public String getTagName(){

return this.tagName;

}

public int getTagNumber(){

return this.tagNumber;

}

\*//\*\* Another edge will tell the agent about the notification

\* **@param** edge the obeserved edge

\*//\*

public void observedEdgeNotification(SystemEdge edge) {

//System.out.println(this.tagName+edge.getParent());

this.currentState = edge.getChild();

if (currentState.getProcessCompleted()!=null){

this.processDone.add(currentState.getProcessCompleted());

if (processDone.containsAll(processNeed)){

processNeed.add((this.tagNumber-1)+"F");

}

}

requestCommand();

}

\*//\*\*

\* If the request command hasn't been run in a while, run it to try to go somewhere

\*//\*

@ScheduledMethod ( start = 27 , interval = 27, priority = 1000)

public void runRequestCommand(){

if(ranRequestCommand == false && currentState !=null){

requestCommand();

}

this.ranRequestCommand = false;

}

\*//\*\*

\* Request a method from another agent

\*//\*

private void requestCommand() {

updateSystemGraph();

DijkstraShortestPath<SystemVertex, SystemEdge> shortestPathGetter = new DijkstraShortestPath<SystemVertex, SystemEdge>(this.systemGraph,weightTransformer);

shortestPathGetter.reset();

this.ranRequestCommand = true;

boolean anyContrallable = false;

ArrayList<SystemEdge> outEdges = new ArrayList<SystemEdge>();

for (SystemEdge outEdge : systemGraph.getOutEdges(currentState)){

if (outEdge.getControllability()){

outEdges.add(outEdge);

anyContrallable = true;

}

}

ShortestPath shortestPath = new ShortestPath(null,10000);

if (anyContrallable){

shortestPathGetter.enableCaching(false);

List<SystemEdge> shortestPathCandidateGraph = new ArrayList<SystemEdge>();

for (SystemVertex vertex:this.systemGraph.getVertices()){

String vertexProcess = vertex.getProcessCompleted();

if(this.processNeed.contains(vertexProcess) && !this.processDone.contains(vertexProcess)){

shortestPathCandidateGraph = shortestPathGetter.getPath(this.currentState, vertex);

Controllers\_GraphHelper.createTextFile(systemGraph, "DEBUG.txt", this.currentState.toString());

for (Object edge:systemGraph.getOutEdges(this.currentState)){

System.out.println(edge);

System.out.println();}

System.out.println(shortestPathGetter.getIncomingEdgeMap(currentState));

//

for (SystemVertex key : shortestPathGetter.getIncomingEdgeMap(currentState).keySet()) {

System.out.println(key.getClass().getSimpleName());

if (!key.getClass().getSimpleName().contains("nc")){

System.out.println("CHECK THIS" + key);

System.out.println(this.systemGraph.getOutEdges(key));

System.out.println();

for (SystemVertex key2 : shortestPathGetter.getIncomingEdgeMap(key).keySet()){

//System.out.println("key2: " + key2);

//System.out.println(this.systemGraph.getOutEdges(key2));

}

}

}

shortestPath = compare(shortestPath,shortestPathCandidateGraph);

}

}

if (shortestPath.getShortestPath() == null || shortestPath.getShortestPath().isEmpty()){

return;

}

SystemEdge requestEdge = shortestPath.getShortestPath().get(0);

if (requestEdge.getControllability()){

try {

if (requestEdge.getActiveObject().getClass().getSimpleName().contains("Robot")){

((RobotSystemController) requestEdge.getActiveObject()).setStopperNoMove();

}

requestEdge.getActiveMethod().invoke(requestEdge.getActiveObject(), requestEdge.getActiveParams());

} catch (IllegalAccessException | IllegalArgumentException

| InvocationTargetException e) {

System.err.println("Check request command in part controller " + requestEdge);

}

}

for (SystemEdge edge:outEdges){

if (!edge.equals(requestEdge)){

try {

if(requestEdge.getDeactiveMethod()!=null){

requestEdge.getDeactiveMethod().invoke(requestEdge.getDeactiveObject(), requestEdge.getDeactiveParams());

