# MULTI-AGENT PATH FINDING

Simulation of manufacturing processes

Chembrolu Surya
Felipe Vianna
Yuu Sakaguchi

# Problem

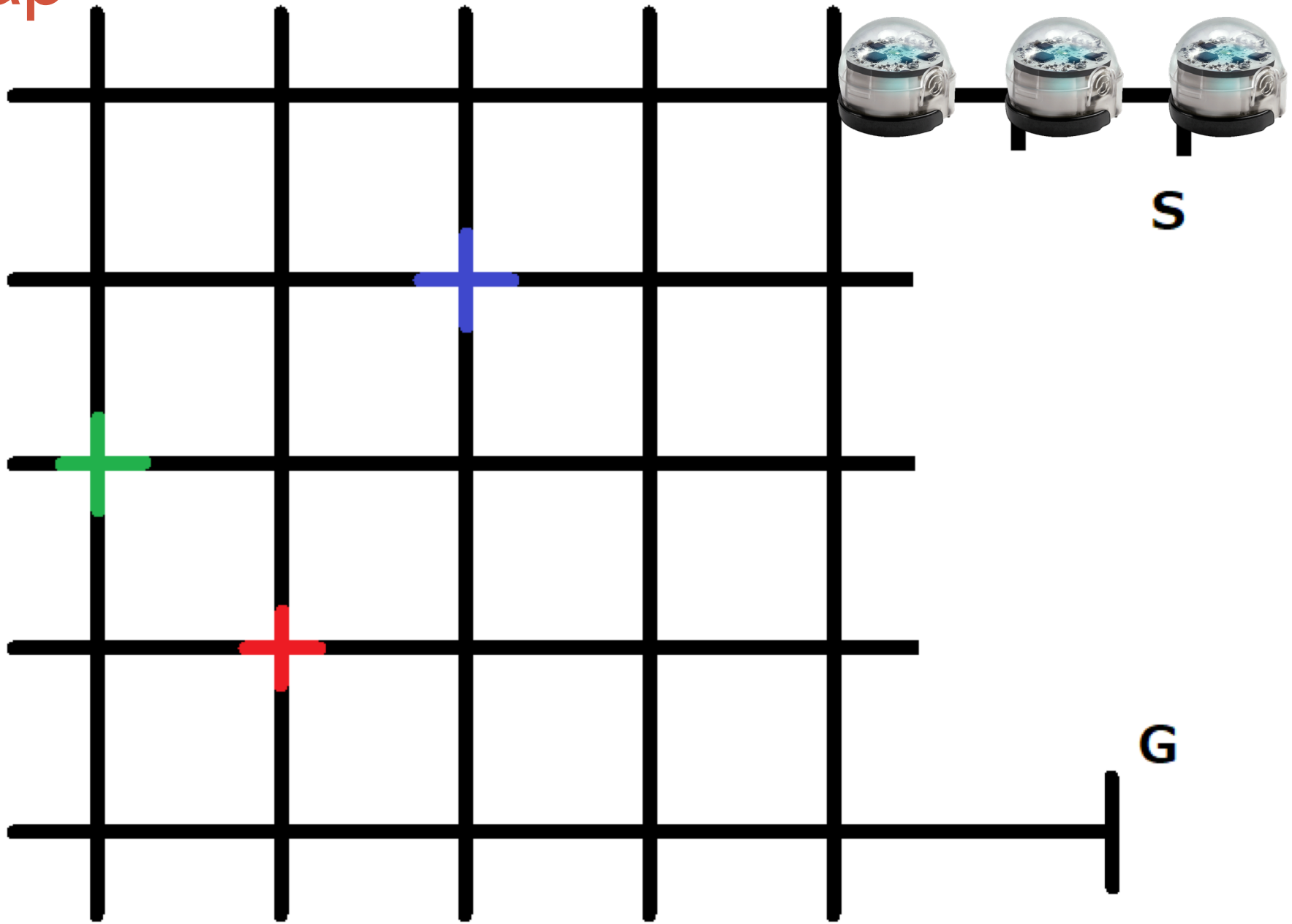
- Find paths for each robot according to sequence of operations they should go to.
- Avoid collision between robots
- Finish all processing in reasonable time

# Given parameters

- Grid map – As adjacent Matrix for paths calculations
- Duration of each operation
- Constraints of operation sequence for each robot

- nodesWeights=[1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 8 1 1 1 7 1 1];

