ABSTRACT

JETLY, GAURAV. A Multi-Agent Simulation of the Pharmaceutical Supply Chain. (Under the direction of Russell E. King.)

The Pharmaceutical Supply Chain is composed of multiple firms interacting to produce and distribute drugs in an uncertain environment. Each supply chain faces tremendous pressure to continuously adapt to changing industry structure and developments. The industry is marked with different kinds of unpredictability related to R&D success, demand, regulatory environment, new entrants, and exclusivity. These changes are frequent and unpredictable and thus adaptation at each level is required to sustain in the industry. In this work, we focus on developing a multi agent simulation of supply chains associated with the pharmaceutical industry. We demonstrate that the operating norms of a particular industry can be accurately represented to create an industry specific model capable of tracing big pharma's evolution. The model is initialized based on 1982 financial data with 30 manufacturers, 60 suppliers and 60 distributors. Three types of drugs: blockbusters, medium, and small; with a twelve year lognormal product life cycle are released by manufacturers based on the random functions normed to FDA and Congressional Budget Office data. Each quarter the distributors bid for future market share of the released products and suppliers bid based on the lowest acceptable margins. Mergers and acquisitions are allowed at each level based on assets and perceived profitability. Each replication is run for a period equivalent to 39 years. The model is validated for various industry parameters.

A Multi-Agent Simulation of the Pharmaceutical Supply Chain

by
Gaurav Jetly

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Industrial Engineering

Raleigh, North Carolina

2010

APPROVED BY:

BIOGRAPHY

I was born on November 22nd, 1982 in Haryana, India. I received my Bachelors of Technology degree in Metallurgical and Materials Engineering from Indian Institute of Technology-Chennai in 2006. After my graduation, I worked as Quality Assurance Engineer in Sterling Commerce for one and a half year. Thereafter, I attended North Carolina State University for my Masters of Science degree in Industrial Engineering. During my Masters, I worked as a Research Assistant under the supervision of Dr. Christian Rossetti and Dr. Robert Handfield in College of Business Administration.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to Dr. Christian Rossetti, co-chair of my committee, who gave me this opportunity to learn and do research under his guidance. I am also thankful to him for his patience and excellent guidance throughout my research work and for sharing his vast knowledge with me. Dr. Rossetti has always been very helpful, motivating, and encouraging which enabled me to enhance my understanding of the subject.

I am also grateful to Dr. Robert Handfield who involved me in his research group and also supported and encouraged me at every stage of the project. His words of encouragement kept me in high spirits through out my research work.

I am also thankful to Dr. Kay and Dr. King. Dr. Kay has helped me in becoming a better programmer. I felt a great difference in my programming abilities after completing my course in Logistics Engineering. Dr. King helped me identify my career goals and also guided me in achieving those goals. I am also thankful to Dr. King and Dr. Kay for a thorough review of my thesis.

I would also like to thank all the faculty members at North Carolina State University who taught me different courses.

Finally, I am thankful to my family especially my parents who have always been with me in the toughest times of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter1

Introduction

1.1 Background

The movement of pharmaceutical product from manufacturers to its consumer involves several players. These are suppliers, manufacturers, distributors, pharmaceutical benefit managers, health insurance companies, and pharmacies/retailers. Competition and uncertainty at each of these levels has made the pharmaceutical supply chain one of the most dynamic sectors of United States economy. Besides being the biggest market of pharmaceutical drugs, the United States is also a pioneer in terms of discovery and development of new drugs. Out of the top 20 largest pharmaceutical companies, 12 are based in the USA. A large contributor to the dynamics of the pharmaceutical industry is new product development. The spending on research and development (R&D) relative to sales revenue is highest in the pharmaceutical industry –spending increased from $5.5 Billion in 1980 to more than $17 Billion in 2003 as per CBO report. Despite large increases in drug R&D spending over the last 25 years, there is little increase in the number of new drugs entering the market each year.

The pharmaceutical industry is a research intensive industry. The drug development process has different types of uncertainties associated with it, like low success rate and uncertain development times. This makes it a capital intensive process. Even after successful drug development, there is an uncertainty associated with the demand for the drug. The drug demand is almost negligible at the end of patent life due to competition from the generics.

Further, there are drug specific contracts between suppliers, manufacturers and distributors that give rise to complex, competitive and dynamic relationships in the pharmaceutical supply chain. The industry is also marked with large number of mergers and acquisitions (M&A) over last 30 years. At the manufacturer level M&A were primarily a response to perceived lower than expected earnings. Distributors merged to expand their market and to reduce operational cost. At the distributor level the consolidation resulted in three large players (AmerisourceBergen, Cardinal, and Mckesson) sharing more than 90 percent of the drug distribution industry.

The dynamics and evolution of the pharmaceutical supply chain is governed by a number of forces and their interaction. In the literature these factors have been studied, but most of the studies focused on individual factors such as R&D, profitability, costs etc. and do not address the role of complex and dynamic relationships that exist between the members of the pharmaceutical supply chain. Also, the existing research on pharmaceutical industry does not consider the impact of competition and the supply chain structure on the profitability of industry.

To offer insights into these relationships as well as expand the body of simulation research addressing industry specific supply chains, we developed a multi-agent model of the PSC. Since the pharmaceutical industry involves different players with different roles and individual objectives, agent-based modeling fits well to address the research questions in consideration (Swaminathan, Smith, and Sadeh, 1998).

1.2 Research Questions

The objective of this study is to develop a model which includes all the governing factors along with the complex, competitive and dynamic relationships that exist in the pharmaceutical supply chain. Due to the importance of the pharmaceutical industry and the dynamic interplay between manufacturing systems, product lifecycle, and government regulation, this research addresses following questions:

1. How should various forces and agents (Manufacturers, Suppliers, and Distributors) in the pharmaceutical supply chain be modeled?
2. Do certain structures result in superior supply chain financial performance?

1.3 Layout of the Thesis

In the next section of the thesis, some applications of multi-agent methodology in different domains are summarized. Chapter 3 is focused on the methodology adopted to develop the model. The methodology section is further divided into subsections explaining the different stages of model development.

Chapter 4 explains how the model was validated using industry specific data obtained from Compustat, Congressional Budget Office, and the FDA and industry reports. Various non-parametric and parametric techniques were used to validate the model.

Chapter 5 includes some results of the simulation and the interpretation of results. In the last section we summarize the potential applications of our model and the further research that follows this study.

Chapter2

Literature Review

2.1 Pharmaceutical Industry

There have been a number of studies focused on different areas in pharmaceutical industry. Each of these studies looked at different aspects of the industry and followed different methodology. The empirical research addresses three intersecting areas: profitability, research intensity, and mergers and acquisitions.

The pharmaceutical industry is considered to be the most profitable industry as measured by returns on investment in R&D. Several studies have investigated returns to R&D expenditure (Grabowski and Vernon, 2003). Most of the studies concluded that the returns were much higher than other industries due to extensive research and innovation. Other studies focused on the efficiency of new product development and if there is a relationship between firm size and research productivity (Henderson and Cockburn, 1996). DiMasi, Hansen and Grabowski (2003) have extensively examined time and cost of new drug development. They derived the costs in different stages of drug development using the phase transition probabilities for an investigational new drug. The authors estimated that out of pocket cost was $403 million per approved drug and total cost of capital was $ 802 million per approved drug.

The pharmaceutical industry is also a research intensive industry. As per Congressional Budget Office (CBO) report (2006), the R&D investment, relative to their sales, is highest in this industry. The pharmaceutical industry R&D intensity remained close to 15 percent since 1980. The reason for an increase in R&D investment is an increase in sales revenue from pharmaceutical products. The drug development process involves various stages and overall has a very low success rate. A large number of new molecular entities need to be tested by manufacturers to discover the molecular entities with a potential to clear the clinical trials. On an average, 5 out of every 5000 compounds tested clear the preclinical stage. Due to low success rate, large scale research efforts are required to discover a new drug.

The industry is also marked by extensive consolidation. Danzon, Epstein, and Nicholson (2003) found that large firms merge mainly because of excess capacity as a result of gaps in product pipeline or anticipated patent expiration. The authors measured excess capacity using a number of variables such as Tobin’s q, percentage of drugs which were launched 9-14 years previously, lagged change in sales, and lagged change in operating expense. Large firms with relatively low expected earnings acquire other firms either to reduce the operating expenses or to fill the gaps in their pipeline. Their R&D pipeline is financed by their current sales. Small firms on the other hand depend on external financing to support their R&D and any setback to their R&D program reduces their market value and makes it difficult for them to raise more capital. Therefore, small firms merge mainly because of financial trouble. Higgins, and Rodriguez (2005) determined that firms that suffer a decline in their product development pipeline, or which experienced a decline in their current sales, were primarily involved in acquisition as a way to replenish their product pipeline. The above studies were focused on different factors/parameters in the industry like R&D productivity, research intensity, returns to R&D investment, and consolidation.

Extrapolating from the previously mentioned empirical results is questionable due to sample size and multiple exogenous shocks. In addition, empirical research on supply chain relationships is lacking. The nature and effects relationships between manufacturers, suppliers, and distributors are relatively untouched by management researchers. This is understandable given the difficulty in collecting detailed data on contracts, R&D, and M&A from three members of a supply chain.

2.2 Multi-Agent Simulation (MAS)

MAS have been used in the past to model many different types of problems considering systems of differing size and structure. A firm level model in the pharmaceutical industry was developed by Solo and Patch (2004). They developed a hybrid simulation approach which incorporated agent-based, discrete event and continuous equation approach to model the structure of pharmaceutical enterprise portfolio. This model can be used for optimal allocation of finite resources of a firm. The researchers modeled a new product development (NPD) pipeline structure, active products in market and limited resources of an enterprise, which act as a constraint. The NPD pipeline structure is a model of compounds which are in the R&D pipeline. Each compound undergoes different phases of testing with a probability of passing that stage. They modeled a probability decision tree with all the stages of product development and different paths a compound may take during the trial stages. They also modeled a decision rule to intentionally discard a compound after it has cleared certain stage or to dynamically add a new compound to the simulation from outside through merger with other company. Their objective was to develop a tool which can be used to allocate limited resources of an enterprise across the available product investment opportunities to achieve the best possible returns. Therefore their study was a firm specific study and does not include other players in the industry like suppliers, distributors and other manufacturers.

Other studies examined the micro level problems determining the impact of different policies on the efficiency of an organization. Siebers et al. (2007) used MAS to understand the relationship between human resource management practices and retail productivity. In their study, the authors determined the effect of employee learning on the overall efficiency of the retail store. Other studies focused on mid level problems in which a number of companies coordinate together to determine an efficient operations plan. Barbuceanu, Teigen, and Fox (1997) developed an agent and coordination technology which can be used to model the distributed supply chains. They simulated a vertically integrated supply chain of personnel computer manufacturer. The MAS included each plant operation (planning, material management, production, transportation etc) as an agent and these agents coordinate among themselves to fulfill customer orders. Fu, Piplani, De Souza, and Wu (2000) used an agent-based simulation methodology to examine collaborative inventory management in supply chain. They also validated their model based on simple personnel computer (PC) assembling case and proposed a theoretical framework for collaborative inventory management. The systems in the above literature involve cooperation and coordination between agents through sharing of information. While the above studies involved different supply chain entities, they lack the competition that prevail at each level in the industry and therefore does not help us understand the impact of this dynamic force on the supply chain relationship.

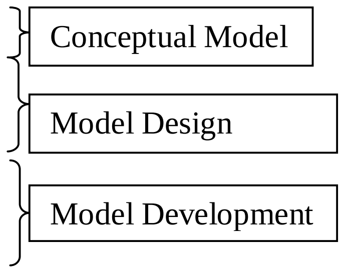
There are some other agent-based simulations which involve competition among agents for available resources. This kind of agent-based modeling was primarily used to model systems in which agents bid for a particular resource based on the accessible information (Wellman, Greenwald, Stone, 2007). Sadeh et al. (2003) incorporated this mechanism in their model in which bids from different supply chain entities were evaluated based on measurable cost and delivery dates. In another study, a web based multi-agent simulation supply chain management game was designed by Arunachalam and Sadeh (2005) in which 20 different teams from around the world participated with different strategies for their agent. In their game, PC assemblers bid on request for quotes from customers and the winner was responsible for procuring parts, assembling, and delivering PCs. While the above models include the competition and driving forces in the supply chain relationships, they do not include the innovation (R&D) and M&A in the industry. In pharmaceutical industry both innovation and mergers and acquisitions are important drivers and the model lacking any of them will give limited insight into the dynamics and evolution of the supply chain. Even though our model can be used by individual companies to understand different aspects of the supply chain, the primary objective of this thesis is to give key insights into the driving forces in the pharmaceutical industry. Therefore the model starts with a realistic number of agents which existed in the industry in 1982. Further this model includes all the key forces, trends and decision strategies (Rossetti, Handfield, and Dooley) that are practiced in the industry by different players.

Chapter 3

Methodology

In the early phase of this study, a conceptual model (Balci and Ormsby, 2007) was developed based on different industry reports and research findings. The conceptual model was used during simulation design and implementation stages. The Simulation model was developed in the following stages:

1. Identify the supply chain structure and interactions,
2. Perform data mining and analysis,
3. Develop algorithm and rules for the agents,
4. Develop the model using the Java programming language,
5. Perform model verification and validation



3.1 Identify the Supply Chain Structure and Interactions

The pharmaceutical supply chain involves multiple players with different roles. In our model, we considered three levels of the supply chain: suppliers, manufacturers, and distributors. These players, at different levels of supply chain, have different roles and therefore are modeled as different types of agent. There are two general types of interaction in pharmaceutical supply chain (refer to Figure 1): vertical interaction and horizontal interaction. Vertical interaction involves product specific contracts and bidding.

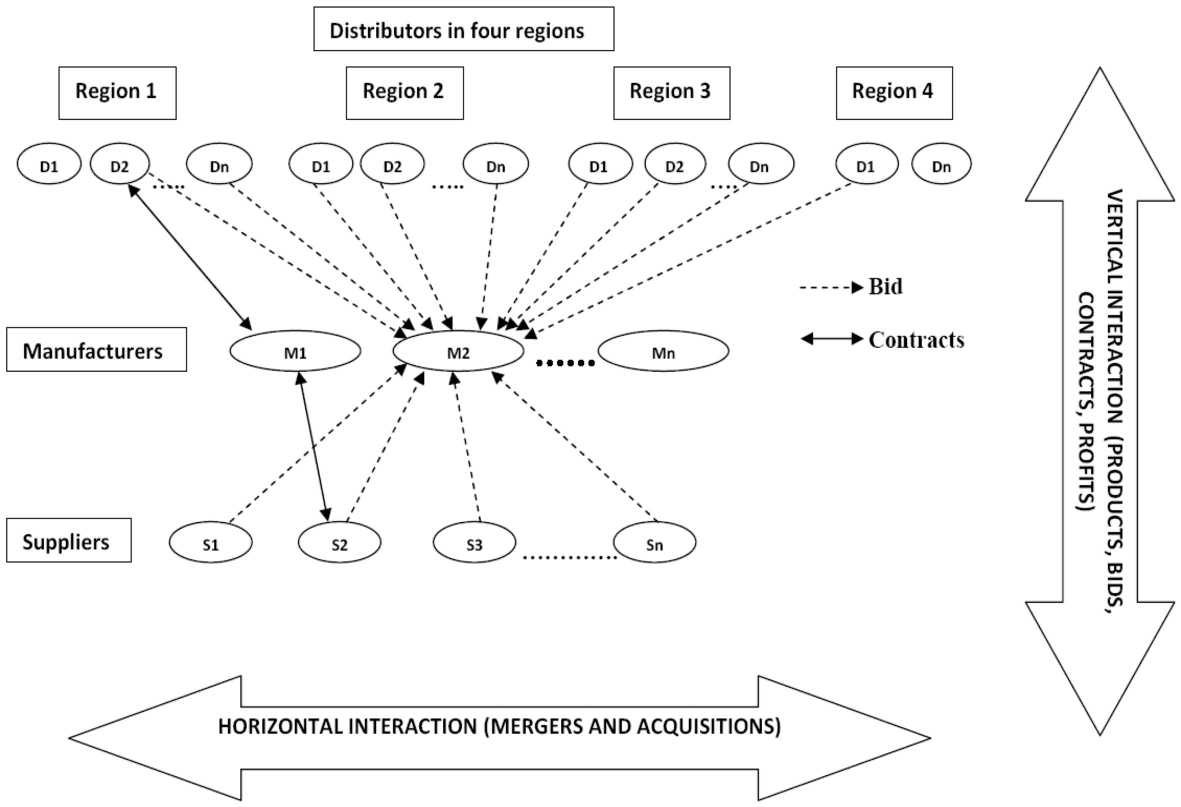

Distributors in four regions
Region 1
Region 2
Region 3
Region 4
D1
D2
Dn
D1
D2
Dn
D1
D2
Dn
D1
Dn
Bid
Contracts
Manufacturers
M1
M2
Mn
Suppliers
S1
S2
S3
Sn
VERTICAL INTERACTION (PRODUCTS, BIDS, CONTRACTS, PROFITS)
HORIZONTAL INTERACTION (MERGERS AND ACQUISITIONS)

Figure 1. Interaction in pharmaceutical supply chain (arrows represents bids for the new product).

Whenever the manufacturer releases a new drug, suppliers bid based on their lowest acceptable margins. Suppliers provide ingredients to the manufacturers. The U.S. consumer market is divided into four regions. The number of distributors and the product sales in each region are proportional to the population in that region. Distributors in each region bid for the exclusive distribution rights for a number of quarters based on estimated demand of the new product. At the end of the contract, other distributors in that region offer competing bids for future exclusive rights to distribute that drug. The objective of horizontal interaction at each level of supply chain is to identify candidates for M&A which are based on profitability, size and drugs.

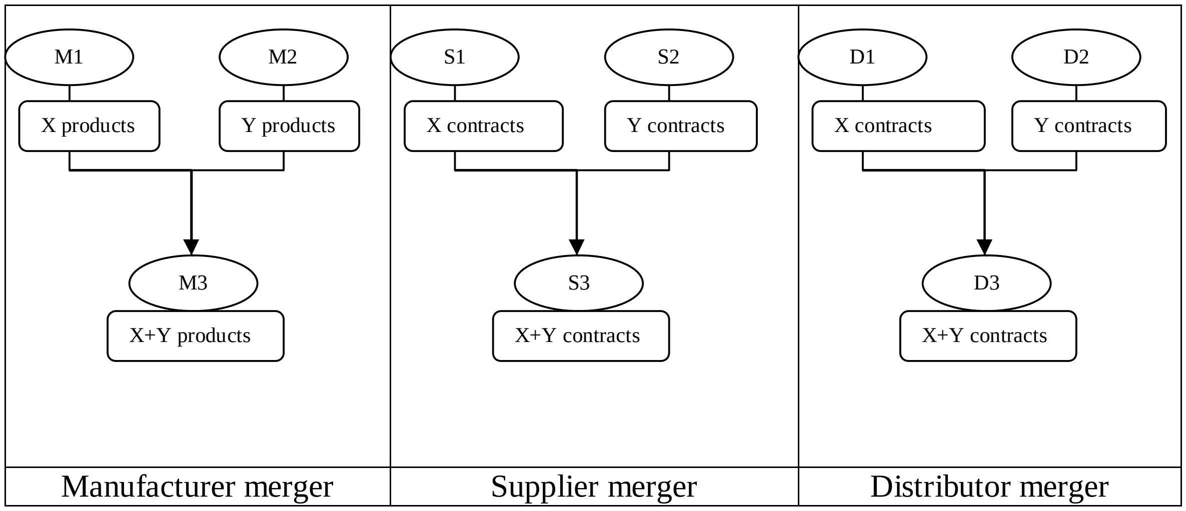


Figure 2. Mergers at each level of pharmaceutical supply chain.

3.2 Data Mining and Analysis

The data were extracted from several databases such as the COMPUSTAT database (2008), the FDA's Orange book (2008), several technical papers and reports. Different simulation parameters and rules were estimated using this data. Random scores, representing the firms' assets, are assigned to each type of player at the start of the simulation. Scores are based on a triangular distribution, TRIA (minimum, mode, maximum) that has been fitted to the log of the firms' total assets in their specific standard industrial code (SIC). The parameters are based on 1982 data from the COMPUSTAT database (2008). For manufacturers the parameters are: TRIA(4.20,7.63,9.00). For suppliers the parameters are TRIA(3.72, 7.15, 8.52). For distributors the parameters are: TRIA(2.00 , 3.10, 7.76). Each time a simulation starts, new starting scores for each type of agent are obtained from these distributions.

Each manufacturer has its own product pipeline under development. In our model, we used the drug development estimates from the findings of DiMasi et al. (2003). In their study, the product development was categorized into two stages: preclinical and clinical. The average drug development period is eight years. The preclinical stage varies for different drugs from 1-6 years. The clinical stage is subcategorized into stage1, stage2, and stage 3. In our model, preclinical stage and each subcategory of clinical stage is spread across 2 years. Based on DiMasi et al. (2003), estimates of drug development cost in each stage of preclinical and clinical trials and using the probabilities of success in each stage, we determined the cost of preclinical and clinical trials as well as the probability that each investigational new drug (IND) would advance to the next stage. The timeframes for each stage and the probabilities of success at the end of each stage are presented in Table 1 (DiMasi, Hansen, and Grabowski, 2003).

Table 1. Different stages of drug development (time and the probability of success)

	Preclinical	**Clinical Stage1**	**Clinical Stage 2**	**Clinical Stage 3**
Time period	2 years	2 years	2 years	2 years
Phase transition probability	0.0004(IND)	0.75	0.48	0.58
Cost per compound (millions of 1982 dollars)	0.0049	2.07	4.23	32.66

*http://www.fdareview.org/approval_process.shtml

Based on the phase transition probabilities, the overall cost (in each stage) of developing a drug successfully, and the number of starting compounds required for a successful product, a cost per compound was estimated in different stages of product development (refer to Figure 3).

