

CM30174

Dr. Julian Padget
Dr. Marina De Vos

***Blazin* – A TAC Classic Agent**

Solutions and Strategies

Amit Kothari (ma0ak@bath.ac.uk)

Brian Ferguson (cs2bcf@bath.ac.uk)

1	Introduction.....	3
1.1	Trading Agent Competition.....	3
1.1.1	Flights – 8 auctions.....	4
1.1.2	Hotels – 8 auctions.....	4
1.1.3	Entertainment – 12 auctions.....	4
1.2	Existing Strategies & Literature Review.....	5
2	Trading strategies.....	7
2.1	Introduction.....	7
2.2	Flight auctions.....	8
2.2.1	Strategy F1: “Early bird”.....	8
2.2.2	Strategy F1.1 “Phased early bird”.....	8
2.3	Hotel auctions.....	9
2.3.1	Package Based Strategy SET.....	10
2.3.1.1	Strategy H1: “Packages randomly allotted into the two pies”.....	10
2.3.1.2	Strategy H1.1: “Packages sorted and entered into the two pies”.....	11
2.3.1.3	Strategy H2: “Packages allotted into the pies with risk grading”.....	11
2.3.1.4	Strategy H3: “Packages sorted into the two pies with reallocation potential”.....	12
2.3.1.5	Strategy H4: “Extreme filler”.....	12
2.3.1.6	Algorithm H5: “Highest Number Wins for Two Packages”.....	13
2.3.1.7	Strategy H6: “Strategy Omnibus”.....	14
2.3.2	Total Need Strategy SET.....	15
2.3.2.1	Strategy HT1: “Parameter decision”.....	15
2.3.2.2	Strategy HT2: “Parameter Decision with reallocation”.....	15
2.3.2.3	Strategy HT3: “High quantities gives license for greater risk”.....	15
2.4	Entertainment Auctions.....	17
2.4.1	Strategy E1: “Allocate and leave”.....	17
2.4.2	Strategy E1.1: “Allocate and sell remainder”.....	17
2.4.3	Strategy E2: “Allocate, sell remainder and purchase the useful”.....	17
2.4.4	Strategy E3: “Allocate, sell remainder and optimal purchase”.....	18
2.4.5	Strategy E4: “E3 with parametric speculative buying”.....	18
2.4.6	Strategy E5: “Hopeful bids”.....	19
2.5	Interactions and Scoring.....	20
2.5.1	Hopeful bids.....	20
2.5.2	Quantity vs. price.....	20
2.5.3	Utility scoring.....	20
2.5.4	Unit pricing.....	21
3	Implementation.....	22
3.1	Introduction.....	22
3.2	Good programming techniques.....	22
3.2.1	Details Object (Flights/Hotels).....	22
3.2.2	Entertainment Object.....	24
3.3	Agent versioning system.....	25
3.4	Choosing and testing agent versions.....	25
3.5	Particular implementation constraints.....	25
4	Improvements.....	26
5	Appendix.....	27
5.1	Bibliography.....	27
5.2	Personal Reports.....	27
5.3	Source Code for Submission.....	28

1 Introduction

1.1 Trading Agent Competition

The TAC Agent Trading Game is a research-oriented environment to enable autonomous competing software agents to score a utility based on their costs, penalties and bonuses, defined amongst each customer (in a set of 8) by Figure 1. It is run annually and features a new type of game this year (2003) based on supply chain management. Our work addresses TAC Classic.

$$\text{Utility} = 1000 - \text{TravelPenalty} + \text{HotelBonus} + \text{FunBonus}$$

Where $\text{TravelPenalty} = 100 * (|\text{ArrivalDate} - \text{PlannedArrivalDate}|) + (|\text{DepartureDate} - \text{PlannedDepartureDate}|)$

Where HotelBonus is XX points for each night in TT

Where FunBonus is the sum of the reservation values of all the entertainment received

An agents' net score is the total client trip utility (added over 8 clients) minus the net expenditures in auctions. Maximum and minimum utility scores for *feasible* packages are as shown below.

$$\text{Order: } Arr | Dep | Hotel | AW(e1) | AP(e2) | MU(e3)$$

$$U_{\min} = 1000 - 300 - 300 + 0 + 0 + 0 + 0 = \mathbf{400}$$

$$U_{\max} = 1000 - 0 - 0 + 150 + 200 + 200 + 200 = \mathbf{1750}$$

Figure 1 – Utility scoring relations

Clients describe their preferences, namely arrival and departure days, a bonus payable for nights stayed at the Tampa Towers Hotel (an overall better class of hotel) and their preference values for each type of entertainment which we shall call e1, e2, e3. Each game lasts 12 minutes.

1.1.1 Flights – 8 auctions

Range from Day 1 to Day 4 incoming and Day 2 to Day 5 outgoing. Flight prices update every 24-32 seconds. A feasible trip must have incoming and outgoing flights, as in the real world, with a penalty being applied for flights that cannot be allocated to within the preferences (see Figure 1 - TravelPenalty). The prices of flights follow an upward bias from initial starting values.

1.1.2 Hotels – 8 auctions

The 8 auctions are for TT (nicer hotel) and SS (not as nice) and span days 1 to 4 only. A feasible trip must include rooms in *one* kind of hotel for all of the nights of a clients' stay, except for the day of departure. There are 16 rooms available for each hotel, every night, sold through ascending 16th price auctions. The auctions themselves are all open at the beginning of the game, and each auction closes in a random order from minute 4 of the game through to minute 12. Quotes for all open hotel auctions are supplied every minute and new bids must exceed the going ask price as well as the quantity which was winning at (ask price + 1). There are 4 TT auctions and 4 SS auctions.

1.1.3 Entertainment – 12 auctions

Agents are endowed with an initial allocation (free of cost) of 12 entertainment tickets at given days and for entertainment types 1, 2 and 3. They may allocate these to their own clients (leave them as they are) or trade them with other agents through a continuous double auction. Each entertainment auction issues bid and ask prices. In the stated entertainment preferences, we do not gain additional utility if all of the 3 entertainment options have already been exercised during a client's stay.

The problem is to use the rules of the game and the *callback driven* API to construct logic and reasoning based on our chosen strategies, preferably a different strategy for flights, hotels and entertainment. Our agent must be capable of entering the TAC competition. Dealing, trading and transacting will all be defined with some level of correctness within our solution and the goal in the end is to consistently obtain a high net score. Our initial aims were targeted within the 1500-2500 points range.

