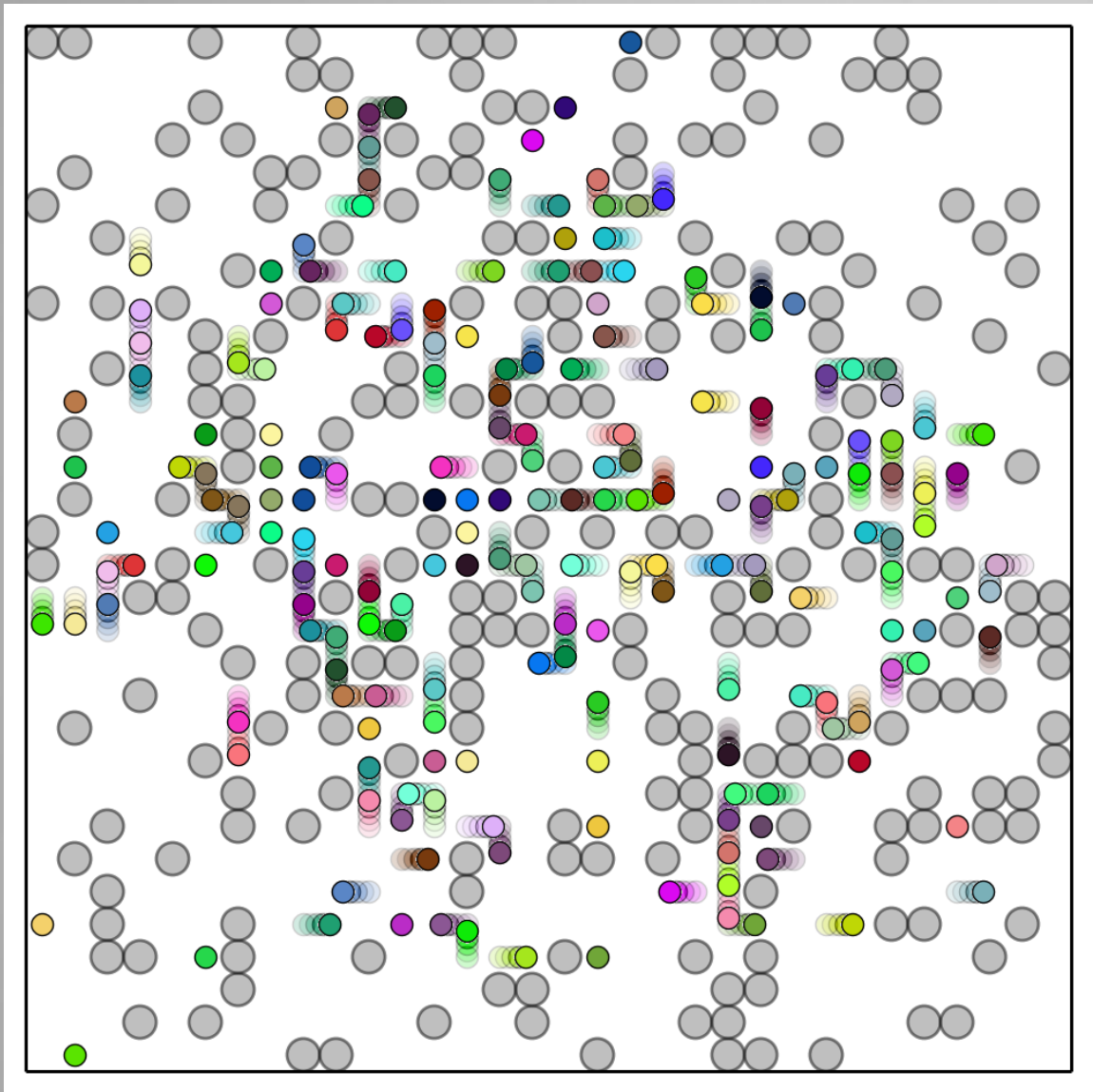
Collaborative A* Pathfinding

YAN GE



(IJCAI 2016 Workshop on Multi-Agent Path Finding)

Problem Analysis

- What is my goal?
- What is the problem with A*?