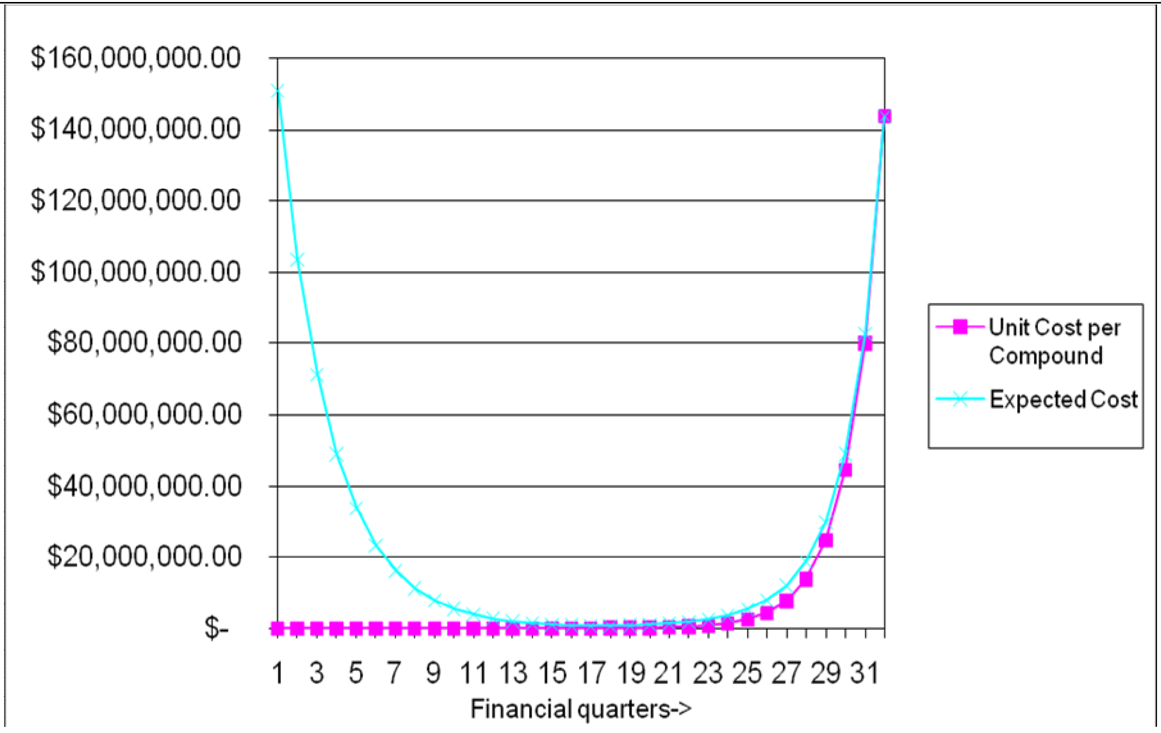


Figure 3. Cost of drug development (where average drug development time is 8 years)

Three kinds of products were considered in the simulation: small, medium and blockbuster. Based on a previous case study and the research findings of Grabowski and Vernon (2003), it was estimated that revenue for each product follows a typical lognormal product lifecycle. The overall sales for each product are equal to the sales values mentioned in the study by Grabowski and Vernon (2003) and are modeled as a lognormal distribution over time. For three kinds of products the distributions were as shown in Figure 4.

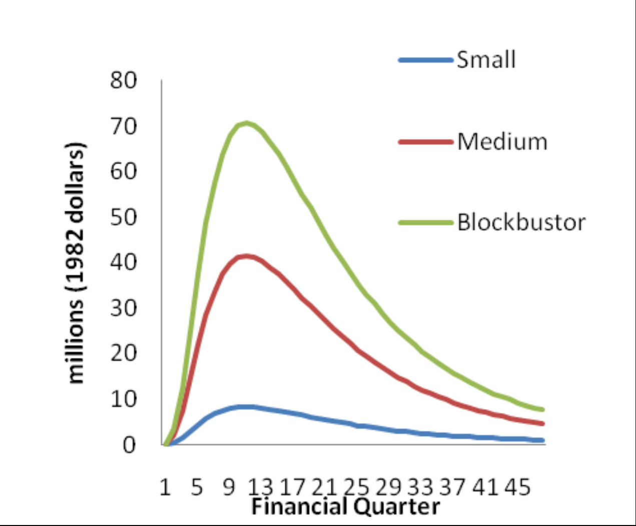


Figure 4. The product life cycle curve for different types of drugs.

Based on an industry report published by Pharmaceutical Research and Manufacturers of America (2006), we considered effective patent life to be 12 years. Further, each drug follows a different sales trajectory with sales concentrated mostly before the patent expiration. The model also includes the drug price increases due to inflation. The Congressional Budget Office (2006) estimates that drug prices have increased at three times the increase in the consumer price index (CPI). Therefore, although base revenue (demand) is driven by the lognormal curve, revenue each quarter is inflated by the Drug Price Index (DPI). We assume that all input costs will rise with inflation and affect each player similarly.

3.3 Develop Algorithm and Rules for the Agents

The simulation flow is shown in Figure 5. At the beginning of the simulation, all the variables are declared and initialized. In our simulation, we have modeled each round as one financial quarter. We ran the simulation for 300 quarters, which is equivalent to 39 years. At the beginning of the simulation game, scores/assets are assigned to each kind of player based on their corresponding triangular distribution. The initial 144 quarters are taken as a warm up period during which scores of different players do not change. The warm up period of 144 quarters represents three complete product life cycles.

During the simulation warm-up period a product pipeline is developed for each manufacturer. Manufacturers spend on R&D based on the square root of their assets. The square root of assets is used to represent the findings that smaller pharma firms tend to have a more focused product pipeline with fewer resources to support its R&D. Large manufacturers tend to have a larger and more diversified product pipeline. During the warm up period, the manufacturers' assets remain constant. That is, products released do not create profits and the pipeline is costless.

Manufacturers use a priority rule to allocate resources based on products closest to launch. The drugs which are in the later stages of clinical trials receive the highest priority for R&D spending. After spending their after tax profits for drugs in stage 3, stage 2, stage 1, and the preclinical stage, manufacturers spend the rest of the amount on IND's. If a manufacturer does not make profit in any quarter, they use a part of their assets to complete the clinical trials for the drugs in the last phase of stage 3. They estimate the cost of the drugs in their pipeline which are closest to the market. Therefore, manufacturers which are not making profit (or sufficient profits to sustain even stage 3 R&D) will use part of their assets on the R&D of the most promising drug that will garner them immediate returns. Further, in each stage some of the drugs are removed from the manufacturer's product pipeline

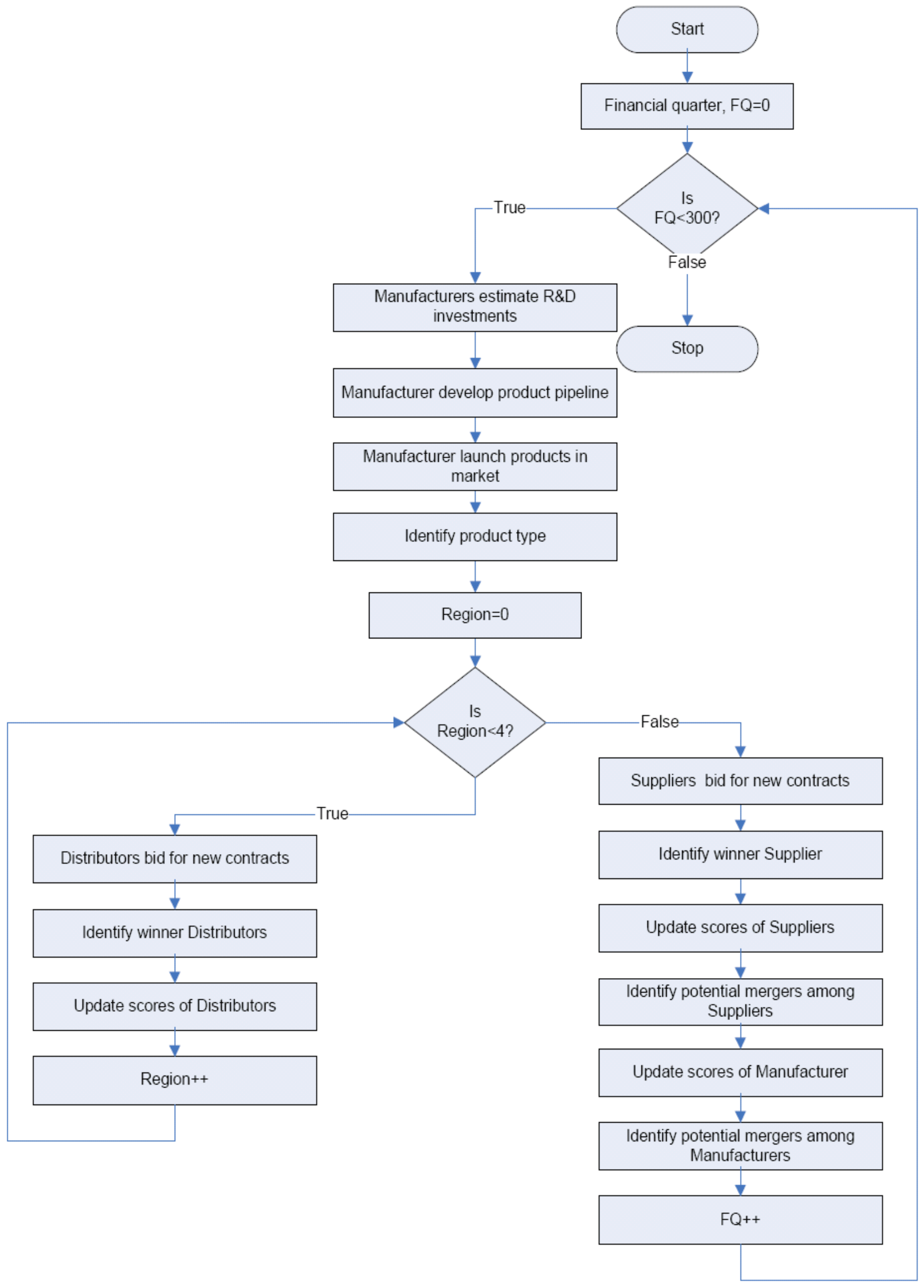

Start
Financial quarter, FQ=0
Is FQ<300?
True
False
Stop
Manufacturers estimate R&D investments
Manufacturer develop product pipeline
Manufacturer launch products in market
Identify product type
Region=0
Is Region<4?
True
False
Distributors bid for new contracts
Identify winner Distributors
Update scores of Distributors
Region++
Suppliers bid for new contracts
Identify winner Supplier
Update scores of Suppliers
Identify potential mergers among Suppliers
Update scores of Manufacturer
Identify potential mergers among Manufacturers
FQ++

Figure 5. Simplified schematic of simulation flow in the model.

based on the conditional probabilities of success in that stage. The actual industry R&D/Sales ratio ranged from 0.15 to 0.21 (Congressional budget office (2006) report mentions the finding of two studies. The first one was based on a report from the Pharmaceutical Research and Manufacturers of America (2005). They found that R&D/Sales ratio varies between 15 to 19 percent. They also mentioned the findings of National Science Foundation study in which it remains close to 15 percent) during the period under consideration (Congressional Budget Office, 2006). In our simulation, manufacturers spend 15 % of their overall sales on R&D in each financial quarter.

When a product is launched in different markets, it follows one of the three product life cycle curves. The sales for these drugs follow a lognormal distribution with different parameters for small, medium and blockbuster drugs based on the findings of Grabowski and Vernon (1993, 2003). These drugs are released by the manufacturers based on the rules described in Table 2.

In our model, we divided the pharmaceutical market into four regions with sales proportional to population in each region. We populated each region with a different number of distributors proportional to the population in that region.

The suppliers and distributors in each region bid to win the contracts for each new product. The winning distributor gets the exclusive distribution rights in its region for the contract time frame. At the end of the contract period, all the distributors bid again. Distributor's bid value represents the future market share they can buy whereas suppliers bids represent their margin. While bidding for a new product by a certain manufacturer, distributors estimate their bid value based on their past experience with that particular manufacturer. Every quarter the distributors update the average returns from the existing products for each manufacturer. For a new product, their estimated bid value is equal to this updated average from each manufacturer. For an exisitng product, distributors estimate the demand in next two years based on historical demand for that product. Based on the bid values received, the manufacturer determines the winning distributor.

Suppliers estimate their bid value based on their performance in the previous two bidding rounds. If they lost the previous two bids, they reduce their next bid value by a small percentage (determined using a random number generator). In cases where they win any of the previous two bids, they increase their next bid value by a small percentage. Every quarter, suppliers update the number of active contracts with each manufacturer. Active contracts with a manufacturer are the number of contracts a supplier won with that manufacturer in the last 12 years. In case the bid value of two suppliers is equal, the manufacturer selects the supplier with greater number of active contracts. This is similar to industry practices where previous relationships hold sway in supplier selection decisions.

Every quarter, different players receive returns from the active products and bear different costs based on the rules described in Table 2. Their profit from each product depends on the product life cycle (Figure 4) and the product type (Table 2).

At the end of each quarter, agents at each level scan competitors to determine ideal candidate for merger based on the rules described in Table 2. M&A rules were based on the findings of Danzon et al. (2003), and Higgins and Rodriguez (2005). The manufacturer with the highest priority to bid is estimated based on a rank order multi-criteria decision. Their rank is determined based on two criteria: return on assets over the last three years and total

Table 2: Rules for individual players and for their interaction

	Manufacturer	Supplier	Distributor
Starting bid	N/A	10	10
Number of Agents	30	60	60(15,24,12,9 in each region)
Bid based on	N/A	Acceptable profit margin	Future market share, historical demand, previous products
Costs	10% of sales(marketing cost), 15% of sales on R&D Suppliers Margin 33% tax on net earnings	5% of Assets	3% of Assets
Profit Margin	80%, 85%, 90% for small, medium and large products.	Based on bid value	10%

Bid Estimation rule	N/A	If lost previous 2 rounds
		Bid= Last round bid – RN*(0-0.1).
		If won at least 1 of the previous 2 rounds
		Bid= Last round bid+RN*(0-0.1).
			If Expected bid > 10 % of Assets then bid= 10% of the Assets.
			If Expected bid < 10 % of Assets then bid= Returns.

M & A rule	Manufacturers merge if: 1. Score1 > 5 X Score2. 2. Ln(HPOR2**)>1 3. HPOR is calculated based on return to assets and their product pipeline in last stages of development. After each round, Odds ratio (HPOR) is updated. 4. Determine the priority given to players for bidding based on multi criteria rank order based on their performance and Assets 5. Merger does not reduce Herfindahl index to less than 0.2	Suppliers(1 & 2) merge if: 1. Assets1 >1.5 X Assets2 2. Ln(HPOR2**)>1 3. HPOR is calculated based on return to assets. After each round, Odds ratio (HPOR) is updated. 4. Determine the priority given to players for bidding based on multi criteria rank order based on their performance and Assets 5. Merger does not reduce Herfindahl index to less than 0.2	Distributors(1 & 2) merge if: 1. Assets1 >5 X Assets2 2. Mergers occur only within a region. 3. Ln(HPOR2**)>1 4. HPOR is calculated based on return to assets. After each round, Odds ratio (HPOR) is updated. 5. Determine the priority given to players for bidding based on multi criteria rank order based on their performance and Assets 6. Merger does not reduce Herfindahl index to less than 0.2
Other	Drugs are released by Manufacturers if the IND under clinical trial clears all stages. The type of drug is decided based on the random number(RN) • RN<=3 => small drug • 3< RN <9 => medium drug • RN >=9 => Block buster. .		Demand is distributed in the 4 regions in the following ratio: 0.25, 0.40, 0.19, 0.16 Distributor gets the distribution rights for the drug till the contract time frame. Once the contract is over, all the distributors in that region bid again.

*Random Number

**HPOR refers to High Performance Odds Ratio. It is estimated using different performance indicators for different players. For suppliers, and distributors, the performance indicator is based upon profitability in recent years. For manufacturers, the performance indicators are based upon products in R&D pipeline and profitability in recent years. These performance indicators are estimated as follows:
Performance1: (Profit in last 3 years)/(Current score)
Performance2= (Number of drugs in stage2 and stage3)/(Current score)

The drugs in Preclinical and Stage 1 were not considered because there is very low probability of success for drugs in these two stages and will not be of much interest to acquirer. The above two performance measures are estimated for last 12 financial quarters. In each quarter if its performance is above median performance, it gets Highperformance=1 otherwise 0. Then we estimate Highperformance count (HPCT)

Table 2 Continued

which is a measure of number of quarters in which a firm performance was above median. Next we estimate Odds ratio from the HPCT as follows:
High performance odds ratio (HPOR) = $((\text{HPCT1}+\text{HPCT2})/(24-\text{HPCT1}-\text{HPCT2}))^2$
If, ln(HPOR) >1 then the company is a suitable target firm. This condition ensures that the company under consideration either has above average product development pipeline or has above average number of products in market which are in early stage of the life cycle. Among the acquiring companies the priority to bid goes to the manufacturer with a higher rank based on weighted average of their Assets and returns on assets (ROA) in last 3 years. Acquirer should have high capital reserve which is required to support the R&D, marketing and manufacturing of products of both the companies. The weight assigned to ROA is less than the weight assigned to Assets. This ensures that the acquiring company has good capital reserves. Another restriction is on the industry concentration. Any acquisition should not result in the industry herfindahl index to cross 0.2 which is considered as a very high concentration level in any industry. For suppliers and distributors, only one performance indicator was considered to measure HPCT and HPOR.

assets, with weights assigned to each criterion to calculate the ranking. This multi-criterion approach is reflected in practice where the largest and most profitable companies have easiest access to capital, which can then be used for acquisitions. Similarly, the most attractive manufacturer to be acquired is determined by the number of products in stage 2 and stage 3 of the product pipeline and their profitability during the last three years. After a merger, the products and contracts of the merging manufacturers are assigned to the new manufacturer. For distributors mergers are based on relative profitability in the previous three years as described in Table 2. The priority bidder will be the one with highest weighted score based on assets and performance in last three years. The new distributor has the combined experience of the type of products launched by the different manufacturers from the history of both distributors. This gives it an advantage in future bidding. Similarly, each supplier determines the ideal candidate for merger according to the rules described in Table 2 and merge. The new supplier has active contracts of both the suppliers.

The next quarter begins with the launch of new products by manufacturers. The price of each drug is inflated each quarter by the inflation factor. Whenever mergers occurs the supply chain network, associated suppliers and distributors of the two manufacturers are assigned to the new manufacturer.

3.4 Develop the Model using the Java Programming Language

The pharmaceutical supply chain multi-agent simulation was developed in JAVA. The simulation was developed in small modules and sub modules following it. Initially two levels of supply chain were developed and later third layer was added. We divided the simulation into a Main class (which forms the flow of the simulation as mentioned in the Figure 5) and other classes each representing different types of agents. There is a separate class for suppliers, distributors, and manufacturers, each with their own specific attributes and methods. Further each agent has its own thread of control. A new thread is generated whenever merger happens and the threads corresponding to old agents become inactive.

3.4.1 Supplier Object

The Suppliers object has several properties and routines to interact with other objects in the simulation and to perform several tasks autonomously like bidding, determining potential candidates for mergers, and estimating their profits from their active contracts.

3.4.1.1 Supplier Object Properties

Each supplier object has several properties to interact with other objects during the simulation. These properties are used to identify the supplier (*Supplier Number*) and to determine which supplier is still in the business and which are bankrupt (or acquired) by other suppliers (*Aliveagent*). *Score* is used to keep track of their overall assets. Another property called *CA* is used by the supplier to keep track of their active contract with different manufacturers. Properties like *Performance* keep track of their Return on Assets over the last 3 years. *Rank1* and *Rank2* determine the relative ranking of suppliers based on their *Performance* and *Score* respectively. Further their relative *Rank* is estimated based on the cumulative score *CMSC* obtained by taking weighted average of *Rank1* and *Rank2*. The higher *Rank* suppliers get priority over other suppliers during the acquisition process. Further potential candidates to be acquired are determined based on *HPOR (High performance odds ratio)* over the last 3 years. *HPOR* is determined based on *Highperf* and *HPCT* (High performance count). Table 3 summarizes all the Properties associated with Supplier Object.

Table 3: Supplier Object Properties

Alives	*Score(s)*	*CA(Number of active contracts)*
Averageprofit	*Performance*	*Sa(Score over the last 12 rounds)*
Supplier Number		
Properties of supplier modeled as shared object(Qs) variables		
Rank2	*Rank*	*CMSC(Cumulative score)*
Highperf	*HPCT(highperformance count)*	*HPOR(high performance odds ratio)*
Rank1		

3.4.1.2 Supplier Object Methods

Suppliers take decisions and perform their actions using a number of methods. These methods implement the rules mentioned in Table 2. Suppliers use these methods to determine the bids, profits, performance and to update their scores. These methods are discussed below.

Bid

Whenever a new product is launched by manufacturer agent in the simulation, supplier bid to get the contract for the entire product life. Each supplier object estimates its bid value based on the previous two bid experiences. If the supplier won any of the previous two bids, it increases its previous bid by a small value estimated using random number generator.

Bidnew

In case the supplier lost both the previous bids, it decreases its bid by a small value determined using random number generator. This is done by the supplier to ensure that it has sufficient business for sustaining in the industry.

Profits

This subroutine is used by the supplier to determine the profits they make from each product for which they won the contract in the last 12 years (Effective patent life). Their bid value represents their profit margin. Based upon this bid value they receive profit every quarter.