1.2 Existing Strategies & Literature Review

The 2001 TAC seemed to have “substantially better agents” (Wellman, Greenwald, et al) as a result of rule changes. Also quoted in this review is a simple view of a “bidding cycle”.

REPEAT

- 1) For each good, do I want to bid now or later?
- 2) For each good that I want to bid on now, what quantity do I want to buy or sell?
- 3) For each quantity I want to exchange, what price should I offer?

UNTIL GAME OVER

(Wellman, Greenwald, et al)

The simplicity of the above inspired us to create a simple flow process for each auction type, which are illustrated later in this paper. There are also two ways to analyse needs – customer by customer and by total need. For the hotels section of this paper, we have divided these strategies accordingly. We note that *TacsMan* was the only entry that optimised travel packages by client, rather than globally in 2001.

We investigated *SouthamptonTAC* (Mighua He, Nicholas Jennings), a top performer in TAC-2002. The approach adapted by this entry was similar in that an autonomous adaptive system reacted to a changing market, moulding its bidding “temperament” in accordance with prevailing competition. An optimal package was dynamically created at run-time and presented an optimal allocation with reference to penalties, prices and availabilities. Especially useful was a section summarised below which classes “trading environments” (pp:4,5).

- Competitive Environments – where hotel prices are very high. Predicting hotel prices is crucial to *SouthamptonTAC* in this environment.
- Non-competitive Environments – where little competition exists and therefore it would be better to buy flights earlier.
- Semi – competitive Environments – where prices remain “reasonable”. There is not severe competition.

We expect our agent to perform well in a non/semi competitive environment since constructing an optimal re-allocation and hotel price prediction module would be unfeasible with our resources. *SouthamptonTAC* has an environment sensor, which parallels with our crude idea (later in this paper) of a risk grade or volatility rating.

A result quoted from *livingagents*, another top 2000 performer gives us hopes that new ground in agent design is not something out of reach in terms of bidding strategy.

>> Introduction : Existing Strategies & Literature Review

“... lead to a completely different strategy than was used by the highest scoring team ATTac in 2000. That year, ATTac tried to delay most of the purchases to the very end of the auction round (Stone et al, 2001). Our strategy was doing exactly the opposite; our agents took most of their decisions at the very beginning of an auction round.” (*livingagents* team - Clemens Fritschi, Klaus Dorer)

Further analysis comparing *livingagents* with *ATTac* is available in the 2001 review conducted by Wellman, Greenwald, et al. In terms of specific approaches, one of the most useful examples we read as a pretext was *Walverine* (Cheng et al. 2003) which used a Walrasian analysis of the travel economy, thereby constructing a decision-theoretic formulation of the optimal bidding problem. *Walverine* further notes that this strategy may be applied to a broader range of trading environments.

The recurring themes in most of the world-class agents were identified as follows:

- Price prediction – whether taken historically, through “training” or by other means. *Walverine* went through “training”. Wellman, Reeves et al. (2002) have discussed price prediction approaches in detail.
- Adaptive strategies – able to score well in any trading environment.
- Adaptive “temperaments” – able to assess the importance of a need and chase it within dynamic constraint sets.

From our literature review and further research that is not quoted herein, it seems unlikely that we have the resources to code an agent that would seed into the first round or derive AI-based strategies for its implementation. Instead, we plan to start with very basic approaches, and build on this groundwork. The notion of a “greedy” agent is unclear to us, as after completing this work, we have found that each of the three markets have specific advantages and disadvantages to being greedy. Further, the environments determine whether one needs to be “greedy” or not. The interaction between flights and hotels is clear and they are inextricably linked. Resources may force us to consider these two markets separately.

2 Trading strategies

2.1 Introduction

It is clear that we cannot have a single strategy to transact flights, hotels and entertainment. However, our review has shown that flights and hotels may be combined into a single system, as demonstrated in the architecture of *Walverine* (2003:pp5).

The differences between each type of trading are what make TAC an active playground for exploiting any differences in the rules for flights, hotels or entertainments.

Hotels involve a range of ideas and solutions but remain elusive for a “holy grail” strategy, mainly because we cannot predict when each hotel auction is about to end. We shall attempt to minimize our risk by cutting exposure to high losses, but the consequence of this will be a loss in high cost profits.

The entertainment trader seems the most complex since we need to see if our initial ticket allocation is useful and then proceed to:

- Sell the entertainment tickets we don't need
- Look to buy the ones we do need at reasonable prices
- Track every customer and ensure no duplicate entertainment fills are made

The system allocates our owned tickets optimally at the end of the game, so we have identified that we are only concerned with:

- Totals needed for each day for each entertainment type, for all customers
- Each customer having 3 unique entertainments, and no more
- If possible, a consideration of each customer's preference prices

If we meet the above targets in the entertainment implementation, we can be assured of a sound entertainment strategy.

Interdependencies have been identified as follows:

- Flights are useless if hotel rooms are not available.
- Hotel rooms are useless if flights are not available.

We see the link between strategy F1.1 in Section 2.2.2 and its' ability to solve the interdependency where flights are useless if hotel rooms are not available. However, for reasons given in Section 2.2.2, which includes implementation constraints, we anticipate buying all flights at the beginning as our realistic solution, taking the risk accordingly on the hotel rooms.

2.2 *Flight auctions*

From the description of the rules given in Section 1.1, it has been stated that prices tend to generally increase towards the end of the auction, so the most economical way to obtain the flights which we must obtain (to avoid the Travel Penalty) is to bid for them first. This presents an obvious strategy and a variant of it which relies on hotel rooms being filled before flights are obtained. Reasons to implement F1.1 include assurance of complete *feasible* packages.

2.2.1 Strategy F1: “Early bird”

This very simple strategy bids for all our flights requirements (since the number of tickets is unlimited) as soon as the game starts. The worst case for this is that prices are slightly higher at the beginning, then drop slightly, the best case is obvious.

2.2.2 Strategy F1.1 “Phased early bird”

A variation on F1, where we are buying flight prices based on hotel fills being *almost* fully made or being *actually* fully made. As we obtain further information about hotel prices and whether we can make the rest of the hotel fill, we buy the rest of the required flights at a future time that is optimal.

For the resources we have, our first study into this possibility has not yielded fruitful results. It will require more implementation time in that we have to make the “optimal” time to buy flights sensitive to hotel “availability” – where the “availability” quotient is itself difficult to quantify. For example, if hotel auctions close randomly, how do we decide that the time to buy flights for a given package is the optimal time?

