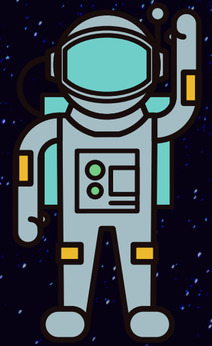
THE MARS COLONIZATION PROGRAM

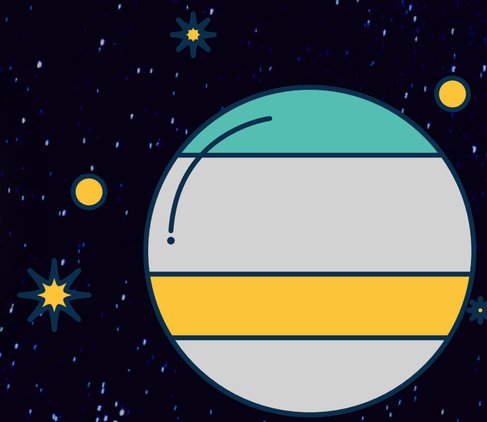
NAVIGATE THE MARS ROVER

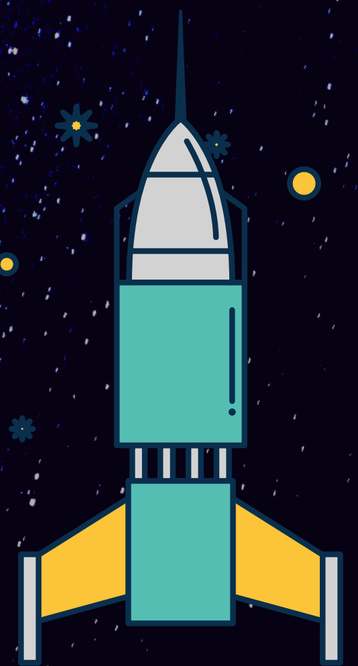


TEAM: MARS MATES

Harshita Malik

Nandini Agrawal





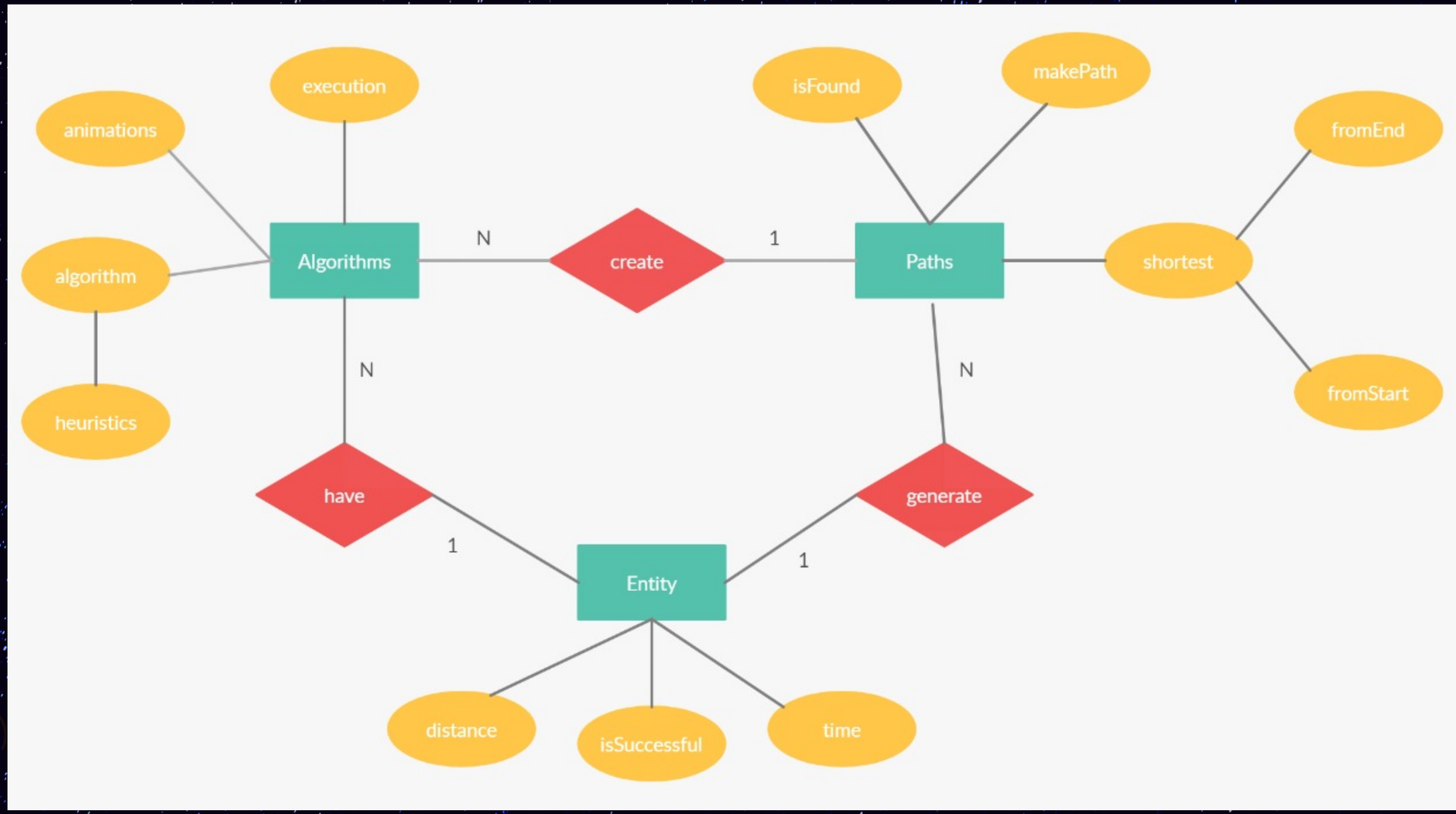
Project Overview:

This project finds the shortest path for the mars rover using the Dijkstra, Breadth First Search, Greedy Best First Search, and A* Algorithms, along with heuristics. It also gives an option for diagonal path for all algorithms and bidirectional path for A* and Breadth First Search Algorithm.

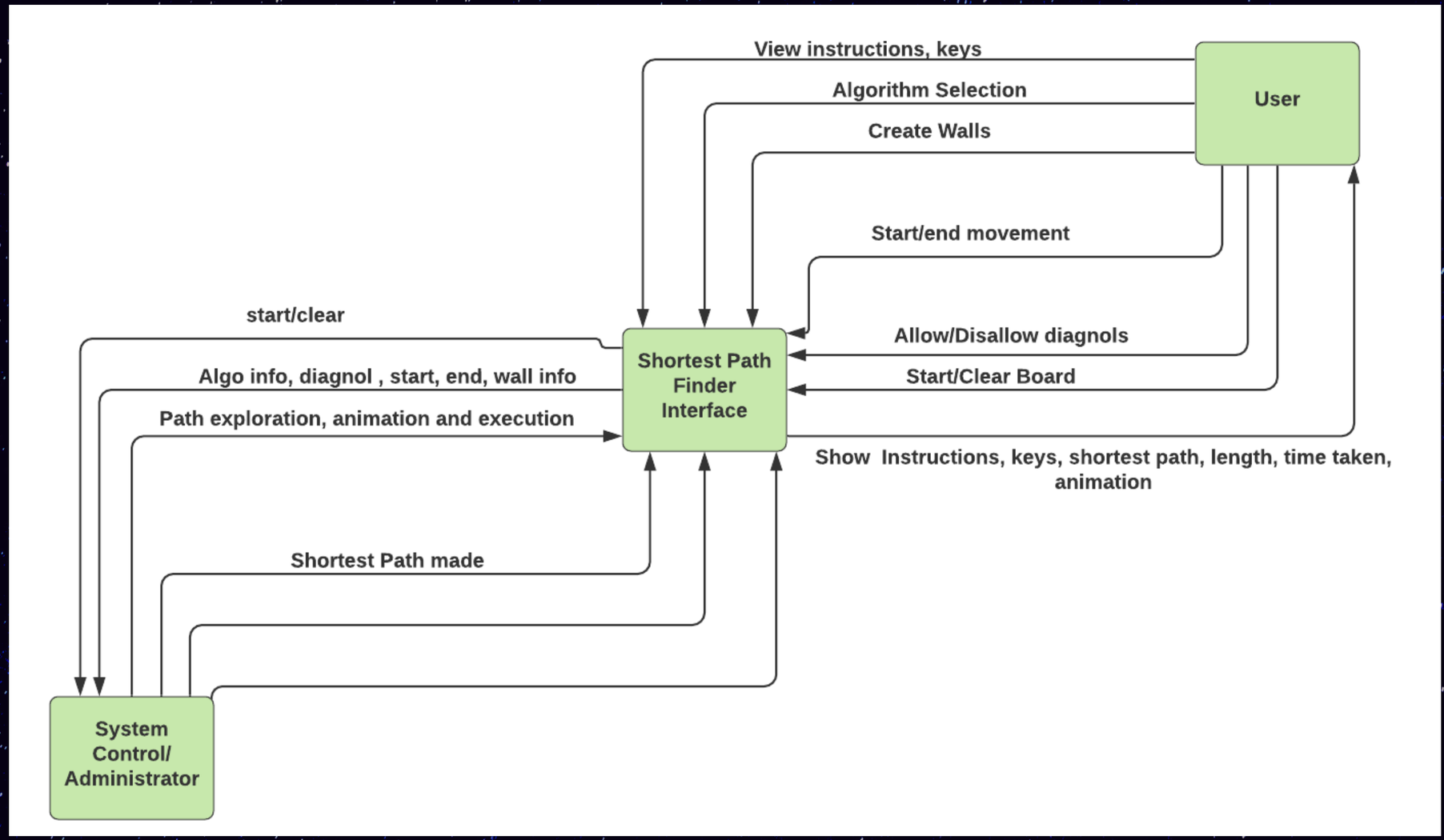
There is one Starting Position from where the path of rover begins and two Destination Points. Out of the two Destination Points, the path is made to the point which is closer to the Starting Position and for which the rover has to travel a shorter distance.

After the user has selected the specifications, the path execution and its making is shown on screen, followed by the display of the final path with preferred destination point, length of path and time taken.