Contractwon

Supplier agents use this subroutine to estimates the number of contracts which were won by the supplier, and the number of active contracts with different manufacturers.

Scoreupdate

Supplier loses the operational cost and updates the profits from active contracts with different manufacturers. Once the score is updated, it is written in the text file.

Inflation

This method is used to estimate the inflation factor in any particular round.

Relativeperformance

This method is used to carry out following three tasks:

1. Update the array which stores the score of supplier in last 12 rounds

2. Update the property called performance = (Current score- score 3 years back)/current score
3. Update the score in last 12 rounds and performance value into the shared object called **Qs**

Sums

Estimate and returns the sum of an array which was passed to this method. This subroutine is used by a number of other subroutine during their execution.

3.4.2 Manufacturer Object

The Manufacturer object also has properties and methods similar to the suppliers which it uses to interact with other players (Table 4).

3.4.2.1 Manufacturer Object Properties

Manufacturers interact with both suppliers as well as distributors in different regions. Also they have other properties like product pipeline and products in market. Therefore, they have much greater number of object attributes as compared to suppliers. Properties associated with Mergers, Scores and Performance is similar to that of supplier. Manufacturer objects have other object properties like *Products* (Index number to identify product) and *TR* (year of launch of product), *B1* (Category of product: small, medium, large). Manufacturer objects have some additional attributes as well. These are *WIND* and *WINS* and are used to track the winner distributor and winner supplier for each product launched by the manufacturer. *Winbidd* and *Winbids* are used to track the winning bid value of distributors and suppliers for each product. Manufacturer interacts with Supplier and distributors using shared objects and these shared objects are used to store the values of some of the above attributes like *Products*(P), *Wind*(Winner distributor), *Wins*(Winner supplier), *Winbidd*(winning bid value of distributors), *Winbids*(winning bid value of supplier) etc.

Table 4: Manufacturer Object Properties

Alivem	*Performance*	*Performancep*
NME(Number of New molecular entities in product development pipeline)	*SM(Score of Manufacturer)*	*SM1(Score of Manufacturer over last 3 years)*
Properties of manufacturer modelled as shared object(Glob) variable		
Products(P)	*Tr(Year of launch of product)*	*B1(Product category)*
Winbids(Bid value of winner supplier)	*Winbidd(Bid value of winner distributor)*	*Wind(Winner distributor)*
Wins(Winner Supplier)		
Properties of manufacturer modelled as shared object(Qm) variable		
Highperf	*Highperfp*	*Rankp*
HPOR	*HPCT*	*Rank1*
Rank2	*Rank*	

3.4.2.2 Manufacturer Object Methods

Manufacturers also take decisions and perform their actions based on a number of methods. They use these methods to determine their profits, performance and to update their scores and to determine their investment in R&D. They also use different methods to select a supplier/ distributor for their new products. These methods are discussed below.

Mini

This subroutine is used by the manufacturer to determine the winning bid value for any of its products. In case of suppliers bidding for a new product, it returns the lowest bid value. If distributors in some region are bidding for future contract, the subroutine returns the maximum bid value.

Winner

This subroutine returns the index number of winner supplier or distributor whenever bidding event occurs. For suppliers, it returns the index number of supplier with the lowest bid value. For distributors, it returns the index number of distributor with the highest bid value.

Nextbidyear

Distributors win the contract for the product for a certain time frame based on their bid value. This subroutine determines the financial quarter in which the next bidding will occur after the end of existing contract. A supplier agent can win a contract for the entire product lifecycle whereas a distributor can win a contract for a maximum of two years.

Profits

Manufacturers use this method to determine their revenue each quarter from the products which they have launched in last 12 years. Their revenue is based upon the type of products and the corresponding profit margin (as mentioned in Table 2).

PipelineCost

Manufacturers use this subroutine to determine the cost of Research and Development (R&D) of their existing pipeline. It calculates the cost of clinical trials for all the products in different stages of clinical/ preclinical testing.

Approval

If the Pipeline Cost estimated by the manufacturer is greater than the capital assigned for R&D, then they use their assets to support the product pipeline in development.

Manufacturers determine if they can support the entire pipeline with their assets or not. If their assets are not sufficient to support the entire product pipeline, they spend their assets on part of their pipeline which is closest to the market. If the capital assigned for R&D is sufficient to support the entire pipeline, they spend the leftover capital on testing New Molecular Entities (NME's). This function also determines the number of drugs which successfully clears each stage and moves over to the next stage. Therefore it also determines the successful drugs entering next stage of drug development for the entire product pipeline.

Basepdapproval

During the simulation warm up period, manufacturers use this subroutine to determine the number of drugs which successfully clear different stages of drug development to reach the next stage.

Scoreupdate

Manufacturer updates its score in following steps:

1. Score = Score - R&D expenditure
2. Score = Score - 10% of profit in previous round
3. Score = Score - 33% corporate tax on the net profit in this financial quarter (Current score - score in previous round)
4. Score = Score + Profit from active products - Suppliers margin based on their bid value
5. Write the updated score in the text file

Assignproductcategory

This method performs the following tasks:

1. Update the year launched for the new products
2. Assign the category to the product based on random number generator:
    - If Random number <= 3 then it is a small product
    - If Random number > 3 and Random number < 9 then it is a medium product
    - If Random number >= 9 then it is a blockbuster product
3. Estimate the next bid year for the new product launched

Relativeperformance

This method performs the following tasks:

1. Update the array which stores the score for last 12 rounds
2. Estimate performance based on score:
3. Performance = (Current score - score 3 years back)/current score
4. Performancep = (Number of products in R&D pipeline in stage 2 and stage3)/current score
5. Update the performance and score data in shared object called Qm

Inflation

Estimate the inflation factor for the current round

3.4.3 Distributor Object

Distributor also has several attributes and methods to interact with manufacturers.

3.4.3.1 Distributor Object Properties

Distributor object attributes are similar to those of supplier. Table 5 summarizes all the Properties associated with distributor objects.

Table 5: Distributor Object Properties

Alived	*SR(score)*	*Movaverage*
SD(score in last 3 years)	*Performance*	*CD(Number of active contract)*
Properties of distributor modelled as shared object(Qd) variable		
Rank2	*CMSC(Cumulative score)*	*Highperf*
HPCT	*HPOR*	*Rank1*

3.4.3.2 Distributor Object Methods

Distributors also take decisions and perform their actions based on a number of methods. These methods implement the rules mentioned in Table 2. They use these methods to determine the bids, profits, performance and to update their scores. These methods are discussed below.

Bid

This method is similar to those of supplier except that in case of distributors if they loose the previous two bids they increase their bid value. This method is used by distributors only during the simulation warm up period.

Bidnew

This method is similar to those of supplier except that distributors use this method if they won atleast one of the previous two bids. This method is used by distributors only during the simulation warm up period.

Update

This method is used by the distributors to determine the category of products launched by the manufacturers in the past.

Profits

This method is used by the distributors to determine the profits from the products for which they won contract in last 12 years.

Contractwon

This method updates the number of contract won by the distributor and the number of active contract of distributor with each manufacturer.

Scoreupdate

1. Distributor loses the operational cost
2. It updates the profit from all the active contracts. Distributor receives 10% of the over all sales of a particular product in its region as its profit.
3. Write the updated score into text file.

Inflation

This method is used to estimate the inflation factor in any particular round.

Relativeperformance

This method is used to carry out following three tasks:

1. Update the array which stores the score of distributor in last 12 rounds
2. Update the property called performance = (Current score - score 3 years back)/current score
3. Update the score in last 12 rounds and performance value into the shared object called **Qd**

Sums

Estimate and returns the sum of an array passed to this method.

3.4.3.3. Shared objects

Agents communicate with each other and share data and information using the shared objects. These shared objects are used to store some of the object attributes mentioned above. The function of these shared objects is to share information between different agents. The shared objects are objects of the classes called ***Glob**, **Q**, **Qs**, **Qd**, and **Qm***.

3.4.4 Global Attributes

There is a separate file which includes all the parameters, declared as global variables, which can be changed for future studies to model different environmental and operating conditions. Table 6 summarizes all the global variables in the simulation.

Table 6: Global Variable Properties

Bidlife(Life of product)	*Region(Number of regions)*	*Experiment(Number of simulation runs)*
Percmp(time period for which performance is determined during merger)	*Successrate(End of stage probability of passing a preclinical/clinical trials)*	*Pro(Revenue associated with different category of product in different stages of product life cycle)*
Preapproval(Cost of testing drugs in different stages of clinical/preclinical trials)	*Pn(Percentage of population in each region)*	*Perc(Percentage of Assets of distributors after deducting operations cost)*
Sup(Percentage of Assets of supplier after deducting operations cost)	*Basepd(Warmup period of simulation)*	*IA(number of suppliers at the start of simulation)*
IM(Number of Manufacturer at the start of simulation)	*ID(Number of distributor at the start of simulation)*	

3.4.5 Statistics Methods

Round

This method is used to round of the data before saving it into the database or the text file.

Sum

This method is used to determine the sum of scores of all the values in an array.

Mean

This method is used to determine the mean of all the values stored in an array.

Minimum

This method is used to determine the minimum value stored in an array.

Maximum

This method is used to store the maximum value stored in an array.

Stdev

This method is used to determine the Standard deviation of all the values stored in an array.

Scorer

This method is used to assign scores to different players (suppliers, distributors, and manufacturers) at the start of the simulation using triangular distribution.

3.4.6 Data Collection and Other Features

To identify each kind of transaction we developed a unique key which can be used to sort and analyze the data. There is a separate class to develop these keys specific to each kind of transaction. The key is composed of 20 letters with the following significance:

XXX-XXX-XXX-XXX-XXX-XXX-XX

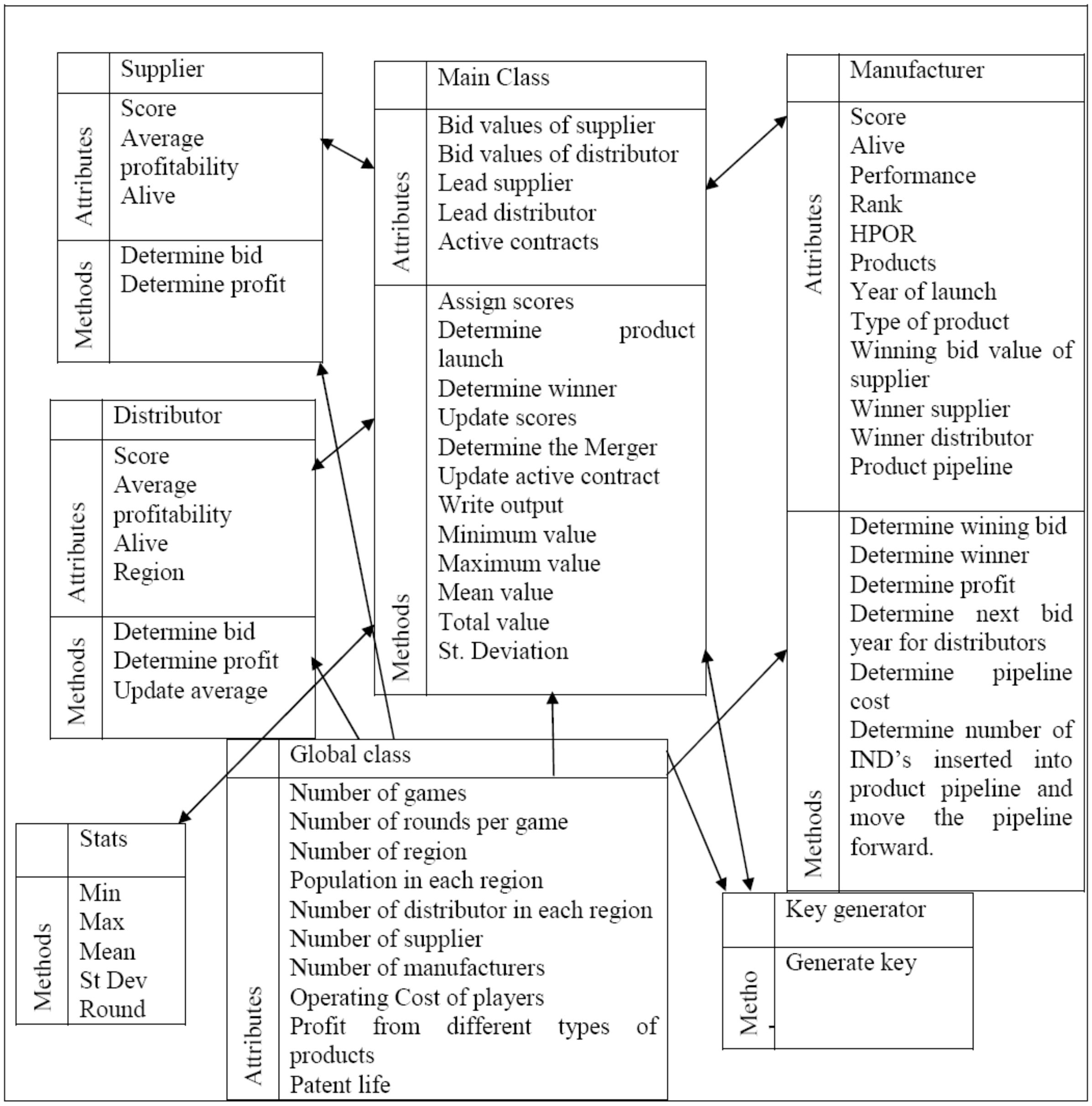


Figure 6. Different classes and their interaction

The first three digits represent simulation game. The next three groups of three digits represent financial quarter, Manufacturer and the product number respectively. The next three digits are for the type of agent corresponding to the given transaction followed by the agent number. The last two digits represent location for the distributor agents.

We wrote the output specific to each layer of supply chain in separate text file. We also calculated statistics for total assets, minimum, maximum and mean and standard deviation of asset values of suppliers, manufacturers, and distributors in each region and for each round. We also collected data for the number of products launched by all the manufacturers and the number of survivors at each level of supply chain in each simulation. The above data was collected so as to develop a relation between the supply chain efficiency and the supply chain parameters.

3.5 Model Verification and Validation

We validated our model using the following three parameters, degree of consolidation, products released, and return on assets (ROA). Each parameter was measured or calculated for the years corresponding to 1982 through 2006. The actual data for each year was obtained from the COMPUSTAT (2008) and FDA Orange Book (2008) databases. This creates three pairs of temporal distributions corresponding to each of our time dependent parameters. For each parameter we stipulated the following null hypothesis: the simulated data and the actual data are from the same distribution. We tested these hypotheses using a Chi-Square and a Kolmogorov-Smirnov (K-S) test. The results of these tests allow us to state if there is a statistically significant difference between the actual and the observed distributions. We also tested the cross section data using the K-S test. This test is less sensitive to serially correlated data and was used to determine if the overall distribution of results was plausible when compared to actual data. We also did a paired t-tested to compare the means of products released and return on assets from the simulated and the actual data.

Results of Validation

We first performed the statistical validation of our model using 1000 runs. At 5 percent significance level, we identified the games for which the difference in the simulation parameter and the corresponding industry parameter is not statistically significant. The summary of validation results is given in Table 7.

Table 7: Number of simulations conforming to industry data at 5 percent significance level

	KS-test(overall distribution)	KS-test (distribution over time)	Chi square test	t-test
Products(before 1995)	928	987	391	859
Products*(with spike distributed)	499	819	19	460
Consolidation(yearly)	205	915	280	-
Return on Assets(before 1995)	146	-	-	423
Return on Assets(entire data)	39	-	-	23

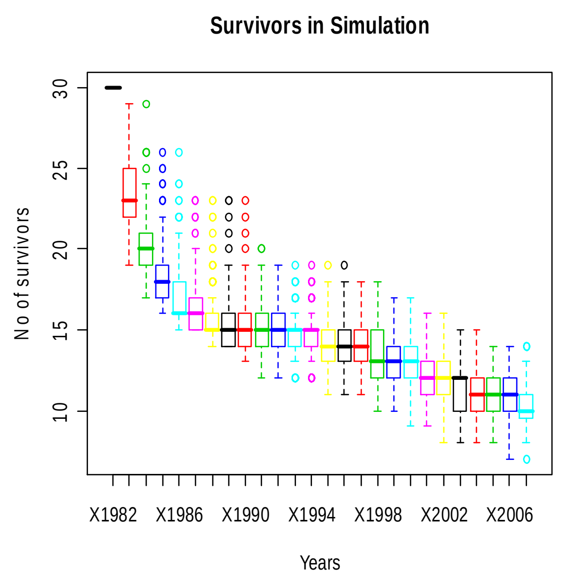
*Spike was distributed using a simple exponential smoothing

Degree of Consolidation

Using results from the 1000 games we determined the change in the number of players in the last 30 years. We compared this result to the actual degree of consolidation each year in 1982 using Compustat database. We tested the null hypothesis:

H_o: Observed Degree of consolidation has the same distribution as the degree of consolidation in the industry.

Out of 1000 games, 280 games has the chi square statistic < $X^2_{0.05}(26)$. Therefore, at α=0.05 significance level, we do not have enough evidence to reject the null hypothesis for 280 games. That is, the data support the hypothesis that degree of consolidation in 280 games reasonably represents the actual degree of consolidation in the industry. The Kolmogorov Smirnov (K-S) test for the above hypothesis was performed to compare the overall distribution of consolidation as well as the distribution of consolidation over time. For the cross-sectional distribution of consolidation, the test statistics for 205 games were less than the critical K-S value. While testing the temporal distribution, 915 games had the test statistics less than the critical K-S value. Using the test related to consolidation over time, we cannot reject the null hypothesis for 915 games.



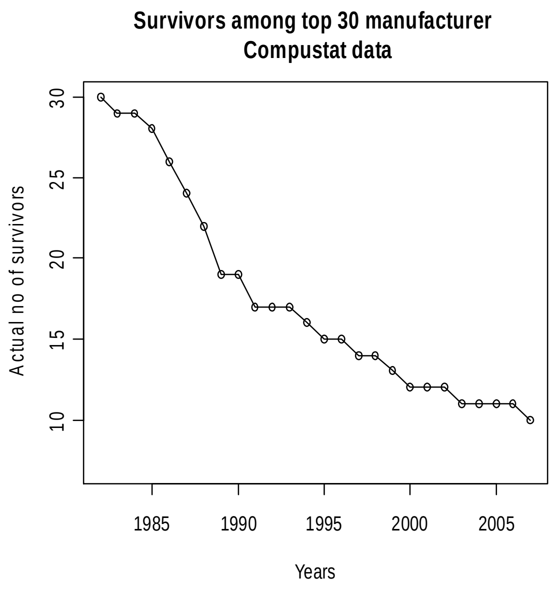


Figure 7a. Simulation output for the games with KS test statistic< critical value.

Figure 7b. Compustat data for the large pharma companies of 1982

Products Released

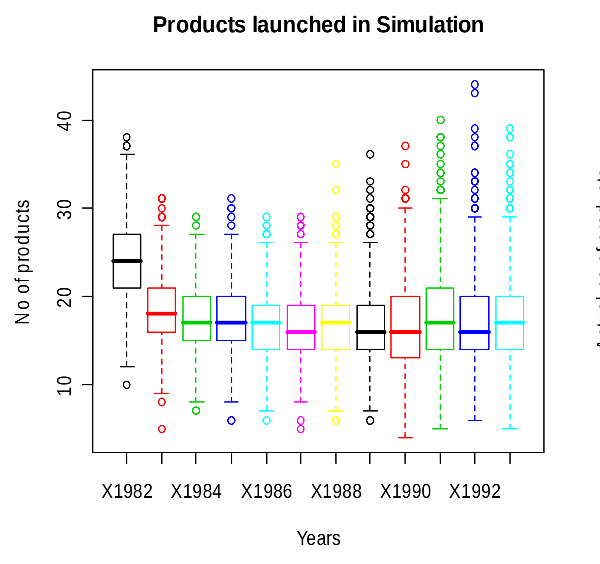
We also verified if the number of products released in our model is representative of the actual values in the industry. We calculated the number of products launched by large pharmaceutical manufacturers from 1982 onward using the Orange Book data and Compustat data. While performing the Chi-Square test for the observed values and the orange book data, we realized that there were five data points in the orange book data which were very much different from the normal trend. These five points represented the number of new drugs approved shortly after 1996. This difference was primarily due to a law passed in 1992, which improved the efficiency of the drug development process in terms of time spent during the clinical trials (http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1113977, accessed 10th July 2009). Since our model will not represent the outliers, we used the data points before 1995 for the statistical tests. Again, we performed Chi square test and Kolmogorov-Smirnov test to determine which games are representative of the actual number of products launched in the industry per the FDA's Orange Book. We tested the null hypothesis:

H_o: Number of products launched in the simulation has the same distribution as the actual number of products launched in the industry.

Out of 1000 games, 391 games has the chi square statistic < $X^2_{0.05}(12)$. Therefore, at α=0.05 significance level, we do not have enough evidence to reject the null hypothesis for 391 games. That is, the data support the hypothesis that number of products launched in 391 games reasonably represents the actual number of products launched in the industry. Using the K-S test, 928 games could not be rejected as fitting the overall distribution of products.

While testing the distribution over time, the K-S test statistics was less than the critical value for 987 games.

We also tested the above null hypothesis after simple exponential smoothing of the orange book data for the period after 1995. Out of 1000 games, 19 games has the chi square statistic < $X^2_{0.05}(27)$. Therefore, at $\alpha=0.05$ significance level, we do not have enough evidence to reject the null hypothesis for 19 games. The KS –test for the overall distribution shows that we cannot reject the null hypothesis for 499 games at $\alpha=0.05$ significance level. While testing the distribution over time, the K-S test statistics was less than the critical value for 819 games.



