CLOUD DRIVEN IMPLEMENTATION FOR MULTI AGENT PATH FINDING

Anusha Datta
Supervised by Dr. Xueyan Tang

School of Computer Science and Engineering
Nanyang Technological University, Singapore

Introduction

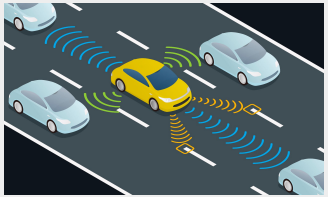
Multi Agent Path Finding (MAPF)

Construct collision-free paths for a set of agents from their respective start to goal positions within a given maze layout

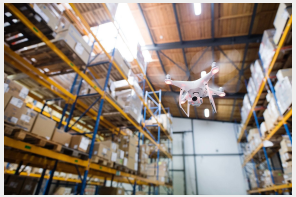
Motivations

- Higher Speed & Efficiency
- Time Cost Savings
- Mitigation of Human Error
- Operational Safety in Hazardous Environments

Applications

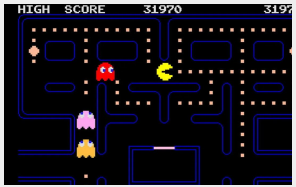


Autonomous Vehicles



Warehouse Drones

Aircraft Management



Video Games

Objectives

- **Explore Conflict Based Search performances for Multi Agent Path Finding**
  - Experimentation on effect of various lower level search (A* Search) heuristics
    i. Manhattan Distance
    ii. Chebyshev Distance
    iii. Euclidean Distance

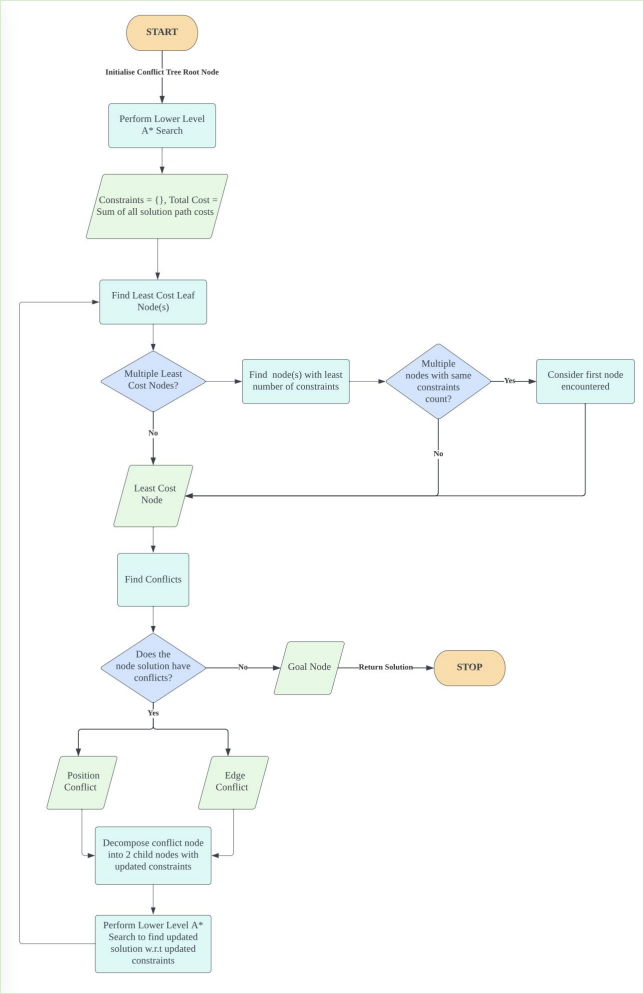
- **Design, Develop and Deploy a Multi Agent Path Finding application**
  - Powered by Cloud Services
  - Authentication differentiated
  - Implement interactive MAPF functionalities
  - Navigation Statistics Pipeline
    i. Predictive Insights
    ii. Navigation Trends
    iii. Intelligent data driven business decisions

MAPF: Conflict Based Search

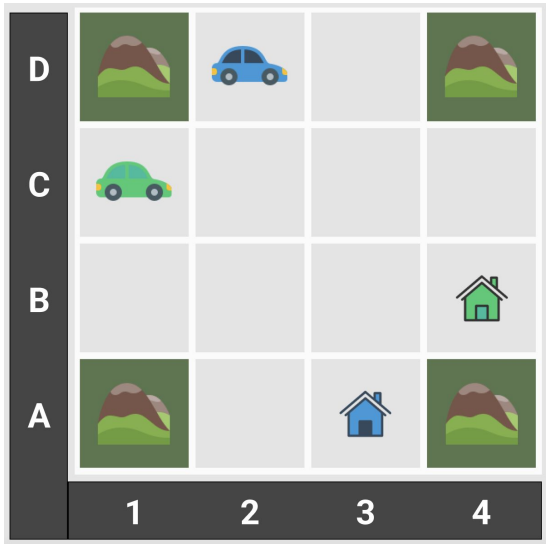
Conflict Based Search (CBS)

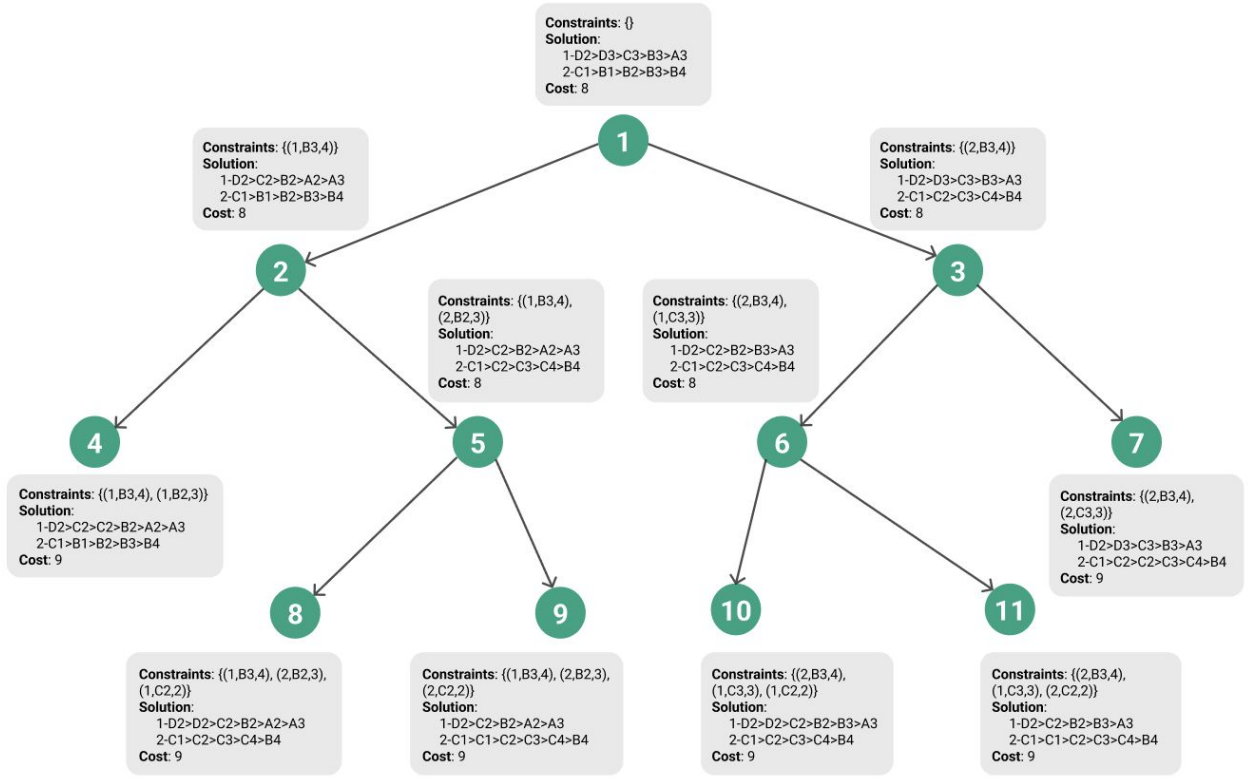
2 Level Algorithm

- **Lower Level Algorithm**
  - Search performed only for a single agent at a time
  - all solution paths combined and passed to the higher level algorithm
- **Higher Level Algorithm**
  - Performs tree based search to identify conflicts between agents
  - Collective solution paths checked for all agents to discover and resolve conflicts



Conflict Based Search: Example

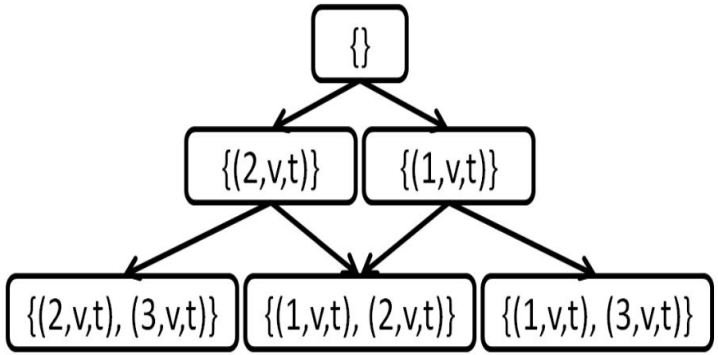




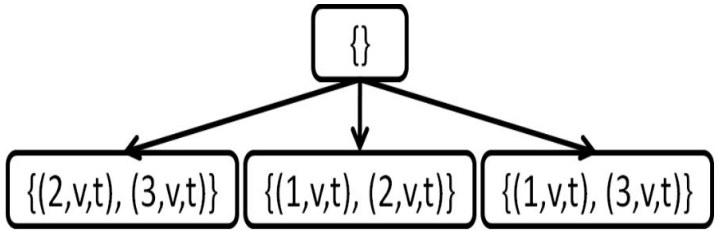
Conflict Based Search: Conflict Resolution