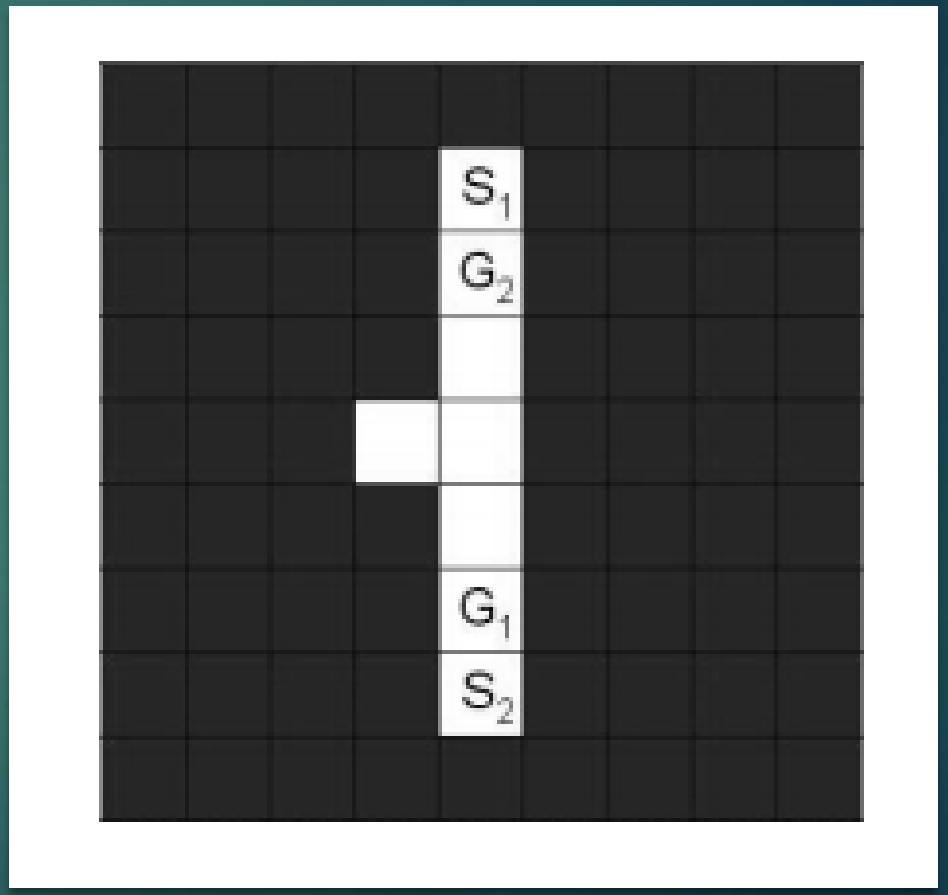
Application Problem Analysis

Say you have two agents,
Agent A: start at S1, want to go to G1
Agent B: start at S2, want to go to G2

Ok, lets go!
S1 reached G1, S2 is dead…
OK, S2 go first, reached G2, S1is dead

It seems hard.
Wait, in the middle, there is a spot on the left corner.

S1: you go there wait!
S2: no, you go there wait!
Agents fight! They both failed.
(Let me hard code it, so I can give you demo.
And researchers actually did it…)



(David Silver, cooperative pathfinding)

CDS Based Issues

Capability Interdependency

- SO's bonded capability
  - need other SO's help
- Decomposition of goals into sub-goals
  - Still need others' help
  - Use windowed approach(David Silver)

Interest Interdependency

- Conflicting interest
  - Avoid collision
- Common interest
  - Common goal is to use minimum recourse and time to complete goal collaboratively.

Knowledge Interdependency

- Shared resource
- Map, some of other SOs' path, window size.
- Locally shared and globally shared resource.
  - Map , global.
  - Agent's path, local.

