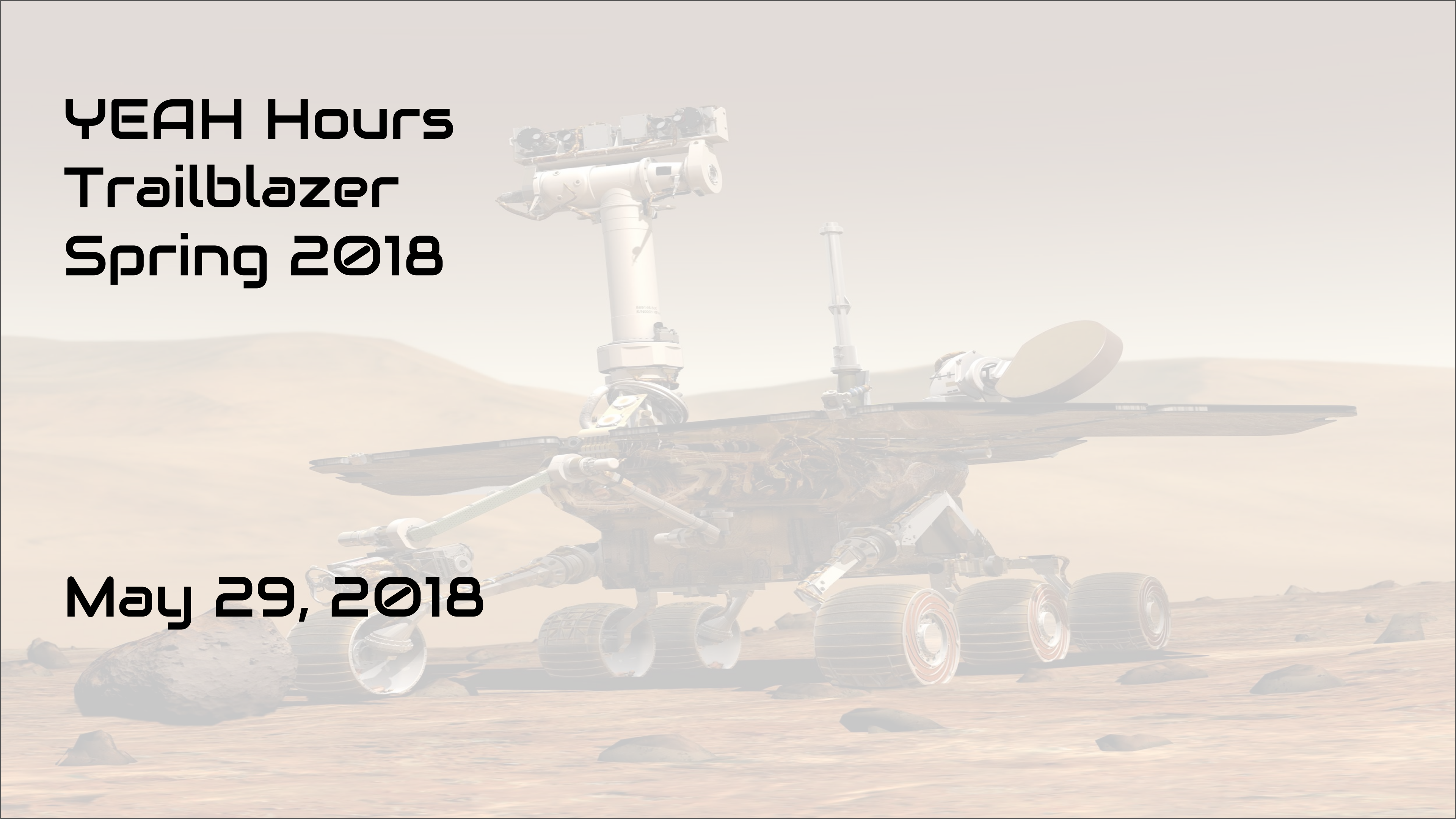


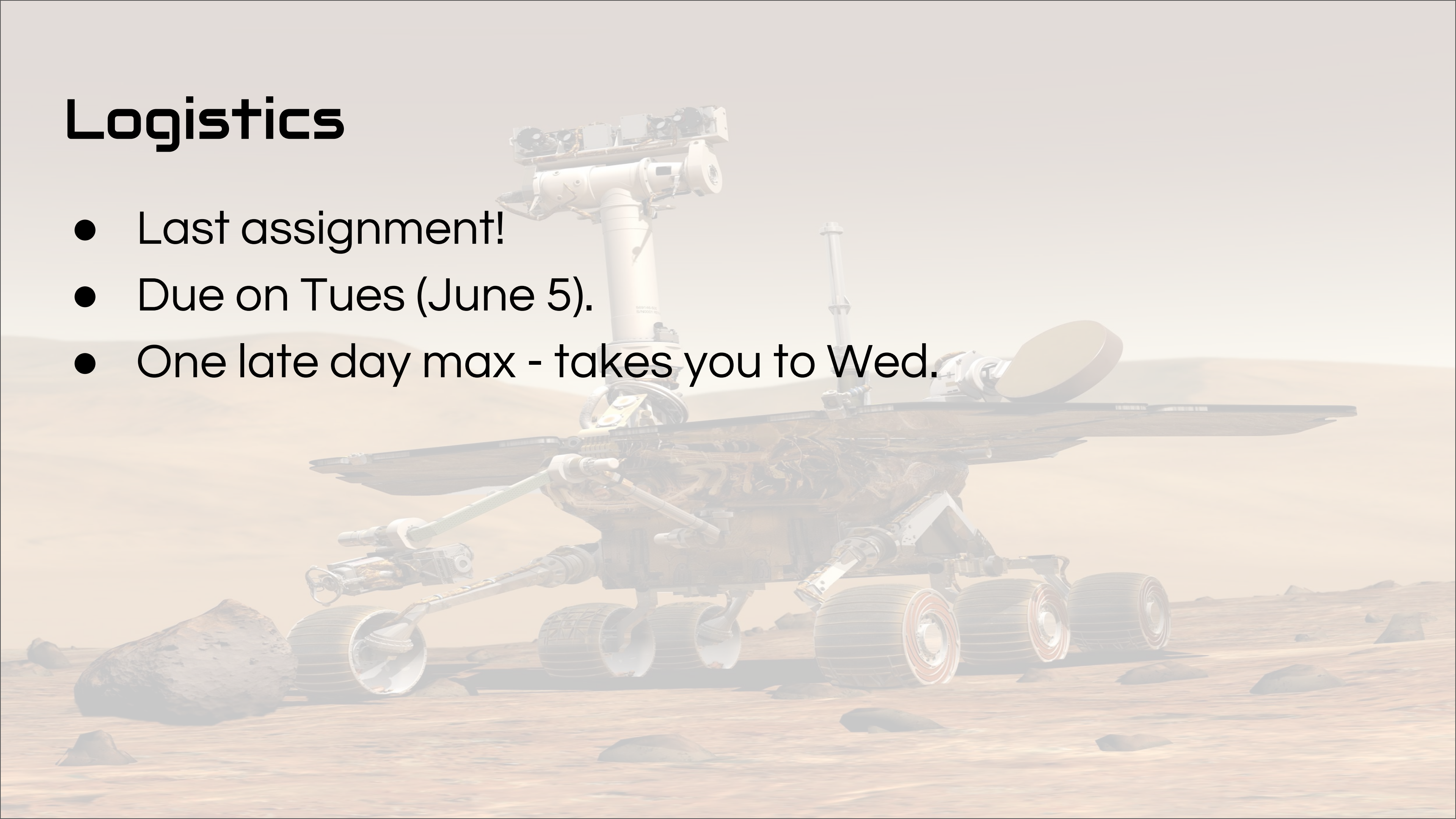
YEAH Hours
Trailblazer
Spring 2018

May 29, 2018

Agenda

- Logistics
- Assignment overview
- Demo
- Deep dive
- Questions
- Feel free to stop me anytime for questions!

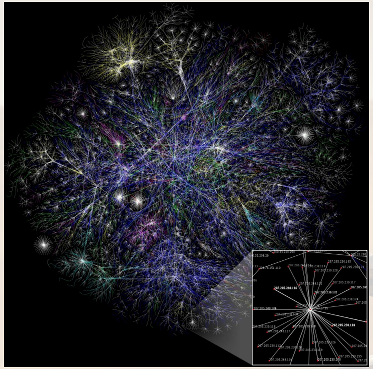
Logistics

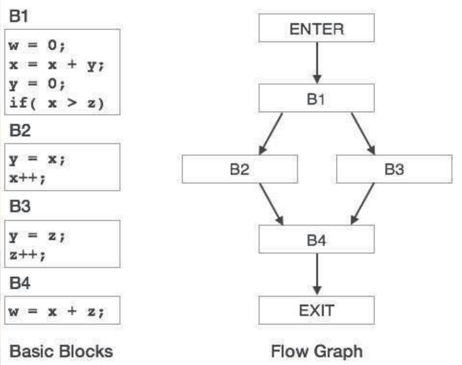


- Last assignment!
- Due on Tues (June 5).
- One late day max - takes you to Wed.