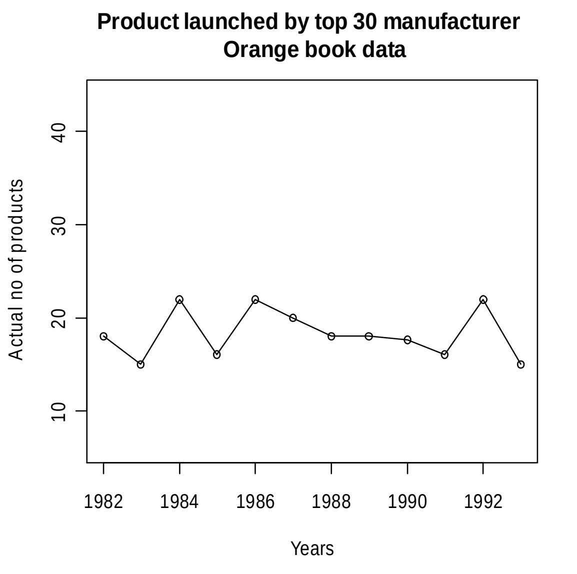


Figure 8a. Simulation output for the games with with KS test statistic< critical value.

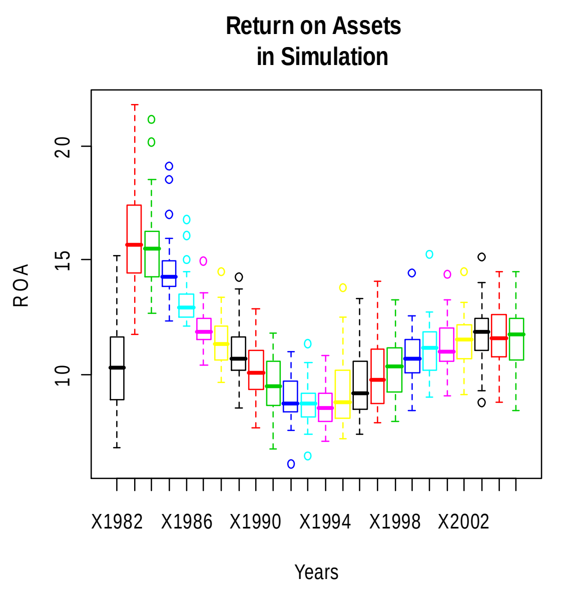
Figure 8b. Orange book data for the large pharma companies of 1982

We also performed paired t-test to test the null hypothesis that the mean of the two-paired samples is equal. At $\alpha=0.05$ significance level, we could not reject the null hypothesis for 859 games for data before 1995, and for 460 games when considered the entire data after exponential smoothing.

Return on Assets

Finally, we tested if our model created representative return on assets. The actual return on assets for the pharmaceutical industry is much higher than other industries. Further, returns are from a variety of products that share common assets. Moreover, the return on assets was different for different markets and varied with the type of products and with the currency fluctuation. Therefore, we can expect some discrepancy between the simulated and the observed return on assets. We tested the null hypothesis

H_o: The return on assets in the simulation has the same distribution as the observed return on assets in the industry.



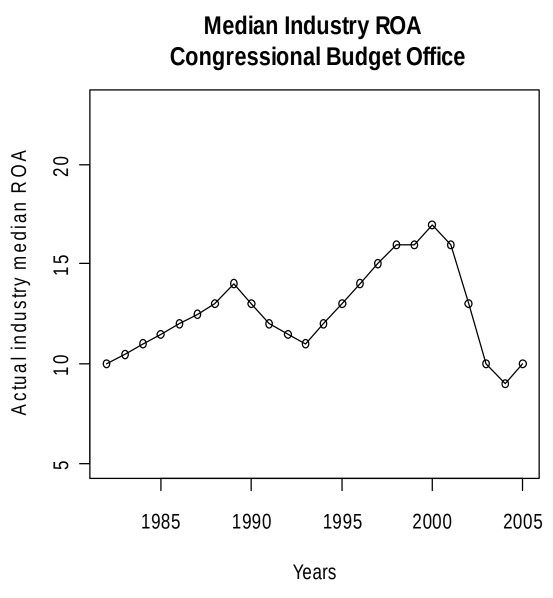


Figure 9a. Simulation output for the games with KS test statistic< critical value.	Figure 9b. Median industry ROA from CBO report.

The K-S test statistics was less than the critical value for 146 games when we consider data before 1995 i.e. before the passing of the new law affecting product launch. If we consider the entire data set, the test statistics for 39 simulation games was less than the critical value. Therefore, we cannot reject the null hypothesis for 39 games if we consider the entire data set.

We performed the paired t-test to check the null hypothesis that the mean ROA of the two-paired samples is equal. At α=0.05 significance level, we could not reject the null hypothesis for 423 games for data before 1995, and for 23 games when the entire data was considered.

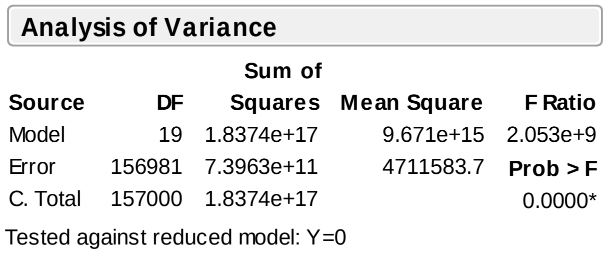
The parametric and non-parametric tests performed above reveal that a significant number of simulations conform to the products released, ROA, and degree of consolidation in the PSC. Therefore, the model can be used to simulate the dynamics of the PSC and to address the research questions considered in this study.

Chapter 4

Results

Given that autonomous agent models are prone to state dependent results, we attempted to discern which factors had the greatest impact on final assets. This simplified analysis uses regression with lagged variables, direct effects, and cross-products with time – business quarters. We analyzed the quarter (linear, square, and cubic), the first and second lag of the assets, each tier of the supply chain's concentration, the number of drugs released by the supply chain, and the interaction of concentration with time. The analysis of variance (see Table 8) shows that the model is significant with an F statistic of 2.053*e^9. The coefficients show that the end state of pharmaceutical supply chain is highly dependent on when mergers and acquisitions occur and when drugs are released.

Table 8: Analysis of Variance



Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	19	1.8374e+17	9.671e+15	2.053e+9
Error	156981	7.3963e+11	4711583.7	**Prob > F**
C. Total	157000	1.8374e+17		0.0000*

Tested against reduced model: Y=0

The financial quarter (time term) shows that the total assets of the supply chain are cubic in nature. The lag effects of Total Score in the previous two quarters are significant. The direct effect and indirect effects of concentration are significant and interesting at the Manufacturer and Supplier levels of the pharmaceutical supply chain. Concentration is defined as the amount of revenue generated by the largest firms in an industry. Higher concentration indicates that the industry may be susceptible to oligopoly or near monopoly pricing. To measure concentration we used the Herfindahl index where Herfindahl index is equal to sum of squared market shares of firms in the industry.

Both direct effects indicate that increased concentration at supplier or manufacturer level leads to higher total assets for the supply chain. This may be because a merger at manufacturer level will prevent bankruptcy which is a much bigger loss for supply chain because it results in loss of the entire product pipeline in development. At the supplier level, mergers result in less competition and therefore higher margins.

The indirect effects indicate something different. Manufacturers' concentration over time leads to higher total assets in the supply chain. The interaction term of manufacturer's concentration with time shows that supply chain performs better if manufacturers merge in the later stages rather than in the early stages of the simulation. The reason for this could be that if manufacturers merge early in the game, then the acquiring manufacturers will not have sufficient amount of capital to sustain its own product pipeline as well as its target firm's pipeline. On the other hand if mergers happen in the later part of the simulation, they result in better performance of the supply chain because acquiring manufacturers have large cash reserves to sustain the product development pipeline of its own as well as the target firm's pipeline. Supplier concentration over time shows the opposite effect. Supplier concentration decreases competition between suppliers and therefore increases revenue for the remaining suppliers. In the model we assume supplier cost pass-through. This means that the supplier's margins will be passed on to the manufacturer. As expected, the number of drugs released has a positive effect on total assets. The interaction effect is also positive and significant. As the game proceeds, drug release becomes more important to supply chain profitability.

Table 9: Regression Analysis

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Financial quarter	-24.4496	1.78157	-13.72	<.0001*
(Financial quarter)^2	0.038331	0.016113	2.38	0.0174*
(Financial quarter)^3	0.0001575	0.000037	4.26	<.0001*
Product_W	173.09641	5.214112	33.20	<.0001*
M_conc	1308.065	207.21	6.31	<.0001*
Sconc	3276.0215	258.3455	12.68	<.0001*
D1_Conc	132.75138	84.76226	1.57	0.1173
D2_Conc	1.6305708	82.87256	0.02	0.9843
D3_Conc	206.75137	99.1582	2.09	0.0371*
D4_Conc	192.8092	93.9236	2.05	0.0401*
(Financial quarter-221)*(Product_W-6.13039)	6.3167143	0.097831	64.57	0.0000*
(Financial quarter-221)*(M_conc-0.17603)	40.375261	4.095373	9.86	<.0001*
(Financial quarter-221)*(Sconc-0.14624)	-9.992852	4.936573	-2.02	0.0429*
(Financial quarter-221)*(D1_Conc-0.19952)	8.1038721	1.798728	4.51	<.0001*
(Financial quarter-221)*(D2_Conc-0.17642)	5.5516437	1.769649	3.14	0.0017*
(Financial quarter-221)*(D3_Conc-0.22204)	11.565085	2.105138	5.49	<.0001*
(Financial quarter-221)*(D4_Conc-0.22483)	11.090451	1.942004	5.71	<.0001*
Totalscore_1	1.85499	0.001453	1276.4	0.0000*
Totalscore_2	-0.851247	0.001499	-568.0	0.0000*

Summary of Fit

RSquare	.
RSquare Adj	.
Root Mean Square Error	2170.618
Mean of Response	717696.8
Observations (or Sum Wgts)	157000

Actual by Predicted Plot

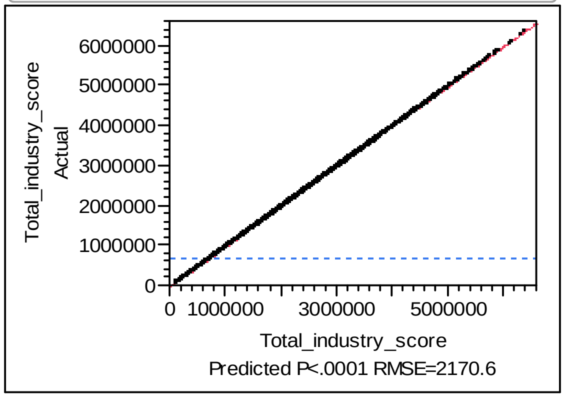


Table 9 Continued

Where the definition of the terms in the regression model is as mentioned below:

Financial quarter	Industry financial quarter
Product_W	Weighted average of active products with the percentage of sales of that product
M_Conc	Sum of squared market share of manufacturer
SConc	Sum of squared market share of suppliers
D1_Conc	Sum of squared market share of distributors in region 1
D2_Conc	Sum of squared market share of distributors in region 2
D3_Conc:	Sum of squared market share of distributors in region 3
D4_Conc	Sum of squared market share of distributors in region 4
Totalscore_1	Total assets of all the manufacturers, distributors and suppliers in previous financial quarter
Totalscore_2	Total assets of all the manufacturers, distributors and suppliers in previous to previous financial quarter

The results reveal several important relationships between different industry factors and the PSC performance. In the future, the PSC will be under ever increasing pressure to develop blockbuster drugs to sustain itself. Subsequently, greater investment in R&D is necessary. Additionally, manufacturer's margins must continuosly be reinvested in innovation for the supply chain to continue. Similarly, consolidation has positive effect on the overall performance of the supply chain as it helps to restore the R&D pipeline of the manufacturers which are financially weak. At supplier level, consolidation results in reduction in competition and therefore results in improved performance. As can be seen, regression analysis of the simulation results demonstrate that a variety of factors (concentration, research productivity, and size) greately influence the performance of the PSC.

Chapter 5

Summary and Future Work

The presented model is able to replicate several important characteristics found in the pharmaceutical supply chain. Based on observations in previous research we have developed realistic rules for three major classes of agents in the PSC. Suppliers and distributors interact with manufacturers to create, manufacture, and distribute three types of drugs: blockbuster, average, and small. Much like the actual PSC all activity revolves around funding and distribution of drugs. Manufacturers in our model finance drugs internally through residual profits or through acquiring another manufacturer with a promising pipeline. It is this merger and acquisition functionality combined with a product development pipeline that separates our model from other industrial models. This creates a test bed to examine the co-evolution of product development and supply chain structure under differing agent rules and environmental conditions.

The current model is able to replicate both the overall variability as well as the evolution of the PSC as measured by several key variables. Using the Chi-Square difference and the Kolmogorov-Smirnov test the vast majority of simulation runs reasonably match drugs released, consolidation, and return on assets. Thirty games reasonably match all three criteria. Given that the simulation covers three decades and does not include environmental shocks, we feel confident that this model can be used to test future research questions.

The existing multi-agent model can be used to study the interaction of different factors and their overall impact on the supply chain performance. We are currently examining the effect of consolidation at various levels of the supply chain on profitability and research productivity. Important questions that this model can help answer include: the effect of the timing of mergers and acquisitions on the acquiring firm's profitability, the effects of gaps in the product pipeline, the effect of relationship length between individual supply chain members, as well as impact of blockbusters on industry structure. Future models will address changes in government regulation, funding sources, and distribution models. These models will likely include retailers and hospitals, pharmacy benefit managers, and government agencies

Several shortcomings of this model detract from its face validity. The first is that it does not include new entrants/technology. We feel this is a minor concern since the vast majority of drugs released as well as economic activity since 1980 has been controlled by top five drug manufacturers, often referred to as 'Big Pharma'. Second, we do not replicate over-the-counter medications. In the past, these drugs provided lower but consistent margins to manufacturers. In the past five years, most manufacturers have sold off these businesses and focused on drug development. Since this is our area of interest, we feel that this compromise limits the complexity of both the model and its output. In the same vain, the model does not include generics. This is a fairly recent addition to the PSC. Its effect is that it shortens the product life cycle of branded drugs. Varying the product life cycle over the course of the simulation is possible for future iterations. Lastly, the game allows horizontal mergers and acquisitions, but does not allow vertical integration. The history of the pharmaceutical industry, like most industries, has been marked by varying degrees of vertical integration. As Cardinal Health's recent sale of its drug making unit[1] demonstrates, these vertical relationships are often less important than peer competition and drug discovery – both accurately reflected in our model.

[1] http://www.reuters.com/article/health-SP/idUSWEN283520070125

REFERENCES

Pharmaceutical research and manufacturers of America. Pharmaceutical industry profile 2006. http://www.phrma.org/files/2006%20Industry%20Profile.pdf

Arunachalam R and Sadeh NM. 2005. The supply chain trading agent competition. Electronic Commerce Research & Applications 4(1):63-81.

Austin DH *et al*. 2006. Congress of the United States. *Congressional budget office*. Research and development in pharmaceutical industry. Washington: U.S. government printing office, 2006. http://www.fdareview.org/approval_process.shtml

Balci O and Ormsby WF. 2007.Conceptual modelling for designing large-scale simulations. Journal of Simulation. August, 2007:175-86.

Barbuceanu M, Teigen R, Fox MS.1997.Agent based design and simulation of supply chain systems. Enabling Technologies: Infrastructure for Collaborative Enterprises. Proceedings Sixth IEEE workshops on , vol., no., pp.36-41, 18-20 Jun 1997

Choi TY, Dooley KJ, Rungtusanatham M. 2001. Supply networks and complex adaptive systems: Control versus emergence. J Oper Manage 19(3):351-66.

Cockburn IM and Henderson RM. 2001. Scale and scope in drug development: Unpacking the advantages of size in pharmaceutical research. J Health Econ 20(6):1033-57.

Danzon PM, Nicholson S, Pereira NS. 2005. Productivity in pharmaceutical–biotechnology R&D: The role of experience and alliances. J Health Econ 24(2):317-39.

Danzon PM, Epstein A, Nicholson S. 2004. Mergers and acquisitions in the pharmaceutical and biotech industries. Cambridge, MA: National Bureau of Economic Research.

DiMasi JA, Hansen RW, Grabowski HG. 2003. The price of innovation: New estimates of drug development costs. J Health Econ 22(2):151-85.

DiMasi JA, Hansen RW, Grabowski HG, Lasagna L. 1991. Cost of innovation in the pharmaceutical industry. J Health Econ 10(2):107-42.

Frank RG. 2003. New estimates of drug development costs. J Health Econ 22(2):325-30.

Grabowski H, Vernon J, DiMasi JA. 2002. Returns on research and development for 1990s new drug introductions. Pharmacoeconomics 20(15):11-29.

Grabowski HG and Vernon JM. 1994. Returns to R&D on new drug introductions in the 1980s. J Health Econ 13(4):383-406.

Henderson R and Cockburn I. 1996. Scale, scope, and spillovers: The determinants of research productivity in drug discovery. Rand J Econ 27(1):32-59.

Higgins MJ and Rodriguez D. 2006. The outsourcing of R&D through acquisitions in the pharmaceutical industry. J Financ Econ 80(2):351-83.

Rossetti CL, Handfield R, Dooley KJ. Forces, trends, and decisions in pharmaceutical supply chain management. Working paper.

Sadeh NM, Arunachalam R, Eriksson J, Finne N, Janson S. 2003. TAC'03: A supply chain trading competition. AI Magazine 24(1):83-91. http://www.aaai.org/ojs/index.php/aimagazine/article/view/1692/1590

Siebers P, Aickelin U, Celia H and and Clegg C. 2007. A multi-agent simulation of retail management practices. In *Proceedings of the 2007 Summer Computer Simulation Conferenc*e (San Diego, California, July 16 - 19, 2007). Summer Computer Simulation Conference. Society for Computer Simulation International, San Diego, CA, 959-966.

Solo K, and Paich M. 2004. A modern simulation approach for pharmaceutical portfolio management. *International conference on health sciences simulation*, San Diego, California, USA. http://www.simnexus.com/SimNexus.PharmaPortfolio.pdf

Swaminathan JM, Smith SF, Sadeh NM. 1998. Modeling supply chain dynamics: A multiagent approach. Decision Sciences 29(3):607-32.

Wellman MP, Greenwald A, Stone P. 2007. Autonomous bidding agents : Strategies and lessons from the trading agent competition. Cambridge, Mass.: MIT Press.

Yonghui F, Piplani R, de Souza R, Jingru Wu. 2000. Multi-agent enabled modeling and simulation towards collaborative inventory management in supply chains. Simulation Conference Proceedings, 2000 Winter 2:1763, 1771 vol.2.

Sources of data

COMPUSTAT database: We obtained a list of all the pharmaceutical manufacturing and distribution companies from 1982 onwards. We collected the data related to number of companies and their financial from 1982 to 2006. The data was collected in 2008 and was used to develop a distribution which is used to assign assets to different players at the start of the simulation and for validating the simulation results related to their degree of consolidation and return on assets.

FDA Orange Book: We obtained data related to number of drugs launched from 1982 to 2006. This data was also collected in the year 2008. This data was used to validate the simulation results for number of products launched during the above time span.