Coordination

Structure

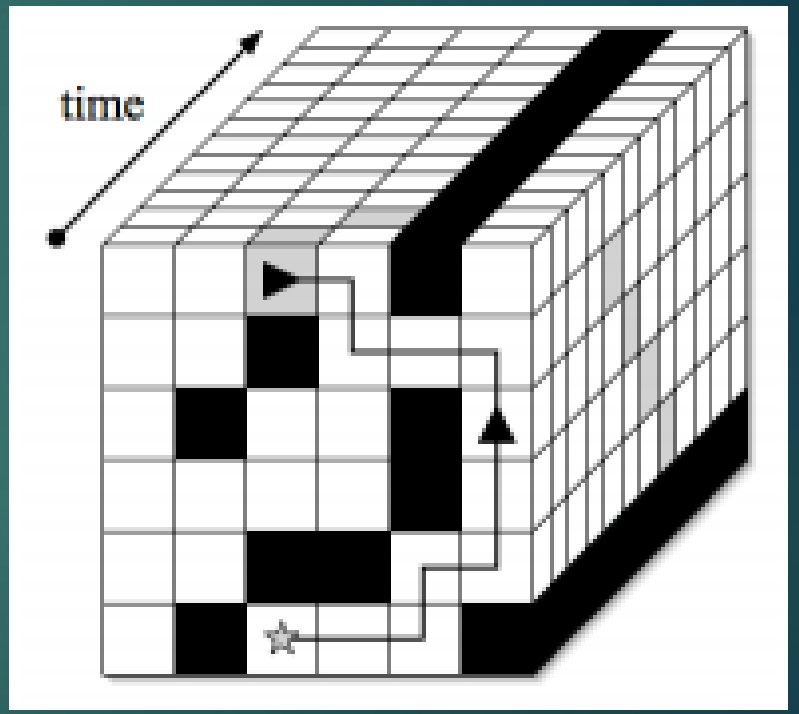
- What are the decision points?
  - Collision for example.
- How to decide if there is a decision point?
  - Pattern of collision. (will see)

Mechanism

- How to address issues of interdependencies?
  - Capability (how bounded?)
  - Interest (collision and common)
  - Knowledge (shared resource, local or global)

Toolkit 1
Time dimension

- What do see on the map?
- Black: obstacle.
- White: walkable region.
- Grey: agent path
- And most importantly, TIME
- Time. Time. Time.



(David Silver, cooperative pathfinding)

Toolkit 2
True distance heuristic

On the left, it gives the distance to goal.

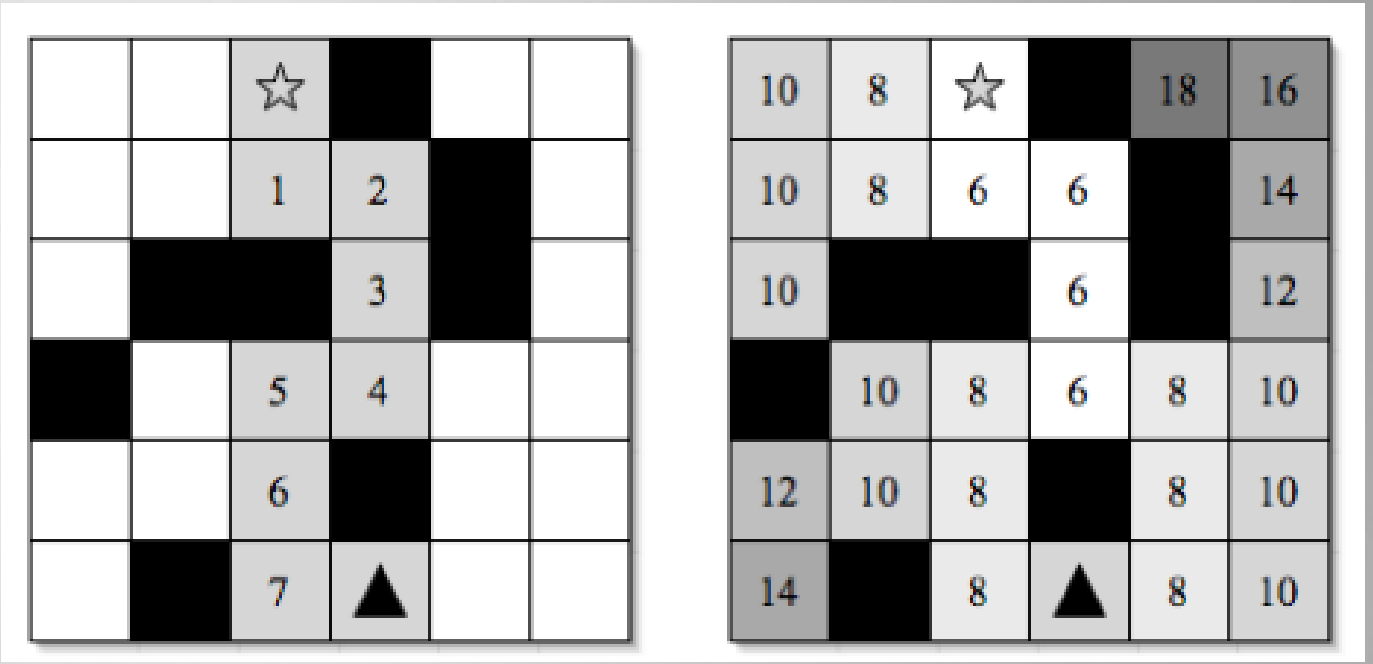
On the right, the number means the distance from start to that cell and from that cell to the goal.