Graphs… the Final Frontier

- They’re everywhere





B1

```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

B2

```
y = x;
x++;
```

B3

```
y = z;
z++;
```

B4

```
w = x + z;
```

Basic Blocks

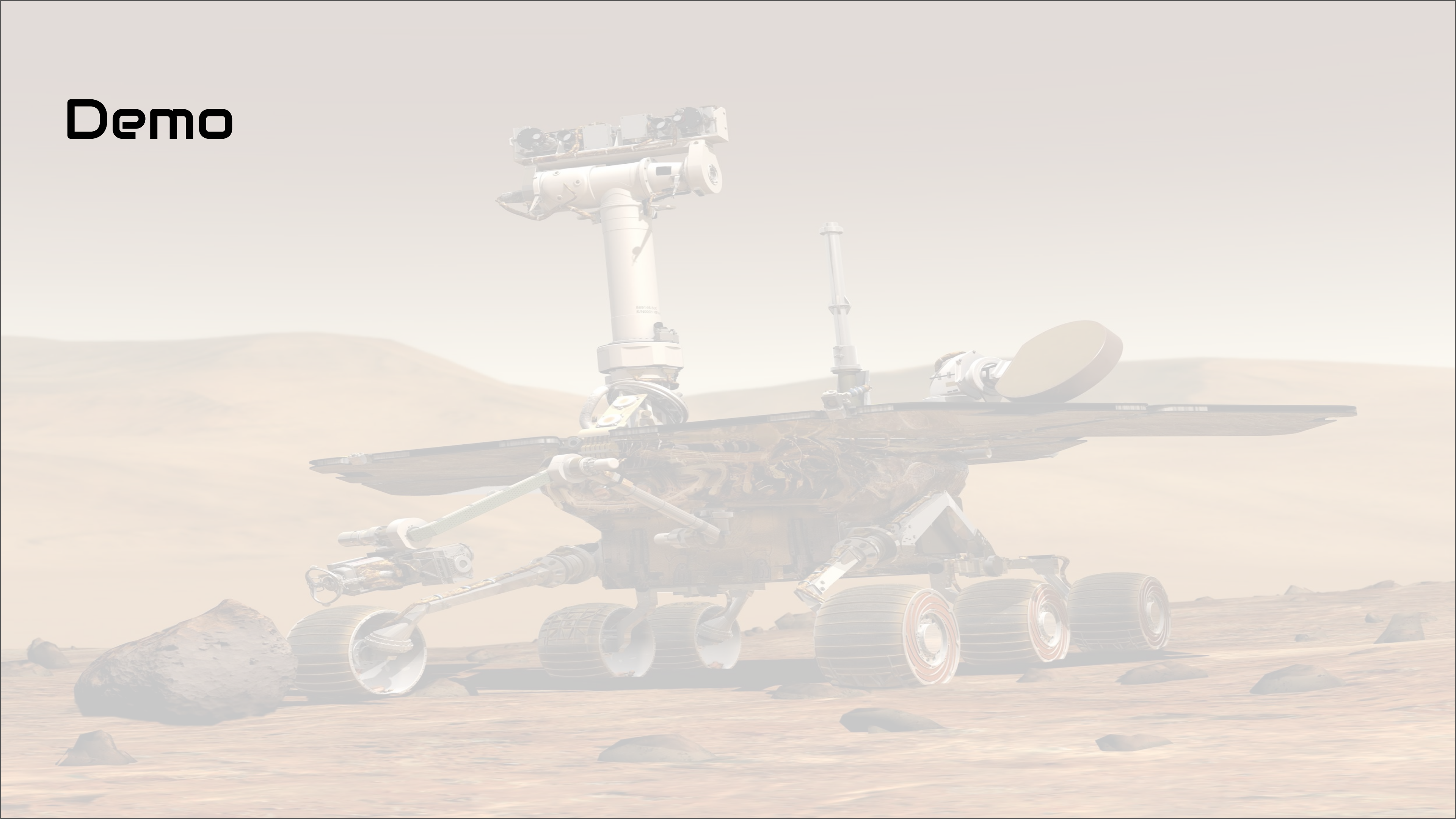
Flow Graph

Image : sanfranciscobusinesstimes.com/photos

Internet Compilers Scheduling

And so much more - come up in every field

Demo



TODO

4 Algorithms

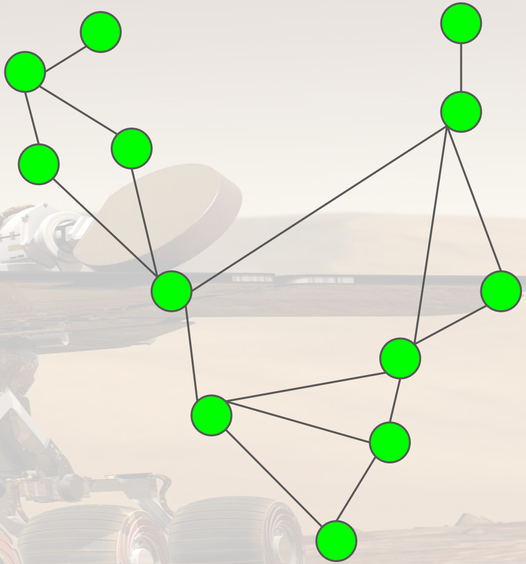
Nice handout here:

http://web.stanford.edu/class/cs106b/handouts/search.html

```