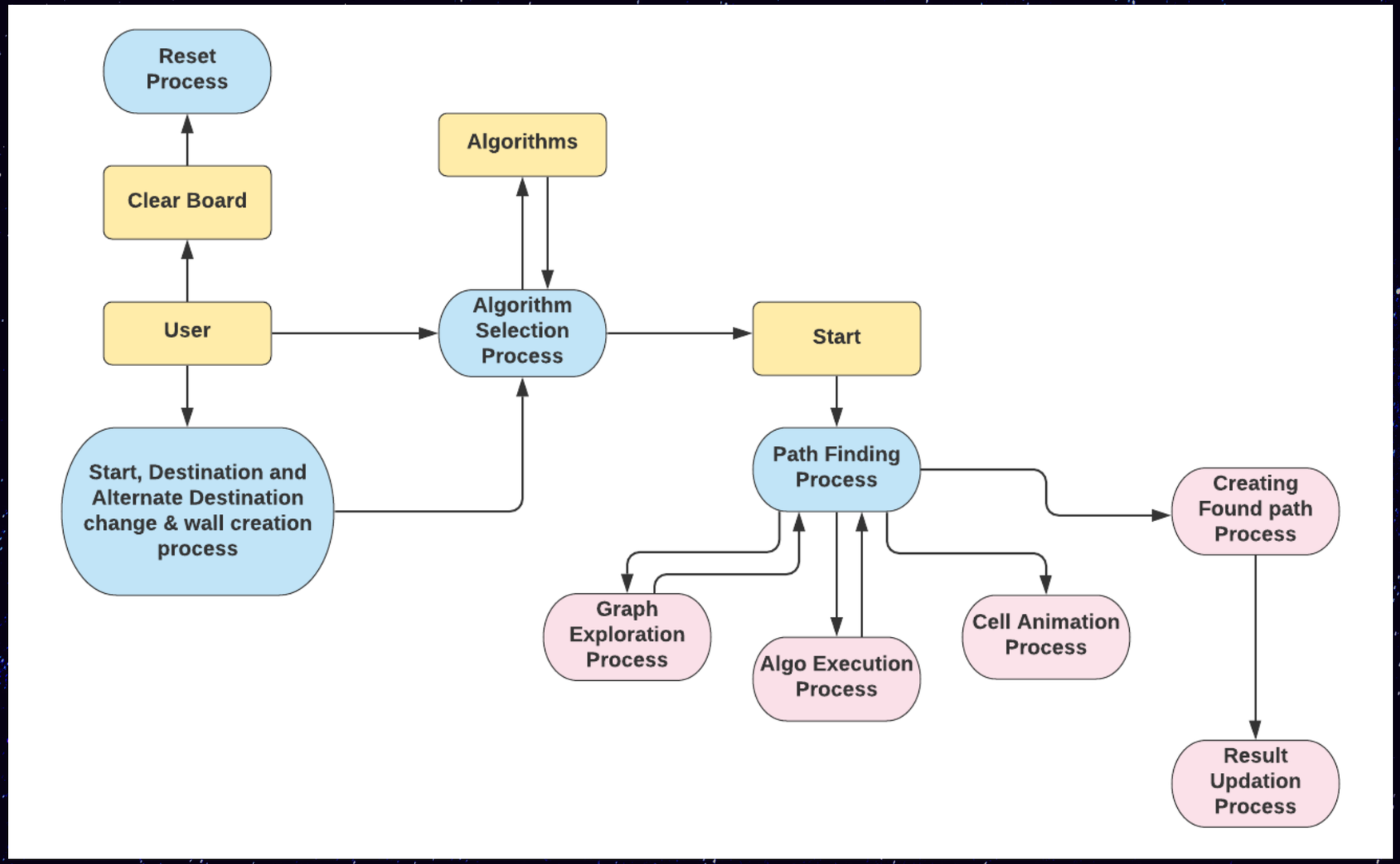
ER Model:


execution
animations
isFound
makePath
fromEnd
algorithm
Algorithms
N
create
1
Paths
shortest
heuristics
N
N
fromStart
have
generate
1
Entity
1
distance
isSuccessful
time

