Class AgentExp

Αρχικάμε υλοποιήσαμε την κλάση **AgentExp** όπου κάνοντας την extend AbstractNegotationParty (βασική υλοποιήση του NegotiationParty) μπορούμε να κάνουμε @Override τις μεθόδους :

1. **init**(**NegotiationInfo** info)
2. **chooseAction**(java.util.List<java.lang.Class<? extends **Action**>> possibleActions)
3. **receiveMessage**(**AgentID** sender, **Action** action)
4. **getDescription**()

που θα μας βοηθήσουν στο set up των μεταβλητών μας.

```
public class AgentExp extends AbstractNegotiationParty {

    // start-up values - init

    private Bid lastReceivedBid = null;
    private Domain domain = null;

    private List<BidInfo> lBids;

    private double threshold_low = 0.9999;
    private double threshold_high = 1.0;

    public void init(NegotiationInfo info) {

    public Action chooseAction(List<Class<? extends Action>> validActions) {

    public void receiveMessage(AgentID sender, Action action) {

    public String getDescription() {

}
```

1. **init**(**NegotiationInfo** info)

```
@Override
public void init(NegotiationInfo info) {

    super.init(info);
    this.domain = info.getUtilitySpace().getDomain();

    // create list of bids & sort it descending
    lBids = new ArrayList<>(AgentTool.generateRandomBids(this.domain, 30000, this.rand, this.utilitySpace));
    Collections.sort(lBids, new BidInfoComp().reversed());

}
```

Αυτή η μέθοδος έχει ως σκοπό την αρχικοποιήση του party καθώς επίσης και για την ενημέρωση του με διάφορες πληροφορίες για το negotiation.

Με την εντολή `this.domain = info.getUtilitySpace().getDomain()` μπορούμε μεσά από το utility space του agent μας να πάρουμε το domain του. Στη συνέχεια μέσω της `lBids = new ArrayList<>(AgentTool.generateRandomBids(this.domain, 30000, this.rand, this.utilitySpace));` δημιουργούμε μια λίστα στην οποία παράγουμε και αποθηκεύουμε 30000 τυχαία bids για το συγκεκριμένο domain.Τέλος με την χρήση της `Collections.sort(lBids, new BidInfoComp().reversed());` ταξινομούμε τα bids κατά φθίνουσα σειρά.

2. chooseAction(java.util.List<java.lang.Class<? extends Action>> possibleActions)

```java
@Override
public Action chooseAction(List<Class<? extends Action>> validActions) {

    // Setting compromise degree

    threshold_high = 1 - 0.1 * timeline.getTime();
    threshold_low = 1 - 0.1 * timeline.getTime() - 0.0001 * Math.exp(this.timeline.getTime());

    System.out.println("max util: "+threshold_high+"\nmin util: "+threshold_low);

    // if time is running out
    // drops standards to reach an agreement
    if (timeline.getTime() > 0.99) {
        threshold_low = 1 - 0.2718 * timeline.getTime();
    }

    /**
    * Function timeline.getTime()
    * Gets the time, running from t = 0 (start) to t = 1 (deadline). The time
    * is normalized, so agents need not be concerned with the actual internal
    * clock.
    *
    * @return current time in the interval [0, 1].
    */


    // Accept Agreement
    if (lastReceivedBid != null) {
        if (getUtility(lastReceivedBid) > threshold_low) {
            return new Accept(getPartyId(), lastReceivedBid);
        }
    }

    // Offer selection of bids
    Bid bid = null;
    while (bid == null) {
        bid = AgentTool.selectBidfromList(this.lBids, this.threshold_high, this.threshold_low);
        if (bid == null) {
            threshold_low -= 0.0001; // every time I don't find a bid, I drop the low threshold (min utility)
        }
    }
    return new Offer(getPartyId(), bid);
}
```

Όταν καλείται η μέθοδος αυτή , ο agent επιλέγει από την action list που του παρέχουμε ποιο action θα επιλέξει δηλαδή είτε θα κάνει accept το offer που υπάρχει ή θα κάνει ένα καινούργιο offer. Αρχικά να πούμε ότι απο την `timeline.getTime()` παίρνουμε τον τρέχον χρόνο (t=0 (start) –