Handling k-Agent Conflicts in Conflict Tree

Decomposition of Conflict Node for k > 2, where k is number of agents involved in conflict



2 child nodes for first of k conflicts



k child nodes for k agent conflicts

CBS Lower Level Solver: A* Search

$$f(x) = g(x) + h(x)$$

where

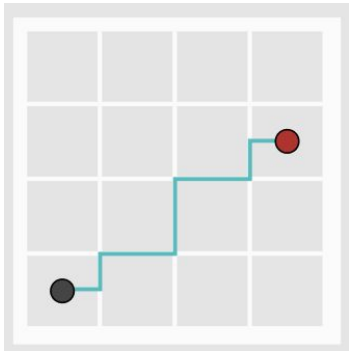
***x** is the current node on the path,*
***f(x)** is the A* evaluation function,*
***g(x)** is the cost function that computes the path cost from the start node to x,*
***h(x)** is a heuristic function that estimates path cost from x to the goal node.*

Worst Case Time Complexity	$O(\|E\|) = O(b^d)$
Worst Case Space Complexity	$O(\|V\|) = O(b^d)$

where
b is branching factor,
d is depth of search tree

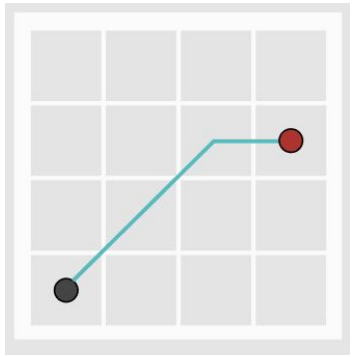
CBS Lower Level Solver: A* Search Heuristics

```
manhattan_distance = abs(x2 - x1) + abs(y2 - y1)
```



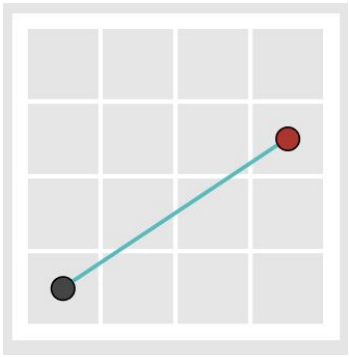
Manhattan Distance

```
chebyshev_distance = max(abs(x2 - x1),abs(y2 - y1))
```



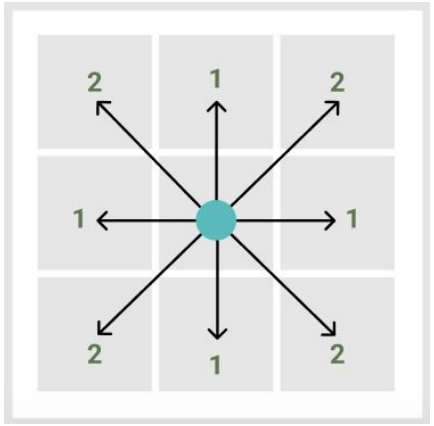
Chebyshev Distance

```
euclidian_distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)
```

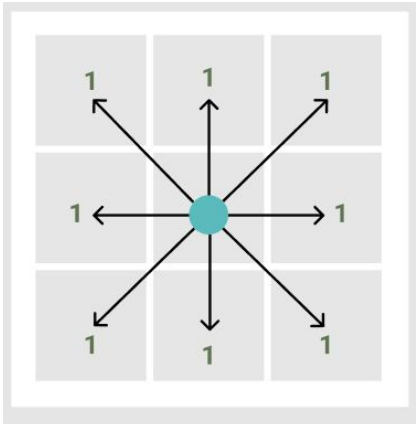


Euclidean Distance

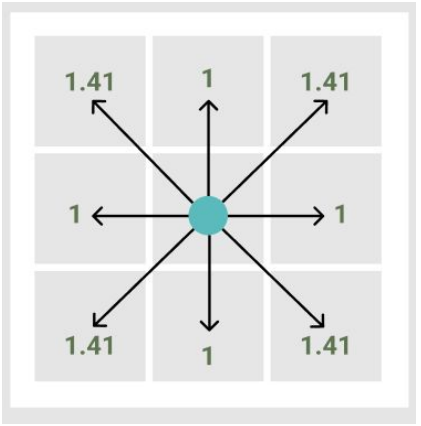
CBS Lower Level Solver: A* Search Heuristics Costs



Manhattan Distance Cost



Chebyshev Distance Cost

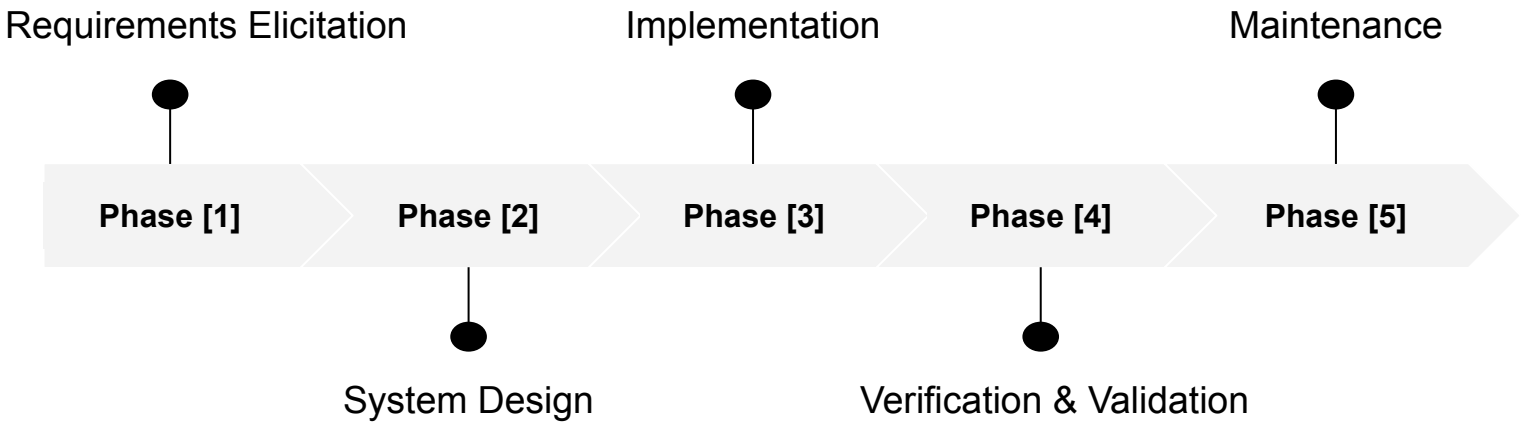


Euclidean Distance Cost

Software Development Life Cycle (SDLC)

WATERFALL MODEL

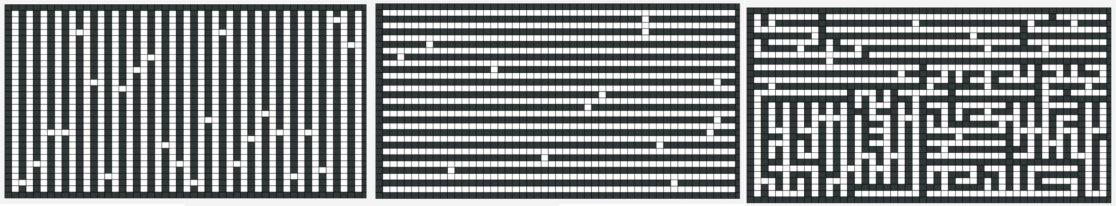
SDLC Phases conducted in a strictly sequential manner



[1] Requirements Elicitation: Functional Requirements

Functional Requirements

- Authentication
- Maze Generation
  - Randomised Verticals
  - Randomised Horizontals
  - Recursive Division



