Exercise 1 – Selbstorganisierende Systeme

Michael Wagner (0827376)
Taylor Peer (0725922)

Iterated Prisoner's Dilemma

Overview

Our example implements a simple simulation of the iterated Prisoner's Dilemma, a game theory example that demonstrates the behavior between "rational" agents. The simulation involves two prisoners, A and B, who must either blame a crime on the other prisoner or remain silent. Their decisions are made independently of one another and neither knows what the other will decide. The impact of their decisions are as follows:

- If A and B each say the other is guilty, each of them serves 2 years in prison
- If A betrays B but B does not implicate A, A will be set free and B will serve 3 years in prison (and vice versa)
- If neither A and B implicates the other, both of them will only serve 1 year in prison

This game is played by the agents and the communication and game results are printed out to the console. The iterative component of the game is that multiple rounds are played, with each prisoner having a memory of the outcome of the previous round. That is, they are aware of a previous betrayal while making their decision for the current round. Depending on the strategy chosen for the prisoner, this may or may not impact their decision during each round of the game.

Agents

PrisonGameMasterAgent: an agent that controls the flow of the game. This agent is started with the local names of the two prisoners and the number of iterations to play. Each round the PrisonGameMasterAgent communicates with the prisoners to retrieve their confession or denials and when the iterations are done, it prints out the game results.

PrisonerAgent: an agent that simulates the prisoners of the game. Two of them are instantiated at runtime with a particular game strategy and each waits for a request from a PrisonGameMasterAgent before sending their decision whether to confess or not. To make that decision they expect a GameInfo object, which includes the last round of the game. Depending on the configured strategy for the agent, the GameInfo object may then be used as part of the agent's decision making process.

Strategies

Default: a naive strategy that causes the *PrisonerAgent* to always confess, regardless of the outcome of previous rounds.

Random: the *PrisonerAgent* randomly choose if to confess or not, regardless of the last rounds.

Retaliation: this strategy makes the *PrisonerAgent* first confess and then subsequently replicate the other prisoners choice from the last round.

Agent Communication Structure

The agent communication between the PrisonGameMasterAgent and the *PrisonerAgents* takes place via ALCMessages that are matched against the FIPANames.InteractionProtocol.*FIPA_QUERY* protocol and the ACLMessage.*QUERY_IF* performative. It is expected that a serialized GameInfo object is included in the request message to the *PrisonerAgents*. If not, some strategies (like *Retaliation*) will lack required information and will revert to a default strategy.

Results

Intuitively, one might assume that the overall most rational gameplay for the prisoners would be to always remain silent, since this would divide the jail time between them equally. However, since each prisoner is working independently to minimize their own jail sentences, it can be beneficial for a prisoner to betray the other. Due to the iterative nature of the simulation, this can break the trust between the two prisoners and lead to retaliatory decisions by the betrayed prisoner. Depending on the strategies used, the individual prison sentences may vary greatly between the two prisoners.

	P2 Default	P2 Random	P2 Retaliation
P1 Default	P1: 2000 years P2: 2000 years	P1: 2480 years P2: 1040 years	P1: 2000 years P2: 2000 years
P1 Random	P1: 966 years P2: 2517 years	P1: 1490 years P2: 1526 years	P1: 1501 years P2: 1504 years
P1 Retaliation	P1: 2000 years P2: 2000 years	P1: 1465 years P2: 1465 years	P1: 2000 years P2: 2000 years

Results from an experiment using various strategy combinations over the course of 1000 iterations

Execution Instructions

To run the simulation, execute the following command:

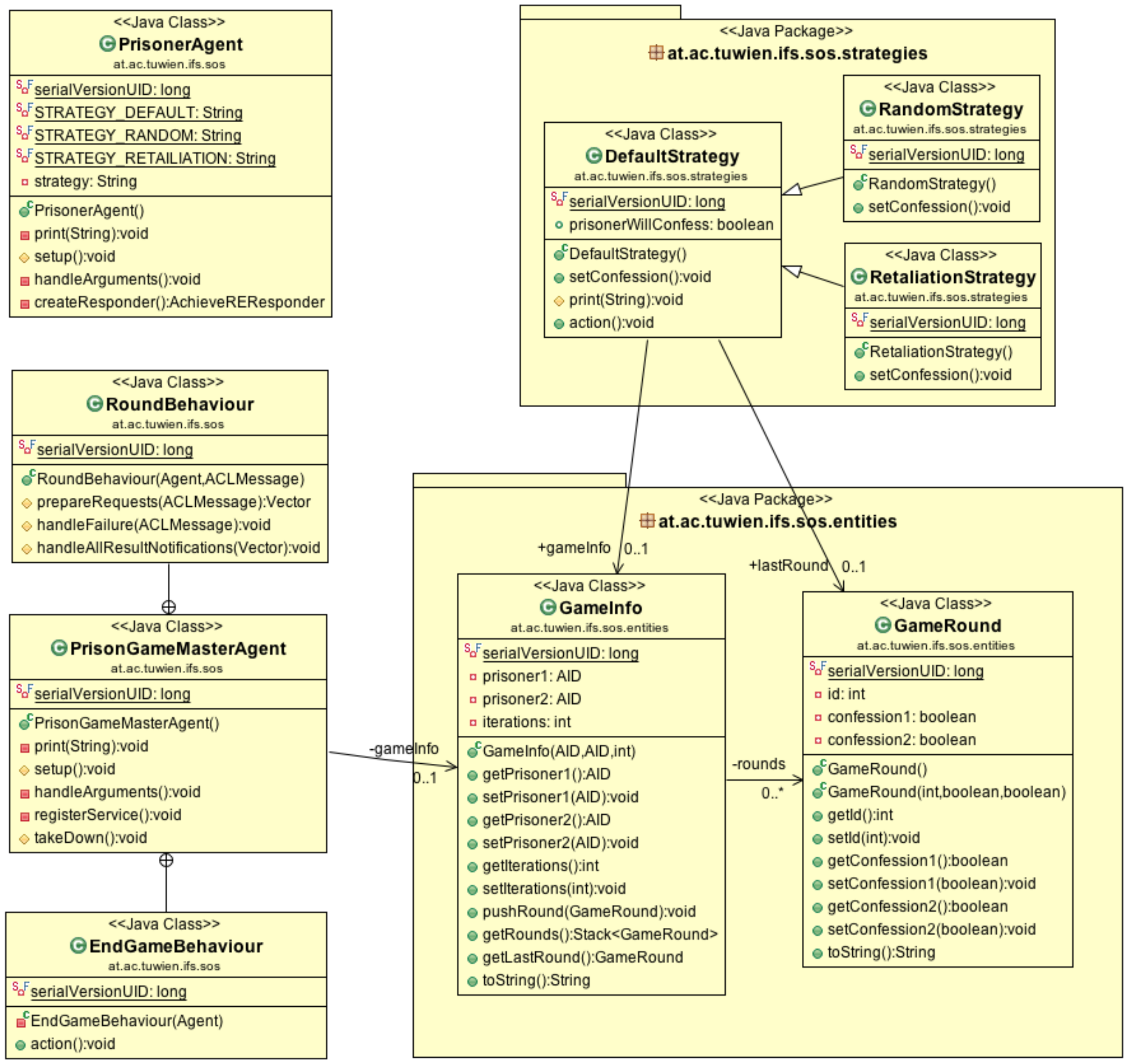
```
java -cp jade-4.3.3.jar;sos-prison-exercise.jar jade.Boot -agents
"prisoner1:at.ac.tuwien.ifs.sos.PrisonerAgent(p1strategy);prisoner2:at.ac.tuwien.ifs.sos.Prison
erAgent(p2strategy);pm:at.ac.tuwien.ifs.sos.PrisonGameMasterAgent(prisoner1, prisoner2,
iterations)"
```

- Supplement *p1strategy*, *p2strategy*, and *iterations* with the string values of the strategy to use for prisoner 1, prisoner 2 and the number of game iterations to play, respectively. Valid game strategy values are *default*, *random* and *retaliation*.
- The -gui parameter may be used to optionally start the JADE interface tool
- On UNIX-like systems it is necessary to replace the semicolon with a colon (*...jade-4.3.3.jar:sos-prison-exercise.jar...*)

Additionally it is possible to create agents in the GUI, e.g. for starting new PrisonGameMasterAgents or new PrisonerAgents waiting to be contacted by a PrisonGameMasterAgent.

It is also possible to build the project with maven from the submitted source code and when successfully built, starting one of the start scripts.

Class Structure:



<<Java Class>>
PrisonerAgent
at.ac.tuwien.ifs.sos
serialVersionUID: long
STRATEGY_DEFAULT: String
STRATEGY_RANDOM: String
STRATEGY_RETAILIATION: String
strategy: String
PrisonerAgent()
print(String):void
setup():void
handleArguments():void
createResponder():AchieveREResponder
<<Java Class>>
RoundBehaviour
at.ac.tuwien.ifs.sos
serialVersionUID: long
RoundBehaviour(Agent,ACLMessage)
prepareRequests(ACLMessage):Vector
handleFailure(ACLMessage):void
handleAllResultNotifications(Vector):void
<<Java Class>>
PrisonGameMasterAgent
at.ac.tuwien.ifs.sos
serialVersionUID: long
PrisonGameMasterAgent()
print(String):void
setup():void
handleArguments():void
registerService():void
takeDown():void
<<Java Class>>
EndGameBehaviour
at.ac.tuwien.ifs.sos
serialVersionUID: long
EndGameBehaviour(Agent)
action():void
<<Java Package>>
at.ac.tuwien.ifs.sos.strategies
<<Java Class>>
DefaultStrategy
at.ac.tuwien.ifs.sos.strategies
serialVersionUID: long
prisonerWillConfess: boolean
DefaultStrategy()
setConfession():void
print(String):void
action():void
<<Java Class>>
RandomStrategy
at.ac.tuwien.ifs.sos.strategies
serialVersionUID: long
RandomStrategy()
setConfession():void
<<Java Class>>
RetaliationStrategy
at.ac.tuwien.ifs.sos.strategies
serialVersionUID: long
RetaliationStrategy()
setConfession():void
<<Java Package>>
at.ac.tuwien.ifs.sos.entities
+gameInfo
0..1
+lastRound
0..1
-gameInfo
0..1
-rounds
0..*
<<Java Class>>
GameInfo
at.ac.tuwien.ifs.sos.entities
serialVersionUID: long
prisoner1: AID
prisoner2: AID
iterations: int
GameInfo(AID,AID,int)
getPrisoner1():AID
setPrisoner1(AID):void
getPrisoner2():AID
setPrisoner2(AID):void
getIterations():int
setIterations(int):void
pushRound(GameRound):void
getRounds():Stack<GameRound>
getLastRound():GameRound
toString():String
<<Java Class>>
GameRound
at.ac.tuwien.ifs.sos.entities
serialVersionUID: long
id: int
confession1: boolean
confession2: boolean
GameRound()
GameRound(int,boolean,boolean)
getId():int
setId(int):void
getConfession1():boolean
setConfession1(boolean):void
getConfession2():boolean
setConfession2(boolean):void
toString():String