t=1(deadline)) ο οποίος είναι normalized έτσι ώστε ο agent να γνωρίζει πότε πλησιάζει σε deadline. Έτσι με την βοήθεια της μεθόδου αυτή μπορούμε να δημιουργήσουμε 2 όρια , threshold_high και threshold_low,(αποτελούν την περιοχή τιμών για το αν θα γίνει accept το offer του opponent)όπου με την πάροδο του χρόνου θα μειώνονται.Το threshold_low μειώνεται πιο γρήγορα για τον λόγο ότι όσο προχωρά ο χρόνος τόσο πιο πολύ ρίχνουμε τις «απαιτήσεις» του agent μας,έτσι ώστε να καταλήξουμε σε κάποιο agreement.Γι' αυτό και όταν είμαστε κοντά σε deadline (`timeline.getTime() > 0.99`) το ρίχνουμε ακόμη περισσότερο. Όπως αναφερθήκαμε πιο πάνω ένα action που μπορεί να κάνει ο agent μας είναι να κάνει accept την συμφωνία μονο εάν το bid που έκανε ο αντίπαλος είναι μεγαλύτερο από το threshold_low (`getUtility(lastReceivedBid) > threshold_low`) . Το δεύτερο action μπου μπορεί να επιλέξει είναι να κάνει ένα bid(offer) ξανά αλλά αυτήν την φορά θα έχει τιμή η οποία θα βρίσκεται ανάμεσα στα όρια του threshold_high και threshold_low (`bid = AgentTool.selectBidfromList(this.lBids, this.threshold_high, this.threshold_low);`) .

3. receiveMessage (AgentID sender, Action action)

```
@Override
public void receiveMessage(AgentID sender, Action action) {
    super.receiveMessage(sender, action);
    if (action instanceof Offer) {
        lastReceivedBid = ((Offer) action).getBid();
    }
}
```

Αυτή την μέθοδο την καλούμε για να ειδοποιήσουμε το party ότι κάποιος agent(sender) έχει επιλέξει κάποιο action.

4. getDescription()

```
@Override
public String getDescription() {
    return "AgentExp - tucANAC2018-19";
}
```

Με την χρήση αυτης της συνάρτησης παίρνουμε μια human-readable περιγραφή για το party.

Class AgentTool

Σε αυτή την κλάση υλοποίησαμε δυο συναρτήσεις :

1. Bid selectBidfromList(List<BidInfo> bidInfoList, **double** higherutil, **double** lowerutil)
2. Set<BidInfo> generateRandomBids(Domain d, **int** numberOfBids,
   Random random,
   UtilitySpace utilitySpace)

```
class AgentTool {

    private static Random random = new Random();

    public static Bid selectBidfromList(List<BidInfo> bidInfoList, double higherutil, double lowerutil) {

    // one-time (init) creation of main bid list
    public static Set<BidInfo> generateRandomBids(Domain d, int numberOfBids, Random random, UtilitySpace utilitySpace) {

}
```

όπου θα μας βοηθήσουν να επιλέγουμε και να δημιουργούμε bids.

1. Bid selectBidfromList(List<BidInfo> bidInfoList, double higheruti , double lowerutil)

```
public static Bid selectBidfromList(List<BidInfo> bidInfoList, double higherutil, double lowerutil) {
    List<BidInfo> bidPool = new ArrayList<>();
    for (BidInfo bidInfo : bidInfoList) {              // select random bids, but only if they meet util requirements add them to bid pool
        if (bidInfo.getutil() <= higherutil && bidInfo.getutil() >= lowerutil) {
            bidPool.add(bidInfo);
        }
    }

    if (bidPool.size() == 0) {      // no bids within min - max thresholds
        return null;
    } else {
        return bidPool.get(random.nextInt(bidPool.size())).getBid();  //pick random bid from 2nd list
    }
}
```

Σε αυτή την συνάρτηση δημιουργούμε ένα List<BidInfo> bidPool = **new** ArrayList<>(); όπου θα αποθηκεύουμε τα τυχαία bids που παράξαμε(στην λίστα στη κλάση AgentExp) τα οποία βρίσκονται εντός των ορίων που θέσαμε `higherutil, lowerutil` . Σε περίπτωση που δεν υπάρχουν bids στην bidPool επιστρέφουμε null αλλιώς επιλέγουμε τυχαία ένα bid από την λίστα αυτή για να το επιστρέψουμε στην κλάση AgentExp όπου θα μπορεί ο agent να κάνει το offer του.

2. Set<BidInfo> generateRandomBids(Domain d, int numberOfBids, Random random, UtilitySpace utilitySpace)

```java
// one-time (init) creation of main bid list
public static Set<BidInfo> generateRandomBids(Domain d, int numberOfBids, Random random, UtilitySpace utilitySpace) {
    Set<BidInfo> randombidsPool = new HashSet<>();
    for (int i = 0; i < numberOfBids; i++) {
        Bid b = d.getRandomBid(random);
        randombidsPool.add(new BidInfo(b, utilitySpace.getUtility(b)));
    }
    return randombidsPool;
}
```

Με την μέθοδο αυτή δημιουργούμαι τυχαία bids (30000) για το συγκεκρίμενο domain του agent μας(αυτή η μέθοδος καλείται στην κλάση AgentExp στη μέθοδο init()).