At worst, we will have to buy the remaining flights at much higher prices, and the question arises as to whether this flight price premium we pay for these late transactions is balanced by savings we make with the hotels. We think the premium will generally be a lot higher than savings made on hotels. The clear benefit here is that we are assured of complete packages rather than spending on flights when hotels may not be assured.

2.3 Hotel auctions

Hotel auctions are active for each of TT and SS for each day. We note initially that clients give a preference bonus for staying in TT. Therefore, we view the basic requirement as “optional” to stay in TT, since this bonus (50-150) is for the *whole* stay. Since hotel auctions close randomly from minutes 4 through 12, we conclude that attempting to predict their closure is fruitless. Therefore, even if auctions that close earlier should be cheaper for us, we cannot know which ones they are. A second thread of discussion revealed that we might be able to obtain a quotient from each auction which we called “risk grade” based on frequency and quantity of bids and past price increases for the first 4 minutes (quotes are supplied every minute). Naturally then, we would focus our clients’ fills into the low risk grade auctions for each of set TT and SS. However, we may not have the resources to actually implement risk grade based selection. Our evolving hotel strategies attempt to narrow down the difference between worst and best case performance.

If a client’s preference price is not higher than any current TT asking price in any open auction, there is no point in us pursuing a TT stay for that client, and the client should be automatically reverted to bidding for SS since they will not be “TT profitable”.

Figure 2 – A Definition of Client Hotel Preferences being “TT profitable”

We have two approaches to bid for hotels:

- **By customer package**
- **By total need for each auction (for all customers)**

If we do it by package, it looked like a tedious affair to implementation, but gives us finer grain control over which customers to put into which auctions and precisely why we want to reallocate them into SS/TT. If we do it by total need, each individual customer is ignored, and all we want to create is a total number that we need for that auction. We shall decide in implementation which to go for, and accordingly, use only one of the two major sections detailed below.

2.3.1 Package Based Strategy SET

2.3.1.1 Strategy H1: “Packages randomly allotted into the two pies”

Here we present a case to have each of our 8 clients’ of either TT or SS in each of the 8 auctions, which is convenient in terms of numbering. Here, four random clients will randomly be allotted into the TT auctions, while the other 4 are randomly allotted into the SS auctions. One pie is our metaphor to represent the entire set of TT or SS auctions – hence we have two pies for hotels.

The advantage with this is that if a certain auction spirals into high prices (since it happens to close last, or for any given reason), another auction (one with the lowest prices or risk grade) might close at a comparably “profitable” price. The end result is a conservative payoff which results in our prices being averaged out. By distributing our packages equally, we have maximum exposure to the full hotel market, and hopefully weather its best and its worst for a reasonable performance.

Clearly, rather than simply randomly allocate our 8 clients to the 2 pies, we may consider:

- Allocating our clients into an ordered set with the 4 highest TT “preference value” packages getting entered first into the 4 TT auctions – giving us a chance for maximum profit at best cost price. The remaining 4 clients are entered into the SS auctions. Notice that we still keep “packages allotted into the two pies” but we optimise which package goes into which pie.
- What happens when some kind of optimal allocation (like our optimal customer in a TT auction) becomes non profitable. In other words, this client for this auction becomes non TT profitable (Figure 2). We have to therefore reallocate this client to an open SS auction.

The simple version of H1 is pseudo-coded below. It does not obey the maxim of being TT profitable since obeying it would require package reallocation, which we deal with later. The additional possibilities above begin to get addressed in variants of H1.

```
gameStarted () { Allocate_Hotels () }

Allocate_Hotels () {
  for (all clients) {
    if (client_number <= total_clients / 2)
      enter_TT (client_number);
    else : enter_ss (client_number);}
}
```

>> Trading strategies : Hotel auctions

2.3.1.2 Strategy H1.1: “Packages sorted and entered into the two pies”

This is a variant where customers with the highest TT bonuses are sorted and allocated to the TT auctions, and the rest are allocated into the SS auctions. The only change from H1 is in the allocation. This is a first attempt at achieving TT profitability, without auction reallocation on breaking TT profitability.

```
Allocate_Hotels () {
    Sort(client_array);
    for (all clients) {
        if (client_number <= total_clients / 2)
            enter_TT (client_number);
        else : enter_ss (client_number);}
}
```

2.3.1.3 Strategy H2: “Packages allotted into the pies with risk grading”

In this strategy, the idea is to create 2 dynamic risk grade quotients for the two pies – SS and TT, with respect to a particular package. Data is gathered until the 4th minute (or the 4th quote we receive) and allocation occurs based on risk grade at that time. The risk grade quotient *might* be created as shown in Figure 3.

Risk grade may be based on two factors – the frequency of bids in the first 4 minutes of the auction, and ask price data for each of the first 4 minutes.

The frequency of bids in the auction before they begin closing may be determined by calling a method on the auction to obtain current going bid *rate*. This might not be possible using the Agentware API. The ask price increases are simply logged into an array and the volatility calculated. We then use a statistical volatility of deviation instrument to see which auction gives us the highest volatility grading. This is based on the logic that highly volatile auctions may show a higher tendency for our agent overpaying for this hotel type/day.

Both factors are assessed a score rated and assigned on two fronts *for each package* – one risk grade for the TT pie and one for the SS pie. Actual allocation can then proceed.

Figure 3 – Risk grade

The aim is to make on-the-fly selection with risk grade, which can give us a better idea of where to put a particular package in the *current* market.

2.3.1.4 Strategy H3: “Packages sorted into the two pies with reallocation potential”

This strategy makes each customer TT profitable and considers reallocating them once they reach a state where they might break their TT profitability.

The decision that it makes about *if* we reallocate the package is a different question which is looked at using an algorithm, etc. later in this section.

The decision is based on choosing any random SS auction which is open. If no SS auction is currently open their package will be forced to continue bidding in their current auction. Objects and their methods are used implicitly below. Allocation methods remain the same and we see the addition of a re-allocation method.

```
gameStarted () { Allocate_Hotels () }

REAllocate (client_number) {
    Arrival = getClientPreference (client_number,ARRIVAL);
    Depart = getClientPreference (client_number,DEPARTURE)
    for (I = Arrival to Depart) {
        Retract(TT, client_number);
        Enter(SS, client_number);
    }
}
```

2.3.1.5 Strategy H4: “Extreme filler”

We thought of an extreme idea which could be employed if we have not made the hotel fill we need and desperately need TT or SS hotel rooms. This should not be employed at less than minute 10 of the game since there is a high probability that we will end up overpaying. We do not see where this might apply for this section (customer by customer handling), but we think this strategy will show much promise in the other way of hotel auction handling (by total need, Section 2.3.2).