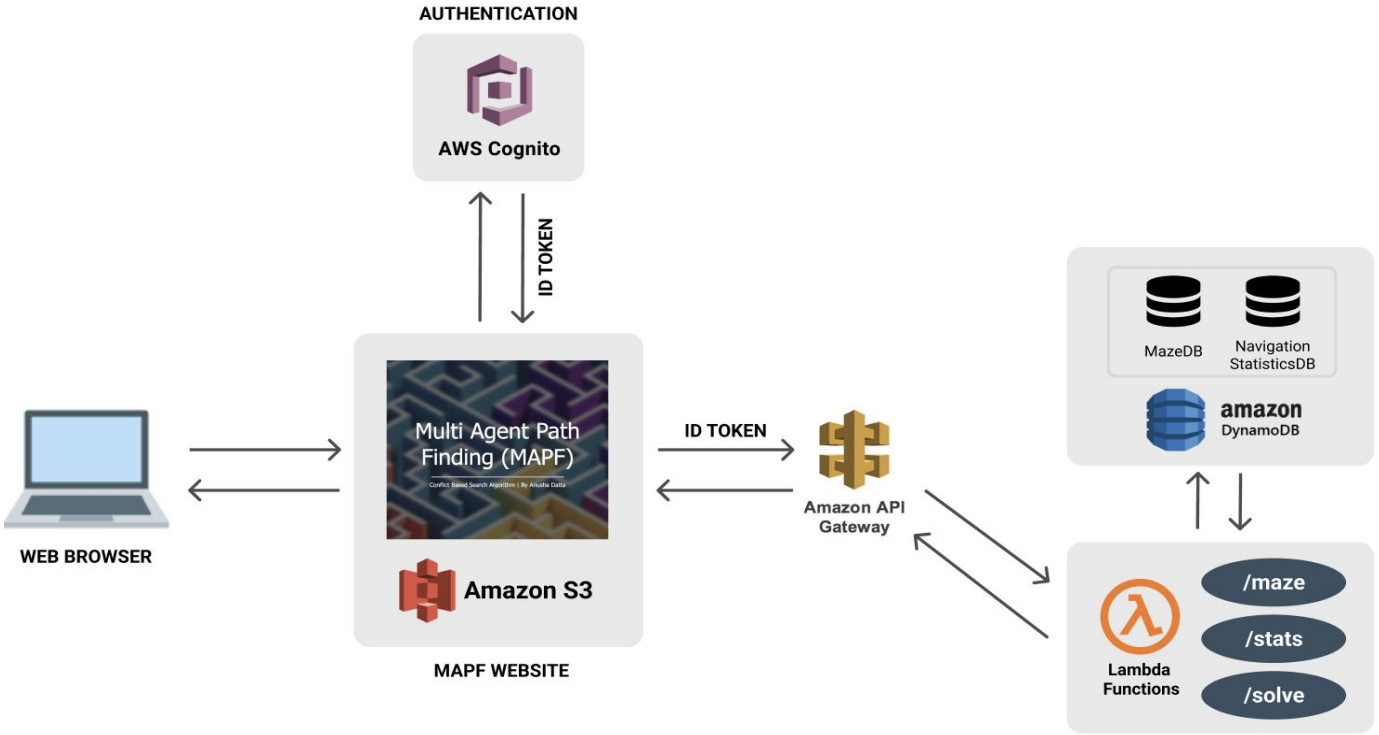
- Maze DB Management
  - Remote Cloud datastore
- Multi Agent Path Finding
  - Conflict Based Search, with A* Search as Lower Level Solver
- Navigation Statistics Pipeline
  - Execution Cost
  - Execution Time
- Application Hosting

[1] Requirements Elicitation: Non-Functional Requirements

Non-Functional Requirements

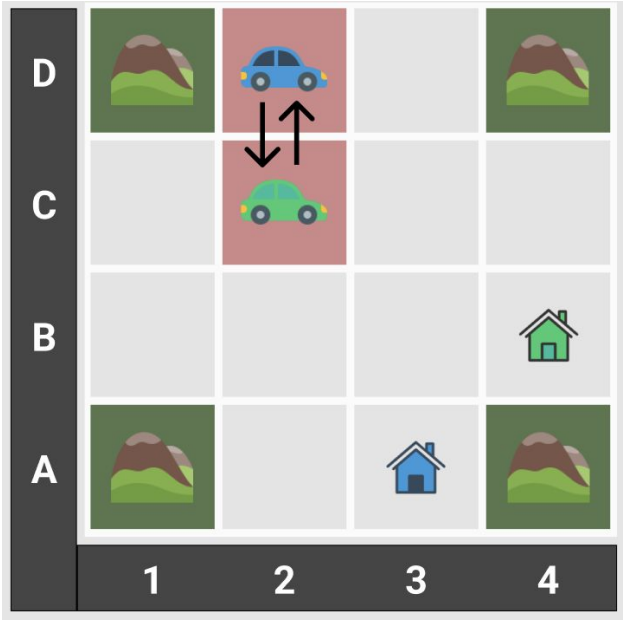
- Performance Requirements
- Security Requirements
- Usability Requirements
  - Schneiderman’s 8 Golden Rules of Interface Design
    - Strive for Consistency
    - Enable shortcuts
    - Offer Informative Feedback
    - Design Dialog to yield closure
    - Offer simple Error Handling
    - Permit Easy Reversal of Actions
    - Support Internal Locus of Control
    - Reduce Short Term Memory Load

[2] System Design: Cloud Architecture



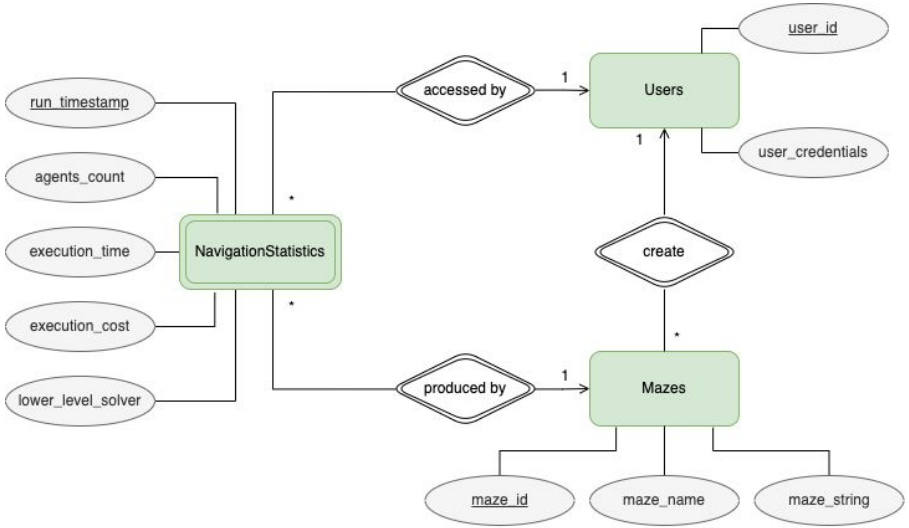
[2] System Design: Modified A* Search

Position Conflict
(Agent X, Position Y, Timestamp Z)



Edge Conflict
(Agent X, [Position A, Position B], Timestamp Z)

[2] System Design: Database Schema

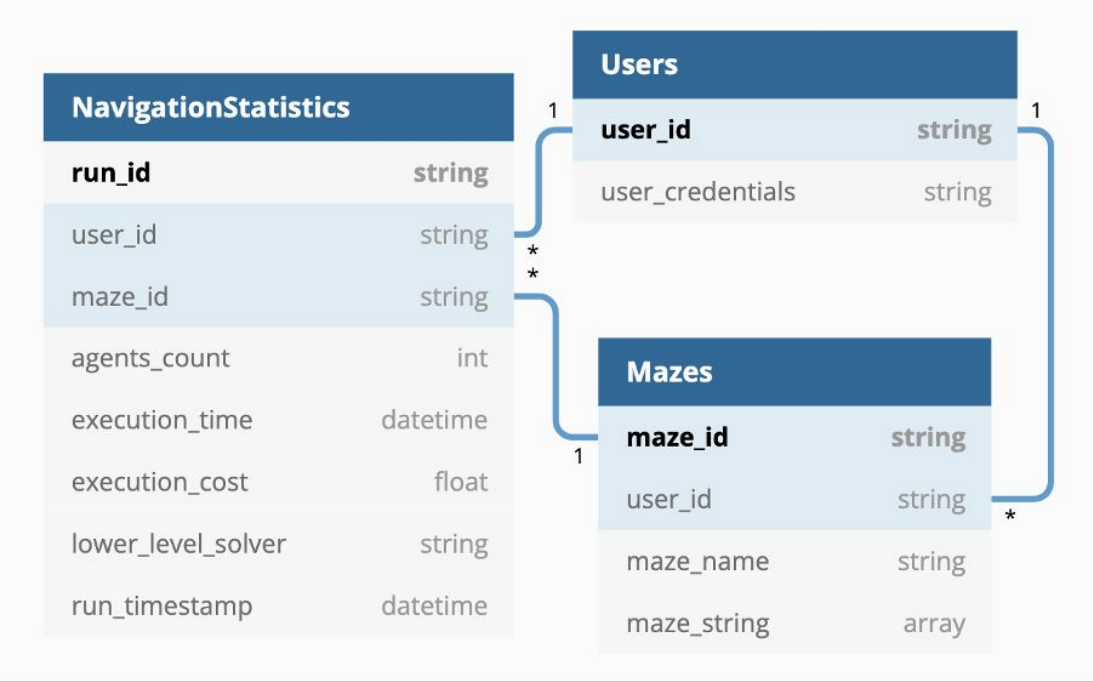


Entity Relationship Diagram

Users(<u>user_id</u>, user_credentials)
Mazes(<u>maze_id</u>, user_id, maze_name, maze_string) where *<u>user_id</u>* is a foreign key referencing *Users*
NavigationStatistics(<u>run_id</u>, user_id, maze_id, agents_count, execution_time, execution_cost, lower_level_solver, run_timestamp) where *user_id* and *maze_id* are foreign keys referencing *Users* and *Mazes* respectively