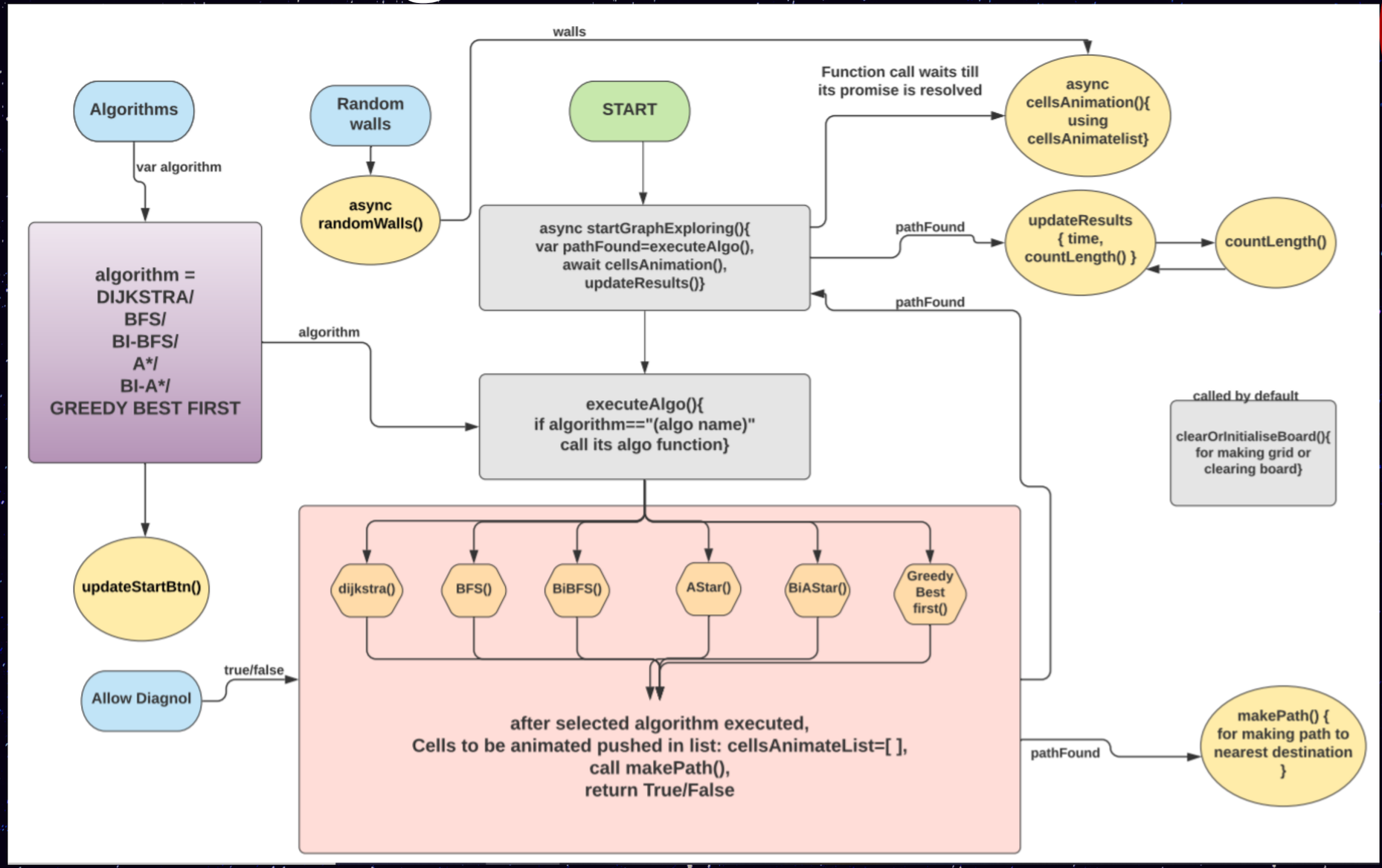
Context Diagram:



Low Level Diagram:



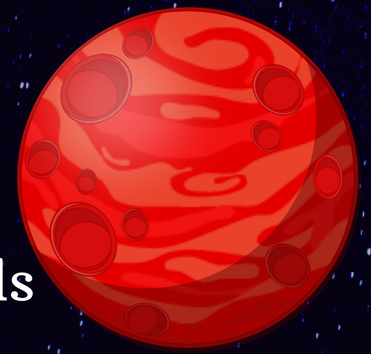
High Level Diagram:



What is Solved:

- Finds the shortest possible path using algorithms:
  a. Dijkstras (Diagonal Path also possible)
  b. Greedy Best First Search (Diagonal Path also possible)
  c. Breadth First Search (Diagonal and Bidirectional also possible)
  d. A* (Diagonal and Bidirectional also possible)
- Allows to choose from four heuristics.
- All options and selections are grouped together in a Navigation Bar.
- Shows color codes and cells nomenclature in Key Button.
- Instructions are shown with the help of a button. All the other functions are temporarily disabled when user is reading the instructions.
- Algorithm which will execute after selection is shown at Start Button.

What is Solved:



- Allows to create walls in the path either by generating random walls or by clicking on the cells and selecting.
- In Unidirectional Search it finds the closer destination, out of the two destinations, and makes path with it.
- In case of equal paths with both destination points, it forms path to any one of them.
- Shows the searching cells and visited cells during the execution of algorithm, with the help of animations, till the destination cell is not found.
- Shows the length of path and time taken in result.