Hotel auction quotes have to be actioned if we want to keep winning the auction with an ask price one bid point higher. If the final quote issued by an auction suggests that we may not for some reason win the auction, we could add a large float number into the bid to finally submit an inflated bid for a quantity within the first 16 places of the order book. We think that the quantity associated with such an inflated bid should not exceed 9 for the following reasons.

- If we bid for 15 rooms and another agent is also playing something similar to extreme filler, we end up paying a *hyper-inflated* price.
- The chance that the rest of the quantities further down the list from 9 will descend into more average prices is worth relying on.

This strategy is not a fully fledged method, merely an idea which is implemented in a certain environment only. It is a cruder version of HT3 whose description follows later in this paper.

2.3.1.6 Algorithm H5: “Highest Number Wins for Two Packages”

We looked into a simple calculated solution to help us make a decision in *one timeframe only* about which of TT or SS to choose for a given package. Figure 4 below illustrates the simple general form and extreme examples. It is simply based on incoming cash (client bonus money) and outgoings - the total cost at ask price. The comparison at the end is similar to looking at a *net* sum on a balance sheet.

IN: HotelQuote, TTBonus

OUT: A reaction to this single quote changing
 ChooseSSpackage = x | ChooseTTpackage = y

ALGORITHM:

- 1) Obtain this customer’s details, then request entire set of prices as follows:

$$\text{sum} (\text{foreach}(\text{day}) \{ \text{negative}(\text{getaskprice}.\text{SS}) \}) = x$$

$$(\text{sum} (\text{foreach}(\text{day}) \{ (\text{negative}(\text{getaskprice}.\text{TT}) \}) + \text{TTBonus}) = y$$

- 2) Return max (x | y)

Figure 4 – TT and SS package comparator

2.3.1.7 Strategy H6: “Strategy Omnibus”

The process of “reallocating” a package by integrating all that we have thus far might best be illustrated with an example diagram.

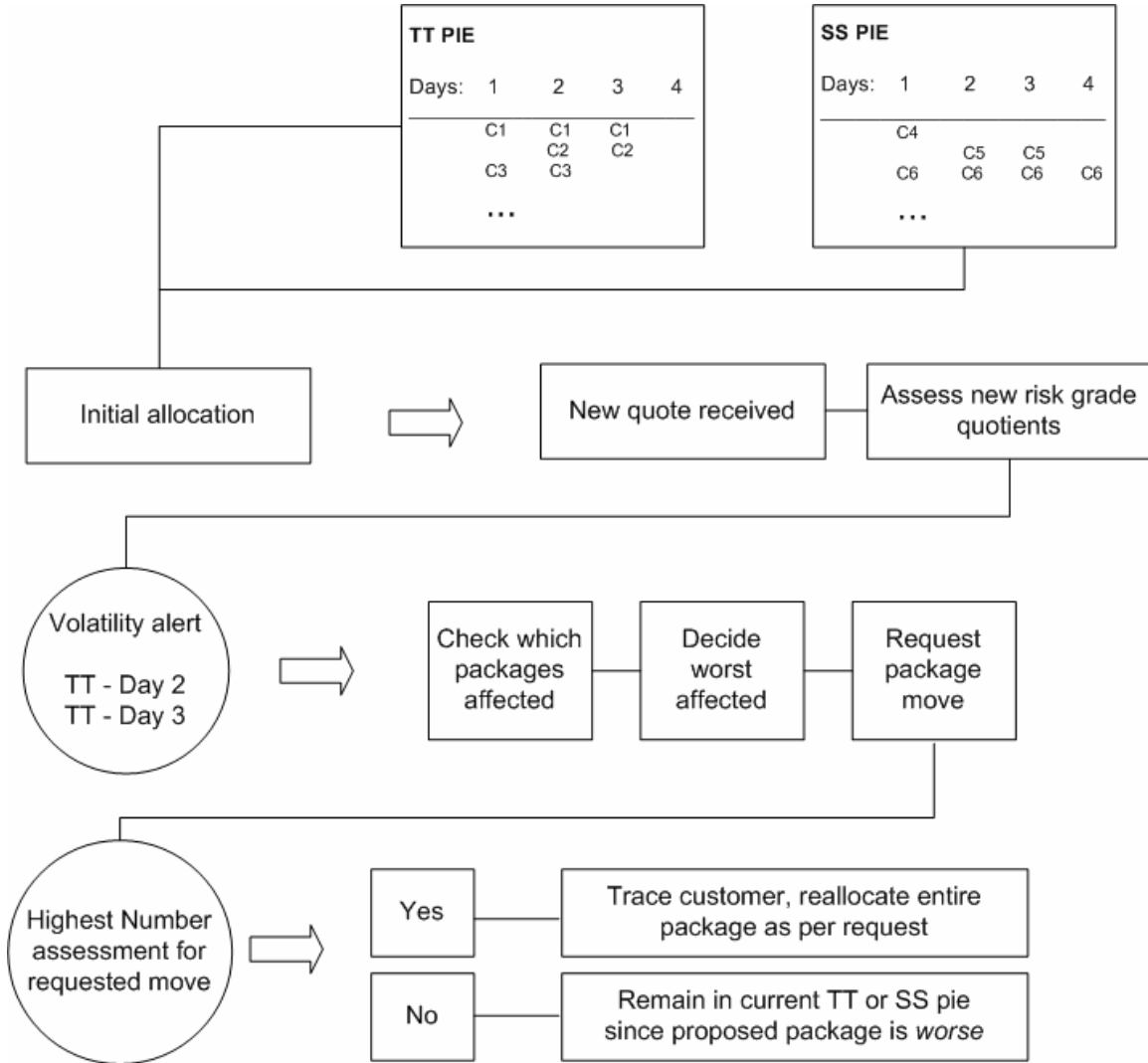


Figure 5 – Multiple strategies and algorithms in potentially one process

2.3.2 Total Need Strategy SET

The total need strategy set gives us the ability to create an array of 28 objects, each holding details that include how much (pre-decided) we need to bid for each individual auction. To find how this array is populated, please visit “Implementation” – Section 3.2.1. The decision to put customers into certain hotels for their stay is made in advance.