BCNF Decomposed Relational Schema

[2] System Design: Database Schema



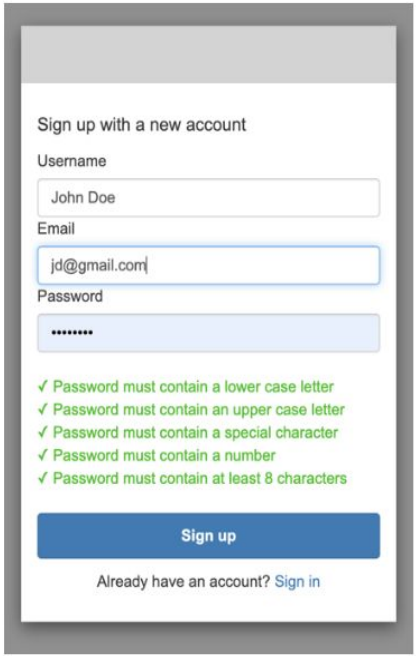
Database Schema

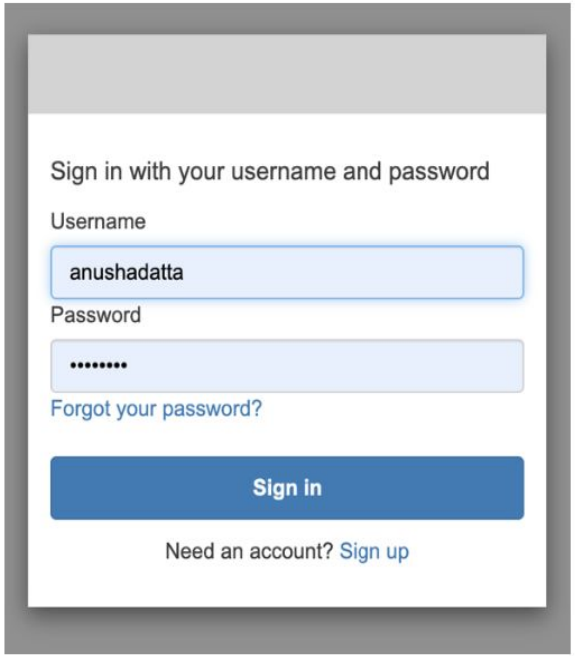
- **Motivation**
  Prevent
  - data redundancy
  - data inconsistencies
  - table join losses
  - insertion/updation/deletion anomalies
- **Normalisation**
  - Boyce Codd Normal Form
  - Armstrong's Axioms
- **Referential Integrity**
  - Enforced via Foreign Keys

[3] Implementation: Authentication

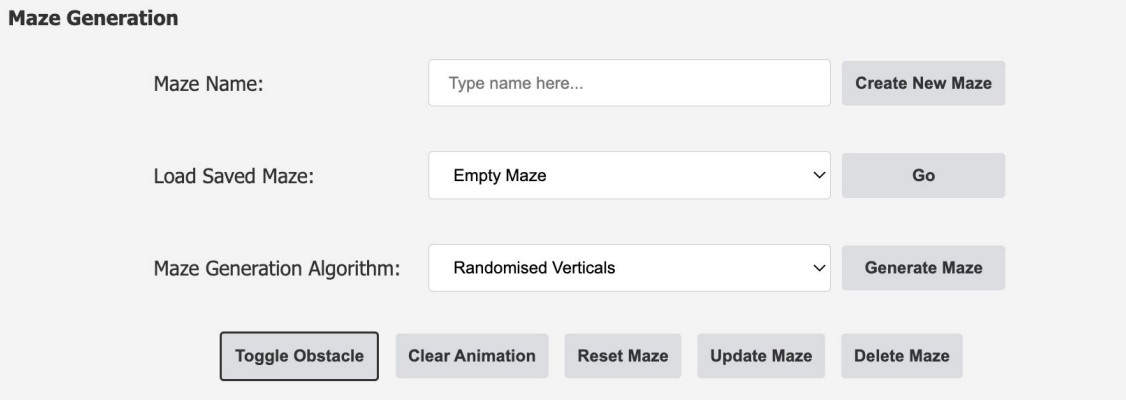
Authentication: AWS Cognito

- Complex Password Policy
- Browser Credentials Caching





[3] Implementation: Maze Management



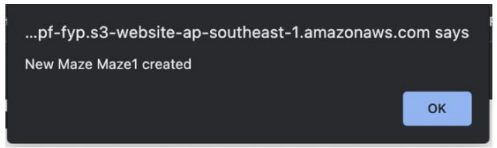
Maze Management

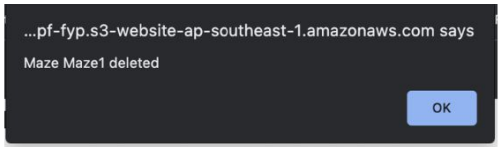
- Maze DB
  - Create/Load/Update/ Delete
- Maze Generation Algorithms
- Other features
  - Clear Animations
  - Reset Maze

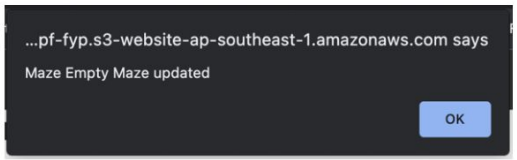
Shneiderman's Golden Rules

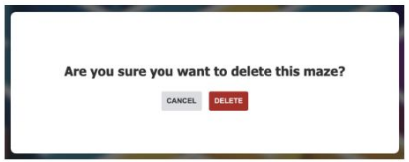
- Informative Dialog Box
- Confirmatory Dialog Box

to ensure Easy Reversal of Actions, offer Informative Feedback and Yield Closure









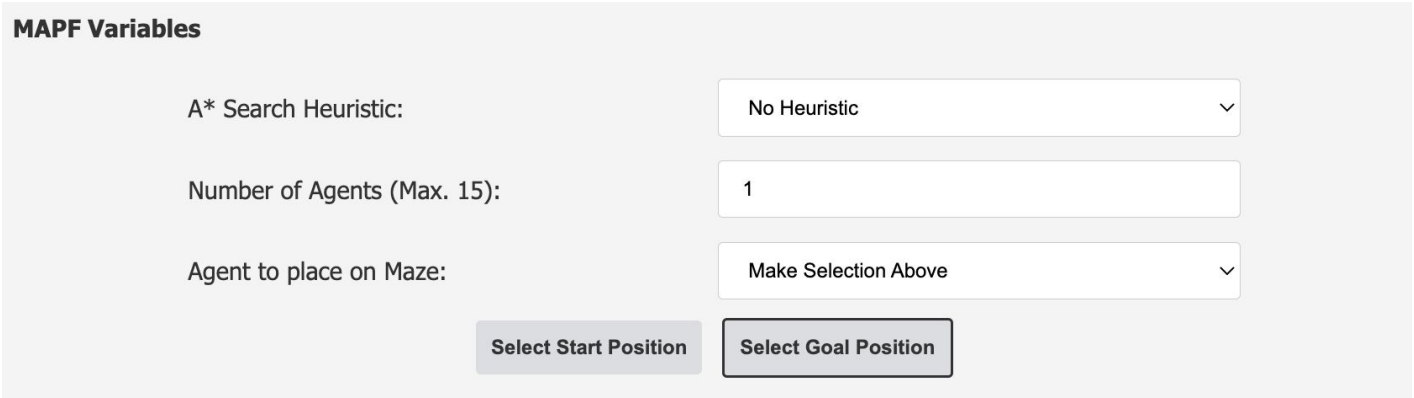
[3] Implementation: Multi Agent Path Finding

Multi Agent Path Finding Parameters

- A* Search Heuristic: No Heuristic, Manhattan Distance, Chebyshev Distance, Euclidean Distance
- Number of Agents: 1 to 15
- Agent Placement: Dynamic dropdown population

Inbuilt Error Checking

- Invalid Agent Count
- Invalid Agent Placement



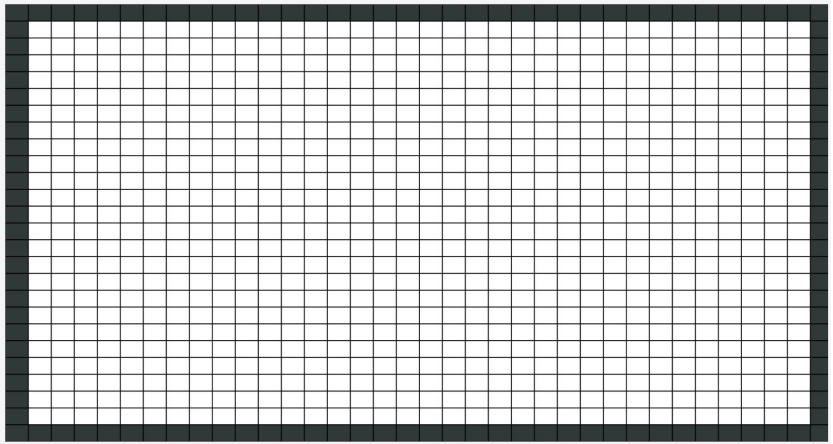
[3] Implementation: Visualisation

Visualisation: p5js Library

- Access hosted version of p5js via CDN
- Define maze layout parameters
- Initialise and set up grid maze

```
const HEIGHT = 500; //pixels
const WIDTH = 950; //pixels
const mazeHeight = 32; //units
const mazeWidth = 52; //units
const mazeHeightUnit = HEIGHT / mazeHeight; // pixels/unit
const mazeWidthUnit = WIDTH / mazeWidth; // pixels/unit
```

```
    p5.setup = function () {
        p5.createCanvas(WIDTH, HEIGHT);
        p5.background(220);
        initGrid(p5);
    };
```



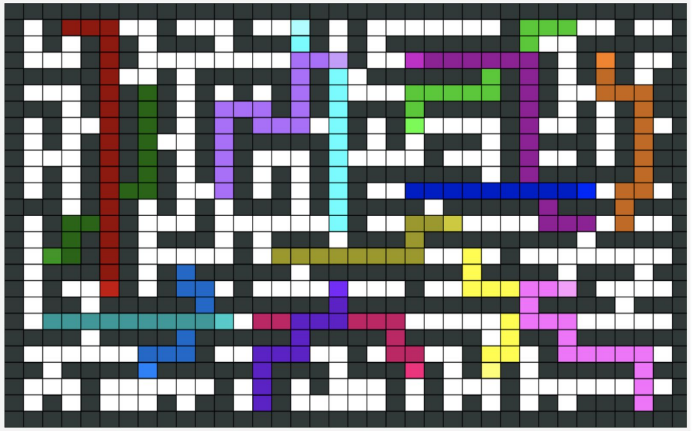
[3] Implementation: Visualisation

Visualisation: p5js

- Inbuilt function touchStarted() for maze manipulation
  - Toggle Obstacle
  - Agent Placements
  - Visualise MAPF Solution