(there is an error on the right map.)

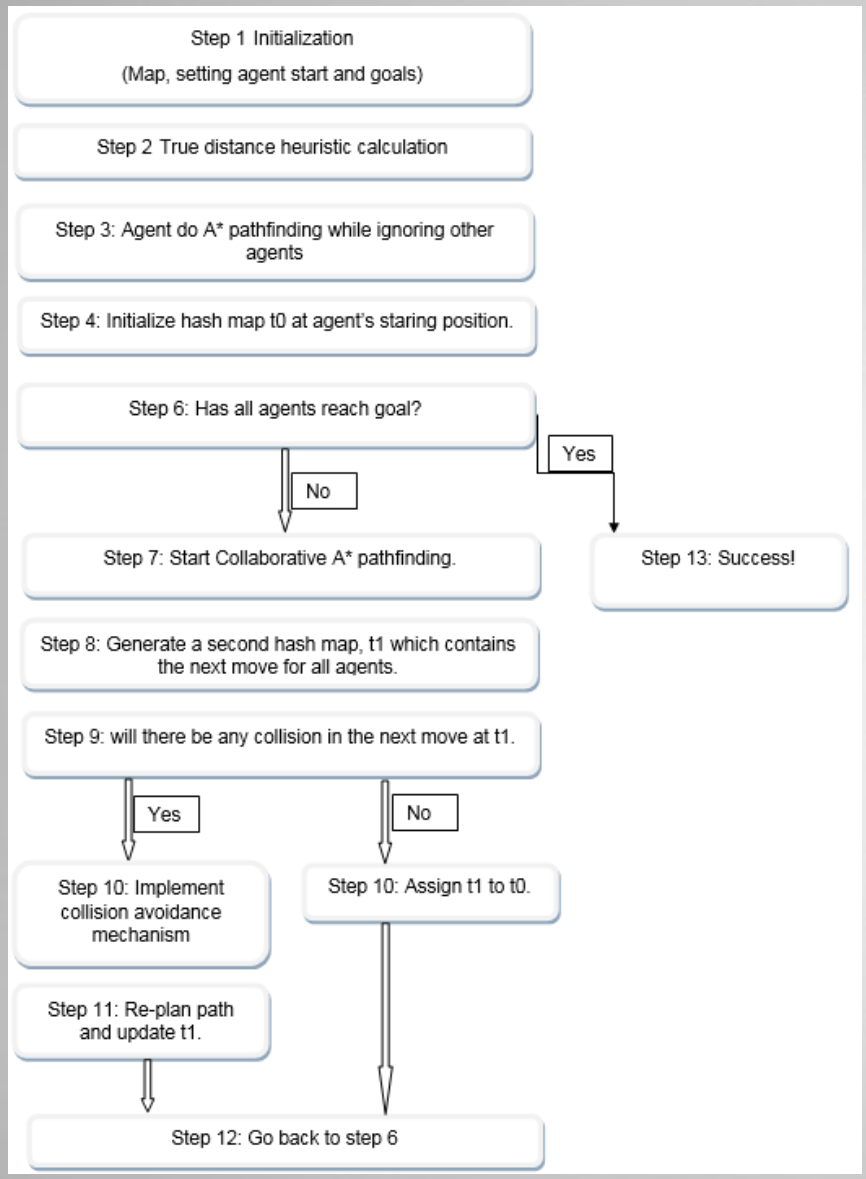
(Assume agent cannot walk diagonal, only up, down, left and right)