2.3.2.1 Strategy HT1: “Parameter decision”

At the beginning, we check each customer’s value of HotelBonus and see if it is above or below our parameter. Hotel bonuses range from 50-150 and we anticipate setting our parameter to the average. This means that each customer’s is allocated according to whether they are likely TT profitable or not, and the ones above our parameter price are “worth it”.

2.3.2.2 Strategy HT2: “Parameter Decision with reallocation”

This is a rather more complicated affair which in practice is beyond our resources to implement. It involves knowing that a current *total* standing bid for a TT day is not worth pursuing. We have to track down which of our customers is involved in this day before re-allocating, and shift each of their *whole* hotel package over to a better set of auctions. We have to do the whole package since customers must stay in one hotel throughout, not just for this current day that we are re-allocating.

Reallocating them subsequently uses a H5-type algorithm to decide where to reallocate to for each of their days and is further done on each customer that we have now removed from this TT day. It is very messy if we are implementing total needs only. How do we know for example which customers have rooms on TT Day 4 if we decide to remove 3 out of the total HQW (Hypothetical Quantity Won) of 5 in this auction? We need to track each of them down and change each of their packages to SS hotels.

2.3.2.3 Strategy HT3: “High quantities gives license for greater risk”

We note that a complete package is not created if a single day we need for a hotel is not won. A customer can be staying Days 1 to 5, but if we do not buy them Day 4, the package is not feasible. Therefore, we must make sure that when a total *quantity* we have going in an auction is >2 , we take active steps to win that auction, because it might break the package of more than two entire clients.

We can work out a factor which escalates the bid points added to our bid dependent on:

- Quantity we have to win (the greater the quantity, the more we must bid extra)
- The time of the game

This is a greedy strategy which gives importance to auctions for hotel days where we have a high quantity that we require. This strategy the previously mentioned “Extreme Filler” dependent on the above factors, and is one the best ways we have come across to use this idea.

A possible relation could be

```
public float get_boost_price(int need) {
    float static_boost = 85;
    float time_boost = (agent.getGameTime() / 72000);
    float quantity_boost = (need/2)*(need/2);
    float boost = time_boost * quantity_boost * static_boost;
    return boost;
}
```

Furthermore, the addition of higher numbers is scaled – it gets higher with quantity and time, so we quietly become more aggressive if we badly need to win this day. From experience, we see that this will be useful since we do not want to be bidding conservatively when we have a large quantity we really must win. Not winning this day/auction total would mean breaking entire packages, resulting in a large loss of entire packages which suddenly become unfeasible.

2.4 Entertainment Auctions

Entertainment auctions are the most complex affair in the game, since the complexity seems to be increasing as we go from flights to hotels to entertainment. They are double auctions where we have to obtain unique entertainment tickets for each of our clients' packages. The last night does not need any entertainment. The maximum number of entertainment tickets for which we get utility is 3, as there are 3 entertainment types.

We foresee two separate approaches to this problem, where the first may lead to the second if resources permit.

- 1) Of our initial allocation of 12, we need to check which ones of these 12 are going to match to our clients, or in other words, which are actually useful to us.
- 2) We need to then obtain more tickets to make further fills up to 3 and also sell the useless tickets we matched in (1).

We confirmed a fact of some importance from *Walverine* (2003:pp9) (quoting a result from *livingagents* (Fritshi and Dorer, 2002)) if we decide to do speculative dealing – the average ticket price seems to be around 80, and our local experiments proved this. In addition, less congested days are quoted to be Day 1 and 4 while more congested days are Days 2 and 3. *Congested* probably means a more liquid market.

2.4.1 Strategy E1: “Allocate and leave”

This is the tickets left as they were allocated. Future strategies require us to create an easy to use data structure. Such an object is described in “Implementation” – Section 3.2.2.

2.4.2 Strategy E1.1: “Allocate and sell remainder”

Of our 12 entertainment tickets, we need to check each ticket to see if any clients will use the ticket. If not, then we place this ticket “for sale”. We allocate what we can use and sell what we don't need. The price at which we offer for sale is around 60-80 to prevent other agents getting an advantage, as it's better to hold useless tickets than to sell them cheap to other agents and contribute to their profits. The 60-80 choice derives from *Walverine*'s analysis of average ticket prices.

2.4.3 Strategy E2: “Allocate, sell remainder and purchase the useful”

We do the same as E1.1, but we now place hopeful bids (See E5 – Section 2.4.6) for the ones we would like, fixed at a price.

We place the tickets we don't need for sale but also calculate (possibly into another data structure) a list of what we require, bearing in mind that we need a maximum of three unique tickets for each client's stay. For each item in this list, we place a hopeful bid (E4) since it is not crucial that we get these entertainment tickets, but it would be nice to get them cheaply. We hope that *dumb agents* who want to sell their tickets might well decide to sell them to us, in which case we have obtained what we need very cheaply.

At best we get entertainment we need at bargain prices. At worst we have tried to "poke" the auction for some bargains without success.

2.4.4 Strategy E3: "Allocate, sell remainder and optimal purchase"

This strategy takes our entertainment dealer to its full capacity.

It implements E2, but instead of the last step of placing "hopeful bids" for the entertainment it still needs, it does the checking to see the preference price of each ticket type of each client, and places each buy offer at a parameter percentage (markup) below the price that clients are willing to pay. We illustrate with an example.

```
Global MARKUP_Percentage = 30%

//Ticket required - e1, Day 4, 40 [Type, Day, Client Pref]

Float BuyOffer.thisclient = (40-(MARKUP_Percentage));

SubmitBid (ticket, BuyOffer);
```

In this way, agents find our offers to buy more attractive and we get more tickets to make our fill. Despite getting more tickets to make our fill, we also profit on them since we paid the parameter percentage *less* for them than the amount the client is paying us.

Finally on the selling front, we sell our remaindered tickets (the useless remnants of the initial 12) at more attractive prices which are highly likely to get accepted.

2.4.5 Strategy E4: "E3 with parametric speculative buying"

Our speculative buying is focused primarily in the minutes when statistically the prices of entertainment tickets are at their lowest. We then attempt to sell these in a fashion which is intrinsically separated from the transactions and decisions for our *own* allocation.

We implement E3 but integrate a different module into the quote handler which fills a "shopping basket" at certain times using facts learned from papers. All items in the shopping basket are for sale at all the times, and their going prices are updated in our internal data structure as well as to the auctions.