}

} catch (IllegalAccessException | IllegalArgumentException

| InvocationTargetException e) {

}

}

}

shortestPathGetter.reset();

}

}

//================================================================================

// Helper functions

//================================================================================

private void updateSystemGraph() {

systemGraph = new DirectedSparseGraph<SystemVertex, SystemEdge>();

for (MaterialHandlingAgent agent : this.materialHandlingList){

systemGraph = Controllers\_GraphHelper.addGraph(systemGraph, agent.getGraph(0));

}

for (ManufacturingProcessAgent agent : this.manufacturingList){

systemGraph = Controllers\_GraphHelper.addGraph(systemGraph, agent.getGraph(0));

}

for (CollectionAgent agent : this.collectionAgentList){

systemGraph = Controllers\_GraphHelper.addGraph(systemGraph, agent.getGraph(0));

}

ArrayList<SystemVertex> point3080 = new ArrayList<SystemVertex>();

for (SystemVertex vertex:systemGraph.getVertices()){

if(vertex.getLocation().x == 30 && vertex.getLocation().y == 80){

point3080.add(vertex);

}

}

}

private ShortestPath compare(ShortestPath shortestPath,

List<SystemEdge> shortestPathCandidateGraph) {

if (shortestPathCandidateGraph.isEmpty()){

return shortestPath;

}

ShortestPath outputPath = shortestPath;

int total\_weight = 0;

int old\_weight = shortestPath.getTotalWeight();

boolean change = true;

for (SystemEdge edge: shortestPathCandidateGraph){

total\_weight = total\_weight + edge.getWeight();

if (total\_weight > old\_weight){

change = false;

break;

}

}

if (change){

outputPath = new ShortestPath(shortestPathCandidateGraph,total\_weight);

}

return outputPath;

}

private class ShortestPath{

private List<SystemEdge> shortestPath;

private int totalWeight;

public ShortestPath(List<SystemEdge> shortestPath ,int totalWeight){

this.shortestPath = shortestPath;

this.totalWeight = totalWeight;

}

public List<SystemEdge> getShortestPath() {

return shortestPath;

}

public int getTotalWeight() {

return totalWeight;

}

}

public void updateResourceList(ArrayList<MaterialHandlingAgent> materialHandlingRAs,

ArrayList<ManufacturingProcessAgent> manufacturingRAs,

ArrayList<CollectionAgent> collectionAgentRAs){

this.systemGraph = new DirectedSparseGraph<SystemVertex, SystemEdge>();

for (MaterialHandlingAgent agent : materialHandlingRAs){

this.materialHandlingList = materialHandlingRAs;

systemGraph = Controllers\_GraphHelper.addGraph(systemGraph, agent.getGraph(0));

agent.addListeningPart(this);

}

for (ManufacturingProcessAgent agent : manufacturingRAs){

this.manufacturingList = manufacturingRAs;

systemGraph = Controllers\_GraphHelper.addGraph(systemGraph, agent.getGraph(0));

agent.addListeningPart(this);

}

for (CollectionAgent agent : collectionAgentRAs){

this.collectionAgentList = collectionAgentRAs;

systemGraph = Controllers\_GraphHelper.addGraph(systemGraph, agent.getGraph(0));

agent.addListeningPart(this);

}

}

//Randomly toggles stuff in the graph;

@ScheduledMethod ( start = 1 , interval = 5)

public void checkSwitch(){

Collection<SystemEdge> allEdges = controllabilityGraph.getEdges();

ArrayList<SystemEdge> arrayEdges = new ArrayList<SystemEdge>(allEdges);

int index = (int) Math.floor(Math.random() \* arrayEdges.size());

SystemEdge useEdge = arrayEdges.get(index);

try{

SystemVertex parentVertex = useEdge.getParent();

for (SystemEdge edge : controllabilityGraph.getOutEdges(parentVertex)){

if (edge.getDeactiveMethod()!=null){

edge.getDeactiveMethod().invoke(edge.getDeactiveObject(), edge.getDeactiveParams());}

}

useEdge.getActiveMethod().invoke(useEdge.getActiveObject(), useEdge.getActiveParams());

}

catch (IllegalAccessException | IllegalArgumentException

| InvocationTargetException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}