Path breadthFirstSearch(RoadGraph graph, RoadNode* start, RoadNode* end)
Path dijkstrasAlgorithm(RoadGraph graph, RoadNode* start, RoadNode* end)
             Path aStar(RoadGraph graph, RoadNode* start, RoadNode* end)
    Path alternateRoute(RoadGraph graph, RoadNode* start, RoadNode* end)
```

Terminology

- RoadGraph == BasicGraph
- RoadNode == Vertex
- Path == Vector<Vertex*>



RoadGraph

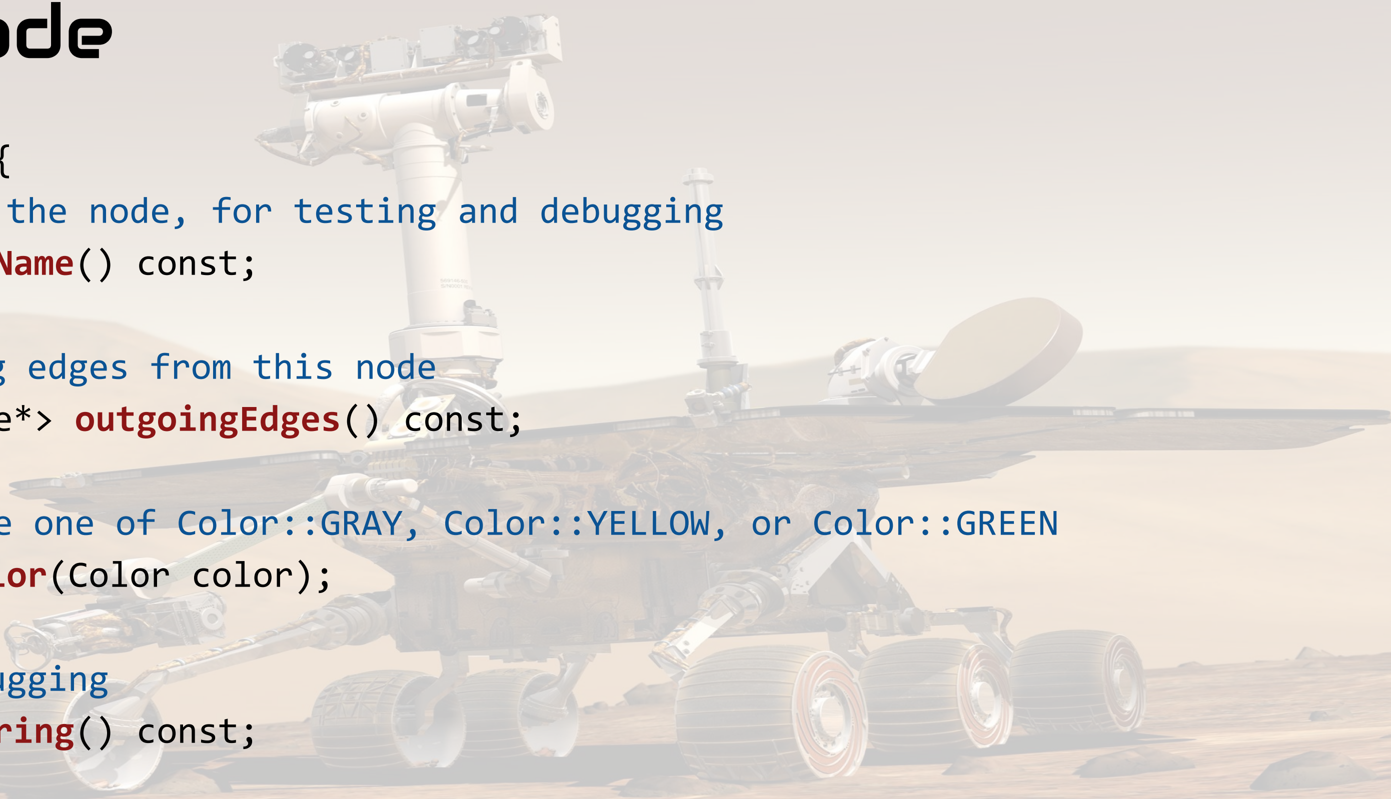
```
class RoadGraph {
    /* Returns the set of all the nodes adjacent to the given node. */
    Set<RoadNode*> neighborsOf(RoadNode* v) const;