How to calculate?

Basic A* search, but backwards!



“Naïve” Architecture



Smart-Object Architecture

- The architecture must be designed as simple as possible, Imagine you have a thousand agents walking.
- The goal of this project to find routes without collision.
- However, agent's goal could change, agent's speed can vary, agent has its own priority, bad agent can block other agents forever.
- But, for now, let's prey, "life is simple, life is good".

An Agent has

- A name
- A starting location
- A goal location
- Individual knowledge
- A path

```
/*************************************************************************/
// Agent class.
class Agent
{
public:
	Agent(char c) : name(c) { path.clear(); };

	bool operator==(const Agent &a) const {
		return (name == a.getName());
	}

	unordered_map<Node, Node> cameFrom; // <child, parent>
	unordered_map<Node, unsigned int> gScore;
	unordered_map<Node, unsigned int> fScore;
	unordered_map<Node, unsigned int> closedSet;  // calculated fscore nodes
	prioriyQueue<Node, std::vector<Node>, comp_node_fScore> openSet; // lowest fscore
on top()

	char getName() const { return name; };

	void setStart(const Node& s) { start = s; path.push_back(s); current_node = s; };
	Node getStart() const { return start; };

	void setGoal(const Node& g) { goal = g; };
	Node getGoal() const { return goal; };

	void setPath();
	void getPath(list<Node> &p);

	uint getTime() const { return time; };
	void setTime(uint t) { time = t; };
```

```
	uint getTime() const { return time; };
	void setTime(uint t) { time = t; };

	Node set_current_node(const Node& n) { current_node = n; };
	Node get_current_node() const { return current_node; };

	void print_path();
	uint get_path_length() const { return path.size(); };
	void insert_path_to_front(const list<Node>& p);

	bool getNextNode(const Node& n, Node& next);
	bool getPrevNode(const Node& n, Node& prev);

	void pop_front_node_from_path() { if (!path.empty()) path.pop_front(); };
	Node get_front_node_from_path() { return path.empty() ? current_node :
path.front(); };

private:
	char name;
	uint time;
	Node start;
	Node goal;
	Node current_node;
	Node next_node;
	list<Node> path;
};
```

Problem-Solver

- Mechanism: pathfinding related methods
  - 1. A-star

```
/*****************************************************************/
// A-Star pathfinding for single agent.
bool aStar(Agent &agent,  Map &map)
```

  - 2. Reverse Resumable A*, True Distance Heuristic.

```
552 /*****************************************************************
553    reverse resumable A* (Backwards Search ignoring other agents)
554    the g value (measured distance to goal) is the true distance heuristic value.
555    if g value of requested node is not known, set goal at requestedNode.
556 *****************************************************************/
557 bool get_true_distance_heuristic(Agent& agent, Map& map, const Node& requestNode)
```

Problem-Solver

- Mechanism: resolving conflict
  - 1. collision detection (involving communication and interaction)

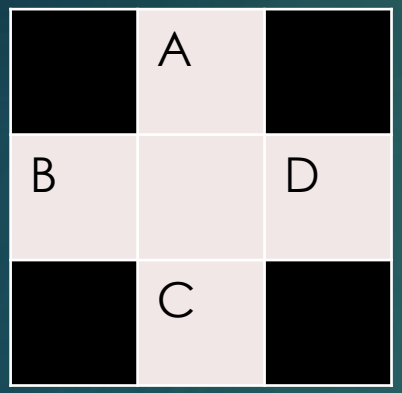
```
748 /*************************************************************************/
749 // check face to face collisions, each colliion involves two agents, where agent1 walking from A to B, agent2 walking from B to A
750 bool check_collision_type2( hash_map& t0, hash_map& t1, vector<vector<Node>>& collision_nodes_pairs)
751 {
```