A disadvantage of taking a short position from tickets we don't own will result in a 200 per ticket penalty being assessed at the end of the game. The only advantage here is a price difference gained by profit in the buy/sell price. We think that this may not work since we don't anticipate our peer agents to have this level of sophistication. Some agents may not even deal entertainment – it is optional after all.

2.4.6 Strategy E5: “Hopeful bids”

See Section 2.5.1. We have identified that “dumb” agents might be willing to accept any price standing to buy or sell their entertainment tickets, so if we submit extreme bids into our auctions at low offers, there is a slim chance that we could fill the order.

Our approach here is to identify all the tickets that we need of all the types that we need, for all the days we need them – and then to submit very low buy offers for these into the auctions.

2.5 Interactions and Scoring

2.5.1 Hopeful bids

We note that bids that are not matched immediately remain in an auction as standing bids, and this is the case in all types of auction. This opens the possibility of using certain auctions to make our fill on one of our utilities based on the low probability that we will strike a bargain. In practice this cannot be achieved for hotel auctions, and is ineffective for flight auctions since

- flight quotes are determined and non-negotiable
- hotel bids have to be at (ask price + 1)

But for entertainment auctions, which does not make or break a *package*, the possibility does exist in that placing a hopeful bid into these might actually be filled by another “dumb agent” that looks to get any price for a given entertainment ticket – giving us a real opportunity to buy cheap.

2.5.2 Quantity vs. price

We were interested in the possibility of submitting single bids for single items in an auction, and “spreading the range” of our costs. We note that for hotel auctions, all buyers pay the 16th highest bid; therefore, there is no advantage in submitting for example 3 bids for 1 room as opposed to 1 bid for 3 rooms, unless we are at the bottom end of the winner’s queue, in which case, 3 bids for 1 room could be at different prices. In practice, this off-chance is difficult to implement.

2.5.3 Utility scoring

Each of our 8 customers provides us a utility based on the following relation:

$$\text{Utility} = 1000 - \text{TravelPenalty} + \text{HotelBonus} + \text{FunBonus}$$

Where $\text{TravelPenalty} = 100 * (|\text{ArrivalDate} - \text{PlannedArrivalDate}|) + (|\text{DepartureDate} - \text{PlannedDepartureDate}|)$

Where HotelBonus is XX points for each night in TT

Where FunBonus is the sum of the reservation values of all the entertainment received

On further consideration, we note that if our objective was to maximise a customer’s utility, we are faced with the constraint of cost, which will increase as we attempt to gain a higher utility score. Our choices can be thus:

- Initially, attempt to maximise utility at any cost and then formulate cost-saving strategies.

- Target a “percentage satisfaction” of utility score if costs prove too high to maintain a top position relative to our peer agents.
- Take a risk – focus in majority on cost and formulate strategies to minimise them for each auction type to make our fill.

We hypothesise that a successful solution may have the following characteristics.

Implement “TOTAL NEEDS” for hotels

Attempt to buy flights early – F1

HT1 with a bias for cheap hotels & HT3 possibly being used as a dynamic calculation

E2 for the entertainments possibly using a refined version of E5

We aim to construct such a solution since:

- We are unsure of the trading strategies of our peer agents. A maximum utility strategy may cost us dearly if the calibre of these agents is very high, and “taking a risk” may cost us dearly in travel penalties or failed fills for required hotels.
- It should ensure a score which is at least average and will not skew to the low end.
- It should be adaptable against other more extreme agents that employ maximum utility or risk-taking strategies.

2.5.4 Unit pricing

We investigated the possibility of pricing each auction, irrespective of type in unit types and upper and lower bounds for what we “should pay” per unit. This option was rejected at implementation since the unit types we would have to deal with would not be distinct enough when compared to risk grade ratings (Section 2.3.4) for say, hotels. We would also need three unit prices, which would have to rely on historical data.

We obtained an idea from an ad-hoc “futures” trading strategy developed for the US Futures and Commodities market in the 1970’s called “Turtle Trading” – see *Original Turtles* (pp 17 - 22). The complete strategy, when *always followed* was (apparently) very successful, with single successful trades forming much of the year’s profit from commodity positions. The synopsis of the idea was to use “buy/sell” indicators based on a “breakout” from the moving average. Traders would take a long or short position based on the direction of the breakout. However, they took their position in real money based on a variable *scale* which was always divided into units. This meant that their exposure was perfectly balanced and normalised to prevailing market conditions. The aspect of this work which was interesting to us was the unit pricing and exposure ideas they adopted. The use of this system required highly liquid markets. We felt the unit price idea was interesting but not applicable.

3 Implementation

3.1 Introduction

Our implementation has been carefully constructed to ensure maximum readability and follows general Java conventions. We decided to construct a few of our own objects to aid various internal record-keeping functions. Section 5.3 (in the Appendix) details our source code. Methods have been packaged into “important” and supporting “library” functions which fulfil precise purposes with decreasing responsibility as the file is traversed.

3.2 Good programming techniques

Our implementation was started from scratch. We organised the methods and objects precisely so that as much re-use and modification potential was generated from the code. As a result, evolving an agent was much simpler than creating the groundwork. Method names and variables names follow consistent norms and rather than provide an exhaustive summary of methods here, the reader is referred to comments in the code. Variables names have been separated from global constants (which are usually all uppercase in our code).

3.2.1 Details Object (Flights/Hotels)

We invented an object called Details, of which there are 28 instances for each of the auctions. The Details Object is stored in an array, and has its' own specific getter/setter methods. We found this much easier to implement than a native array. The advantage of this object is that we know on an auction to auction basis (not a customer to customer basis) our total requirements within that auction – See introduction of Section 2.3.

- For flight objects (8), we note down how many incoming flights and outgoing flights our customers prefer, and then find out which auctions are holding these. We then submit precise bids into each auction for the *total* quantity we require, avoiding having to check flight preferences again.
- For hotel objects (8), we make decisions about which clients will go for which hotels based on factors such as those in Section 2.3.2.3. Then a total requirement needed per day per TT/SS auction is added up and logged in the array as NEED. NEEDS are dynamically set and unset through the game.

The actual construct that held these objects was an array, and the decision making processes which led to a calculation of our needs were conducted as per Figures 6, 7 and

8 for each auction. There were 28 Details Objects with all the details that we required for each auction in existence.