    /* Given a start and end node, returns the edge that links them, or
     * nullptr if there is no such edge. */
    RoadEdge* getEdge(RoadNode* start, RoadNode* end) const;

    /* Returns the highest speed permitted on any road in the network. */
    double getMaxRoadSpeed() const;

    /* Returns the "straight-line" distance between the two nodes; that is,
     * the distance between them if you just drew a line connecting them. */
    double getCrowFlyDistance(RoadNode* start, RoadNode* end) const;
};
```

RoadNode



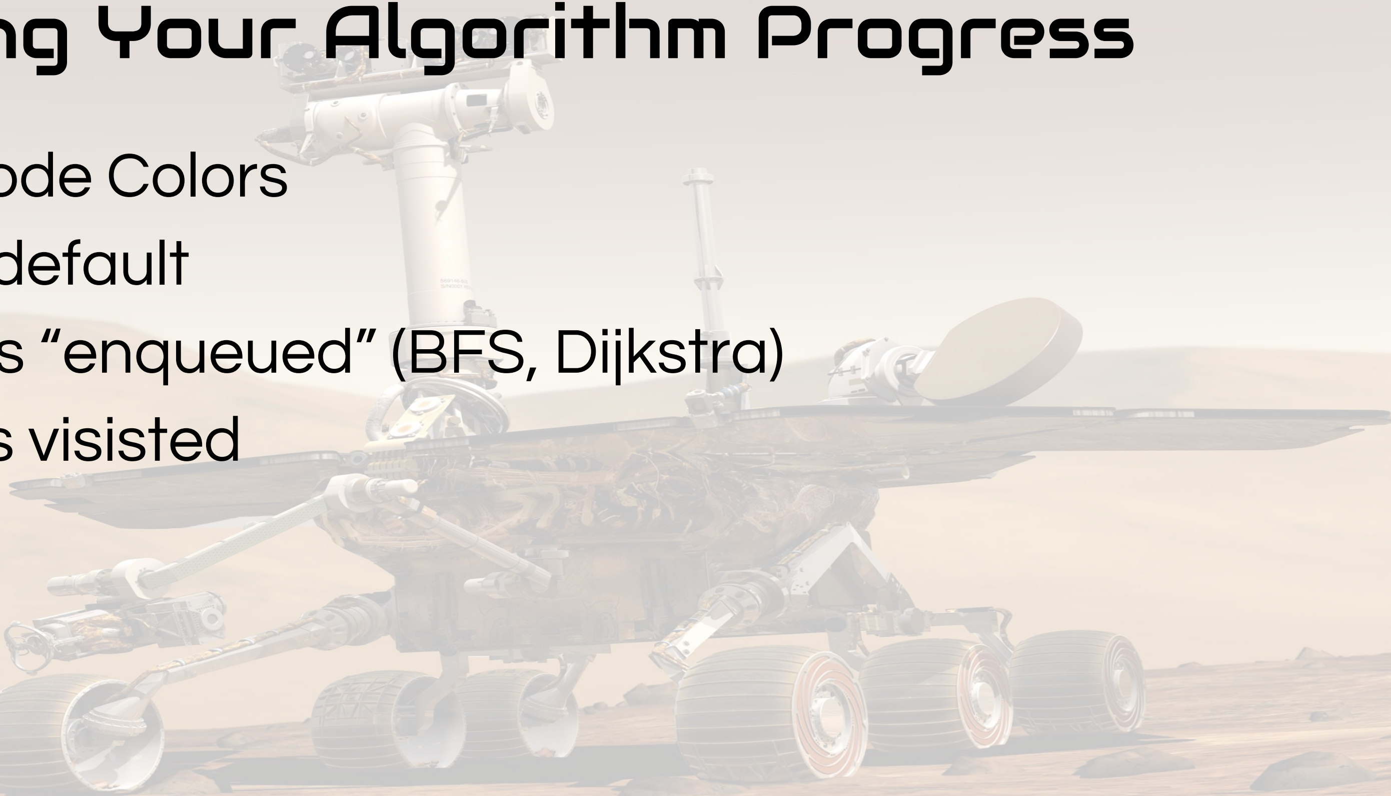
```
class RoadNode {
    // Name of the node, for testing and debugging
    string nodeName() const;