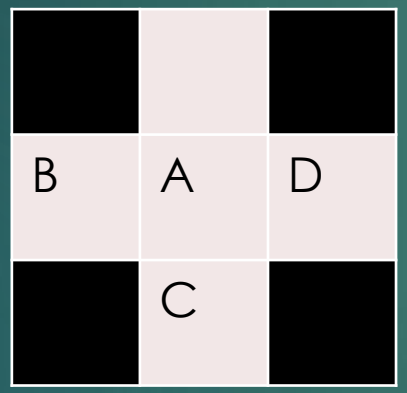
  - 2. collision fix

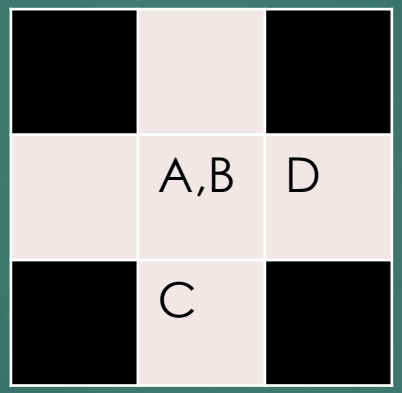
```
916 /*************************************************************************/
917 void fix_agents ( hash_map& hmap_t0,  hash_map& hmap_t1, vector<vector<Node>>& collision_nodes_pairs, Map& m )
```

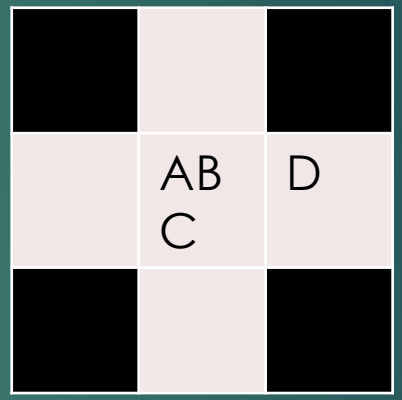
Collision Type 1

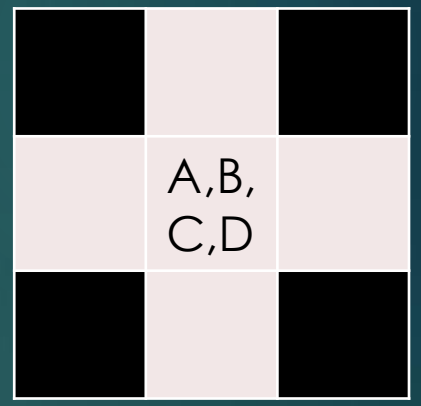
"one spot - many agents".





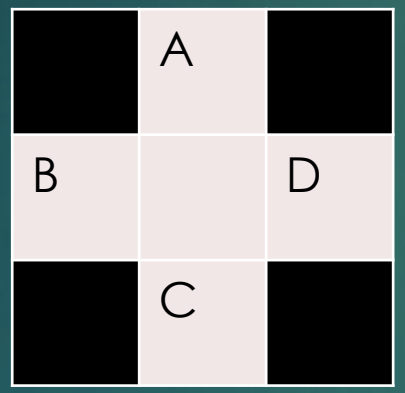


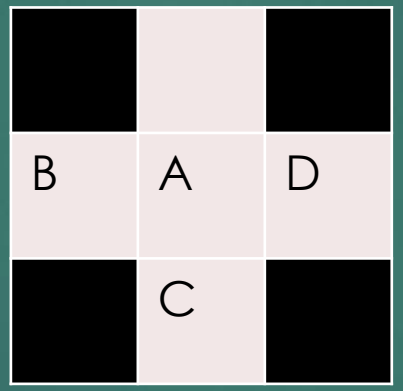


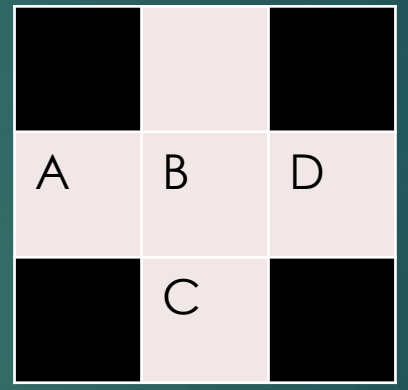


Collision Type 2

"face-to-face"







Why agents are not happy?