# Map



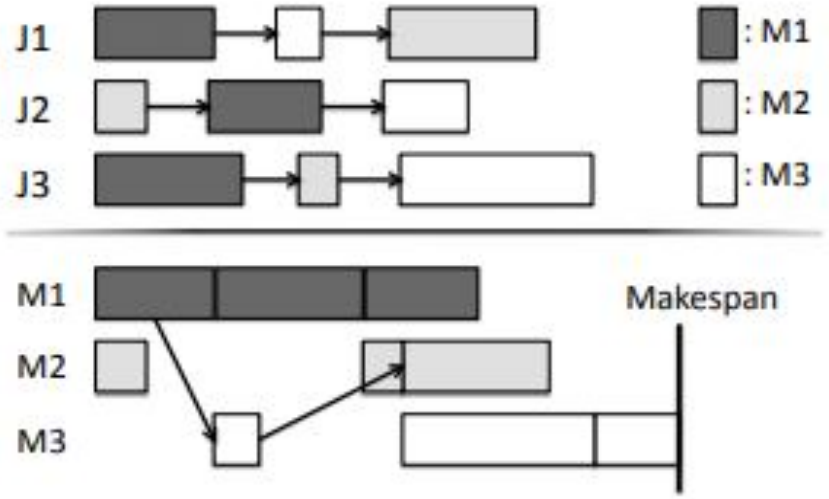
# Scheduling Agent Actions

- Machine environment(Resources) → Multiple operations

Each job j consists of 3 operations and each operation specific with processing time.

- Job description(Tasks) → Non-preemption i.e cannot be interrupted, no precedence relations between tasks.

- Objective is to minimize maximum makespan.



# Scheduling Implementation

- The motivation in implementing problem is job shop scheduling problem.
  In job shop problem different jobs have different order of operations and usually each resource is visited at most once by each job.

- Programmed in python using ortools which comes with constraint solver.

Outline of Solver:

- We consider a resource allocation i.e, assigning agents to resources perform actions which is consider as a matrix.

- Another matrix of processing times for each task is represented.
  Here each job considers 3 operations that are to be performed by one agent.

- DisjunctiveConstraint - checks for tasks of the same agent from overlapping in time.

- ConjunctiveConstraint – checks for consecutive operations of the same resource from overlapping in time.

scheduling to Planning

- Output of job shop scheduler implemented is a sequence of operation node where each row corresponds to one agent actions.
  example: [ 12, 19, 23]

- After obtaining the schedule agents start pathfinding from start node reach every operation node in the order and finally reach the goal node.

# Implementation – Path Finding

- Old way: BFS + DFS to find many possible paths ordered by number of steps, then try combinations of paths until collision free paths.

- Now: A* with heuristics checking for forbidden steps and best improvement step. Paths for robots are searched sequentially. First robot get free grid, second has available nodes not occupied by first, and so on.

- Lines of code: 312(old) vs 114 (new)

# Implementation – Path Finding

- From the sequence of operations, calculate paths. Example:
  - Sequence: [1 12 19 23 5]
  - Find path from 1 to 12, from 12 to 19, from 19 to 23 and from 23 to 5.
  - According to duration of operations (weight of node), repeat nodes in the path to mark them as occupied.
  - Concatenate all sections of paths
- For subsequent robots, repeat the same, but at each step number, check if another robot is already at this position. If so, heuristic function of A* will return infinite
- As robots are not released simultaneously, for each robot subsequent we add 1 step more at initial position. This allows for better prediction of possible collisions. This added steps are removed at final output text files, used later

# Implementation – Path Finding

- 4 functions:
  - A_pathFinding:
    - Define adjacency matrix, weights for nodes
    - Generate paths for bots calling A_findPath, print output file
  - A_findPath:
    - Call A* to find each section path, monitoring total current number of steps used to check for heuristic
    - Concatenate all sections into the full path
  - A_star:
    - Typical A*. Best improvement step is queued.
  - A_heuristics:  for each possible step, check the distance to target. If it's invalid move (node is occupied by other robot, returns inf), otherwise return manhattan distance.

# OzoBlockly generator

- Python script to generate XML, which can be loaded to OzoBlockly.
- Input a set of sequences of nodes, generates moves for whole path.
- XML is coded using <*block*>, <*field*>, <*next*> tags.
- Modules for each actions; move forward, backward, left, right, wait, etc.
- Possible to store actions in a “list’ in OzoBlockly, so that bots can go back to previous node or more.

# Results

- Find non-conflict paths for as many robots and operations desired.

- No optimal solution assured: Depending on which robot got priority in path, the total length of paths are different. Not optimal, but good enough =]

# Next Steps Promised

- Implement a scheduler for the operations sequencing. For now it was just decided manually.
- Increase number of operations and robots.
- Implement reactive decisions using sensors. Initial idea was have path finding algorithm embedded in ozobot, but current interface doesn't allow that. Sensors fail in most of the cases. To secure better, collisions in path finding are avoided in safer approach.
- Switch from BFS/DFS try-error approach to A*

# Results

- Video