    // Outgoing edges from this node
    Set<RoadEdge*> outgoingEdges() const;

    // Should be one of Color::GRAY, Color::YELLOW, or Color::GREEN
    void setColor(Color color);

    // For debugging
    string toString() const;
};
```

Watching Your Algorithm Progress



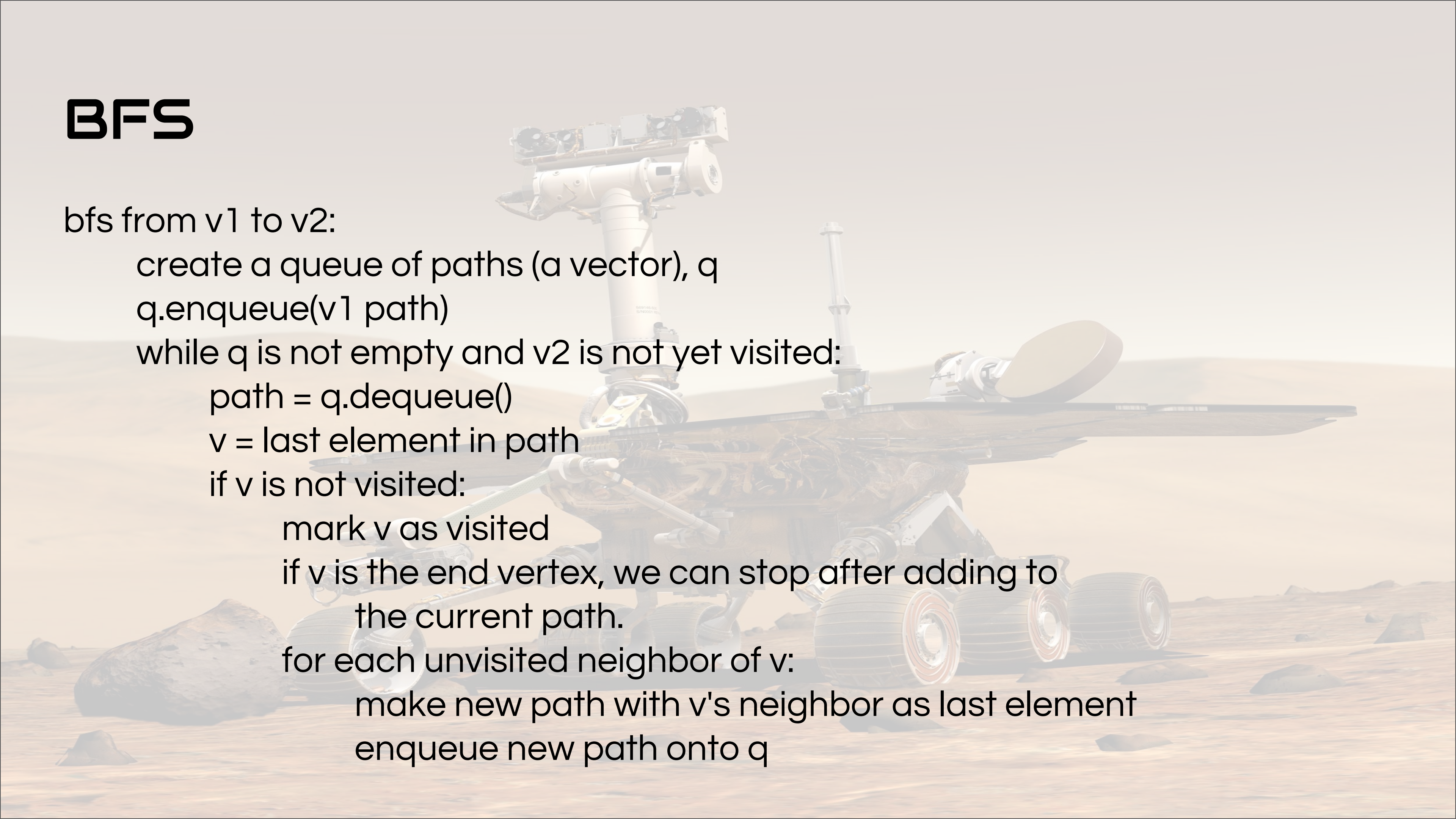
● RoadNode Colors

○ Grey is default

🟡 Yellow is “enqueued” (BFS, Dijkstra)

🟢 Green is visisted

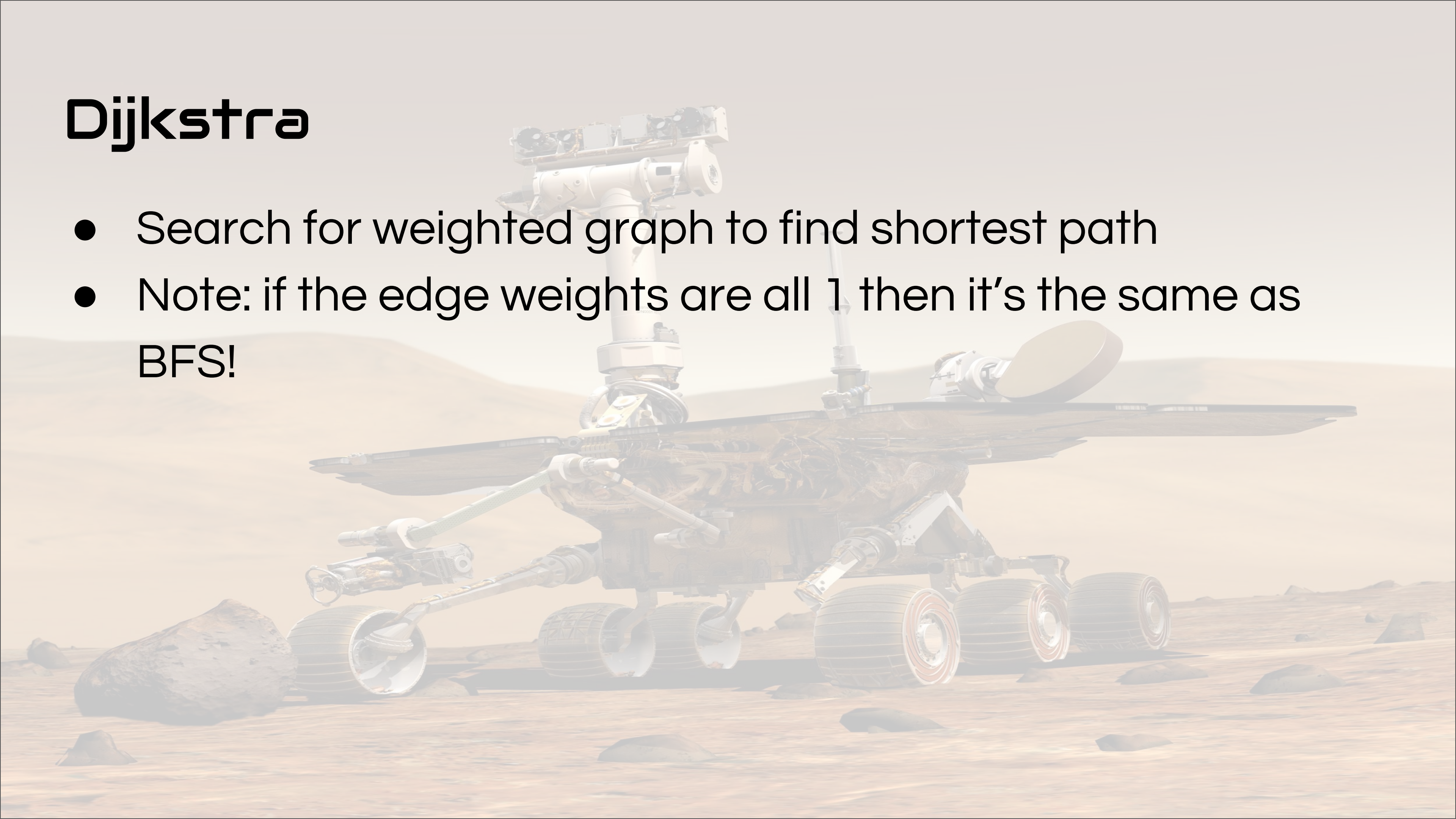
BFS



