Exercise 1: Making the planning less naïve

The dummy agent constructs a plan only at the start of the game. During the lessons it became clear that this is not something we want to have. We should change our plan if certain items become unavailable or too expensive. We first have to program a timer to make this possible:

1. Create a directory, for example ‘exercise’.
2. Download “TAC Classic Java AgentWare” from http://www.sics.se/tac/.
3. Unzip this package in the directory made in step 1.
4. In the directory ‘exercise’ you create a directory ‘myagent’.
5. Copy DummyAgent from exercise/se/sics/tac/aw to exercise/myagent.
6. Rename this DummyAgent.java to MyAgent.java.
7. Open MyAgent in a Java editor.
8. Change ‘package se.sics.tac.aw’ to ‘package myagent’. Now you have added this code to your own package (group of source code).
9. We still have to change the class name: replace ‘public class DummyAgent’ with ‘public class MyAgent’.
10. Go to ‘import java.util.logging.*;’ give enter and add ‘import javax.swing.Timer’.
11. Minimize the editor and open agent.conf in the ‘exercise’ directory.
12. Change ‘agentimpl=se.sics.tac.aw.DummyAgent’ to ‘agentimpl=myagent.MyAgent’ this tells the AgentWare software which implementation to use.
13. We assume that you will be running your own server. If not, then you should change ‘host = localhost’ to ‘host = other_server’.
14. Now reopen MyAgent.java go to the calculateAllocation method.
15. After ‘calculateAllocation() {‘ you give an enter and add: ‘agent.clearAllocation();’. This will clear the current allocation before we make a new one.
16. Now go to ‘import javax.swing.Timer;’ and add ‘import se.sics.tac.aw.*;’ We have to add this import because we placed our agent outside the TAC package. Some classes won’t be found if we don’t do this.

Now we have made the basis for timed allocation. We now have to decide the best time interval. One good time interval is updating every minute. Why? And why would one choose a different (smaller) interval? One shouldn’t make a interval smaller the time needed to make the allocation. Why?

17. We will continue by going to the calculateAllocation method.

18. Add de following code **before** 'for (int d = inFlight; d < outFlight; d++) {':

```
//a switch, with it we will change the hotel type
boolean current = (type == TACAgent.TYPE_GOOD_HOTEL);
     for (int d = inFlight; d < outFlight; d++) {
     //cheap hotels start at 8:
     int start = 8;
     //expensive hotels start at 12:
     if(current)
          start = 12;

     //Do we own enough if we assume
     //all previous clients want the room:
boolean enough = (agent.getOwn(start + d) > (i+1));

     //Is the auction closed?
     Quote quote = agent.getQuote(start + d);
     if(quote.isAuctionClosed() && !enough){
          //Change the switch to change the hotel type:
          current = !current;
          //don't continue:
          break;
     }
}

//Change the hotel type with the current switch:
if (current) {
     type = TACAgent.TYPE_GOOD_HOTEL;
} else {
     type = TACAgent.TYPE_CHEAP_HOTEL;
}
```

Now the allocation algorithm will change the hotel type if the travel package with the other hotel type becomes infeasible. Of course this is far from optimal, but this exercise is only mend to show how one could implement allocation strategies.
Note that we don't want to switch the hotel if we own enough. We only partially solve this. One should construct a better solution in order to make a good agent.

19. Then we add the following code to the gameStarted method:

```
ActionListener taskPerformer = new ActionListener() {
     public void actionPerformed(ActionEvent evt) {
          calculateAllocation();
     }
};

updateTimer = new Timer(1*60*1000, taskPerformer);
updateTimer.start();
```

20. To gameStopped we add: `updateTimer.stop();`

21. We still have to make class field updateTimer. Go to the other private fields and add: `private Timer updateTimer;`
22. Finally we have to import both ActionListener and ActionEvent. Add the following import: `import java.awt.event.*;`

This timer updates the allocation every minute. The server will close auctions and the agent will change it planning accordingly.

Exercise 2: Bidding when the time is right

It has been show in the lessons that flight prices usually rise and that the agent pays the price the server is asking. Logically we would want to bid at a certain time. Why is that?

This is implemented as follows:

1. Go to sendBids. Remove de following lines:

```
case TACAgent.CAT_FLIGHT:
if (alloc > 0) {
  price = 1000;
}
break;
```

2. Now create the following method:

```
private void updateBids(){
     for (int i = 0, n = 7; i < n; i++) {

int alloc = agent.getAllocation(i) - agent.getOwn(i);

          if(alloc > 0){
               Bid bid = new Bid(i);
               bid.addBidPoint(alloc, 1000.0f);
               agent.submitBid(bid);
          }
     }
}
```

This method calculates how many flights we need and places a bid. If we would like to bid after 8 minutes of play we could just make a timer that calls this method after 8 minutes. However in the future you might want to construct a allocation algorithm that shifts in-flights and out-flights. Why would we want that? So again we look at the above method and we will change it:

3. change ‘if(alloc > 0){‘ into
   ‘if(alloc > 0 && agent.getGameTime()> 8*60*1000){‘
4. Then we add the following code to the gameStarted method:

```
ActionListener taskPerformer2 = new ActionListener() {
     public void actionPerformed(ActionEvent evt) {
```

```
            updateBids();
        }
    };

    bidTimer = new Timer(30*1000, taskPerformer2);
    bidTimer.start();
```

5. To gameStopped we add: `bidTimer.stop();`
6. We still have to make class field bidTimer. Go to the other private fields and add: `private Timer bidTimer;`

Every 30 seconds we will update the flight bids and the agent will start bidding after 8 minutes.