- Agent Placements represented by gradients of same colour

```
// p5js inbuilt method to detect screen touches
p5.touchStarted = () => {
    place = onTouchFunctions[onClickFlag];
};
```



Agent 1: Start	Agent 1: Goal
Agent 2: Start	Agent 2: Goal
Agent 3: Start	Agent 3: Goal
Agent 4: Start	Agent 4: Goal
Agent 5: Start	Agent 5: Goal
Agent 6: Start	Agent 6: Goal
Agent 7: Start	Agent 7: Goal
Agent 8: Start	Agent 8: Goal
Agent 9: Start	Agent 9: Goal
Agent 10: Start	Agent 10: Goal
Agent 11: Start	Agent 11: Goal
Agent 12: Start	Agent 12: Goal
Agent 13: Start	Agent 13: Goal
Agent 14: Start	Agent 14: Goal
Agent 15: Start	Agent 15: Goal

[3] Implementation: Visualisation Model

Subscriber – Publisher Architectural Design Pattern [Flag System]

Achieves maze editing functionality for diverse elements

- Maze Flag
- Visualise Solution Flag
- Clear Flag
- Clear Animation Flag

Custom Time Delay Function

- Rapid Refresh Cycle: p5js Render Cycle invoked nearly 60 times every second
- Time Delay to intercept cycle and make path traversal visualisation more perceptible

```
function delay(time) {
    return new Promise(resolve => setTimeout(resolve, time));
}
```

[3] Implementation: Multi Agent Path Finding

MAPF Agent Start & Goal Placements

[3] Implementation: Multi Agent Path Finding

MAPF Solution Path Traversal Visualisation

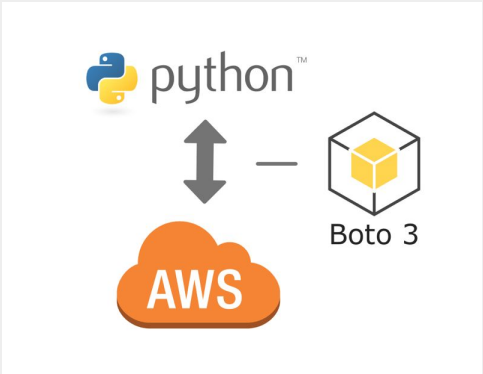
Execution Cost	197	Execution Time	1.8419

- Execution Statistics recorded for every MAPF run
- Subsequently funnelled into Navigation Statistics Pipeline

[3] Implementation: Server Side

Boto3

- Python AWS SDK
- Python API for AWS Infrastructure Services



Serverless Application Manager (SAM)

- Aid configuration and deployment of MAPF Application
- Access via AWS CLI

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
$ sudo installer -pkg AWSCLIV2.pkg -target /
```

```
#Step 1 - Initialise application
sam init

#Step 2 - Build application
cd sam-app
sam build

#Step 3 - Deploy application
sam deploy --guided
```

[3] Implementation: Server Side

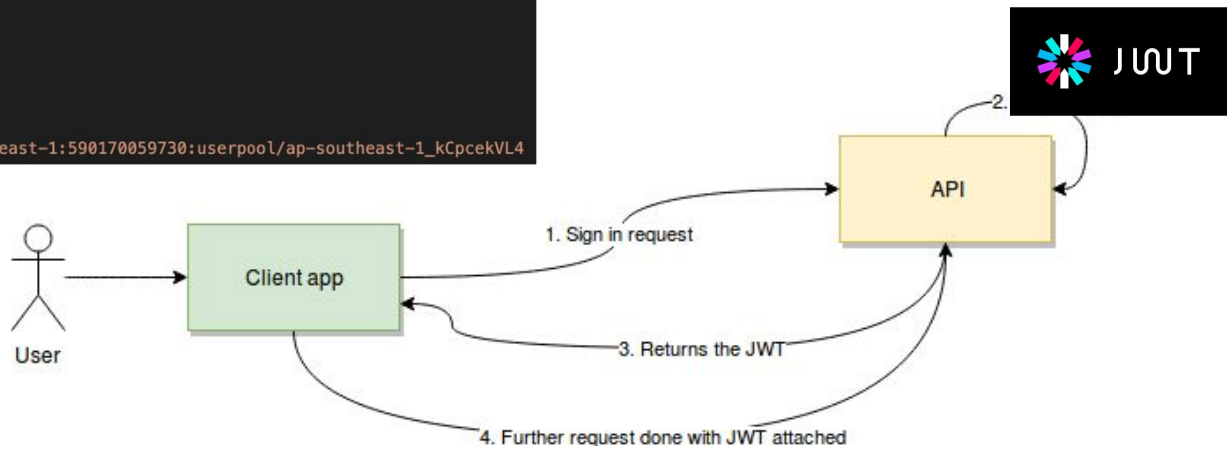
Authentication: AWS Cognito

- Configure Authorisers for existing AWS Cognito Pool
- Successful Authentication produces JWT, used to access AWS services via API Gateway
- JWT (ID Token) expires every 30 minutes, requiring re-authentication

```
ApiGateway:
  Type: AWS::Serverless::Api
  Properties:
    StageName: default
    Cors: "'*'"
    Name: MapfGateway
    Auth:
      DefaultAuthorizer: CognitoAuthorizer
      Authorizers:
        CognitoAuthorizer:
          UserPoolArn: arn:aws:cognito-idp:ap-southeast-1:590170059730:userpool/ap-southeast-1_kCpcekVL4
```

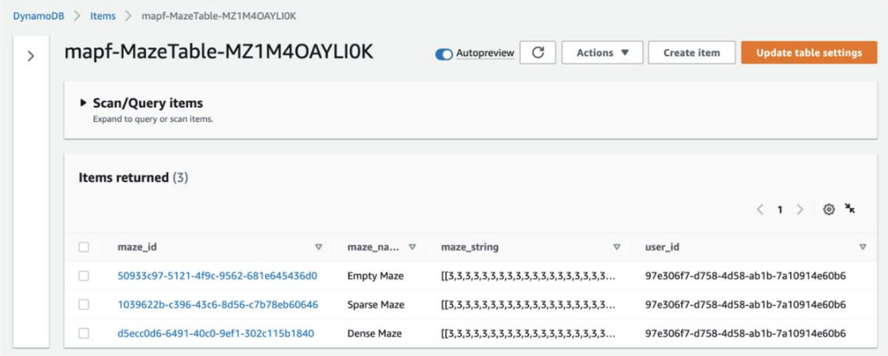
> HEADER
> PAYLOAD
> SIGNATURE

Java Web Token (JWT)
Base64-encoded JSON string that contains user information claims



[3] Implementation: Cloud Datastore

Cloud Datastore: AWS DynamoDB



DynamoDB > Items > mapf-MazeTable-MZ1M4OAYLI0K

mapf-MazeTable-MZ1M4OAYLI0K

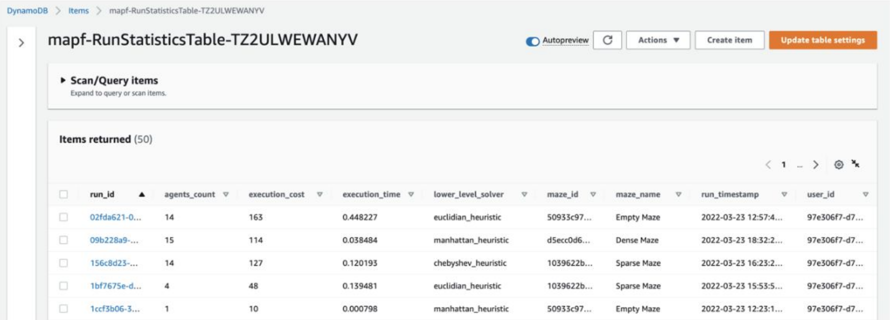
Autopreview | Actions | Create item | Update table settings

Scan/Query items

Expand to query or scan items.

Items returned (3)

maze_id	maze_na...	maze_string	user_id
50933c97-5121-4f9c-9562-681e645436d0	Empty Maze	[[3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3...	97e306f7-d758-4d58-ab1b-7a10914e60b6
1039622b-c396-43c6-8d56-c7b78eb60646	Sparse Maze	[[3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3...	97e306f7-d758-4d58-ab1b-7a10914e60b6
d5ecc0d6-6491-40c0-9ef1-302c115b1840	Dense Maze	[[3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3...	97e306f7-d758-4d58-ab1b-7a10914e60b6



DynamoDB > Items > mapf-RunStatisticsTable-TZ2ULWEWANYV

mapf-RunStatisticsTable-TZ2ULWEWANYV

Autopreview | Actions | Create item | Update table settings

Scan/Query items

Expand to query or scan items.

Items returned (50)

run_id	agents_count	execution_cost	execution_time	lower_level_solver	maze_id	maze_name	run_timestamp	user_id
02fda621-0...	14	163	0.448227	euclidian_heuristic	50933c97...	Empty Maze	2022-03-23 12:57:4...	97e306f7-d7...
09b228a9-...	15	114	0.038484	manhattan_heuristic	d5ecc0d6...	Dense Maze	2022-03-23 18:32:2...	97e306f7-d7...
156c8d23-...	14	127	0.120193	chebyshev_heuristic	1039622b...	Sparse Maze	2022-03-23 16:23:2...	97e306f7-d7...
1bf7675e-d...	4	48	0.139481	euclidian_heuristic	1039622b...	Sparse Maze	2022-03-23 15:53:5...	97e306f7-d7...
1ccf3b06-3...	1	10	0.000798	manhattan_heuristic	50933c97...	Empty Maze	2022-03-23 12:23:1...	97e306f7-d7...

- Modifications to DynamoDB made to template.yaml configuration file and deployed using SAM
- Referential Integrity enforced for maze and corresponding statistics

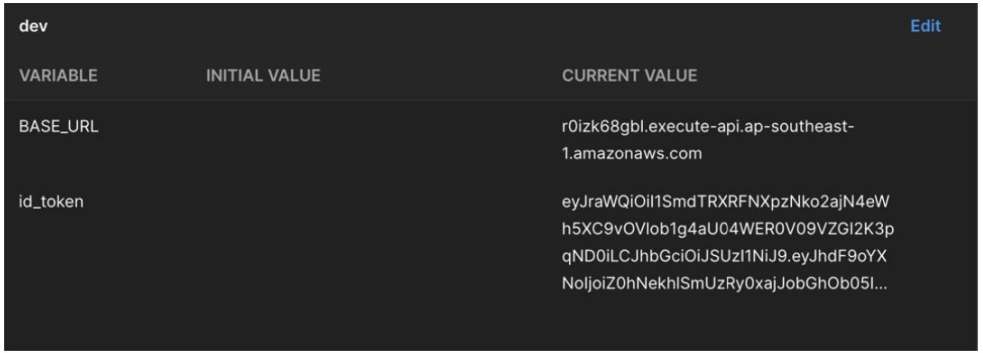
- Principle of Least Privilege
  - AWS Permissions Model
  - Function Policy for each Table