Interaction & communication

- All inside a big while() loop
- If time allowed, I would love to go though the code.

```
1022  // big while loop
1023  while(!all_agents_find_goal)
1024    {
1025
1026      if(!had_collision)
1027  {
1028    // fill hmap_t0 , hmap_t1
1029    for (auto& agent : agent_list)
1030      {
1031        Node current_n = agent->get_front_node_from_path();
1032        if (current_n != agent->get_current_node())
1033    agent->set_current_node(current_n);
1034
1035        hmap_t0[current_n].push_back(agent);
1036
1037        Node next_n;
1038        if (agent->getNextNode(current_n, next_n))
1039    hmap_t1[next_n].push_back(agent);
1040        else
1041    hmap_t1[current_n].push_back(agent); // agent stays still.
1042      }
1043  }
1044
1045      if(had_collision)
1046  {
1047    // only update hmap_t1, no change for the current node.
1048    for (auto& agent : agent_list)
1049      hmap_t1[ agent->get_front_node_from_path() ].push_back(agent);
1050  }
```

Interaction & communication

- Hash map is used for efficiency
- Each iteration, two hash maps are prepared
- Then we check if all agents reach goal.

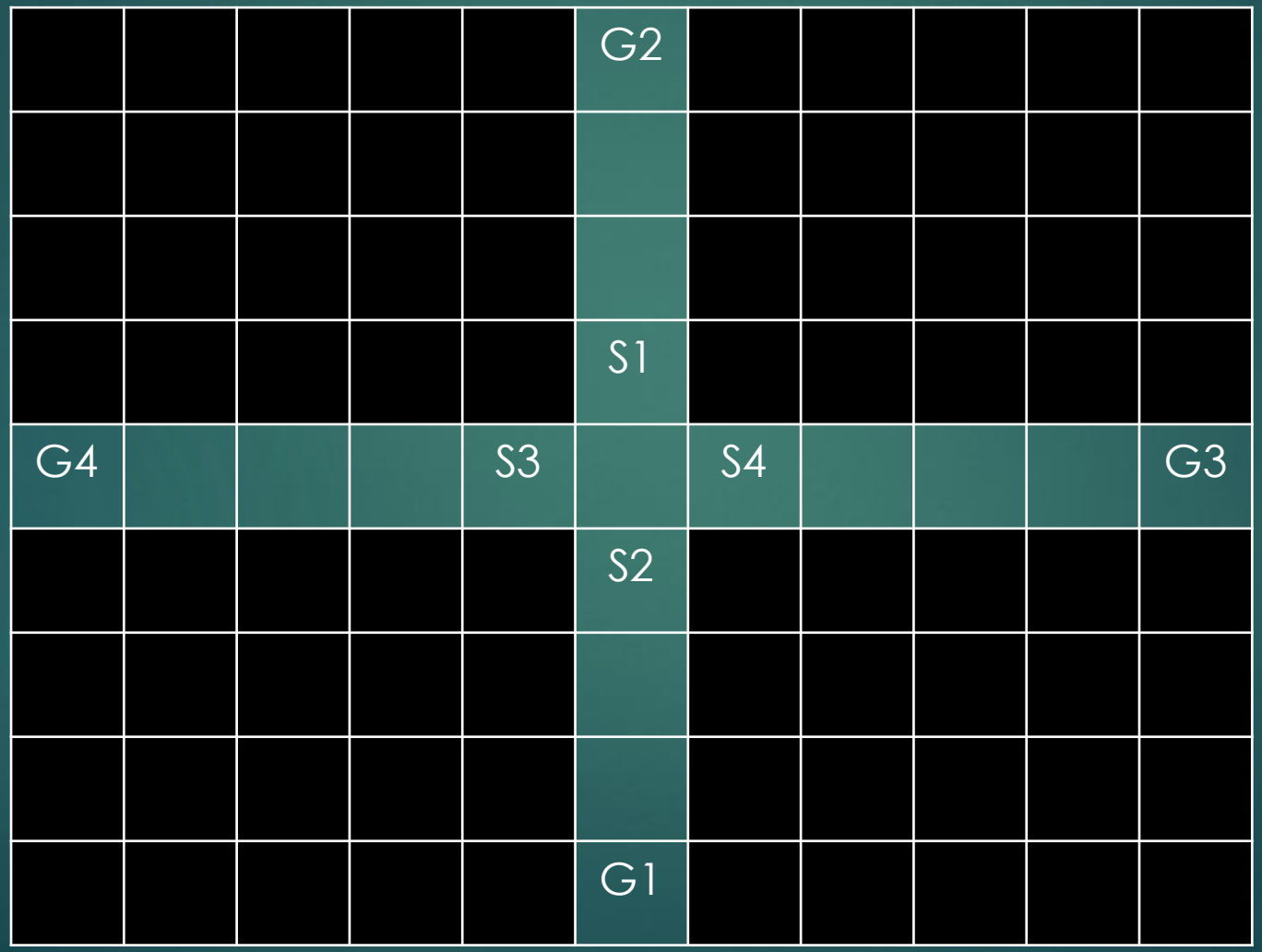
```
1045        if(had_collision)
1046    {
1047      // only update hmap_t1, no change for the current node.
1048      for (auto& agent : agent_list)
1049        hmap_t1[ agent->get_front_node_from_path() ].push_back(agent);
1050    }
1051
1052        cout << "hmap_t0\n";
1053        print_hash_map(hmap_t0);
1054        cout << "hmap_t1\n";
1055        print_hash_map(hmap_t1);
1056
1057        ////////////////////////////////////////////////////////////////////////////////
1058        // STEP 1: GOAL CHECK
1059        cout << "--------------- Do all agents reach destination? -------------\n";
1060        for (auto i = 0; i < n_agents; ++i)
