**Details needed from the log file**

For every decision that the robot makes, we need to print the following:

1. Location of the robot

2. Name of the robot

3. Actions available

4. Tier at which the final decision is taken i.e. tier 1 or tier 2 or tier 3

5. Comment strength of every tier 3 Advisor on every available action

6. Information received by every tier 3 Advisor from the descriptive manager

7. Goal (currently it would be visit target location)

8. Goal details (target location, initial location)

4, 6, 7 and 8 have no immediate use but are for the future.

**Format of the log file**

TBD

**Classes**

Class RobotDecision

{

int decisionId;

Cell robotCell;

String robotId;

Goal robotGoal;

Map<String, String> infoUsed;

List<AdvisorComments> AdvisorCommentsList;

int actionChoice;

}

Class Cell

{

Static int cellSize

int xCoordinate

int yCoordinate

}

Class Goal

{

}

Class VisitNode extends Goal

{

}

Class MoveBox extends Goal

{

}

Class AdvisorComments

{

int AdvisorId

List<float> comments

}

Class Digression

{

int startDecisionId

int endDigressionId

}

Class RSWL

{

List<RobotDecision> parseLog()

{

}

List<List<RobotDecision>> extractSuccessfulDecisionSequences (List<RobotDecision> rawDecisions)

{

}

List<float> computeWeights(List< List<RobotDecision>> successfulDecisionLists);

{

initialize weights to k;

for every SDi in successfulDecisionLists

{

List <RobotDecision> positiveSDi = SDi;

List <Digression> cumulativeDigression = null;

for every Dj in SDi

{

Digression digression = findDigression(SDj from 1 to Dj);

if(digression != null)

{

cumulativeDigression = addtolist(cumulativeDigression, digression);

weights = addPenalty(digression, SDi, weights);

}

}

positiveSDi = removeDigression(SDi, cumulativeDigression);

for every Dj in positiveSDi

{

weights = addRewards(weights, Dj);

}

}

return weights;

}

Pair findDigression (decisionSequence List<RobotDecision>)

{

decision = lastof(decisionSequence);

hasDigression = false;

for i from the n-1 to 1

{

if (checkEqual(decisionSequence[i],decision)

{

Digression digression = new digression(decisionSequence[i].id, decisionIds);

hasDigression = true;

return digression;

}

}

if(hasDigression == false)

{

return null;

}

}

Boolean checkEqual(RobotDecision d1, RobotDecision d2)

{

if(d1.robotCell.xCoordinate == d2.robotCell.xCoordinate && d1.robotCell.yCoordinate == d2.robotCell.yCoordinate)

{

return true;

}

else

{

return false;

}

}

List<Digression> mergeOverlaps (List<Digression> cumulativeDigression)

{

cumulativeDigression = mergeSort(cumulativeDigression);

List<Digression> mergedCumulativeDigression;

for(int i = 1; i = length(cumulativeDigression) ; i++)

{

checkMerge = false;

if(cumulativeDigression[i+1].startDecisionId <= cumulativeDigression[i].endDecisionId)

{

if(cumulativeDigression[i].startDecisionId > cumulativeDigression[i+1].startDecisionId)

{

start = cumulativeDigression[i].startDecisionId;

}

else

{

start = cumulativeDigression[i+1].startDecisionId;

}

end = cumulativeDigression[i+1].endDecisionId;

Digression digression = new Digression(start, end)

mergedCumulativeDigression.addTolist(digression)

checkMerge = true

}

else

{

if(checkMerge == false)

{

mergedCumulativeDigression.addtoList(cumulativeDigression[i]);

checkMerge = false;

}

}

}

return mergedCumulativeDigression;

}

List<RobotDecision> removeDigression (List<RobotDecision> decisionSequence, List<Digression> cumulativeDigression)

{

cumulativeDigression = mergeOverlaps(cumulativeDigressions);

for(i = 1; i <= length(cumulativeDigression); i++)

{

decisionSequence = decisionSequence.removeFromList(cumulativeDigression[i])

}

}

List<int> addPenalty(Digression digression, List<RobotDecision> decisionSequence, List<int> weights)

{

wrongDecision = decisionSequnce.findinlist(digression.startDecisionId);

List<AdvisorComments> advisorCommentsList = wrongDecision.advisorCommentsList;

int actionChoice = wrongDecision.actionChoice;

for(i = 1; i<= length(advisorCommentsList);i++)

{

if(isAdvisorInvolved(advisorCommentsList[i]) == true)

{

int relativeSupport = computeRelativeSupport(advisorCommentsList[i],actionChoice);

problemHardness = 1;

if(actionSupport > 0)

{

weights[i] = weights[i] + problemHardness \* relativeSupport;

}

}

}

return weights;

}

Boolean isAdvisorInvolved(AdvisorComments advisorComments)

{

List<int> comments = advisorComments.comments;

for(i = 1; i <= length(comments); i++)

{

if(comments[i] != 0)

{

return true;

}

}

return false;

}

List<List<RobotDecision>> extractSuccessfulDecisionSequences(List<RobotDecision> rawDecisions)

{

List<List<RobotDecision>> successfulDecisionSequence;

for(i = 1; i <= size(rawDecisions); i++)

{

List<RobotDecision> robots[numberOfRobots];

RobotDecision decision rawDecisions[i];

if(decision.tier == 3)

{

if(decision.goalState == false)

{

robots[decision.robotId].addtoList(decision)

}

else

{

successfulDecisionSequence.addtoList(robots[decision.robotId]);

robots[decision.robotId].removeAllElements;

}

}

}

}

}

RSWL()

{

List<RobotDecision> rawDecisions = parseLog();

List<List<RobotDecision>> successfulDecisionLists = extractSuccessfulDecisionSequences(rawDecisions);

List<float> weights = computeWeights(successfulDecisionLists);

Void writeIntoConfigFile(weights);

}

In situations where there are multiple digressions, it might happen that some of the digression overlap.

Consider a decision sequence from d1 to d20 and robotState(d10) = robotState(d3) and robotState(d12) = robotState(d7). Then findDigression function would return digression (3,10) and (7,12)

mergedTotalDigression **mergeOverlaps**(totalDigression)

sort totalDigression based on the index of the starting decision of the digression

FOR every digression in the sorted totalDigression

checkMerge = false;

nextDigression is the next digression in totalDigression

IF(nextDigression.startDecisionId <= digression.endDecisionId)

IF(digression.startDecisionId > nextDigression.startDecisionId)

newStartDecisionId = digression.startDecisionId;

ELSE

newStartDecisionId = nextDigression,startDecisionId;

ENDIF

endDecisionId = nextDigression.endDecisionId;

newDigression = (newStartDecisionId,newEndDecisionId)

mergedTotalDigression.addtoList(newDigression)

checkMerge = true

ELSE

IF(checkMerge == false)

mergedTotalDigression.addtoList(newDigression);

ENDIF

ENDIF

ENDFOR

return mergedTotalDigression;

End