```yaml
Policies:
  - DynamoDBCrudPolicy:
      TableName: !Ref RunStatisticsTable
  - DynamoDBCrudPolicy:
      TableName: !Ref MazeTable
```

[3] Implementation: API Endpoints

Endpoint	HTTP Method	Action
/maze	GET	Retrieve all mazes created by authenticated user from Maze Table
/maze	POST	Create new maze under authenticated user from Maze Table
/maze	PUT	Update a maze record in Maze Table by modifying the maze layout
/maze	DELETE	Delete a maze in Maze Table
/stats	GET	Retrieve all navigation statistics from Run Statistics Table associated with all execution runs performed on user's mazes
/solve	POST	Perform a MAPF execution run, return computed solution and update Run Statistics Table with the corresponding navigation statistics for that execution run

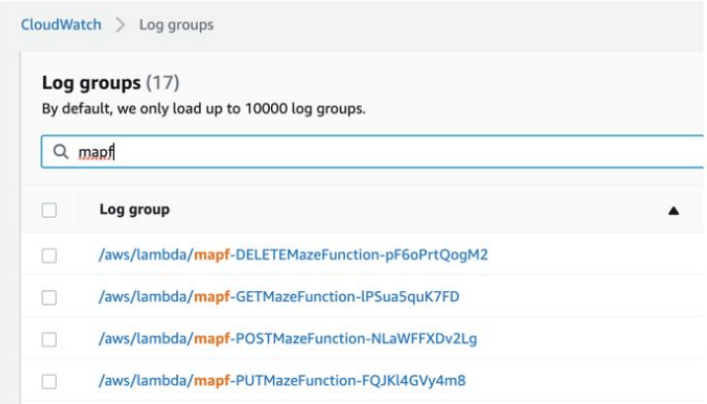
- HTTP method request to an API Gateway endpoint invokes corresponding Lambda function
- Postman for unit testing, troubleshooting and error isolation

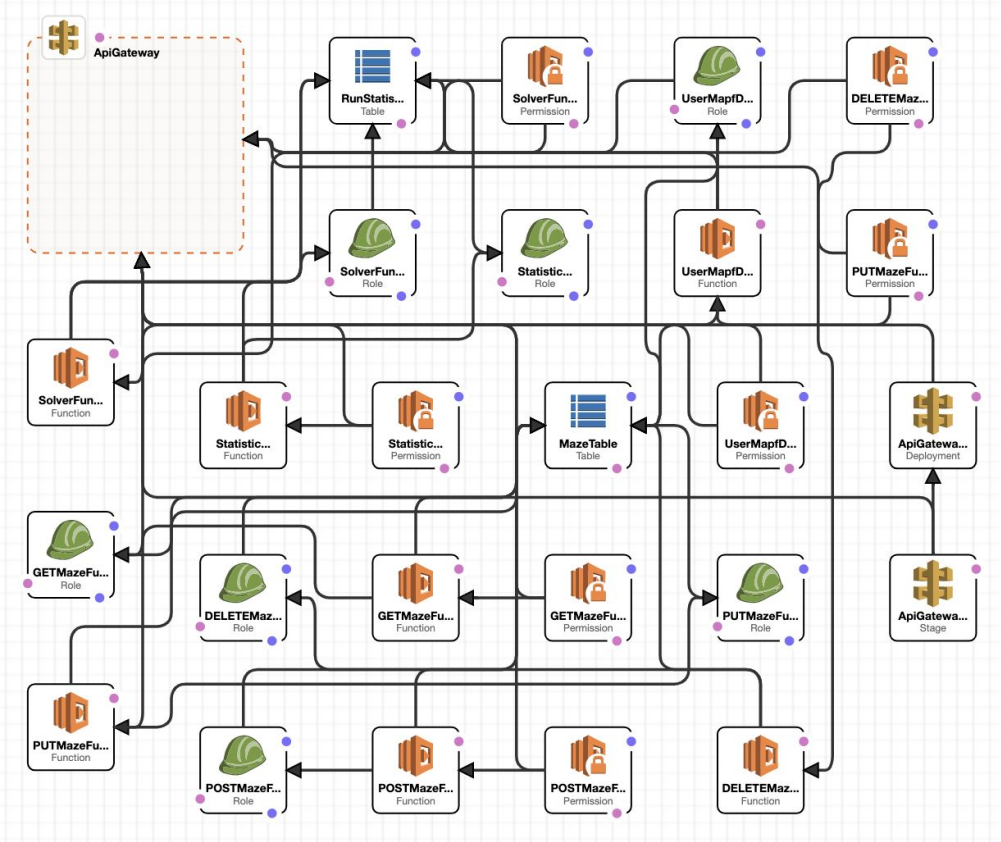


[3] Implementation: Server Side

CloudWatch
Log Group for API > Log Stream for each Invocation > Log Events (Invocation details)

CloudFormation
Visualise backend interactions between Tables, Lambda Functions, API Gateway Stages, Roles and Permissions



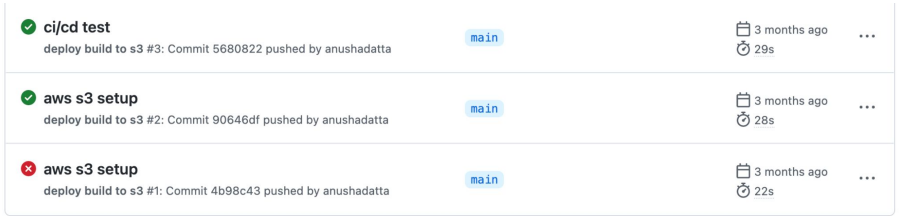


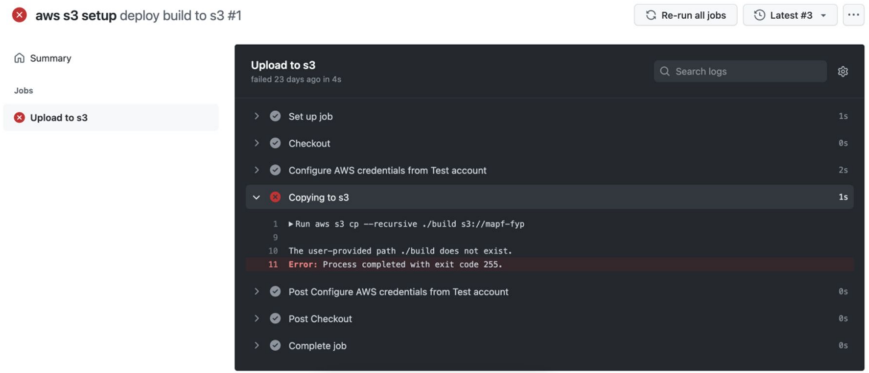
[3] Implementation: CI/CD Workflow

Continuous Integration/Development
Approach to automate application development stages from building & testing to delivery & deployment

GitHub Actions

- Workflows to enforce CI/CD
- Workflow Run Execution History
- Workflow Specific Logs





[4] Verification & Validation

Verification

Verify that final software developed conforms to specifications outlined in the system design phase

- Investigative Periodic Reviews