1061    if (agent_list[i]->get_current_node() == agent_list[i]->getGoal())
1062      num_agents_at_goal++;
1063
1064        if (num_agents_at_goal == n_agents) {
1065    all_agents_find_goal = true;
1066    cout << "Success! All agents at goal!\n";
1067    return 0;
1068        }
1069        else
1070    cout << "--------------- Nope -------------\n";
1071        ////////////////////////////////////////////////////////////////////////////////
```

Interaction & communication

- Agents communicate and interact through hash map.
- The key of hash map is node on the map.
- The value stored is the agents.
- If collision happened, fix agent's route, and rehash.
- Only agents who are about to have collision need to communicate, good agents don't.
- Therefore, we reduce the cost of communication.

```
1086        // if collision, hmap_t1 is wrong do not do any thing to
1087        if ( check_collision_type2( hmap_t0, hmap_t1, collision_nodes_pairs ))
1088    {
1089      had_collision = true;
1090      cout << "-------------- fixing collision --------------\n";
1091      fix_agents( hmap_t0, hmap_t1, collision_nodes_pairs, test_map );
1092      cout << "-------------- collision fixed --------------\n";
1093    }
1094        else
1095    {
1096      cout << "-------------- Good, no collision, keep moving! --------------\n";
1097      had_collision = false;
1098      super_hash_map.push_back(std::move(hmap_t0));
1099
1100    }
1101
1102        hmap_t0.clear();
1103        hmap_t1.clear();
1104
1105        for (auto& agent : agent_list)
1106    {
1107      agent->pop_front_node_from_path();
1108      agent->print_path();
1109    }
```

If time allowed, I suggest we play a game.



I hope I did not waste your time.
Thank you for being patient.
Next time, demo.
I hope it works!
If it doesn't work, I can't date woman…