APPENDIX

Appendix: Java program for different classes in the simulation

Appendix includes the code in Java programming language. The simulation is composed of number of classes. The functions of these classes were described in the simulation development section. These classes are as follows:

1. Main class
2. Manufacturer class
3. Supplier class
4. Distributor class
5. Global variables class
6. Glob class
7. Statistics class
8. Semaphores class(Q)
9. Shared variable class between manufacturers(Qm)
10. Shared variable class between suppliers(Qs)
11. Shared variable class between distributors(Qd)
12. Key generator class

1. Main class

```
import java.io.*;
import java.util.*;
import java.lang.Object.*;
import java.math.*;
import java.lang.*;
import java.sql.*;
public class Main {
public static void main(String[] args) throws Exception {
BufferedWriter pr = new BufferedWriter(new FileWriter("C:\\Output\\prod.txt"));
BufferedWriter wm = new BufferedWriter(new FileWriter("C:\\Output\\Mscore.txt"));
BufferedWriter ws = new BufferedWriter(new FileWriter("C:\\Output\\Sscore.txt"));
BufferedWriter wd1 = new BufferedWriter(new FileWriter("C:\\Output\\D1score.txt"));
BufferedWriter wd2 = new BufferedWriter(new FileWriter("C:\\Output\\D2score.txt"));
BufferedWriter wd3 = new BufferedWriter(new FileWriter("C:\\Output\\D3score.txt"));
BufferedWriter wd4 = new BufferedWriter(new FileWriter("C:\\Output\\D4score.txt"));
BufferedWriter mr  = new BufferedWriter(new FileWriter("C:\\Output\\Mergers.txt"));
BufferedWriter profs  = new BufferedWriter(new FileWriter("C:\\Output\\sProfit.txt"));
BufferedWriter profd  = new BufferedWriter(new FileWriter("C:\\Output\\dProfit.txt"));
BufferedWriter profm  = new BufferedWriter(new FileWriter("C:\\Output\\mProfit.txt"));
BufferedWriter es  = new BufferedWriter(new FileWriter("C:\\Output\\Endscore.txt"));
BufferedWriter msry  = new BufferedWriter(new FileWriter("C:\\Output\\MSummary.txt"));
BufferedWriter ssry  = new BufferedWriter(new FileWriter("C:\\Output\\SSummary.txt"));
BufferedWriter d1sry  = new BufferedWriter(new
FileWriter("C:\\Output\\D1Summary.txt"));
BufferedWriter d2sry  = new BufferedWriter(new
FileWriter("C:\\Output\\D2Summary.txt"));
BufferedWriter d3sry  = new BufferedWriter(new
FileWriter("C:\\Output\\D3Summary.txt"));
BufferedWriter d4sry  = new BufferedWriter(new
FileWriter("C:\\Output\\D4Summary.txt"));
BufferedWriter product  = new BufferedWriter(new FileWriter("C:\\Output\\Product.txt"));
//  BufferedWriter psry  = new BufferedWriter(new
FileWriter("C:\\Output\\Productdistribution.txt"));
BufferedWriter testing  = new BufferedWriter(new FileWriter("C:\\Output\\Testing.txt"));
BufferedWriter prdev  = new BufferedWriter(new
FileWriter("C:\\Output\\Productdevelopment.txt"));
BufferedWriter dbid  = new BufferedWriter(new FileWriter("C:\\Output\\dbidding.txt"));
BufferedWriter report  = new BufferedWriter(new
FileWriter("C:\\Output\\ReportSummary.txt"));
BufferedWriter sbid  = new BufferedWriter(new FileWriter("C:\\Output\\sbidding.txt"));
BufferedWriter Rd  = new BufferedWriter(new FileWriter("C:\\Output\\RnD.txt"));
```

```
for(G.sim=0; G.sim<G.experiment;G.sim++)
  {
 /* declaring variables for the Agent and manufacturer.Declaring array for the bid
values,scores and active bids */
   Glob g; //= new Glob();
   Q q ;   //= new Q();
   Q q2;   // = new Q();
   Q q3;   // = new Q();
   Q q1;   // = new Q();
   Qs qs;  // = new Qs();
   Qm qm;  // = new Qm();
   Qd qd;  // = new Qd();
   Runtime z = Runtime.getRuntime();
   Agent a[]= new Agent[G.agent];                  //declaring objects of class Agent
   Manufacturer ma[]= new Manufacturer[G.man];
   distributor de[][]= new distributor[G.Region][G.distributor];
   Random generator = new Random();   //creating an object of class Random to generate
random numbers
   keygen key1=new keygen();
    int newsupp = 0;      // variable to generate a new supplier out of merger
    int newman = 0;                  // variable to generate a new manufacturer out of merger
    int[][][] nb;                        // Array to save the next bid year
    int[]count;                      //Array to count the number of bids won by a distributor with an
particular manufacturer
    float[]sum;                 // Array to keep track of Average returns to a distributor from the
old products from a particular manufacturer
    float[] delta;                   // Array to store the value of all the returns to the suppliers for all
the product of a particular manufacturer
    int[] newdist;                      // A variable to create a new distributor. Each time a distributor
is created , its value is incremented by 1
    float[][] maxbid;                   // An array to store maximum bid for each product launched
by any manufacturer
    int sn =0;
    Random rng = new Random();
    int tpr=0;
    int[] nme;
    int productcount=0;
    float Rnd;
    float clintrialcost=0f;
    float coc=0;
    int counter;
    int mp=0;
    int medianp=0;
```

```
//INITIALIZE THE VALUES TO THE VARIABLES
    g= new Glob();
    q = new Q(g);
    q2 = new Q(g);
    q3 = new Q(g);
    q1 = new Q(g);
    qs = new Qs(g);
    qm = new Qm(g);
    qd = new Qd(g);
    maxbid=new float[G.man][G.spacealloc];
    sum = new float[G.man];
    int bl=0;
    int leadman=0;
    float margin=0;
    int leadsupp=0;
    newsupp = 0;
    newman=0;
    newdist =new int[G.Region];
    delta= new float[G.man];
    count= new int[G.man];
    nb= new int[G.Region][G.man][5*G.spacealloc];
    G.j=1;
    G.start=0;  //Prevent manufacturer thread from starting until the simulation clock starts
//Initializing manufacturers. Start the new thread for each manufacturer agent
  for(int n=0;n<G.im;n++)
     {           // Assigning one product to each manufacturer in the second round
     ma[n]= new Manufacturer(1,Stat.scorer(1),n,q,qm,q1,q2,q3,g,profm,key1,dbid,Rd); //
alivem, score
    g.modifyalivem(n,1);
    g.modifymborn(n,1);
     }
//Initializing supplier. Start the new thread for each supplier agent
 for(int n=0;n<G.ia;n++)
   {
     float temp =Stat.scorer(2);
     a[n]= new Agent(1,temp,0,n,q,qs,q1,q2,q3,g,profs,key1);
     g.modifyalives(n,1);
     g.modifyaborn(n,1);
     }
//Initializing distributors. Start the new thread for each distributor agent
for(int n1=0;n1<G.Region;n1++)
{ for(int n=0;n<G.id[n1];n++)
   {
     de[n1][n]= new
distributor(G.dcn[n1][n],n1,n,0,1,Stat.scorer(0),n,q,qd,q1,q2,q3,g,profd,key1);
     g.modifyalived(n1,n,1);
     g.modifydborn(n1,n,1);
    }
  }
int y2=0;
System.out.println("Simulation :"+G.sim);
G.start= 0;
//INITIALIZATION IS OVER
//NEXT STEP: MANUFACTURERS INTRODUCING NEW PRODUCTS EACH YEAR
for(G.j=1;G.j<G.bid;G.j++)
{  // System.out.println( "Round"+G.j);
    G.start= 1;  // Start the manufacturer threads
    if(G.j>G.basepd && G.j<G.basepd+25)
    Thread.sleep(200);
    q2.nextround(3);
    if(G.j>G.basepd)
    G.coc=(float)Math.pow(1.06,((G.j-G.basepd)/4));
    else G.coc=1;
    // Wait for all the threads to finish mergers and acquisition process
    q1.determinewinner2(3);
    G.incyear=0;
    int merger=0;
   // Estimate the relative performance of all the suppliers, their ranking based on
   // their performance and score
   if(G.j>G.percmp)
       {qs.estrelativeperformance();
        qs.estrelativeperformance2();
        }
   ///Suppliers Merger
   String str5="";
   if(G.j>G.basepd+1)
   for(int t2=0; t2<qs.s1max;t2++)
   { if(qs.lhpor[qs.s1rank[t2]]>1 && g.Aliveagent[qs.s1rank[t2]]==1)
     {
      for(int temp3=0; temp3< qs.s2max;temp3++)
      {
          if(qs.score[qs.s2rank[temp3]][G.percmp-1] > 5*qs.score[qs.s1rank[t2]][G.percmp-1]
&& g.Aliveagent[qs.s1rank[t2]]==1 && g.Aliveagent[qs.s2rank[temp3]]==1 )
      {
       // Determine the Herfindahl index of the supply chain at suppliers level
          float hi=0;
```

```
         for(int j1=0; j1<G.agent; j1++)
        {if(g.Aliveagent[j1]==1 && j1!=qs.s1rank[t2] && j1!=qs.s2rank[temp3])
            {hi= hi+ (float)Math.pow((qs.score[j1][G.percmp-1]/qs.totalsc),2);
            }
         }
          hi= hi+(float)Math.pow(((qs.score[qs.s2rank[temp3]][G.percmp-
1]+qs.score[qs.s1rank[t2]][G.percmp-1])/qs.totalsc),2);
// Check if the merger will increase the concentration to decrease the herfindahl index
// by the minimum allowed concentration level
 if(hi < 0.2)    // ||(hi >=0.2 && sm[r3]!=scsort[G.man-1] && sm[r3]!=scsort[G.man-2] &&
sm[r3]!=scsort[G.man-3] && sm[r3]!=scsort[G.man-4] && sm[r3]!=scsort[G.man-5] &&
sm[r2]!=scsort[G.man-1] && sm[r2]!=scsort[G.man-2] && sm[r2]!=scsort[G.man-3] &&
sm[r2]!=scsort[G.man-4] && sm[r2]!=scsort[G.man-5]  ))
       {
         int ca2=a[qs.s1rank[t2]].ca[G.j]+a[qs.s2rank[temp3]].ca[G.j];
         int ca1=a[qs.s1rank[t2]].ca[G.j-1]+a[qs.s2rank[temp3]].ca[G.j-1];
         for(int h=0;h<G.man;h++)
            {
               if(g.Aliveman[h]==1)
                {
                for (int r11=0;r11<=g.p[G.j][h];r11++)
                   { if((g.wins[h][r11]==qs.s1rank[t2] || g.wins[h][r11]==qs.s2rank[temp3] )&&
(G.j-g.tr[h][r11])<G.bidlife && g.Aliveman[h]==1)
                     g.wins[h][r11]= G.ia+newsupp;
                   }
              int tempvar2= g.acts[qs.s1rank[t2]][h]+g.acts[qs.s2rank[temp3]][h];
              int tempvar =G.ia+newsupp;
              g.modifyacts(tempvar,h,tempvar2);
               }
              }
          float oldbid=a[qs.s2rank[temp3]].oldbid;
         float temp=(qs.score[qs.s2rank[temp3]][G.percmp-
1]+qs.score[qs.s1rank[t2]][G.percmp-1]);
           for(int r2=0;r2<G.percmp;r2++)
             {
qs.score[G.ia+newsupp][r2]=qs.score[qs.s1rank[t2]][r2]+qs.score[qs.s2rank[temp3]][r2];
qs.Highperf[G.ia+newsupp][r2]=(qs.Highperf[qs.s1rank[t2]][r2]*qs.score[qs.s1rank[t2]][r2]
+qs.Highperf[qs.s2rank[temp3]][r2]*qs.score[qs.s2rank[temp3]][r2])/(qs.score[G.ia+newsup
p][r2]);
              }
qs.performance[G.ia+newsupp]= (qs.score[G.ia+newsupp][G.percmp-1]-
qs.score[G.ia+newsupp][0])/(qs.score[G.ia+newsupp][G.percmp-1]);
```

```
 a[G.ia+newsupp]= new
Agent(1,temp,0,G.ia+newsupp,q,qs,ca1,ca2,oldbid,qs.performance[G.ia+newsupp],qs.score[
G.ia+newsupp],q1,q2,q3,g,profs,key1);
g.Aliveagent[G.ia+newsupp]=1;
int tempvar =G.ia+newsupp;
g.modifyalives(tempvar,1);
g.modifyaborn(tempvar,G.j);
  str5= " ";
str5="\n"+G.sim+"\t"+G.j+"\t"+"S0"+"\t"+qs.s1rank[t2]+"\t"+qs.s2rank[temp3]+"\t"+(G.ia+
newsupp)+"\t"+Stat.Round(qs.score[qs.s1rank[t2]][G.percmp-
1])+"\t"+Stat.Round(qs.score[qs.s2rank[temp3]][G.percmp-
1])+"\t"+Stat.Round(qs.score[G.ia+newsupp][G.percmp-1])+"\t";
mr.write(str5);
 a[qs.s1rank[t2]].s=0;
 a[qs.s2rank[temp3]].s=0;
 a[qs.s1rank[t2]].performance=0;
 a[qs.s2rank[temp3]].performance=0;
  g.Aliveagent[qs.s2rank[temp3]]=0;
  g.Aliveagent[qs.s1rank[t2]]=0;
  g.modifyalives(qs.s2rank[temp3],0);
  g.modifyalives(qs.s1rank[t2],0);
  a[qs.s2rank[temp3]].thrd.stop();
  a[qs.s1rank[t2]].thrd.stop();
  merger++;
// Update the raking and relativeperformance of the suppliers
 //      qs.estrelativeperformance();
 //      qs.estrelativeperformance2();
 // RANKS UPDATED
newsupp=newsupp+1;
Thread.sleep(100);
//          t2=0;
//          temp3=0;
      }
     }
    }
   }
   }
//MANUFACTURERS MERGERS
// Estimate the relative performance of all the manufacturers, their ranking based on
// their performance, score, and products
 if(G.j>G.percmp)
  {qm.estrelativeperformance();
    qm.estrelativeperformance2();
   }
 if(G.j> G.basepd+1)
 for(int t2=0; t2<qm.m1max;t2++)
   { if(qm.lhpor[qm.m1rank[t2]] >1 && g.Aliveman[qm.m1rank[t2]]==1)
     {
      for(int temp3=0; temp3<qm.m2max;temp3++)
    {
      if(qm.score[qm.m2rank[temp3]][G.percmp-1] > 5*qm.score[qm.m1rank[t2]][G.percmp-
1] && g.Aliveman[qm.m1rank[t2]]==1 && g.Aliveman[qm.m2rank[temp3]]==1)
      {  float hi=0;
          for(int j1=0; j1<G.man; j1++)
          {if(g.Aliveman[j1]==1 && j1!=qm.m1rank[t2] && j1!=qm.m2rank[temp3])
              {hi= hi+ (float)Math.pow((qm.score[j1][G.percmp-1]/qm.totalsc),2);
              }
          }
 hi= hi+(float)Math.pow(((qm.score[qm.m2rank[temp3]][G.percmp-
1]+qm.score[qm.m1rank[t2]][G.percmp-1])/qm.totalsc),2);
// Check if the merger would not increase the concentration to decrease the herfindahl index
// by the minimum allowed concentration level
 if(hi < 0.2)    // ||(hi >=0.2 && sm[r3]!=scsort[G.man-1] && sm[r3]!=scsort[G.man-2] &&
sm[r3]!=scsort[G.man-3] && sm[r3]!=scsort[G.man-4] && sm[r3]!=scsort[G.man-5] &&
sm[r2]!=scsort[G.man-1] && sm[r2]!=scsort[G.man-2] && sm[r2]!=scsort[G.man-3] &&
sm[r2]!=scsort[G.man-4] && sm[r2]!=scsort[G.man-5]  ))
        {
         int r11=1;
           g.p[G.j-G.bidlife-1][G.im+newman]=0;
               nme= new int[G.preapproval.length+1];
            for(int r5=0;r5<G.preapproval.length+1;r5++)
              {nme[r5]=ma[qm.m1rank[t2]].nme[r5] +ma[qm.m2rank[temp3]].nme[r5];
              }
            for(int r5=0;r5<G.percmp;r5++)
        {qm.score[G.im+newman][r5]= qm.score[qm.m1rank[t2]][r5]
+qm.score[qm.m2rank[temp3]][r5];
 qm.Highperf[G.im+newman][r5]=
(qm.Highperf[qm.m1rank[t2]][r5]*qm.score[qm.m1rank[t2]][r5]
+qm.Highperf[qm.m2rank[temp3]][r5]*qm.score[qm.m2rank[temp3]][r5])/(qm.score[G.im+
newman][r5]);

qm.Highperfp[G.im+newman][r5]=(qm.Highperfp[qm.m1rank[t2]][r5]*qm.score[qm.m1ran
k[t2]][r5]
+qm.Highperfp[qm.m2rank[temp3]][r5]*qm.score[qm.m2rank[temp3]][r5])/(qm.score[G.im
+newman][r5]);
            }
            qm.performance[G.im+newman]=(qm.score[G.im+newman][G.percmp-1]-
qm.score[G.im+newman][0])/qm.score[G.im+newman][G.percmp-1];
        qm.performancep[G.im+newman]= (nme[16]
+nme[17]+nme[18]+nme[19]+nme[20]+nme[21]+nme[22]+nme[23]+nme[24]+nme[25]+nm
e[26]+nme[27]+nme[28]+nme[29]+nme[30]+nme[31]
)/(qm.score[G.im+newman][G.percmp-1]);
            float temp =qm.score[qm.m2rank[temp3]][G.percmp-
1]+qm.score[qm.m1rank[t2]][G.percmp-1];
            int tempvar=G.im+newman;
            System.out.println("Main thread manufacturer"+tempvar);
            ma[G.im+newman]= new
Manufacturer(1,temp,tempvar,q,qm,nme,qm.score[G.im+newman],qm.performance[G.im+n
ewman],qm.performancep[G.im+newman],q1,q2,q3,g,profm,key1,dbid,Rd);

                        String str8= "";
                        str8=
"\n"+G.sim+"\t"+G.j+"\t"+"M0"+"\t"+qm.m2rank[temp3]+"\t"+qm.m1rank[t2]+"\t"+(G.im+
newman)+"\t"+Stat.Round(qm.score[qm.m2rank[temp3]][G.percmp-
1])+"\t"+Stat.Round(qm.score[qm.m1rank[t2]][G.percmp-
1])+"\t"+Stat.Round(qm.score[G.im+newman][G.percmp-1]);
                mr.write(str8);
              qm.score[qm.m1rank[t2]][G.percmp-1]=0;
              qm.score[qm.m2rank[temp3]][G.percmp-1]=0;
              qm.performance[qm.m1rank[t2]]=0;
              qm.performance[qm.m2rank[temp3]]=0;
              qm.performancep[qm.m1rank[t2]]=0;
              qm.performancep[qm.m2rank[temp3]]=0;
              ma[qm.m1rank[t2]].sm=0;
              ma[qm.m2rank[temp3]].sm=0;
              tempvar=G.im+newman;
              g.modifyalivem(tempvar,1);
              g.modifymborn(tempvar,G.j);
              g.modifyalivem(qm.m1rank[t2],0);
              g.modifyalivem(qm.m2rank[temp3],0);
               // qm.merge[tempvar]=1;
             merger++;
             tempvar = G.im+newman;
             g.p[G.j-G.bidlife-1][G.im+newman]=0;
             for(int r5=1;r5<=G.j;r5++)
             {g.p[r5][G.im+newman]= g.p[r5][qm.m1rank[t2]]+g.p[r5][qm.m2rank[temp3]];

             if((g.p[r5][G.im+newman]-g.p[r5-1][G.im+newman])>0)
             for(int r6=g.p[r5-1][G.im+newman]+1;r6<=g.p[r5][G.im+newman];r6++)
```

```
                    {
                      g.modifyyear(tempvar,r6,r5);
                     }
          for(int ct=g.p[r5-1][qm.m1rank[t2]]+1;ct<=g.p[r5][qm.m1rank[t2]];ct++)
          {
             g.modifybase(tempvar,r11,g.b[qm.m1rank[t2]][ct]);
             g.wins[G.im+newman][r11]=g.wins[qm.m1rank[t2]][ct];
             for(int lo=0;lo<G.Region;lo++)
               { g.wind[lo][G.im+newman][r11]=g.wind[lo][qm.m1rank[t2]][ct];
             g.modifynb(lo,tempvar,r11,g.nb[lo][qm.m1rank[t2]][ct]);
               }
             r11=r11+1;
          }
          for(int ct=g.p[r5-1][qm.m2rank[temp3]]+1;ct<=g.p[r5][qm.m2rank[temp3]];ct++)
          { g.modifybase(tempvar,r11,g.b[qm.m2rank[temp3]][ct]);
            g.wins[G.im+newman][r11]=g.wins[qm.m2rank[temp3]][ct];
            for(int lo=0;lo<G.Region;lo++)
              {g.wind[lo][G.im+newman][r11]=g.wind[lo][qm.m2rank[temp3]][ct];
                g.modifynb(lo,tempvar,r11,g.nb[lo][qm.m2rank[temp3]][ct]);
              }
                ma[qm.m1rank[t2]].thrd.stop();
                ma[qm.m2rank[temp3]].thrd.stop();
                r11=r11+1;
          }

         }
// Estimate the relative performance of all the manufacturers, their ranking based on
// their performance, score, and products
  // qm.estrelativeperformance();
  // qm.estrelativeperformance2();
   newman=newman+1;
   Thread.sleep(100);
 //  t2=0;
 //  temp3=0;
      }
      }
     }
    }
   }

// DISTRIBUTORS MERGER
// Estimate the relative performance of all the distributors, their ranking based on
// their performance, and score
```

```
 if(G.j>G.percmp)
  {qd.estrelativeperformance();
   qd.estrelativeperformance2();
   }

if(G.j>G.basepd)
 for(int loc=0; loc<G.Region;loc++)
{
 for(int t2=0; t2< qd.locmax1[loc];t2++)
   {
    if(qd.lhpor[loc][qd.d1rank[loc][t2]]>1 && g.Alivedist[loc][qd.d1rank[loc][t2]]==1)
     {
      for(int temp3=0; temp3< qd.locmax2[loc];temp3++)
    { if(qd.score[loc][qd.d2rank[loc][temp3]][G.percmp-1] >
5*qd.score[loc][qd.d1rank[loc][t2]][G.percmp-1] &&
g.Alivedist[loc][qd.d1rank[loc][t2]]==1 && g.Alivedist[loc][qd.d2rank[loc][temp3]]==1 )
      {
// Determine the herfindahl index at distributor level
            float hi=0;
            for(int j1=0; j1<G.distributor; j1++)
             {if(g.Alivedist[loc][j1]==1 && j1!=qd.d1rank[loc][t2] &&
j1!=qd.d2rank[loc][temp3])
                      {hi= hi+ (float)Math.pow((qd.score[loc][j1][G.percmp-1]/qd.totalsc[loc]),2);
                      }
             }

             hi= hi+(float)Math.pow(((qd.score[loc][qd.d2rank[loc][temp3]][G.percmp-
1]+qd.score[loc][qd.d1rank[loc][t2]][G.percmp-1])/qd.totalsc[loc]),2);

// Check if the merger would not increase the concentration to decrease the herfindahl index
// by the minimum allowed concentration level
   if(hi < 0.4)    // ||(hi >=0.2 && sm[r3]!=scsort[G.man-1] && sm[r3]!=scsort[G.man-2] &&
sm[r3]!=scsort[G.man-3] && sm[r3]!=scsort[G.man-4] && sm[r3]!=scsort[G.man-5] &&
sm[r2]!=scsort[G.man-1] && sm[r2]!=scsort[G.man-2] && sm[r2]!=scsort[G.man-3] &&
sm[r2]!=scsort[G.man-4] && sm[r2]!=scsort[G.man-5]  ))
       {
      float temp=qd.score[loc][qd.d1rank[loc][t2]][G.percmp-
1]+qd.score[loc][qd.d2rank[loc][temp3]][G.percmp-1];

   sum = new float[G.man];
   count = new int[G.man];
   for(int h=0;h<G.man;h++)
          {
                if(g.Aliveman[h]==1)
                 { int temp1 = G.id[loc]+newdist[loc];
                 int
tempvar3=g.actd[loc][qd.d1rank[loc][t2]][h]+g.actd[loc][qd.d2rank[loc][temp3]][h];
                 g.modifyactd(loc,temp1,h,tempvar3);
                 for (int r11=0;r11<=g.p[G.j][h];r11++)
              { if(( g.wind[loc][h][r11]==qd.d1rank[loc][t2] ||
g.wind[loc][h][r11]==qd.d2rank[loc][temp3] )&& (G.j-g.tr[h][r11])<G.bidlife &&
g.Aliveman[h]==1)
                         {
                              temp1 = G.id[loc]+newdist[loc];
                              g.modifywind(loc,h,r11,temp1);
                         }
             }
               float oldbid = de[loc][qd.d2rank[loc][temp3]].oldbid;

sum[h]=((de[loc][qd.d1rank[loc][t2]].sum[h])*(de[loc][qd.d1rank[loc][t2]].count[h])+(de[loc
][qd.d2rank[loc][temp3]].sum[h])*(de[loc][qd.d2rank[loc][temp3]].count[h]))/(de[loc][qd.d2
rank[loc][temp3]].count[h]+de[loc][qd.d1rank[loc][t2]].count[h]);

count[h]=(de[loc][qd.d2rank[loc][temp3]].count[h]+de[loc][qd.d1rank[loc][t2]].count[h]);
                 }
                }
      for(int r2=0;r2<G.percmp;r2++)
       {qd.score[loc][G.id[loc]+newdist[loc]][r2]= qd.score[loc][qd.d2rank[loc][temp3]][r2]
+qd.score[loc][qd.d1rank[loc][t2]][r2];
       qd.Highperf[loc][G.id[loc]+newdist[loc]][r2]=
(qd.Highperf[loc][qd.d2rank[loc][temp3]][r2]*qd.score[loc][qd.d2rank[loc][temp3]][r2]
+qd.Highperf[loc][qd.d1rank[loc][t2]][r2]*qd.score[loc][qd.d1rank[loc][t2]][r2])/(qd.score[l
oc][G.id[loc]+newdist[loc]][r2]);
       }
      qd.performance[loc][G.id[loc]+newdist[loc]]=
(qd.score[loc][G.id[loc]+newdist[loc]][G.percmp-1]-
qd.score[loc][G.id[loc]+newdist[loc]][0])/(qd.score[loc][G.id[loc]+newdist[loc]][G.percmp-
1]);


              de[loc][G.id[loc]+newdist[loc]]= new
distributor(G.dcn[loc][G.id[loc]+newdist[loc]],loc,
G.id[loc]+newdist[loc],0,1,temp,G.id[loc]+newdist[loc],q,qd,qd.performance[loc][G.id[loc]+
newdist[loc]],qd.score[loc][G.id[loc]+newdist[loc]],sum,count,q1,q2,q3,g,profd,key1);

             String str8="";
             str8=
"\n"+G.sim+"\t"+G.j+"\t"+"D"+loc+"\t"+qd.d1rank[loc][t2]+"\t"+qd.d2rank[loc][temp3]+"\t
"+(G.id[loc]+newdist[loc])+"\t"+Stat.Round(qd.score[loc][qd.d1rank[loc][t2]][G.percmp-
1])+"\t"+Stat.Round(qd.score[loc][qd.d2rank[loc][temp3]][G.percmp-
1])+"\t"+Stat.Round(qd.score[loc][G.id[loc]+newdist[loc]][G.percmp-1]);
           mr.write(str8);
       qd.performance[loc][qd.d2rank[loc][temp3]]=0;
       qd.performance[loc][qd.d1rank[loc][t2]]=0;
       qd.score[loc][qd.d1rank[loc][t2]][G.percmp-1]=0;
       qd.score[loc][qd.d2rank[loc][temp3]][G.percmp-1]=0;
       de[loc][qd.d1rank[loc][t2]].sr=0;
       de[loc][qd.d2rank[loc][temp3]].sr=0;
       g.modifyalived(loc,qd.d1rank[loc][t2],0);
       g.modifyalived(loc,qd.d2rank[loc][temp3],0);
       int tempvar=G.id[loc]+newdist[loc];
       g.modifyalived(loc,tempvar,1);
       g.modifydborn(loc,tempvar,G.j);
       de[loc][qd.d1rank[loc][t2]].thrd.stop();
       de[loc][qd.d2rank[loc][temp3]].thrd.stop();
       merger++;
// Estimate the relative performance of all the distributors, their ranking based on
// their performance, and score
   //   qd.estrelativeperformance();
   //   qd.estrelativeperformance2();

       newdist[loc]=newdist[loc]+1;
       Thread.sleep(100);
 //         t2=0;
 //         temp3=0;
      }

     }
    }

   }
   }
}

String str10= "\n"+G.sim+"\t"+G.j;
for(int i=0;i<G.man;i++)
 {
    str10=str10+"\t"+g.p[G.j][i];
 }
```

```
  pr.write(str10);
  str10="";
  tpr=0;
 for(int i=0;i<G.man;i++)
 {if(g.Aliveman[i]==1)
   {
     tpr= tpr+g.p[G.j][i]-g.p[G.j-1][i];
   }
 }

for(int i=0;i<G.man;i++)
 {if(g.Aliveman[i]==1)
  {
    String str33="";
    str33=
"\n"+G.sim+"\t"+G.j+"\t"+i+"\t"+ma[i].nme[0]+"\t"+ma[i].nme[1]+"\t"+ma[i].nme[2]+"\t"+
ma[i].nme[3]+"\t"+ma[i].nme[4]+"\t"+ma[i].nme[5]+"\t"+ma[i].nme[6]+"\t"+ma[i].nme[7]+"
\t"+ma[i].nme[8]+"\t"+ma[i].nme[9]+"\t"+ma[i].nme[10]+"\t"+ma[i].nme[11]+"\t"+ma[i].nm
e[12]+"\t"+ma[i].nme[13]+"\t"+ma[i].nme[14]+"\t"+ma[i].nme[15]+"\t"+ma[i].nme[16]+"\t"
+ma[i].nme[17]+"\t"+ma[i].nme[18]+"\t"+ma[i].nme[19]+"\t"+ma[i].nme[20]+"\t"+ma[i].nm
e[21]+"\t"+ma[i].nme[22]+"\t"+ma[i].nme[23]+"\t"+ma[i].nme[24]+"\t"+ma[i].nme[25]+"\t"
+ma[i].nme[26]+"\t"+ma[i].nme[27]+"\t"+ma[i].nme[28]+"\t"+ma[i].nme[29]+"\t"+ma[i].nm
e[30]+"\t"+ma[i].nme[31]+"\t"+ma[i].nme[32]+"\t";
    prdev.write(str33);
   }
  }

  String str8="";
str8="\n"+sn+"\t"+G.j;
for(int loc=0;loc<G.Region;loc++)
{  str8="";
  str8="\n"+sn+"\t"+G.j;
for(int h2=0;h2<G.distributor;h2++)
   if(g.Alivedist[loc][h2]==1)
   str8=str8+"\t"+Stat.Round(de[loc][h2].sr);
     else str8= str8+"\t"+0;
     if(loc==0)
     { wd1.write(str8);
       }

     if(loc==1)
     {wd2.write(str8);
      }
      if(loc==2)
      {wd3.write(str8);
      }
      if(loc==3)
      {wd4.write(str8);
      }
}
   String str7="";
   str7= "\n"+0+"\t"+G.j;
   for(int r2=0;r2<G.agent;r2++)
     if(g.Aliveagent[r2]==1)
   str7= str7+"\t"+Stat.Round(a[r2].s);
    else  str7=str7+"\t"+0;

   str8="";
   str8= "\n"+0+"\t"+G.j;
   for(int r2=0;r2<G.man;r2++)
   if(g.Aliveman[r2]==1)
    str8=str8+"\t"+Stat.Round(ma[r2].sm);
    else str8=str8+"\t"+0;

   wm.write(str8);
   ws.write(str7);

     float[] arr= new float[Stat.sumi(g.Aliveman)];
     float[] arr1= new float[Stat.sumi(g.Aliveman)];
     int cnt=0;
     for (int i=0; i < G.man; i++)
                    {if( g.Aliveman[i]==1)
                            {arr[cnt] = ma[i].sm;
                        arr1[cnt] = g.Aliveman[i];
                             cnt++;
                            }
                    }
String rpt="";
 str8="";
 str8= "\n"+G.sim+"\t"+
G.j+"\t"+Stat.Round(Stat.sum(arr))+"\t"+tpr+"\t"+Stat.sum(arr1)+"\t"+Stat.Round(Stat.mini
mum(arr))+"\t"+Stat.Round(Stat.maximum(arr))+"\t"+Stat.Round(Stat.mean(arr,arr1))+"\t"+
Stat.Round(Stat.stdev(arr,arr1))+"\t";
   msry.write(str8);
   rpt= str8;
   str8="";
```

```
  arr= new float[Stat.sumi(g.Aliveagent)];
  arr1= new float[Stat.sumi(g.Aliveagent)];
  cnt=0;
  for (int i=0; i < G.agent; ++i)
                {if( g.Aliveagent[i]==1)
                        {arr[cnt] = a[i].s;
                      arr1[cnt] = g.Aliveagent[i];
                         cnt++;
                        }
                }
 str8= "\n"+G.sim+"\t"+
G.j+"\t"+Stat.Round(Stat.sum(arr))+"\t"+tpr+"\t"+Stat.sum(arr1)+"\t"+Stat.Round(Stat.mini
mum(arr))+"\t"+Stat.Round(Stat.maximum(arr))+"\t"+Stat.Round(Stat.mean(arr,arr1))+"\t"+
Stat.Round(Stat.stdev(arr,arr1))+"\t";
  ssry.write(str8);

rpt=rpt+"\t"+Stat.Round(Stat.sum(arr))+"\t"+tpr+"\t"+Stat.sum(arr1)+"\t"+Stat.Round(Stat.m
inimum(arr))+"\t"+Stat.Round(Stat.maximum(arr))+"\t"+Stat.Round(Stat.mean(arr,arr1))+"\t
"+Stat.Round(Stat.stdev(arr,arr1))+"\t";;
  for (int temp=0; temp < G.Region; ++temp)
  {arr= new float[Stat.sumi(g.Alivedist[temp])];
   arr1= new float[Stat.sumi(g.Alivedist[temp])];
  cnt=0;
  for (int i=0; i < G.distributor; ++i)
                {if( g.Alivedist[temp][i]==1)
                        {arr[cnt] = de[temp][i].sr;
                    arr1[cnt] = g.Alivedist[temp][i];
                         cnt++;
                        }
                }
  str8="";
  str8= "\n"+G.sim+"\t"+
G.j+"\t"+Stat.Round(Stat.sum(arr))+"\t"+tpr+"\t"+Stat.sum(arr1)+"\t"+Stat.Round(Stat.mini
mum(arr))+"\t"+Stat.Round(Stat.maximum(arr))+"\t"+Stat.Round(Stat.mean(arr,arr1))+"\t"+
Stat.Round(Stat.stdev(arr,arr1))+"\t";

rpt=rpt+"\t"+Stat.Round(Stat.sum(arr))+"\t"+tpr+"\t"+Stat.sum(arr1)+"\t"+Stat.Round(Stat.m
inimum(arr))+"\t"+Stat.Round(Stat.maximum(arr))+"\t"+Stat.Round(Stat.mean(arr,arr1))+"\t
"+Stat.Round(Stat.stdev(arr,arr1))+"\t";
  if(temp==0)
  d1sry.write(str8);
  if(temp==1)
  d2sry.write(str8);
```

```
   if(temp==2)
   d3sry.write(str8);
   if(temp==3)
   d4sry.write(str8);
     }
 report.write(rpt);
  String strg="";
  String strgr="";
  String strgr8="";
   for(int i=0;i< G.man;i++)
   {
     if((g.p[G.j][i]-g.p[G.j-1][i])>0 && g.Aliveman[i]==1)
             {for(int count1=g.p[G.j-1][i]+1;count1<=g.p[G.j][i];count1++)
                {
              if(g.b[i][count1]==0)
                   strgr=strgr +"\t"+1;
              if(g.b[i][count1]==1)
                   strgr=strgr +"\t"+2;
              if(g.b[i][count1]==2)
                   strgr=strgr +"\t"+3;

/// WRITING SUPPLIERS BID AND WINNER SUPPLIER TO THE FILE
SBIDDING.TXT
                                    String str18="";
                                    str18= "\n"+G.sim+"\t"+ G.j+"\t"+"S0"+"\t"+i+"\t"+count1+"\t"+
g.wins[i][count1]+"\t";
                                    for(int dct= 0; dct<G.agent; dct++)
                                    {str18= str18+g.rfq[i][count1][dct] +"\t";
                                    }
                                    sbid.write(str18);
///WRITING DISTRIBUTORS BID AND WINNER DISTRIBUTOR IN EACH REGION
TO THE FILE DBIDDING.TXT.
                }
              strg=strg +"\t"+(g.p[G.j][i]-g.p[G.j-1][i]);
             }
     else{strg=strg+"\t"+0;
             }
   }
   str8= "\n"+G.sim+"\t"+ G.j+"\t"+tpr+strg+"\t";
   product.write(str8);
   tpr=0;
String str3="";
if(G.j>=G.basepd)
```

```
for(int loc=0;loc<G.Region;loc++)
{
for(int d1=0;d1<G.distributor;d1++)
{   if(g.Alivedist[loc][d1]==1)
      {String keyd= "";
      keyd = key1.Keygenerator(G.sim,G.j,0,0,1,d1,loc);
      str8="";
      str8=
"\n"+keyd+"\t"+G.j+"\t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+loc+"\t"+0+"\t"+d1+"\t"+0+"\t"+0+"\
t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+Stat.Round(de[loc][d1].sr);
      es.write(str8);
      }
}
}
if(G.j>=G.basepd)
for(int r2=0;r2<G.agent;r2++)
  {if(g.Aliveagent[r2]==1)
    {
    str3="";
    String key ="";
    key = key1.Keygenerator(G.sim,G.j,0,0,2,r2,0);
    str3 =
"\n"+key+"\t"+G.j+"\t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+00+"\t"+00+"\t"+r2+"\t"+0+"\t"+0+"\t"+0+"\t
"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+Stat.Round(a[r2].s);
    es.write(str3);
    }
  }
if(G.j>=G.basepd)
for(int r2=0;r2<G.man;r2++)
  {if(g.Aliveman[r2]==1)
    {
         String key ="";
         key = key1.Keygenerator(G.sim,G.j,0,0,3,r2,0);
         int drugs= g.p[G.j][r2]-g.p[G.j-1][r2];
         str3="";
         str3="\n"+key+"\t"+G.j+"\t"+0+"\t"+0+"\t"+drugs+"\t"+0+"\t"+00+"\t"+00+"\t"+r2+"
\t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+0+"\t"+Stat.Round(ma[r2].sm);
         es.write(str3);
  }
  }

           float infla= (float)Math.pow(1.09,((G.j-G.basepd)/4));
            G.totalsc=0;
           for(int n=0;n< 3 ;n++)
           {if(ma[n].alivem==1)
           G.totalsc= G.totalsc+ma[n].sm;
           }
       G.totalsc1= 0;                          //assigninig totalsc= 0 at the beginning of
each round
           for(int n=0;n<3 ;n++)
             {   if(ma[n].alivem==1 && ma[n].sm >1 )
                   G.totalsc1= G.totalsc1+(float)Math.log(ma[n].sm);
             }
         }
```

```
/// Stopping all the other threads before the main thread stops

                    for(int n=0;n<G.man;n++)
                    {  //if(G.Aliveman[n]==1)
                       try { g.modifyalivem(n,0);
                               do{ma[n].thrd.stop();
                                  } while(ma[n].thrd.isAlive());
                                 ma[n].thrd.join();
                      }catch (Exception e) {
                    }

                    }
                for(int n=0;n<G.agent;n++)
                    {
                    try {  g.modifyalives(n,0);
                               do{a[n].thrd.stop();
                               }while(a[n].thrd.isAlive());
                      a[n].thrd.join();
                    }catch (Exception e) {
                    }
                    }
                    for(int n1=0;n1<G.Region;n1++)
                    {
                    for(int n=0;n<G.distributor;n++)
                     {
                       try { g.modifyalived(n1,n,0);
                               do{de[n1][n].thrd.stop();
                           }while(de[n1][n].thrd.isAlive());
                                de[n1][n].thrd.join();
                            }catch (Exception e) {
                        }
                          }
                   }

              g= null;
              q= null;
              q1= null;
              q2= null;
              q3= null;
              qs= null;
              qm= null;
              qd= null;

         z.gc();
/// Stopping all the other threads before the main thread stops
 }
 pr.flush();
 Rd.flush();
  wm.flush();
 ws.flush();
 wd1.flush();
 wd2.flush();
 wd3.flush();
 wd4.flush();
 mr.flush();
 profs.flush();
 profd.flush();
 profm.flush();
 es.flush();
 msry.flush();
 ssry.flush();
 d1sry.flush();
 d2sry.flush();
 d3sry.flush();
 d4sry.flush();
 product.flush();
 sbid.flush();
 testing.flush();
 prdev.flush();
 dbid.flush();
 report.flush();
 Rd.close();
 pr.close();
 wm.close();
 ws.close();
 wd1.close();
 wd2.close();
 wd3.close();
 wd4.close();
 mr.close();
 profs.close();
 profd.close();
 profm.close();
 es.close();
 msry.close();
 ssry.close();
 d1sry.close();
 d2sry.close();
 d3sry.close();
 d4sry.close();
 product.close();
 sbid.close();
 testing.close();
 prdev.close();
 dbid.close();
 report.close();
    }
}
```