```
bfs from v1 to v2:
    create a queue of paths (a vector), q
    q.enqueue(v1 path)
    while q is not empty and v2 is not yet visited:
        path = q.dequeue()
        v = last element in path
        if v is not visited:
            mark v as visited
            if v is the end vertex, we can stop after adding to
                the current path.
            for each unvisited neighbor of v:
                make new path with v's neighbor as last element
                enqueue new path onto q
```

Dijkstra

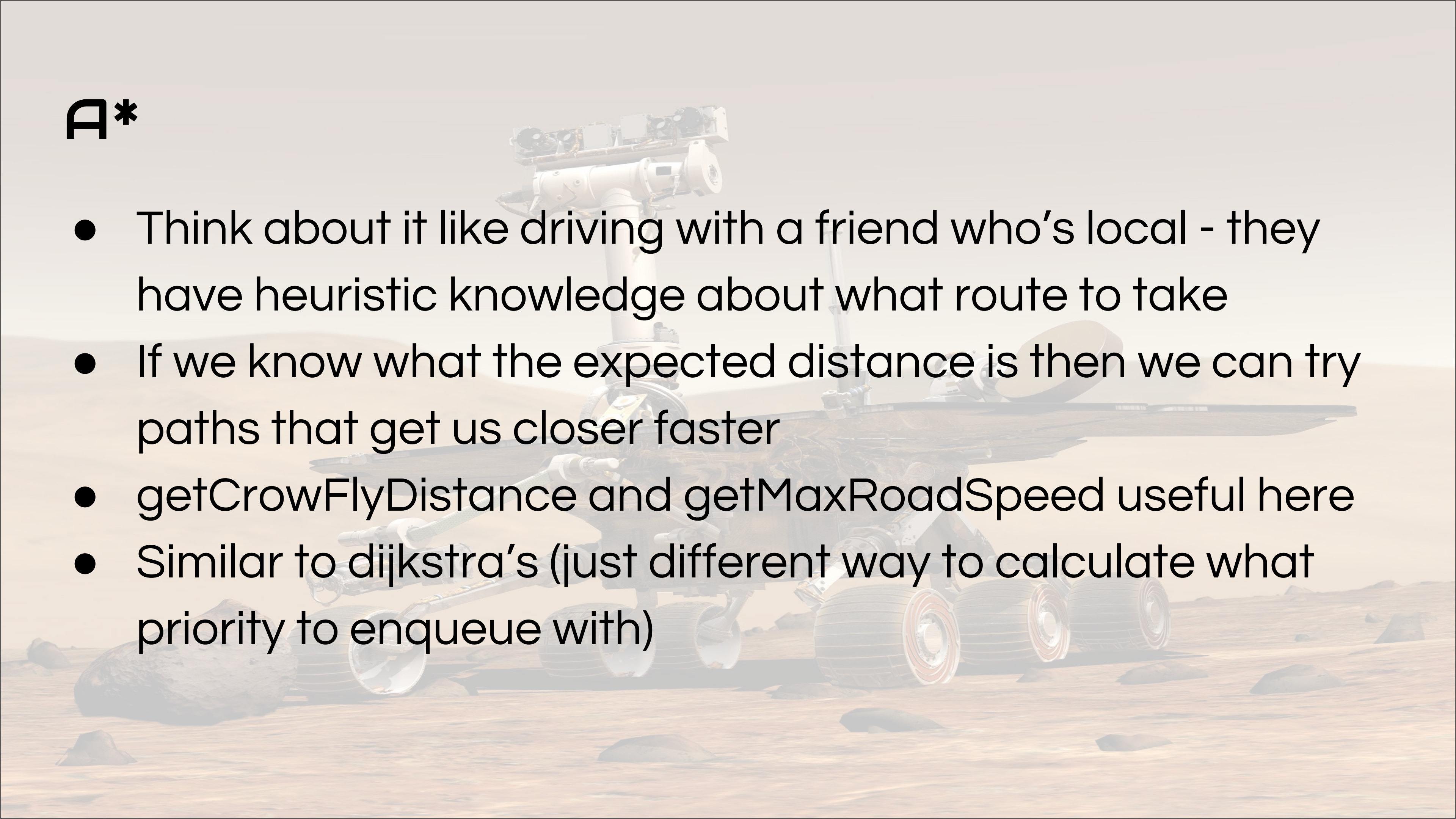
- Search for weighted graph to find shortest path
- Note: if the edge weights are all 1 then it’s the same as BFS!



Dijkstra

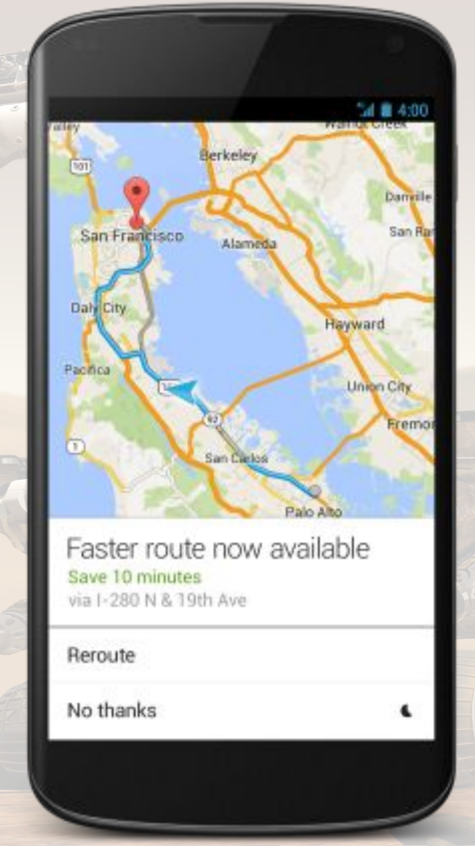
```
dijkstra's from v1 to v2:
create a priority queue of paths (a vector), q
q.enqueue(v1 path)
while q is not empty and v2 is not yet visited:
    path = q.dequeue()
    v = last element in path
    mark v as visited
    if v is the end vertex, we can stop.
    for each unvisited neighbor of v:
        make new path with v's neighbor as last element
        enqueue new path onto q
```

A*



