) **Get tile with *LOWEST heuristic***
- (b) **Adjacent** = Top, Down, Left, Right
- (c) ***Calculate UCS*** *for each adjacent tile (prior cost + 1)*
- *(d) Add EACH adjacent tile to the queue*
  - *(i) Continuously check for…*
    - 1) Empty Queues
    - 2) Goal Nodes Reached
    - 3) Move is Valid (in bounds)

(5) ***Goal Node Found*** → Mark the FIRST path that found it in the Grid

- (a) Relabel tiles in 2D Array maze grid
- (b) Update Display Maze

Q - Learning



(1) Define Training Parameters →

(a) Epochs = 15000

(b) Learning Rate = 0.1

(c) Discount Factor = 0.9

(d) Epsilon = 1.0

(2) Initialize Q-Values for EACH Tile-Move pair

(3) Choose either Random or Best Action for Current State

(a) Calculate Rewards for EACH Move

(b) Update Q-values

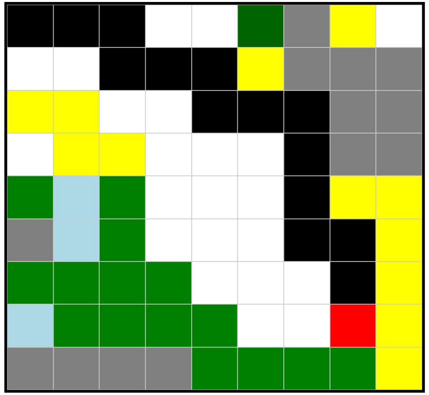
(4) Repeat until Goal Node found… Epoch times

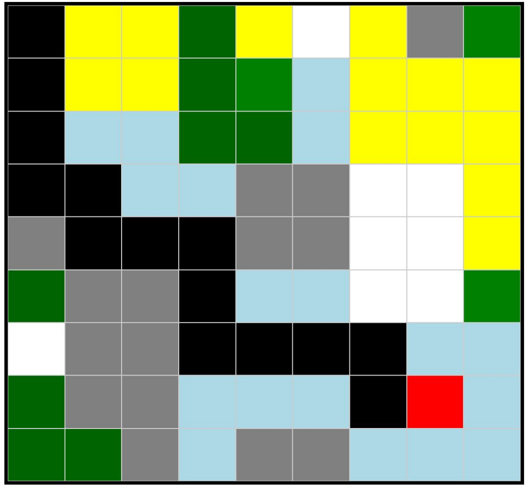
(a) Decay Exploration → 0.99

(5) Calculate MAX Q-value for EACH adjacent node… … Follow Tile with Max Q-value

(a) Repeat Recursively until reach Goal node

Q - Learning (w/ Map-Terrain-Vehicle Criterion)





- Tiles are Clustered together
  - HIGHER probability of being Tile Type of previous Row and Column

- BLACK → *Path*
- RED → *Goal Node*
- *6 Tile Types* →
  - *Snow*
  - *Water*
  - *Grasslands*
  - *Plains*
  - *Forest*
  - *Desert*

  ... EACH with DIFFERENT Terrain Costs
- *6 Vehicle Types* →
  - *Car*
  - *Boat*
  - *Airplane*
  - *Off-Road Truck*
  - *Snowmobile*
  - *Dune Buggy*
- *... EACH with Different Terrain Strengths/Weaknesses*