2. Manufacturer class

```
import java.io.*;
import java.util.*;
import java.lang.Object.*;
import java.math.*;
import java.lang.*;

public class Manufacturer implements Runnable{
  Glob g;
  Q q2;
  Q q3;
  Q q;
  Q q1;
  Qm qm;
  Thread thrd;
  float c1=0;
  float minb;
  float maxb;
  int win1;
  int win2;
  int x=0;
  float y=0;
  int z=0;
  public int alivem;          // Binary variable to determine if a manufacturer is alive
  public float sm;            //array to store score of manufacturer after each round
  public float[] sm1= new float[G.percmp];
  public float performance;     // Array to store the performance of each manufacturer after
each round.(= no of drugs launched in last 8 rounds/ Current assets)
  public float performancep;   //performance based on number of products in pipeline
  public int[] Highperfp= new int[G.percmp];
  public float cmsc;
  public int[] Highperf= new int[G.percmp];      // If rank > median then Highperf=1 else 0
  public float[] mins= new float[G.spacealloc];              // Array to store the minimum of
bid values of Suppliers in each round
  public float[][] max=new float[G.bid][G.Region];        //array to store maximum of the
bid values of distributors in each round
  public int[] nme = new int[G.preapproval.length+1];
  private float cost;
  private float coc;
  private int productcount;
  private int tempproduct;
  private float tempnum;
  private float tempscore=0;
  public int index;
  Random generator = new Random();
  public int rcount;
  float Rnd;   // Variable to store the capital allocated for R&D
  int mesg;
  BufferedWriter profm;
  keygen key1;
  BufferedWriter dbid;
  BufferedWriter Rd;

Manufacturer(int alive, float score,int ind,Q q,Qm qm,Q q1,Q q2,Q q3,Glob
g,BufferedWriter profm, keygen key1,BufferedWriter dbid,BufferedWriter Rd)
   { thrd = new Thread(this);
     this.alivem= alive;
     this.sm= score;
     this.index= ind;
     thrd.start();
     this.rcount=G.j-1;
     this.mesg= 0;
     this.g=g;
     this.q=q;
     this.qm=qm;
     this.q1=q1;
     this.q2=q2;
     this.q3=q3;
     this.profm=profm;
     this.key1= key1;
     this.dbid= dbid;
     this.Rd=Rd;
     this.tempscore=sm;

   }
  Manufacturer(int alive, float s,int ind,Q q,Qm qm,int[] nm,float[] score,float perf,float
perfp,Q q1,Q q2,Q q3, Glob g,BufferedWriter profm, keygen key1,BufferedWriter
dbid,BufferedWriter Rd)
   { thrd = new Thread(this);
     this.alivem= alive;
     this.sm= s;
     this.index= ind;
     thrd.start();
     this.rcount=G.j;
     this.mesg= 0;
     this.g=g;
     this.q=q;
     this.q1=q1;
     this.qm=qm;
     this.q2=q2;
     this.q3=q3;
     this.performance=perf;
     this.performancep=perfp;
     this.profm=profm;
     this.key1= key1;
     this.dbid= dbid;
     this.Rd=Rd;
     this.tempscore=sm;
     g.modifyalivem(index,1);
     for(int r5=0;r5<G.percmp;r5++)
     {sm1[r5]=score[r5];
     }
     for(int r5=0;r5<G.preapproval.length+1;r5++)
     {this.nme[r5]=nm[r5];
     }

    }
    public void run(){
           do{   do{
               }while(G.start==0);

                     if(g.mborn[index]==1)
                     {
                      if(G.j<= G.basepd)
                       {
                        Rnd= 1.57f*((float)Math.sqrt(sm))+700;
                        }
                       else
                       { // float year= Stat.Round((G.j-G.basepd)/4);
                         // float percrnd= (0.16f*year+12)/100;
                            float percrnd=0.15f;       // Represents the percentage of sales
manufacturers will use for R&D
// If the R&D expenses allocated are insufficient to completely test all the drugs in stage 3
ofclinical stage
// then they estimate the pipeline cost which they can support using their assets.
if(percrnd*(sm1[G.percmp-1]-tempscore)> pipelinecost(G.j,(G.successrate.length-4)))
                                  { Rnd= percrnd*(sm1[G.percmp-1]-tempscore);
                                  }
                          else {
                              Rnd=0;
                              int counter=G.successrate.length-1;
                              while(Rnd==0 && alivem==1 && counter>0)
                              {   if(pipelinecost(G.j, counter)> 0)
                                    {   if(pipelinecost(G.j, counter) < sm)
                                             Rnd=pipelinecost(G.j,counter);
                                        else{ alivem=0;
                                                 sm=0;
                                             }
                               }
                              counter=counter-1;
                              }
                         }
                      }
// This part of the code writes their R&D expenses to the text file "RnD"
                   float temprnd=Stat.Round(Rnd);
                   float tempsales=Stat.Round(sm1[G.percmp-1]-tempscore);
                   float tempratio=Stat.Round(Rnd/(sm1[G.percmp-1]-tempscore));
String str3="";
str3="\n"+G.sim+"\t"+G.j+"\t"+index+"\t"+temprnd+"\t"+tempsales+"\t"+tempratio;
                      try{
                              Rd.write(str3);
                              }catch(Exception e) {}
                  float clintrialcost=0;
// Determine the number of NME's manufacturer can test based on the capital left
// after testing the drugs in the later stages of drug development
                  if(alivem==1)
                    {
                      if(G.j > G.basepd)
                      {nme[0]= approval(G.j, Rnd);
                      }
                      else
                      {nme[0]= basepdapproval(G.j, Rnd);
                      }
                  }
// Products which clear stage 3 of clinical trial are launched in the market
                      g.p[G.j][this.index] =  g.p[G.j-1][this.index]+ nme[G.successrate.length];
                      nme[G.successrate.length]=0;
//Update the score of the manufacturer
                      if(G.j> G.basepd)
                      {scoreupdate();
                      }
// Assign category to the new products launched: small, medium , blockbustor
```

```
                        assignproductcategory();
// Inform Suppliers and distributors to bid for the new product
                        q.put(0,index,2);
// Wait till the bids from all the suppliers and distributors is received
                        q3.determinewinner(2);
// Determine the winner supplier for each product
                        float temv;
                        for(int n1= g.p[G.j-1][this.index]+1;n1<= g.p[G.j][this.index];n1++)
                        {temv=mini(n1,0,1,g.rfq[index][n1],G.agent);
                        g.modifywinbids(index,n1,temv);
                        }
// Determine the winner distributor for each product in each location and also
// determine the next bid year for the existing product
                      int start=0;
                       for(int loc=0;loc<G.Region;loc++)
                        {if(G.j> G.bidlife)
                                   { start= g.p[G.j-G.bidlife][this.index];
                                   }
                                   else start = g.p[G.successrate.length][this.index];
                    for(int n1= start+1;n1<= g.p[G.j][this.index];n1++)
                        {   if(g.nb[loc][index][n1]==G.j)
                            {
                            temv=mini(n1,1,loc,g.drfq[index][n1][loc],G.distributor);
                            g.modifywinbidd(loc,index,n1,temv);
                            int nxtbdyr = nextbidyear(g.tr[index][n1],G.j,temv,g.b[index][n1],loc);
                            g.modifynb(loc,index,n1,nxtbdyr);

                           String str18="";
                           str18= "\n"+G.sim+"\t"+ G.j+"\t"+loc+"\t"+index+"\t"+n1+"\t"+
g.wind[loc][index][n1]+"\t"+g.nb[loc][index][n1]+"\t";
                                    for(int dct= 0; dct<G.distributor; dct++)
                                    {str18= str18+g.drfq[index][n1][loc][dct] +"\t";
                                    }
                         try {
                                    dbid.write(str18);
                                    } catch (Exception ex){
                                                  }

                              }
                        }
                        }

// Estimate the performance relative to other manufacturers. This is used later
// to determine a good candidate for merger
            if(G.j> G.percmp)
                      {
                      Relativeperformance();
                      }
                      rcount++;
                      q1.determinewinner2(2);
             }
             System.out.println(" Manufacturer no:"+index);
              if(g.mborn[index]!=1)
              {
               g.modifymborn(index,1);
              }
             if(rcount >= G.j)
             q2.nextround(2);
            if(sm<=0 || g.Aliveman[index]==0)
                        {
                         g.modifyalivem(index,0);
                         rcount=G.bid+1;
                         alivem=0;
                         sm=0;
                         try {
                                    thrd.stop();
                                  } catch (Exception ex){
                            }

                        }

             }
             while(rcount< (G.bid));
             g.modifyalivem(index,0);

       try {thrd.stop();
             thrd.interrupt();
             thrd.join();
       } catch (InterruptedException ex){
       }
       }
// Mini: This function is used to determien the winner supplier or winner
// distributor based on the bid values received from all the agents
// Inputs: Supplier/Distributor, location, bids array, Survivors
       public float mini(int h,int u,int loc, float[] b2,int size)
       {   minb=100f;
```

```
        maxb=0f;
        int t5=0;  // start with the first value
        win1=0;
           win2=0;
  // Estimate winner Supplier and winning bid value
  //*Find out if there are more than 1 supplier with the minimum bid value*//
  //*In case there are 2 bids with the same value compare Their active bids *//
        if(u==0)
        {for (int c2=0; c2< size; c2++)
                    {   if(g.Aliveagent[c2]==1)
                 if (b2[c2] < minb)
                          {
                        minb= b2[c2];
                        win1=c2;
                    }
               }
        if(minb==100)
          minb=0;
          for(int c2=0;c2<size;c2++)
           {if(b2[c2]==minb && c2!=win1 && g.Aliveagent[c2]==1 )
            { if( g.acts[c2][index]> g.acts[win1][index])
              {     win1=c2;
              }
            }
           }

       g.modifywins(index,h,win1);
        return( minb);
        }
   // Estimate winner distributor
   //*Find out if there are more than 1 term with the maximum bid value*//
   //*In case there are 2 bids with the same value compare Their ACTIVE BIDS *//
         if(u==1)
          {  for (int c2=0; c2< size; c2++)
                         { if(g.Alivedist[loc][c2]==1)
                       if (b2[c2] > maxb)
                                   {
                        maxb= b2[c2];
                        win2=c2;
                       }
                     }
          for(int c2=0;c2<size;c2++)
                    {if(b2[c2]==maxb && c2!=win2 && g.Alivedist[loc][c2]==1)
                     { if( g.actd[loc][c2][index]> g.actd[loc][win2][index])
                       {      win2=c2;
                       }
                     }
                    }
            g.modifywind(loc,index,h,win2);
              return( maxb);
              }
            return( 0);
     }
   public int winner(int u,int loc)
     { if(u==0)
       return (win1);
      else
       return (win2);
     }

   // Distributors can win a contratc for a maximum of 2 years. Based upon the
   // length of their contract, they determine the next bid year for each product
   // Inputs: year of launch, current year, bid value, type of product, location //
   public int nextbidyear(int tr, int j,float u,int base, int location)
     { if(j>G.basepd)
               {u=(u*100)/((G.Pn[location])*(G.mar[base]));
               }
      x= j-tr;
      y= (0.01f)*(G.Pn[location])*G.pper[base][j-tr];
      z=1;
      if(u<=y)
              z=1;
      if(u>y)
      for(int i=(j-tr+1);i<G.bidlife ;i++)
      {
        if(y<u)
               {y=y+(0.01f)*(G.Pn[location])*G.pper[base][i];
                       z++;
               }

      }
      if((z<=8)&& ((j+z)<(tr+G.bidlife)) )
               return(j+z);
      else{ if((z<=8)&& ((j+z)>(tr+G.bidlife)) )
               { return(0);
               }
               else{  if((z>8)&& ((j+8)<(tr+G.bidlife)) )
                     return(j+8);
                        else return(0);
               }
      }
      }

// This function is used to determine the profits which they will receive for
// each product that they launched in the last twelve years
//Inputs: current year, base value, year in which product was lauched//
     public float profits( int y, int b,int base)
     {
      int x= y-b-1;
      if(x< G.bidlife && x>=0)
      {
      return G.pro[base][x];
      }
      else
               return 0;
     }
// Determine the cost of the product pipeline
// Inputs: Current Quarter, length of the pipeline for which cost is to be estimated
    public float pipelinecost( int qrt, int time)
    {         cost =0f;
               coc=0f;
               tempnum = this.nme[time];
               for(int i= time; i<G.successrate.length;i++)
               { if(qrt> G.basepd)
                         coc = (float)Math.pow(1.06, ((qrt-G.basepd)/4));
                 else coc=0;
                 cost= cost+tempnum*G.preapproval[i]*coc;
                 if(i <G.preapproval.length)
                 tempnum=tempnum+ this.nme[i+1];
               }
               return cost;
    }

// Use the R&D cost to make progress in the product pipeline
// Inputs: Current financial quarter, Capital allocated to R&D
     public int approval( int qrt, float Rd)
    {
               coc=0f;
               for(int ct= G.preapproval.length; ct>0;ct--)
```

```
            { productcount=0;
              tempproduct=0;
              if(qrt> G.basepd)
                    coc = (float)Math.pow(1.06, ((qrt-G.basepd)/4));
             else coc=1;
          if(Rd>0)
             {   generator= new Random();
                 if(this.nme[ct-1]*G.preapproval[ct-1]*coc< Rd)
                 {      if(G.successrate[ct-1]<1)
                             {
                              for(int ct1=1; ct1<= this.nme[ct-1];ct1++)
                              {if(generator.nextInt(1000)<=(1000*G.successrate[ct-1]))
                                       productcount=productcount+1;
                            }
                             }
                            else{
                                     productcount= this.nme[ct-1];
                                 }
                             Rd= Rd-this.nme[ct-1]*G.preapproval[ct-1]*coc;
                             this.nme[ct]= productcount;
                             this.nme[ct-1]= 0;
// System.out.println("Stage: "+ct+"count"+ this.nme[ct]+"NME:"+this.nme[ct-1]+"
Approved: "+productcount);
                }
                else { tempproduct= (int)(Rd/(G.preapproval[ct-1]*coc));
                             for(int ct1=1; ct1<= tempproduct;ct1++)
                              {if(generator.nextInt(1000)<=(1000*G.successrate[ct-1]))
                                       productcount=productcount+1;
                            }
                             Rd =0;
                             this.nme[ct]= tempproduct;
                             this.nme[ct-1]= this.nme[ct-1]-tempproduct;
                          }
                 }

            }
            return((int)(Rd/(G.preapproval[0]*coc)));
    }
// Product approval during the warmup period
// Inputs: Current financial quarter, R&D
public int basepdapproval( int qrt, float Rd)
    {
            coc=0f;
```

```
               for(int ct= G.preapproval.length; ct>0;ct--)
               { productcount=0;
                 tempproduct=0;
                  generator= new Random();
              if(G.successrate[ct-1]<1)
                              {for(int ct1=1; ct1<= this.nme[ct-1];ct1++)
                                {           if(generator.nextInt(1000)<=(1000*G.successrate[ct-1]))
                                               productcount=productcount+1;
                            }
                              }
                              else{ productcount= this.nme[ct-1];
                              }
                               this.nme[ct]= productcount;
                               this.nme[ct-1]= 0;

               }
             return((int)(Rd));
     }
// This function is used to update the score of manufacturer. They receive profits
// based on their margin and loose the ingredients cost to suppliers based on their
// bid value. The output is written to the file mProfit
   public void scoreupdate()
    {
      tempscore= sm;
           float temppr= 0.0f;
           float tempinfl= inflation();
            for (int r1=g.p[G.j-G.bidlife][index]+1;r1<=g.p[G.j-1][index];r1++)
              {if(r1>0)
                        {temppr =
(tempinfl)*(G.margin[g.b[index][r1]])*(profits(G.j,g.tr[index][r1],g.b[index][r1]))*(1-
g.winbids[index][r1]/100);
                          sm= sm + temppr;
                          String str6="";
                          String key ="";
                          key = key1.Keygenerator(G.sim,G.j,index,r1,3,index,0);
                          float temp1=Stat.Round(temppr);
                          float s1=Stat.Round(this.sm);
                          String str3="";

str3="\n"+key+"\t"+G.j+"\t"+index+"\t"+g.tr[index][r1]+"\t"+r1+"\t"+(g.b[index][r1]+1)+"\t
"+00+"\t"+00+"\t"+(this.index)+"\t"+0+"\t"+0+"\t"+temp1+"\t"+0+"\t"+0+"\t"+0+"\t"+s1;
                          try{
                                 profm.write(str3);
                        }catch(Exception e) {}
                }

            }

            for(int r1=0;r1<(G.percmp-1);r1++)
            {sm1[r1]= sm1[r1+1];
             }
             sm1[G.percmp-1]= sm;
             if(G.j>G.basepd && alivem==1)
             sm= sm-Rnd;
        if((sm1[G.percmp-1]-sm1[G.percmp-2])>0 && G.j > G.basepd)
                {     sm= sm-0.10f*(sm1[G.percmp-1]-tempscore);
                      sm= sm- 0.33f*(sm -tempscore);
                }
                Rnd=0f;
   }
// Assign category to the product just launched in the market: small, medium, and
// blockbuster. This is done using a random number generator.
public void assignproductcategory()
   {
           for(int ct= g.p[G.j-1][index]+1;ct<=g.p[G.j][index];ct++)
            {
                g.modifyyear(this.index,ct,G.j);
                    int tempbs = generator.nextInt(10);
                    int tempvar1=0;
                    if(tempbs<=3)
                       tempvar1 =0;
                    if(tempbs>3 && tempbs<9)
                       tempvar1 =1;
                   if(tempbs>=9)
                       tempvar1 =2;
                   g.modifybase(this.index,ct,tempvar1);
                   for(int loc=0;loc<G.Region;loc++)
                    {
                            g.modifynb(loc,index,ct,G.j);
                    }
                   }
   }
// Estimate the inflation factor during each financial quarter
   public float inflation()
        {  float infla= (float)Math.pow(1.09,((G.j-G.basepd)/4));
            return infla;
        }
// Estimate the performane of manufacturer based on the finanical results in last
// three years. Performance is estimated based on product pipeline and return on assets
// in last three years
  public void Relativeperformance()
        {
        if(G.j>=(G.percmp+1))
            {
             float tempt =(sm1[G.percmp-1]-sm1[0]); //determining the performance of each
manufacturer= no of products launcehd in last 8 rounds/ current assets
                performance= tempt/(sm);
                performancep=(nme[16]
+nme[17]+nme[18]+nme[19]+nme[20]+nme[21]+nme[22]+nme[23]+nme[24]+nme[25]+nm
e[26]+nme[27]+nme[28]+nme[29]+nme[30]+nme[31] )/(sm);
             }
     else
           { performance= 0.3f;
             performancep= 0.3f;
           }
           qm.updatemergerinf(index,sm1,performance, performancep);
        }
}
```