- Application Walkthrough Inspections

Validation

Validate software is free of any bugs, meets users' needs and functions as per intended use

- Unit Testing
- White Box Testing
- Black Box Testing
- Integration Testing

[5] Maintenance

Corrective Maintenance
Reactive changes to mitigate software defects or bugs

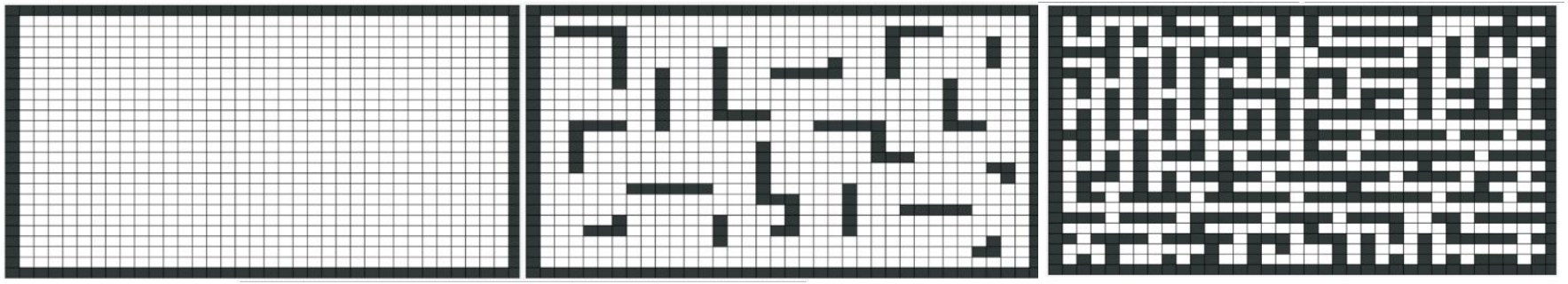
Preventive Maintenance
Detect and mitigate software errors before they take effect

Adaptive Maintenance
Modification of software after change in operating environment

Perfective Maintenance
Target improvements in software functionality and usability

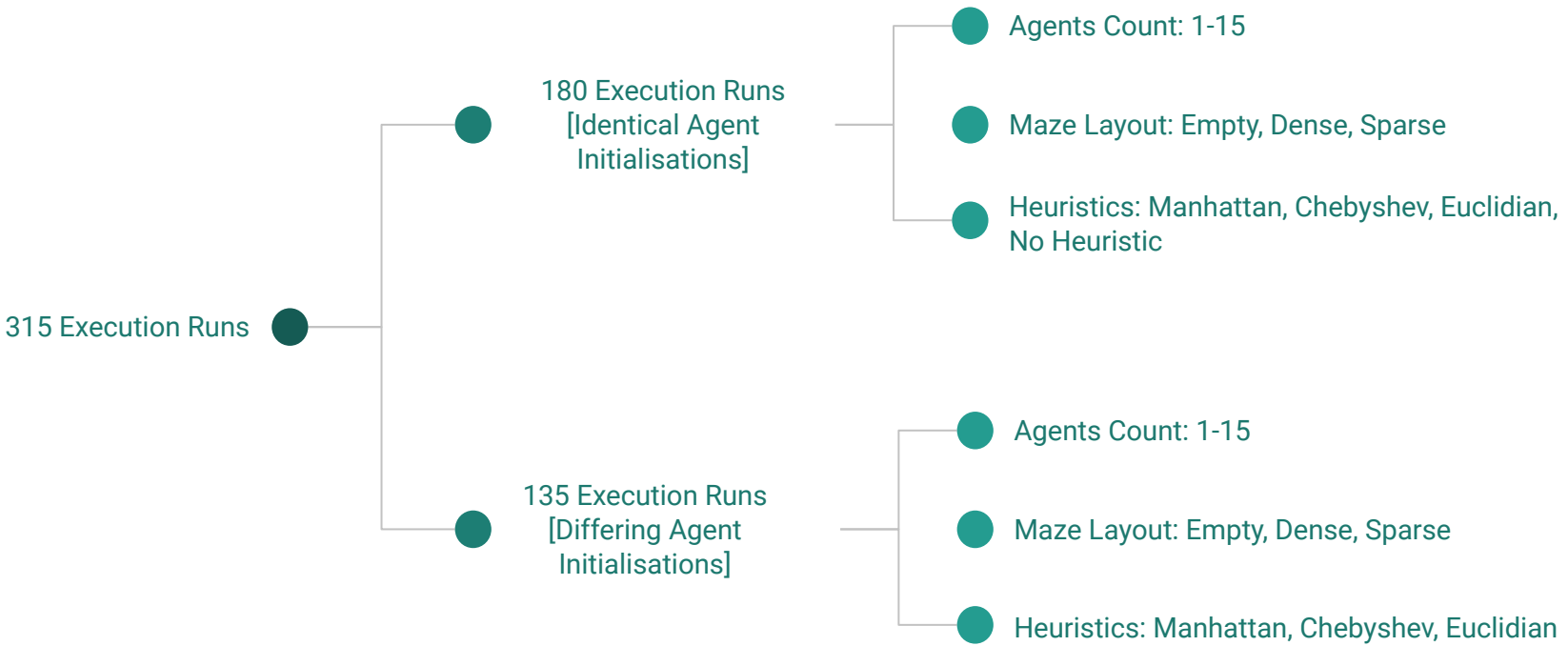
Experiments

Agents Count: 1 to 15
Lower Level Search: A* Search
Heuristics: Manhattan, Chebyshev, Euclidian, No Heuristic
Maze Density: Empty, Sparse, Dense

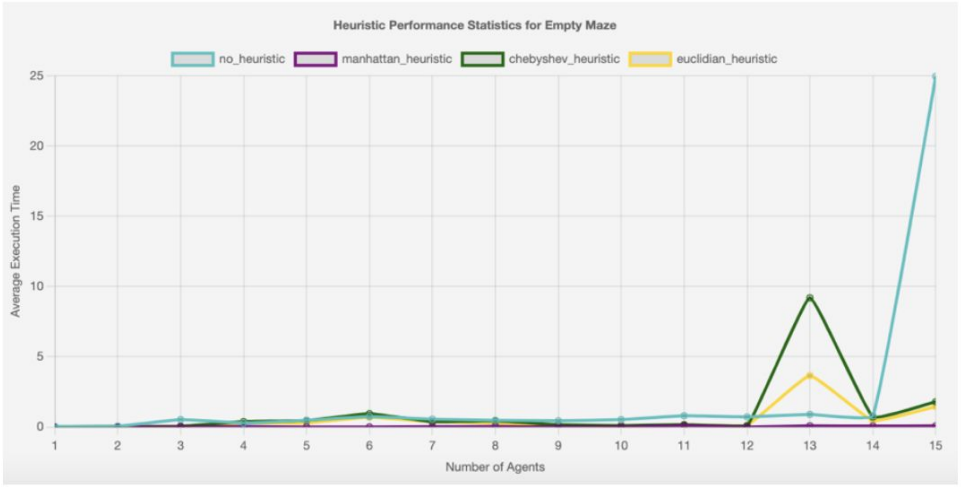


- **Identical Agent Initialisations:** Compare Execution Performances (w.r.t. Execution Time)
- **Differing Agent Initialisations:** Observe Execution Distribution (w.r.t. Execution Cost)

Experiments

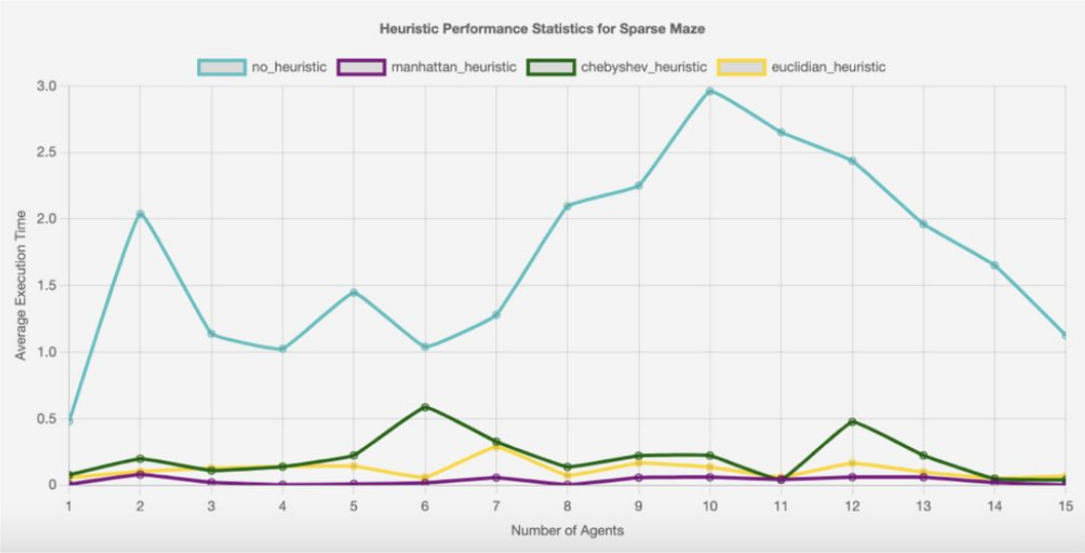


Experimental Findings: Execution Time for Empty Maze



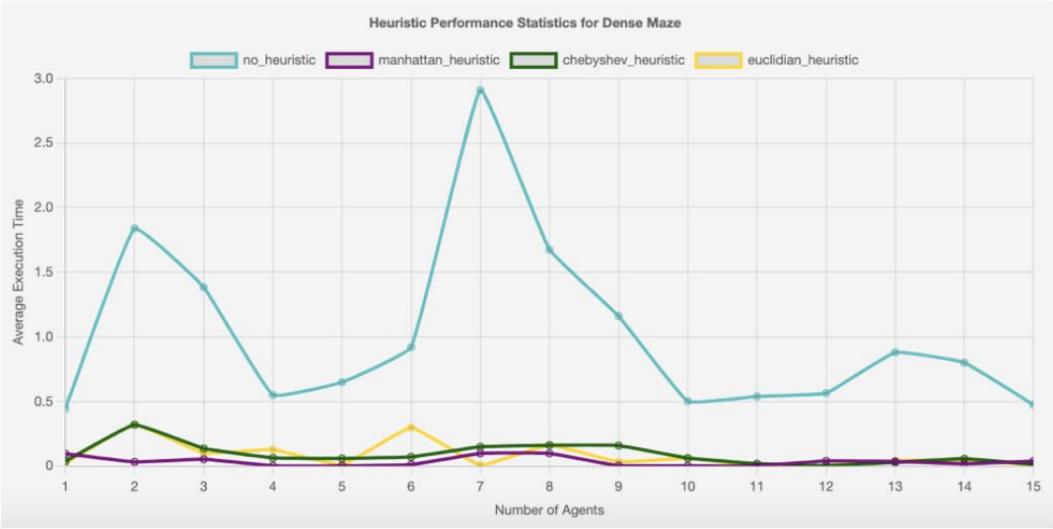
- No Heuristic performs the worst, peaking at a higher agent count
  - No Heuristic simulates Uniform Cost Search which is highly inefficient
- Manhattan Distance Heuristic performs the best with the lowest execution time
  - Most appropriate heuristic for four allowable movement actions
- Comparable performances across heuristics, potentially due to lack of obstacles in Empty Maze
  - Fails to provide a sufficiently challenging maze environment
  - Cannot well distinguish various heuristic performances, especially for lower agent counts

Experimental Findings: Execution Time for Sparse Maze



- No Heuristic performs significantly worse