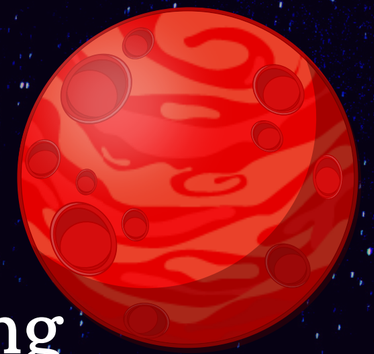
What is Not Solved:

- IDA* and Jump Search Algorithms have not been included.
- Pause button has not been included.
- Operations performed have not been shown in results.

Functions Used:

- generateGrid(): creates the grid on screen
- moveStartOrEnd(newIndex, startOrEnd): moves the selected cell on grid
- updateStartButton(): updates selected algorithm on Start
- randomWalls(): creates walls randomly
- startGraphExploration(): starts exploring the graph for finding path
- executeAlgo(): executes selected algorithm

Functions Used:



- calcHeuristicDistance(x, y): calculates distance using selected heuristic
- update(message): updates message to be displayed
- updateResults(duration, pathFound, length): updates results
- makePath(path, pathFound, isEnd, tempNode): makes the found path according to specifications
- pathLength(): calculates length of shortest path

Functions Used:

- cellsAnimation(): animates cells
- clearOrInitialiseBoard(keepWalls): clears all paths & walls from grid
- makeWalls(): makes walls
- neighborsThatAreWalls(): finds neighbors that are walls
- createDistances(): makes distances between cells
- createPath(): makes paths between cells
- getNeighbors(x, y): finds all the neighbors of the cell

Functions Used:

- createWallsVisited(): creates all the walls which have been visited
- cellIsAWall(x, y, cells): checks if a cell is a wall or not
- manhattanDist(x, y): calculates distance using Manhattan
- euclideanDist(x, y): calculates distance using Euclidean
- octileDist(x, y): calculates distance using Octile
- chebyshevDist(x, y): calculates distance using Chebyshev

ALGORITHMS

- PSEUDO CODES
- FUNCTION CALLS
- EXAMPLES (SCREENSHOTS)

1. DIJKSTRA (UNI)
2. BREADTH FIRST SEARCH (UNI & BI)
3. GREEDY BEST FIRST SEARCH (UNI)
4. A* SEARCH (UNI & BI)

Dijkstras Algorithm: Pseudo Code

```
Mark distance at start cell = 0
Insert it into minHeap [distance, cell]
Add cell to animate cell list and mark it as searching
while minHeap is not empty do
    extract minimum and mark it visited
    break out of loop if either of destination cells reached
    for all neighbors do
        find distance and update if shorter distance found
        mark searching and add in animate cell list
        add to path
make the path that has been found to closer destination cell
return the path found
```

Dijkstras Algorithm: Function Calls

- minHeap()
- createDistances()
- createWallsVisited()
- createPath(path)
- getNeighbors(x coordinate, y coordinate)
- makePath(path, pathFound, whichDestinationPoint)

Breadth First Search: Pseudo Code

```
Add starting cell to Queue and mark visited
Add cell to animate cell list and mark it as searching
while Queue is not empty do
    pop the top cell out of Queue
    mark it visited and add to animate cell list
    break out of loop if current cell is destination or alternate destinaion cell
    find all neighbours of current cell and do
        mark them searching and add in animate cell list, add to Queue
        add to path
make the path that has been found to closer destination cell
return the found path
```

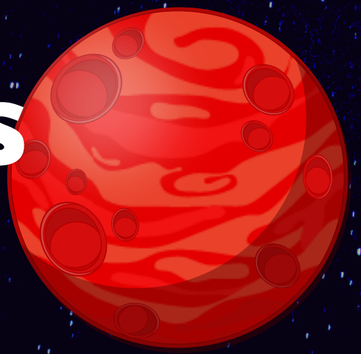
Breadth First Search: Function Calls

- Queue()
- createWallsVisited()
- createPath(path)
- getNeighbors(x coordinate, y coordinate)
- makePath(path, pathFound, whichDestinationPoint)