3. Supplier class

```
import java.io.*;
import java.util.*;
import java.lang.Object.*;
import java.math.*;
import java.lang.*;
public class Agent implements Runnable{
Glob g;
Q q2;
Q q3;
Q q;
Q q1;
Qs qs;
public int b; // public access
public float r;
public int randomInt;
public int j;
public float t;
public float p;
public int alives;        // Binary variable to determine if a Supplier is alive or not
public float s;          // Array to store the score of suppliers after each round
public float[] sa = new float[G.percmp];
public float avprofit;    // Array to store the average profit over the last 4 years for each
supplier
int base=0;
public float performance; // Array to store the performance of each manufacturer after each
round.(= no of drugs launched in last 8 rounds/ Current assets)
Thread thrd;
public int index;
public int y1=0;
private int count;
public int rcount;
Random generator = new Random();
float oldbid;
BufferedWriter profs;
keygen key1;
float m=0.8f;
public int[] ca = new int[G.bid]; // Array to store the number of bids won by any supplier till
any round
Agent(int alive,float temp,int zer, int ind,Q q,Qs qs,Q q1,Q q2,Q q3, Glob g,BufferedWriter
profs, keygen key1)
{thrd = new Thread(this);
 this.count=0;
 this.alives=alive;
 this.s= temp;
 this.index= ind;
 this.rcount= G.j-1;
 this.oldbid=10;
 this.g=g;
 this.q=q;
 this.q1=q1;
 this.qs=qs;
 this.q2=q2;
 this.q3=q3;
 this.profs=profs;
 this.key1= key1;
 g.modifyalives(index,1);
 g.modifyaborn(index,1);
 for(int i=0;i<G.percmp;i++)
 sa[i]= zer;

 for(int l=0;l<G.bid;l++)
  {  ca[l]=1;
   }
   thrd.start();
}
Agent(int alive,float temp,int zer, int ind,Q q,Qs qs, int temp1, int temp2,float temp3,float
temp4,float[] temp5,Q q1,Q q2,Q q3,Glob g,BufferedWriter profs, keygen key1)
{thrd = new Thread(this);
 this.count=0;
 this.alives=alive;
 this.s= temp;
 this.index= ind;
 this.rcount= G.j;
 this.oldbid=temp3;
 this.g=g;
 this.q=q;
 this.q1=q1;
 this.qs=qs;
 this.q2=q2;
 this.q3=q3;
 this.profs=profs;
 this.key1= key1;
 g.modifyalives(index,1);
 this.performance=temp4;
 for(int i=0;i<G.percmp;i++)
```

```
sa[i]= temp5[i];
ca= new int[G.bid];
ca[G.j]=temp2;
ca[G.j-1]= temp1;
thrd.start();
}
// Function to determine the bid if the supplier won any of the previous two bids
public float bid(int randomInt, float t)
  {
    r= t+ (float)(((float)randomInt)/1000);
    if(r<0)
    {r=0;
    }
    return (r);
  }
// Function to determine the bid if the supplier lost both the previous two bids
public float bidnew(int randomInt, float t)
 {
   p = t-(float)(((float)randomInt)/1000);
   if(p>0)
    return p;
   else
    return 0;
 }

// Function to determine the profits from the existing contracts
// Inputs: current year, base value, year in which product was lauched//
public float profits( int y, int b,int base)
   {
    int x= y-b-1;
    if(x<G.bidlife && x>=0)
    return G.pro[base][x];
    else return 0;
   }
public void run(){
          do{
                    if(g.aborn[index]==1)
                    {
// Wait for the manufacturers to launch the product
                    q.put(0,index,0);
                    if(rcount<G.j)
                    {      float temp=100;
                          float temp1=0;
                          if(temp<100 || temp1 >0)
                          if((ca[G.j-1]==ca[G.j-2]) && (ca[G.j-2]==ca[G.j-3]))
                              {oldbid = temp;
                               }
                               else{oldbid = temp1;
                               }

// For products launched by each manufacturer, estimate the bid value
                            for(int n=0; n<G.man;n++)
                            { if(g.Aliveman[n]==1)
                                      {for(int c11=g.p[G.j-1][n]+1;c11<=g.p[G.j][n];c11++)
                                        {    if((ca[G.j-1]==ca[G.j-2]) && (ca[G.j-2]==ca[G.j-3]))
                                              {float temv= bidnew(generator.nextInt(100),oldbid);
                                               g.modifyrfq(n,c11,index,temv);
                                              }
                                              else{float temv= bid(generator.nextInt(100),oldbid);
                                                    g.modifyrfq(n,c11,index,temv);
                                              }

                                              if(g.rfq[n][c11][index]<temp)
                                              temp=g.rfq[n][c11][index];
                                              if(g.rfq[n][c11][index]>temp1)
                                              temp1=g.rfq[n][c11][index];
                                              }
                                      }
                            }
                    }

// Update the total contracts won by the supplier
                    if(G.j> G.basepd-4)
                    {contractwon();
                    }
// Update the score of supplier based on all the active contracts
                    if(G.j> G.basepd)
                    {scoreupdate();
                    }
// Estimate the performance of the supplier
                    if(G.j> G.percmp)
                    {
                     Relativeperformance();
                    }

// Wait for manufacturers to determine the winner supplier based on the bids
```

```
                q3.determinewinner(0);
                rcount++;
// Notify the main thread to start the procedure for Mergers and acquisitions
                q1.determinewinner2(0);

            }
             if(g.aborn[index]!=1)
             g.modifyaborn(index,1);

// Wait for the Main thread to finish mergers and acquisitions process
             if(rcount >= G.j)
             q2.nextround(0);
             if(g.Aliveagent[index]==0)
                    {
                      s=0;
                      rcount=G.bid+1;
                      alives=0;
                      try {
                                thrd.stop();
                               } catch (Exception ex){
                          }

                     }

              if(s<=0)
                     {g.modifyalives(index,0);
                      rcount=G.bid+1;
                      alives=0;
                      try {
                                  thrd.stop();
                               } catch (Exception ex){
                          }

                     }
            }
            while(rcount< (G.bid));
            g.modifyalives(index,0);
            try {
            thrd.stop();
            thrd.interrupt();
            thrd.join();
                } catch (InterruptedException ex){
    }
    }

// Function to determine the active contracts of supplier
 public void contractwon()
 {
       for(int m=0;m<G.man;m++)
           {if(g.Aliveman[m]==1)
                     {        int tempvar2=0;
                              for(int m1=0;m1<=g.p[G.j][m];m1++)
                              {if(g.wins[m][m1]==index)
                                        { ca[G.j]=ca[G.j]+1;
                                                  if(g.tr[m][m1]>=G.j-G.bidlife)
                                                  {
                                                            tempvar2= tempvar2+1;
                                                  }
                                        }
                              }
                              g.modifyacts(index,m,tempvar2);
                     }
           }
 }

//Function to determine the profits from the active contracts of the supplier
// Write proft value from each product to the text file sProfit
  public void scoreupdate()
    {
           s = G.sup*s;
           float temppr= 0.0f;
           float tempinfl= inflation();
           for (int h=0;h<G.man;h++)
           { if(g.Aliveman[h]==1)
              {
                for (int r1=g.p[G.j-G.bidlife][h]+1;r1<=g.p[G.j-1][h];r1++)
                     {     if(g.wins[h][r1]== this.index && r1>0)
                              {
                              temppr=
(tempinfl)*(g.winbids[h][r1])*((profits(G.j,g.tr[h][r1],g.b[h][r1]))/100);
                               s = s+ temppr;
                               String str6="";
                               String key ="";
                               key = key1.Keygenerator(G.sim,G.j,h,r1,2,index,0);
                               float mins1=Stat.Round(g.winbids[h][r1]);
```

```
                                float temp1=Stat.Round(temppr);
                                float s1=Stat.Round(this.s);
                                String str3="";

str3="\n"+key+"\t"+G.j+"\t"+h+"\t"+g.tr[h][r1]+"\t"+r1+"\t"+(g.b[h][r1]+1)+"\t"+00+"\t"+00
+"\t"+(this.index)+"\t"+mins1+"\t"+0+"\t"+temp1+"\t"+0+"\t"+0+"\t"+0+"\t"+s1;
                                try{
                                profs.write(str3);
                                }catch(Exception e) {}
                               }
                        }
                  }
              }
    }
// Estimate inflation factor in any quarter
  public float inflation()
          {  float infla= (float)Math.pow(1.09,((G.j-G.basepd)/4));
             return infla;
          }
// Estimate the performance of supplier based on its Return on Assets in last 3 years
  public void Relativeperformance()
          {for(int r1=0;r1<(G.percmp-1);r1++)
             {sa[r1]= sa[r1+1];
              }
           sa[G.percmp-1]= s;
           if(G.j>=(G.percmp+1))
           {
           performance=(sa[G.percmp-1]-sa[0])/(s);
           }
           qs.updatemergerinf(index,sa,performance);
          }
// Estimate the sum of any array
// Inputs: Array with values
public int sums(int[] arr) {
                              int add = 0;
                              int size = arr.length;
                              for (int i=0; i < size; ++i)
                                      add = add+arr[i];
                              return add;
          }
 }
```