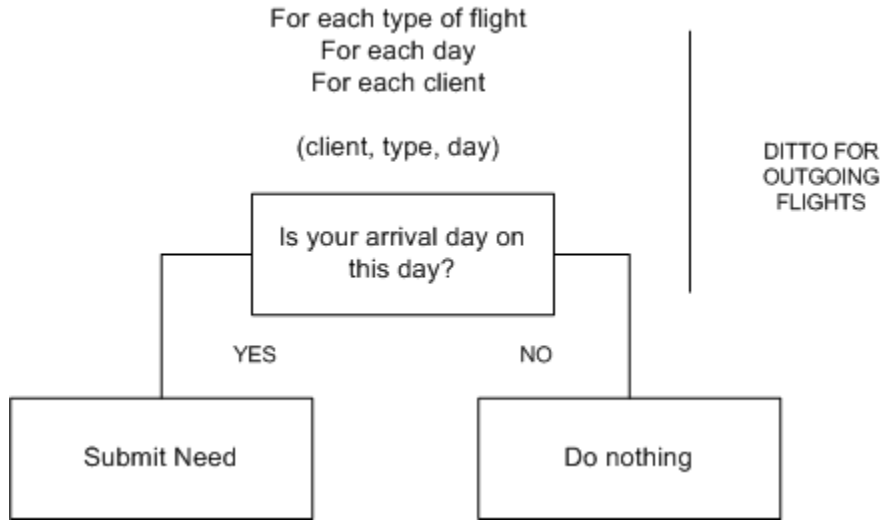


Figure 6 – Needs model for flights

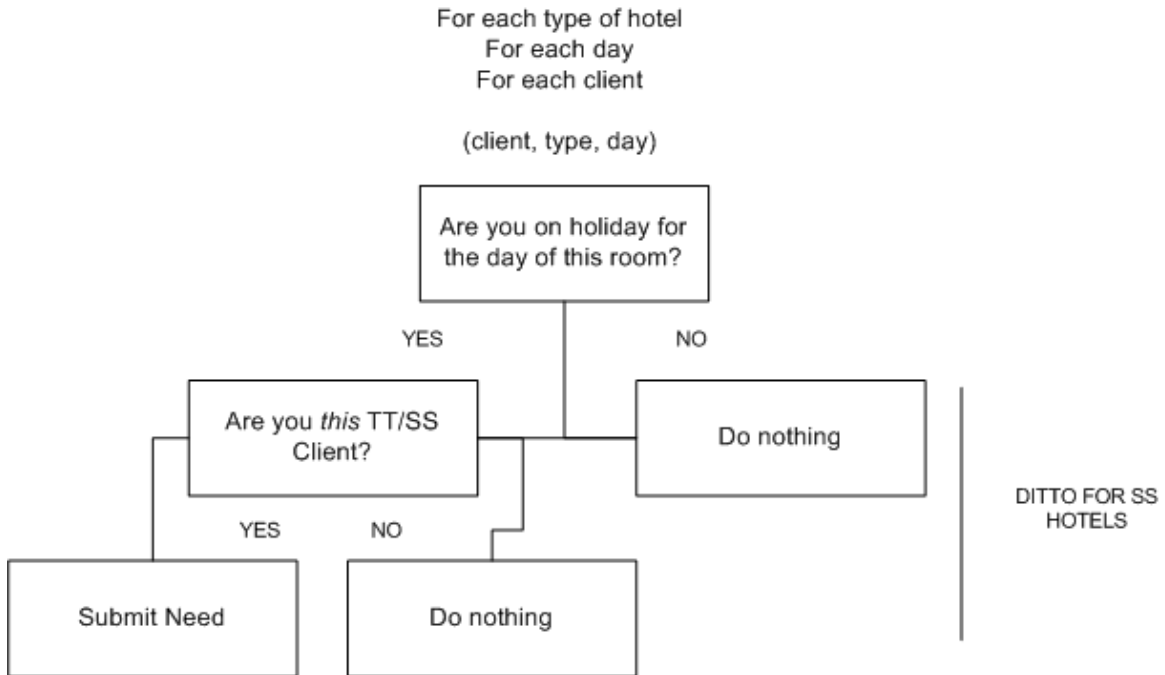


Figure 7 – Needs model for hotels

>> Implementation : Good programming techniques

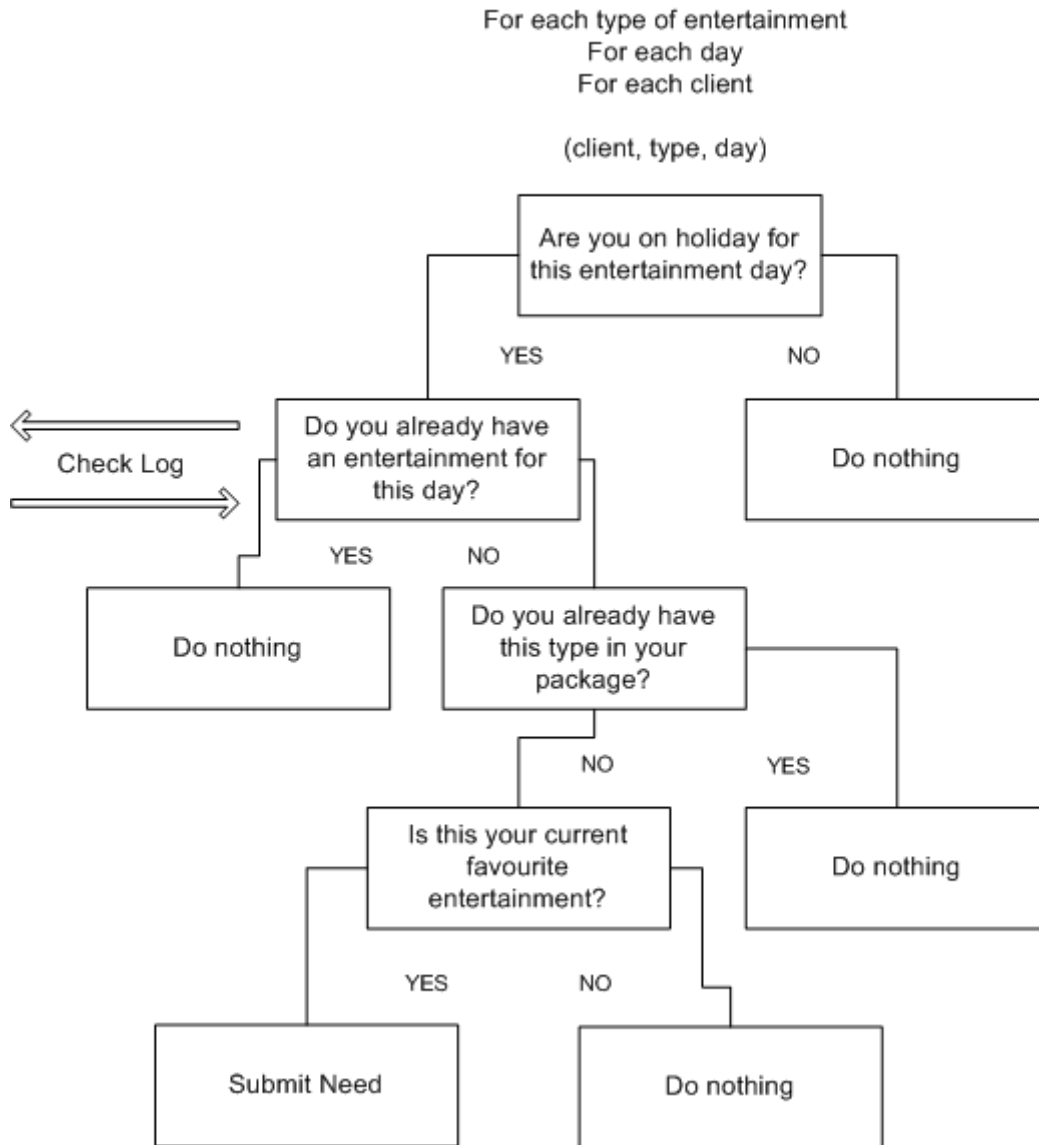


Figure 8 – Needs model for Entertainments

3.2.2 Entertainment Object

The Entertainment Auctions presented us with the challenge of “noting what we have allocated so far” and “flagging” what we wish to allocate as yet. This *log* object was intended to support our relatively complicated flowchart model for making entertainment decisions, which turned out to be more difficult to code in practice. However, our testing was based on each decision in the if-else statements and we are confident to a reasonable degree that the filling of our true needs – which is what we want to sell and what we need has been *precisely calculated*.

>> Implementation : Good programming techniques

3.3 Agent versioning system

Our agents evolved from alpha versions. Initially, we constructed simple strategy agents to familiarise ourselves with the game and then copied and incremented these version with new code to map to one of our defined strategies as given in this document.

3.4 Choosing and testing agent versions

Testing our agent version was the only indicator that they were working to plan and doing well. Extensive use of log outputs was made to resolve Java NullPointerException and Array OutofBounds exceptions. We attempted to put our agent versions against each other in one game, to see which came out top. The disadvantage is that the agents involved are playing to our own defined strategies, not real ones from our peers (except on certain occasions). The magnitude of the loss of Dummy Agents was a fair indicator as to how we performed relative to them. We were later able to test against our peers and last year's old agents. The final testing was severely hampered by the servers being offline or being congested and this delayed our hypotheses when we tweaked certain methods or parameters.

3.5 Particular implementation constraints

A critical analysis of our implementation follows, together with justification for our choices of certain parameters.

- We ought to have pursued the fourth level check for entertainment needs. This final check, which we called `is_favourite()` in concept, would reject the current log entry from being appended if the type was not the favourite. The favourite type is the highest customer preference ticket. We initially implemented a complex object using an array inside a class but this led to implementation issues when it came to a simple sort using Java comparators as flags. This final check would have all that was required to fill our master Details array with EXACTLY the correct entertainments we wanted, filtered down to those we also wanted first. This is so that a client, going, say, just for 1 day is only allocated the highest profitable ticket that we buy the client.
- Entertainment tickets are sold at 60 and bought at 100, choices which we decided upon from experience and reading.