- Think about it like driving with a friend who’s local - they have heuristic knowledge about what route to take
- If we know what the expected distance is then we can try paths that get us closer faster
- getCrowFlyDistance and getMaxRoadSpeed useful here
- Similar to dijkstra’s (just different way to calculate what priority to enqueue with)

Alternative



Faster route now available

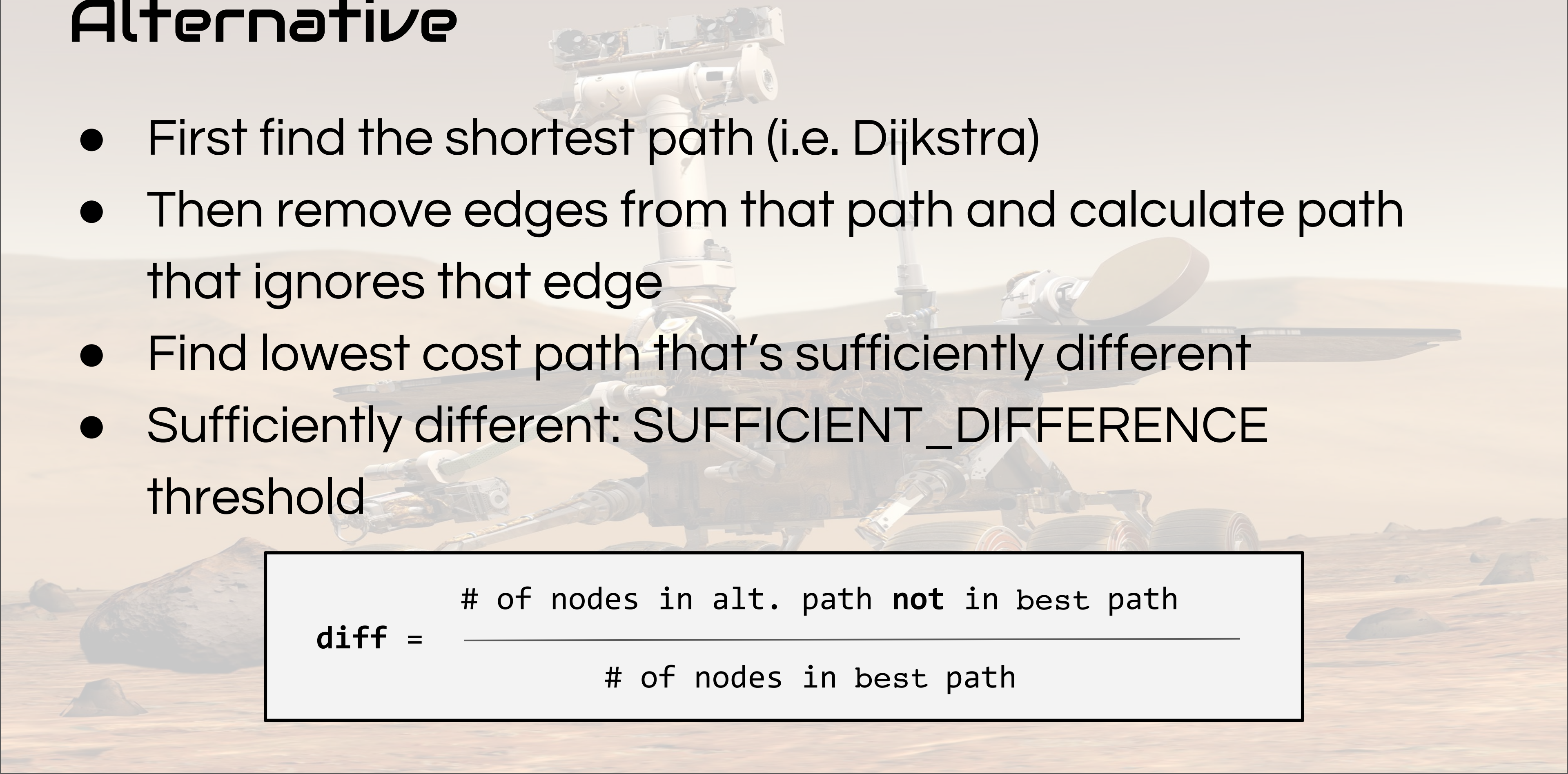
Save 10 minutes

via I-280 N & 19th Ave

Reroute

No thanks

Alternative

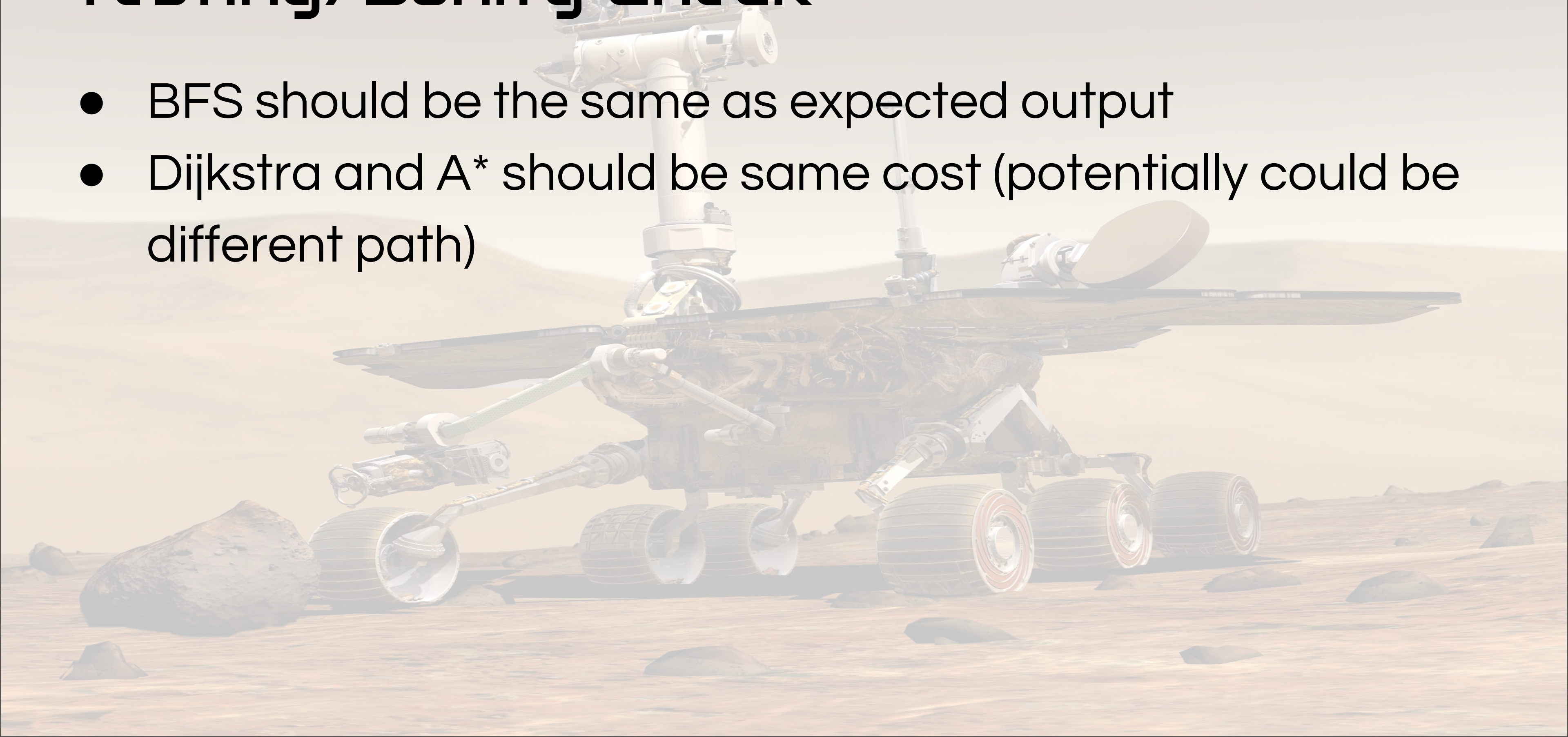


- First find the shortest path (i.e. Dijkstra)
- Then remove edges from that path and calculate path that ignores that edge
- Find lowest cost path that's sufficiently different
- Sufficiently different: SUFFICIENT_DIFFERENCE threshold

$$\textbf{diff} = \frac{\texttt{\# of nodes in alt. path }\textbf{not}\texttt{ in best path}}{\texttt{\# of nodes in best path}}$$

Testing/Sanity Check

- BFS should be the same as expected output
- Dijkstra and A* should be same cost (potentially could be different path)



Creative

- Create your own map!

Suggestions

- In order from easiest to hardest
- Pick small map/routes at first and trace through for debugging

Good Luck!