  - Inefficient, as previously established
  - Difference in performance may be augmented due to presence of obstacles
- Manhattan Distance Heuristic performs the best, with the lowest execution times
  - Most appropriate heuristic

Experimental Findings: Execution Time for Dense Maze



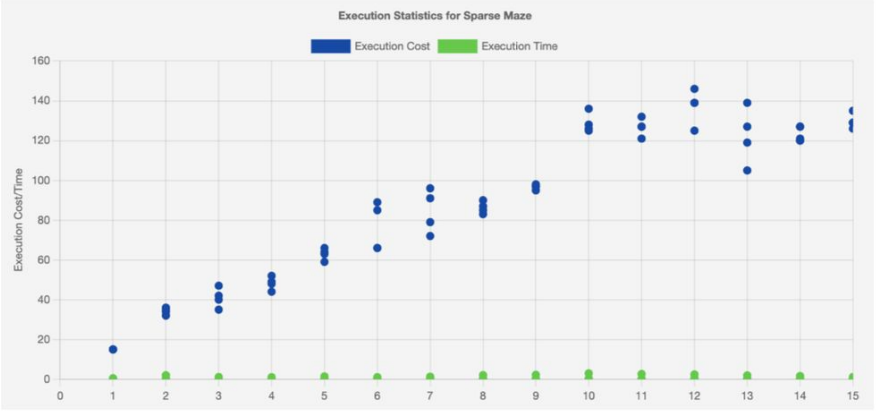
- No Heuristic performs significantly worse
  - Inefficient, as previously established
  - Difference in performance may be augmented due to concentrated presence of obstacles

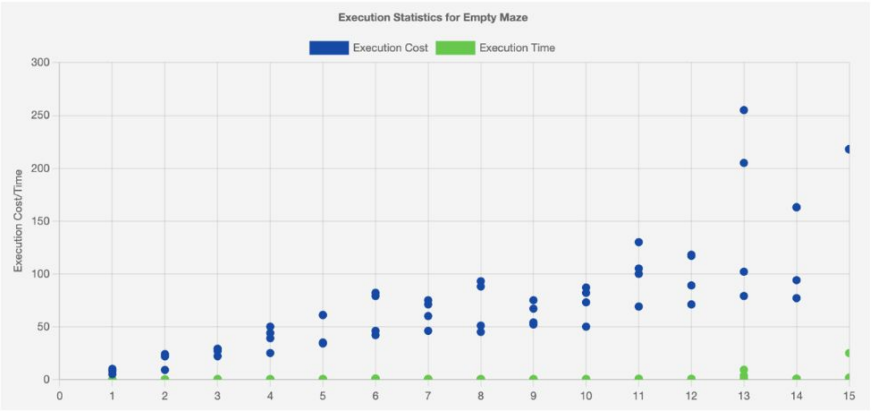
- All Heuristics perform comparably, but Manhattan Distance Heuristic performs the best, with the lowest execution times
  - Most appropriate heuristic

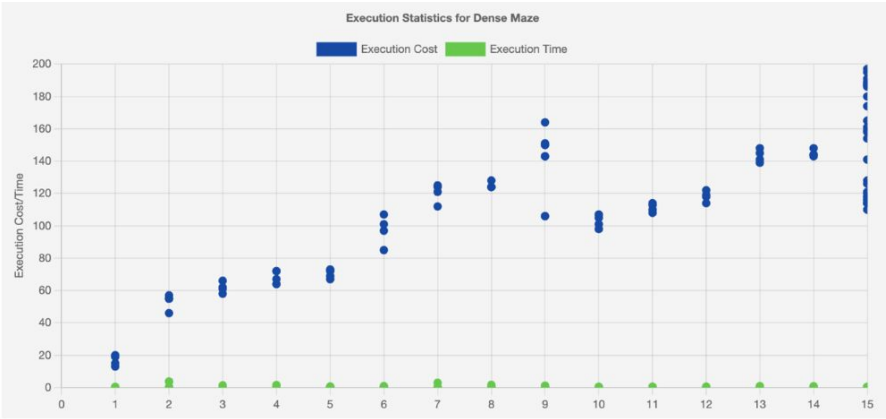
Experimental Findings: Execution Statistics

Execution Statistics

- Positive Correlation between Agent Count and Execution Cost & Time, regardless of maze layout
- Execution Time & Cost Low for Empty maze, Higher for Sparse Maze and Highest for Dense Maze



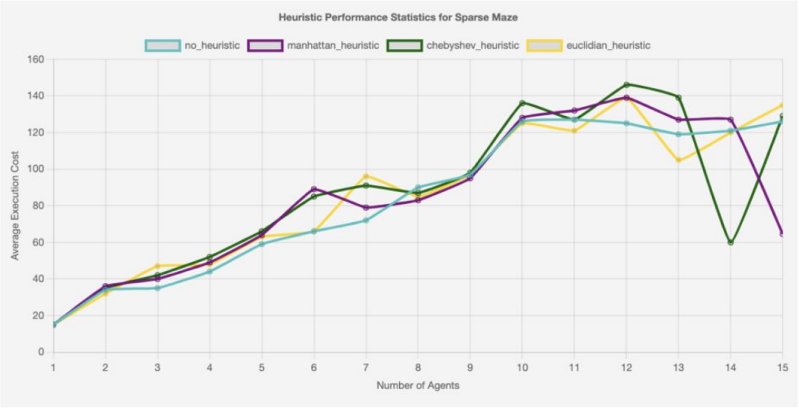


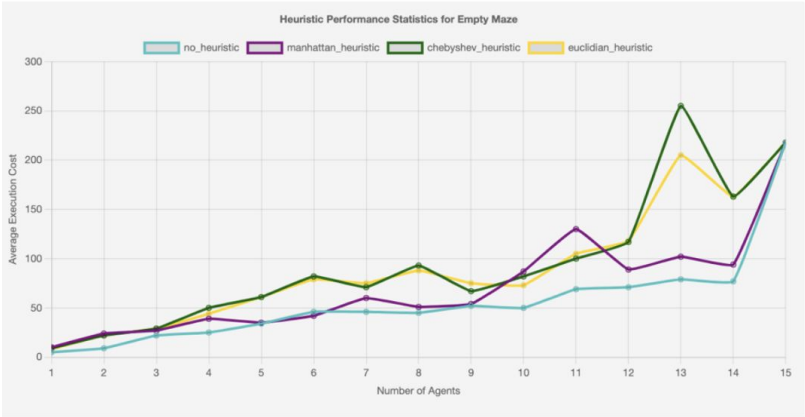


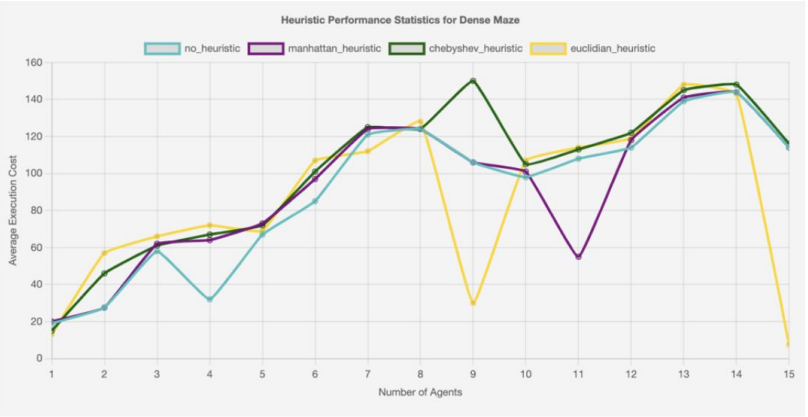
Experimental Findings: Execution Cost Distributions

Heuristic specific Average Execution Costs

- Cost distributions provide insights into agent initialisation patterns
- High execution costs insinuate agent initialisations sparsely positioned
- Low execution costs suggest agent initialisations are closely positioned







Future Works

- Explore other lower level search algorithms
  - Breadth First Search
  - Dijkstra’s
- Upload user configuration file for agent positions
  - suggested format JSON
- Implement Live MAPF application walkthrough tutorial
- Conduct User Acceptance Testing (UAT)
- Extend to Android and iOS platforms

Thank you!

Questions?