BiBreadth First Search: Pseudo Code

```
Add start/end cell to start/end list respectively
Mark them as open visited
while start list and end list is not empty do
    break out of loop if current cell is destination cell
    pop first node from start/end list and mark close visited
    add to animate cells list as visited
    find all neighbors of current cell and do
        if neighbor not close visited but open visited do
            if node already inspected by reverse search then pathfound, break
            add neighbor to start/end list, add to path, mark searching and add in
            animate cell list, mark open visited true
make the path that has been found from start and end
return the found path
```

BiBreadth First Search: Function Calls



- createWallsVisited()
- createPath(path)
- createDistances()
- getNeighbors(x coordinate, y coordinate)
- makePath(path, pathFound, destinationPoint, neighbor)

Greedy Best First Search: Pseudo Code

```
Insert start cell and its cost in minHeap
Add cell to animate cell list and mark it as searching
while minHeap is not empty do
    extract minimum from minHeap
    if not visited do
        mark visited and add to animate cells list
        break out of loop if either of destination cells reached
        for all neighbors do
            find new cost using Heuristics till neighbor for both destinations
                compare costs of both destinations and choose the lesser one as newCost
                if newCost smaller do update cost, path, minHeap, animate cell list searching
make the path that has been found to closer destination
return the path found
```

Greedy Best First Search: Function Calls

- minHeap()
- createWallsVisited()
- createPath(path)
- createDistances()
- getNeighbors(x coordinate, y coordinate)
- calcHeuristcDistance(x, y)
- makePath(path, pathFound, whichDestinationPoint)

A* Search Algorithm: Pseudo Code

Initialize distance and cost of start cell to zero
Insert into minHeap and searching in animate cells list
while minHeap is not empty do
    extract minimum and mark visited and add to animate cells list
    break out of loop if ether of destination cells reached
    find all neighbours of current cell and do
        calculate newDistance till neighbor & if shorter than distance
            update path, mark searching and add in animate cell list
        calculate cost for both destination cells and select lesser cost as newCost
        if lesser than cost do update and insert in minHeap
make the path that has been found to closer destination
return the found path

A* Search Algorithm: Function Calls

- minHeap()
- createWallsVisited()
- createPath(path)
- createDistances()
- getNeighbors(x coordinate, y coordinate)
- calcHeuristcDistance(x, y)
- makePath(path, pathFound, whichDestinationPoint)

BiA* Search Algorithm: Pseudo Code

Mark start and end cell open visited and distance as zero
Add start/end cell to start/end open minHeap respectively
Add start cell to animate cells list as searching
while start list and end list is not empty do
    pop minimum from start/end list and mark close visited
    find all neighbors of current cell and do
        if neighbor open visited & not close visited do
            if node already inspected by reverse search then pathfound, break
        calculate newDistance till neighbor

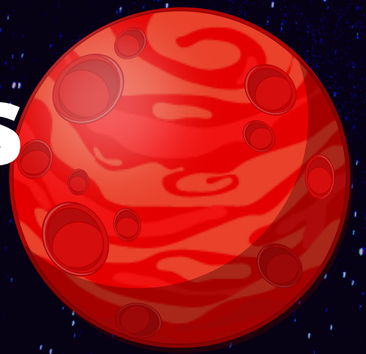
(continued...)

BiA* Search Algorithm: Pseudo Code

(continued...)

```
        if neighbor not open visited or newDistance is shorter do
            update distance, add to path, mark searching and add in animate
            cells list, calculate newCost using Heuristics with neighbor
            if not open visited do
                mark open visited, (update bylist-for checking reverse search)
            if newCost lesser do
                update cost & add cell & cost to start/end minHeap
    make path that has been found from start and end
    return the path found
```

BiA* Search Algorithm: Function Calls



- minHeap()
- createWallsVisited()
- createPath(path)
- createDistances()
- getNeighbors(x coordinate, y coordinate)
- calcHeuristcDistance(x, y)
- makePath(path, pathFound, destinationPoint, neighbor)

Thank You!

:)