- The sorting parameter (i.e. 120) for hotel preferences was chosen since we wanted a *skewed* bias to allocating cheap hotels. We discussed this at length and noted that good hotels can get very expensive to pursue if we are not using reallocation.

4 Improvements

The reader is referred to the Literature Review conducted in Section 1.2.

We would look to the winning agents of the worldwide competition for ideas on improvement. These are the points we have gleaned out from our Review:

- *Learn*
Winning agents are adaptive and change their strategies on re-assessment of the market.
- *Optimal packages*
Some of the most successful agents create optimal packages dynamically and attempt to change their bidding strategy and holdings.
- *Focus on the entertainment market*
World class agents may find making money easier with other world class agents if their entertainment strategies are more refined. We think that flights are without negotiation, hotels are a matter of “saving” money, and entertainment is where money is made. We think this is so because the winner of TAC 2002 (*whitebear*) made substantial headway with making entertainment profits, as quoted in *Walverine* (2003:pp19).
- *Adaptive agents*
A good example is Southampton TAC (Minghua He, Nicolas Jennings) and may be a catalyst to create simple algorithms but dynamic strategy changes in-game.
- *Hotels – attempt to predict prices*
As difficult as this may sound, attempting to predict hotel prices allows one to know which auction to best bid within. We thought of this in our own small way with the risk grade per auction pair.

5 Appendix

5.1 Bibliography

Clemens Fritschi, Klaus Dorer. Agent-oriented software engineering for successful TAC participation. International Conference on Autonomous Agents. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, 2002.

Michael P. Wellman, Daniel M. Reeves, Kevin M. Lochner, and Yevgeniy Vorobeychik. Price prediction in a trading agent competition. University of Michigan, 2002.

Michael P. Wellman, Amy Greenwald, Peter Stone, and Peter R. Wurman, The 2001 Trading Agent Competition. *Electronic Markets* **13**(1), 2003. Earlier version appeared in *Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 935-941, Edmonton, 2002.

Minghua He and Nicholas R. Jennings. SouthamptonTAC: An adaptive autonomous trading agent. To appear, *ACM Transactions on Internet Technology*.

Original Turtles (Trading Rules) – [WWW]
<http://www.originalturtles.org/docs/turtlerules.pdf> (13 November 2003)

Shih-Fen Cheng, et al. Walverine: A Walrasian trading agent. Revised version of a paper to appear in *Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Melbourne, 2003.

Stone P., Littman L., Singh S., and Kearns M. ATTac-2000: An Adaptive Autonomous Bidding Agent. *Proceedings of the 5th Int. Conference on Autonomous Agents 2001*.

5.2 Personal Reports

Our personal report forms follow this page.

5.3 Source Code for Submission

The pages that follow display the code for our submission.