Exercise 3: Reacting on the competition

Now we bid on hotel auction with a constant rate of 50 (ask price + 50). In this exercise we are going to change this. We are going to use a very crude technique: take two previous ask prices, calculate their difference, store this difference and use this instead of 50:

1. First add: `private float diff[] = new float[28];`
2. And add: `private float lastAskPrice[] = new float[28];`
3. Then we go to updateBids and we add the following line:

```
        for (int i = 8, n = 15; i < n; i++) {
            Quote quote = agent.getQuote(i);
            if(quote.getAskPrice() > lastAskPrice[i]
            && lastAskPrice[i] != 0){

                float safety = 3.0f;
                diff[i] = (quote.getAskPrice() -
                lastAskPrice[i]) + 3.0f;

            } else if(lastAskPrice[i] == 0){
                diff[i] = 50.0f;
            }
            lastAskPrice[i]=quote.getAskPrice();
        }
```

4. Go back to quoteUpdated(Quote quote) and change

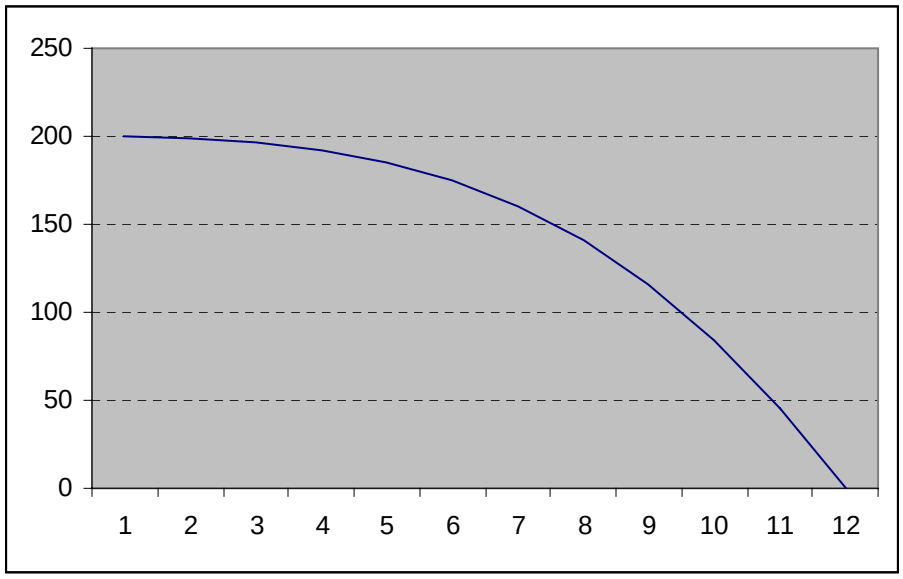
```
prices[auction] = quote.getAskPrice() + 50;
```

to

```
prices[auction] = quote.getAskPrice() + diff[auction];
```

Exercise 4: Changing your selling behavior

In the end of the game we would really like to lose our redundant entertainment tickets. You should answer for yourself why one would like that. We are going lower our selling price in the following way:



This is accomplished with the following simple calculation:

$$x^3 = 200$$

$$X = 5{,}8480$$

0,4873	1
5,8480	12

$$\text{BID\_PRICE} = (\text{TIME} * 0{,}4873)^3$$

Implementing this is also not very complex:

1. First we go to quoteUpdated(Quote quote)
2. In this method we replace the following lines:

```
if (alloc < 0)
prices[auction] = 200f - (agent.getGameTime() * 120f) /
720000;
```

With

```
if (alloc < 0){

double time = ((double)agent.getGameTime())/(60.0*1000.0);
time = time * 0.4873;
long power = 200 - Math.round(Math.pow(time,3));
prices[auction] = (new Float("" + power)).floatValue();

}
```

Exercise 5: Compiling & Running

Compiling

First we will create a manifest file. This file tells the 'java' runner which file in the jar contains the class with the main method. Note that most professionals use Apache Ant for compiling and packaging their Java source.

1. Make a directory 'build', 'dist' and 'manifest' in your exercise directory.
2. Go to the manifest directory and create a file called MANIFEST.
3. Add the following line to this file:

```
Main-Class: myagent.MyAgent
```

Now we will construct a batch file that will compile and package your code:

4. Create a file called compile.bat in the exercise directory.
5. Open this file in an editor and add the following code:

```
javac -d build -classpath . .\se\sics\tac\aw\*.java
javac -d build -classpath . .\se\sics\tac\util\*.java
javac -d build -classpath . .\myagent\*.java
jar cvfm dist\agent.jar manifest\MANIFEST -C build\ .
```

This will tell the compiler to compile you agent, place it in the directory build and jar it into agent.jar.

Running

In order to run your agent, you first have to run the server. I assume that you are working at the university.

1. Download the server from http://www.sics.se/tac/. (Note: the BINARY VERSION)
2. Go to the nettmp directory and make your own directory.
3. Unzip the server package in this directory.
4. Make a batch file called runserve.bat with the following lines of code:

```
cd java
java -jar infoserver.jar
```

5. Now you can first run the TAC Server with tacserver.exe
6. Then you can run the info server with runserve.bat
7. If you get a null pointer exception: close the info server and rerun it.
8. Open http://localhost:8080 in a explorer window.
9. Register a new user gadgit with the password gogadgit.

Open a dos prompt and go to were ever you have your agent.jar. You can probably find this in exercise/dist. To run your agent use the following command:

```
java -jar agent.jar -agent gadgit -password gogadgit -config
agent.conf
```

Now wait and see what happens.

Note

- And don't forget to place agent.conf in the same directory!