4. Distributor class

```
import java.io.*;
import java.lang.Object.*;
import java.math.*;
import java.lang.*;
import java.util.*;
public class distributor implements Runnable{
    Glob g;
    Q q2;
    Q q3;
    Q q;
    Q q1;
    Qd qd;
    float average=0;
    int b11=0;
    public float d = 0;
    public ArrayList<Integer> lon = new ArrayList<Integer>();
    public ArrayList<Integer> lat = new ArrayList<Integer>();
    public int dc ;
    public int alived;       // Binary array to determine if a distributor is alive
    public float sr;         // variable to store score of distributor in each location after each
round
    public float movaverage; //variable to store the moving average of this distributor in last 4
rounds
    public float[] sd= new float[G.percmp];
    public float r;
    public float t;
    public float p1;
    Thread thrd;
    public float performance;      // Array to store the performance of each manufacturer after
each round.(= no of drugs launched in last 8 rounds/ Current assets)
    public int rank1;              // Array to store the rank of each manufacturer which is
determined based on performance
    public int rank2;
    public int rank;
    public float cmsc;
    public int[] Highperf= new int[G.percmp];       // If rank > median then Highperf=1 else 0
    public int hpct;               // Array to store the High performance count(= sum of variable
Highperf over the last 8 rounds
    public float hpor;             // Array to store the high performance odd's ratio for each
manufacturer
    public float lhpor;            // Array to store the logarithm of high performance odd's ratio
    public int index;
```

```
   public int rcount;
   public int location;
   public int perscore;
   public float oldbid;
   public int[] count= new int[G.man];
   public float[] sum= new float[G.man];
   public int[] ca = new int[G.bid+2];
   BufferedWriter profd;
   keygen key1;
   Random generator = new Random();
   Random gen1 = new Random();
   Random gen2 = new Random();
   public int[] cd = new int[G.bid];  // Array to store the number of bids won by any
distributor in any region till any round
   distributor( int i, int loc, int n, int zer, int alive, float score,int ind,Q q,Qd qd, Q q1,Q q2,Q
q3,Glob g,BufferedWriter profd, keygen key1)
{      thrd = new Thread(this);
       alived= alive;
       sr= score;
       this.dc = i;
       this.index=ind;
       this.rcount=G.j-1;
       this.location=loc;
       this.oldbid= 10;
       this.perscore=10;
       this.g=g;
       this.q=q;
       this.q1=q1;
       this.qd=qd;
       this.q2=q2;
       this.q3=q3;
       this.profd=profd;
       this.key1= key1;
       g.modifyalived(location,index,1);
       for( int j1=0; j1<G.man; j1++)
         {this.sum[j1]=0;
          this.count[j1]=0;
         }
         /*
       for( int j1=0; j1<i; j1++)
          {lon.add(j1,G.x[location][j1][n]);
           lat.add(j1,G.y[location][j1][n]);
          }
```

```
            d = distance(i, loc);
             */
             for(int l=0;l<G.bid;l++)
            {   cd[l]=1;
             }
            for( int j1=0; j1<5; j1++)
            {sd[j1]=zer;
            }
      thrd.start();
  }

distributor( int i, int loc, int n, int zer, int alive, float score,int ind,Q q,Qd qd,float perf,float[]
sc,float[] su, int[] coun,Q q1,Q q2,Q q3, Glob g,BufferedWriter profd, keygen key1)
{       thrd = new Thread(this);
        alived= alive;
        sr= score;
        this.dc = i;
        this.index=ind;
        this.rcount=G.j;
        this.location=loc;
        this.oldbid= 10;
        this.perscore=10;
        this.g=g;
        this.q=q;
        this.q1=q1;
        this.qd=qd;
        this.performance=perf;
        this.q2=q2;
        this.q3=q3;
        this.profd=profd;
        this.key1= key1;
        g.modifyalived(location,index,1);
        for( int j1=0; j1<G.man; j1++)
          {this.sum[j1]=su[j1];
           this.count[j1]=coun[j1];
          }
       for(int l=0;l<G.bid;l++)
            {   cd[l]=1;
                }
       for( int j1=0; j1<G.percmp; j1++)
            {sd[j1]=sc[j1];
            }
        thrd.start();
 }

distributor(  int zer, int alive, float score,int ind)
{          thrd = new Thread(this);
           this.index=ind;
           alived= alive;
           sr= score;
           for( int j=0; j<5; j++)
            sd[j]=zer;
            thrd.start();
 }


public float update( int y, int b11,int e, float av)      // current year, year in which product was
lauched,base value..//
{
     average= (av + G.pro[b11][y])/e;
     return average;

}
/*
public float distance( int i ,int loc) {
                    float totaldist= 0.0f;
                    float dist= 0.0f;
                    for (int c=0; c < G.cty[loc]; c++)
                    {float min = Float.MAX_VALUE;
                    for (int r=0; r < i; r++)
                    { int x= lon.get(r).intValue();
                      int y = lat.get(r).intValue();
                      dist= (float)Math.sqrt((x-G.city[1][loc][c])*(x-G.city[1][loc][c])+(y-
G.city[2][loc][c])*(y-G.city[2][loc][c]));
                     if(min > dist)
                     {min= dist;
                     }
                    }
             totaldist= totaldist+min;
             }
                    return totaldist;
          }
          */
// Function to estimate the bid value if distributor won any of the previous
// two bids
```

```
public float bid(int randomInt, float t)
  {
     r= t+ (float)(((float)randomInt)/1000);
     return (r);
  }
// Function to estimate the bid value if distributor lost both the previous
// two bids

public float bidnew(int randomInt, float t)
 {
    p1 = t-(float)(((float)randomInt)/1000);
    if(p1>0)
     return p1;
    else
     return 0;
 }

// Function to estimate the profit from the products
// Inputs: current year, base value, year in which product was lauched//
public float profits( int y, int b,int base)
    {
      int x= y-b-1;
      if(x<G.bidlife && x>=0)
      return G.pro[base][x];
      else return 0;
    }

public void run()
    {      do{      if(g.dborn[location][index]==1)
                         {
// Wait for the manufacturers to launch the product
                              q.put(0,index,1);
                              if(rcount<G.j)
                           {  float temp=0;
                              float temp1=100;
                              if(temp>0 || temp1<100)
                              if((ca[G.j-1]==ca[G.j-2]) && (ca[G.j-2]==ca[G.j-3]))
                               {oldbid = temp;
                               }else{oldbid= temp1;
                               }
// Determine the bid value for all the products for which manufacturers are conducting
// bidding
                              for(int n=0; n<G.man;n++)
                              { if(g.Aliveman[n]==1)
                                    {         int start=0;
                                              if(G.j <= G.bidlife)
                                                      start=0;
                                              else start=(g.p[G.j-G.bidlife][n]);

                                              for(int c11=start+1;c11<=g.p[G.j][n];c11++)
                                              {
// In the warmup period, bidding is conducted using the rules based on random number
generator
 if(G.j<=G.basepd)
        { if(g.nb[location][n][c11]==G.j && (g.nb[location][n][c11]-g.tr[n][c11])<G.bidlife)
                    {
                                  if((ca[G.j-1]==ca[G.j-2]) && (ca[G.j-2]==ca[G.j-3]))
                                              {float temv= bid(generator.nextInt(100),oldbid);
                                                g.modifydrfq(n,c11,location,index,temv);
                                              }
                                  else{float temv= bidnew(generator.nextInt(100),oldbid);
                                                g.modifydrfq(n,c11,location,index,temv);
                                              }
                                              if(g.drfq[n][c11][location][index]>temp)
                                              temp=g.drfq[n][c11][location][index];
                                              if(g.drfq[n][c11][location][index]<temp1)
                                              temp1=g.drfq[n][c11][location][index];

                                          }
                                        }
                                        float temv=0;
                                        float tempbid;
// After the warm up period, bidding is conducted based on the forecast by the ditributors
// Distributors can get a contract for a maximum of 2 years of product life cycle.
// If it is a nwe product, distributor determine the bid value based on their previous
// experience with the manufacturer. If it is an old product,the distributor determine
// the future sales using on a linear function based on first two years of sales
 if((G.j>G.basepd) && (g.nb[location][n][c11]==G.j))
        {
         if(g.nb[location][n][c11]==g.tr[n][c11])
         {
             if(sum[n]>0)
              {tempbid= sum[n];
                 }else{  int nt=0;
               for(int cnt=0;cnt< sum.length;cnt++)
                   {if(sum[cnt]>0)
```

```
                              nt++;
                         }
                    tempbid= (Stat.sum(sum))/nt;
          }

          if((ca[G.j-1]==ca[G.j-2]) && (ca[G.j-2]==ca[G.j-3]))
          {
            if(perscore<100)
            perscore= perscore+10;
            if(tempbid <((perscore)*sr*(0.01f)))
           { temv=tempbid;
              g.modifydrfq(n,c11,location,index,temv);
            }
           else{temv=((perscore)*sr*(0.01f));
             g.modifydrfq(n,c11,location,index,temv);
           }
          }
           else{      if(tempbid <((0.1f)*sr))
                       { temv=tempbid;
                          g.modifydrfq(n,c11,location,index,temv);
                       }
                       else{temv=((0.1f)*sr);
                             g.modifydrfq(n,c11,location,index,temv);
                            }
                }
       }
     else{  temv=0;
            float slope=0;
            int sign=1;
            int startx=0;
            if(G.j-g.tr[n][c11] >=8)
            { slope= ((G.pro[g.b[n][c11]][G.j-g.tr[n][c11]])*(inflation((G.j-1)))-
(G.pro[g.b[n][c11]][G.j-g.tr[n][c11]-8])*(inflation(G.j-8)))*(0.01f)*(G.Pn[location])/(8);
             }else{slope= ((G.pro[g.b[n][c11]][G.j-g.tr[n][c11]])*(inflation((G.j-1)))-
(G.pro[g.b[n][c11]][0])*(inflation(g.tr[n][c11])))*(0.01f)*(G.Pn[location])/(G.j-g.tr[n][c11]);
                                                                      }
            if(gen2.nextInt(10)>5)
            { sign=1;
             }else{ sign =-1;
                 }
if(this.index== g.wind[location][n][c11])
   {slope= slope+(sign)*(slope)*(0.05f)*(gen1.nextFloat());
   }else slope= slope+(sign)*(slope)*(0.1f)*(gen1.nextFloat());
```

```
if(G.j-g.tr[n][c11] >=8)
   { startx= 8;
    }else{ startx=(G.j-g.tr[n][c11]);
            }
for(int countx=(startx+1);countx<startx+8;countx++)
 {temv= temv+ (G.pro[g.b[n][c11]][G.j-g.tr[n][c11]])*(inflation((G.j-
1)))*(0.01f)*(G.Pn[location])+ slope*(countx-startx);
  }

g.modifydrfq(n,c11,location,index,temv);
 }
}

}
}
}
}
// Update the score of distributor from all the active contracts
if(G.j> G.basepd)
 {scoreupdate();
 }
// Update the performance of distributor
if(G.j> G.percmp)
  {
   Relativeperformance();
   }
// Notify the manufacturers for the completion of the biding submission
q3.determinewinner(1);

// Determine the average returns from products o ach manufacturer
// Update the active contracts of distributor with each manufacturer
if(G.j> G.successrate.length)
{       prodestimate();
        contractwon();
}
 rcount++;
// Notify the main thread to finish of the mergers and acquisition process
 q1.determinewinner2(1);
 }
if(g.dborn[location][index]!=1)
        g.modifydborn(location,index,1);
// Wait for the Main thread to finish the mergers and acquisition process
if(rcount >= G.j)
```

```
            q2.nextround(1);

if(g.Alivedist[location][index]==0)
{
 rcount=G.bid+1;
 alived=0;
 try {
        thrd.stop();
       } catch (Exception ex){
                                        }
}

 if(sr<=0)
{
    g.modifyalived(location,index,0);
    rcount=G.bid+1;
    alived=0;
    try {
              thrd.stop();
          } catch (Exception ex){
}
}
}
while(rcount< (G.bid));
 g.modifyalived(location,index,0);
 try {
            thrd.stop();
            thrd.interrupt();
            thrd.join();
       } catch (InterruptedException ex){
                                                          }


    }

// Function to determine the total active contracts of distributor with each manufacturer
 public void contractwon()
 {
      for(int m=0;m<G.man;m++)
        {if(g.Aliveman[m]==1)
                       {      int tempvar2=0;
                              for(int m1=0;m1<=g.p[G.j][m];m1++)
                                 {    if(g.wind[location][m][m1]==index)
```

```
                                          { ca[G.j]=ca[G.j]+1;
                                            if(g.tr[m][m1]>=G.j-G.bidlife)
                                                    {
                                                     tempvar2= tempvar2+1;
                                                    }
                                           }
                                 }
                                 g.modifyactd(location,index,m,tempvar2);
                        }
              }

    }
// Function to update the average returns from the products of each manufacturer
// for which the distributor won the contract
 public void prodestimate()
 {
       for(int m=0;m<G.man;m++)
           {if(g.Aliveman[m]==1)
                     {
                              for(int m1=g.p[G.j-1][m]+1;m1<=g.p[G.j][m];m1++)
                              {
                               if(g.wind[location][m][m1]==index)
{sum[m]=(sum[m]*count[m]+(0.01f)*(G.Pn[location])*(G.mar[g.b[m][m1]]))/(count[m]+1);
                                         count[m]=count[m]+1;
                                     }
                              }
                     }
           }
  }
// Determine the inflation factor for the year passed to the function
public float inflation(int temyr)
          {  float infla= (float)Math.pow(1.09,((temyr-G.basepd)/4));
             return infla;
          }
// Update the score of the diistributor for all the active contracts and reduce
// the operational cost. Write the profit from each product to dProfit.txt
 public void scoreupdate()
   {    sr= G.perc*sr;
        float temppr= 0.0f;
        float tempinfl= inflation(G.j);
        for (int h=0;h<G.man;h++)
        { if(g.Aliveman[h]==1)
         {
            for (int r1=g.p[G.j-G.bidlife][h]+1;r1<=g.p[G.j-1][h];r1++)
            { if(g.wind[this.location][h][r1]== this.index && r1>0)
                {temppr =
(tempinfl)*(0.001f)*(G.Pn[location])*(profits(G.j,g.tr[h][r1],g.b[h][r1]));
                                sr = sr + temppr;
                                String key= "";
                                 key = key1.Keygenerator(G.sim,G.j,h,r1,1,this.index ,this.location);
                                 String str4="";
                                 float max1 =Stat.Round(g.winbidd[location][h][r1]);
                                 float temp1=Stat.Round(temppr);
                                 float sr1 = Stat.Round(this.sr);
                                 String str8="";

str8="\n"+key+"\t"+G.j+"\t"+h+"\t"+g.tr[h][r1]+"\t"+r1+"\t"+(g.b[h][r1]+1)+"\t"+location+"\
t"+G.Pn[location]+"\t"+
this.index+"\t"+max1+"\t"+0+"\t"+temp1+"\t"+0+"\t"+0+"\t"+0+"\t"+sr1;
                                 try{
                                   profd.write(str8);
                                  }catch(Exception e) {}
                                 }
                       }
               }
            }
    }

// Determine the performance of distributor based on Return on Assets in last three
// years
    public void Relativeperformance()
           {  for(int r1=0;r1<(G.percmp-1);r1++)
               {sd[r1]= sd[r1+1];
                }
               sd[G.percmp-1]= sr;
               if(G.j>=(G.percmp+1))
                 {
                  float tempt =(sd[G.percmp-1]-sd[0]); //determining the performance of each
manufacturer= no of products launcehd in last 8 rounds/ current assets
                  performance= tempt/(sr);
                 }
        else
               { performance= 0.3f;
               }
             qd.updatemergerinf(location,index,sd,performance);
           }
```

```
// Function to determine the summ of array passed to the function
   public int sums(int[] arr) {
                              int add = 0;
                              int size = arr.length;
                              for (int i=0; i < size; ++i)
                              add = add+arr[i];
                              return add;
          }
   }
```

5. Global variable class

```
public class G
{
public static int agent =120;    // Represents maximum number of supplier after mergers
public static int ia =60;           //Represents number of supplier
public static int man =60;    // Represents maximum number of manufacturers after mergers
public static int im =30;       //Represents number of manufacturers
public static int bid = 300;
public static int distributor = 48; //Represents maximum number of distributors after mergers
public static int retailer[]={300,480,240,180} ;
public static int id[]={15,24,12,9} ;        //Represents number of distributors
public static int bidlife = 48;              // life of a product
//public static int compare=1000;
public static int Pn[]={25,40,19,16};         // Population proportion of each region
public static int Region= Pn.length;
public static float mar[]={67.34f, 269f, 2618f}; // This variable represents the sum of all the
returns from each type of product.
public static float margin[]={0.80f, 0.85f, 0.90f};   // Array which stores the margin for each
kind of product
public static int experiment =1000;          // Variable to store the number of simulation
rounds
public static int spacealloc=1500;
public static float d=1f;
public static float perc=0.975f;             // Variable that represents the assets value of
distributors left after each financial quarter
public static float sup=0.97f;              // Variable that represents the assets value of supplier
left after each financial quarter
public static int basepd=144;                        // Simulation warm up period to develop a
pipeline
public static int percmp=12;                         // number of quarters for which we compare
the performance
public static int nmerate=35;
public static int cty[]= {60,80,70,90};             // population in each city
public static float AssetsRnd=0.04f;
public static int x =0;
public static int j =1;                       //Represents the financial Quarter
public static int incyear =0;
public static int start = 0;
public static float totalsc=0f;
public static float totalsc1=0f;
public static float coc=0f;
public static int smedian=0;
public static float[] dmedian= new float[Region];
public static float mmedian=0f;
public static int sim =0;
public static float
successrate[]={1f,1f,1f,1f,1f,1f,1f,0.0004f,1f,1f,1f,1f,1f,1f,1f,0.75f,1f,1f,1f,1f,1f,1f,1f,0.48f,1
f,1f,1f,1f,1f,1f,1f,0.576f};   // This array represents probability of success in each round of
product develoopment process
public static float preapproval[]=
{0.000611712f,0.000611712f,0.000611712f,0.000611712f,0.000611712f,0.000611712f,0.00
0611712f,0.000611712f,0.259199819f,0.259200026f,0.259200026f,0.259200026f,0.2592000
26f,0.259200026f,0.259200026f,0.259200026f,0.528768f,0.528768f,0.528768f,0.528768f,0.
528768f,0.528768f,0.528768f,0.528768f,4.082400072f,4.082400072f,4.082400072f,4.08240
0072f,4.082400072f,4.082400072f,4.082400072f,4.082400072f};
//This array represents costs associated with different stages of drug development
// public static float
successrate[]={0.965f,0.965f,0.965f,0.965f,0.965f,0.965f,0.965f,0.965f,0.912f,0.912f,0.912f,
0.912f,0.912f,0.912f,0.912f,0.912f,0.932f,0.932f,0.932f,0.932f,0.932f,0.932f,0.932f,0.932f};
public static float pro[][]=
{{0.0236451f,0.429517453f,1.486567709f,2.926763541f,4.406307084f,5.708148605f,6.738
450046f,7.481033757f,7.959594505f,8.214122645f,8.28803552f,8.221915417f,8.050953877
f,7.804332247f,7.505532224f,7.173031699f,6.821105171f,6.460593749f,6.099587683f,5.74
4004388f,5.398064397f,5.064675995f,4.74574205f,4.442402409f,4.155223996f,3.88434895
4f,3.629609467f,3.390616291f,3.166826627f,2.957595859f,2.762216695f,2.579948539f,2.4
10039288f,2.251741276f,2.104322721f,1.967075729f,1.839321658f,1.7204145f,1.60974276
2f,1.506730226f,1.410835904f,1.321553385f,1.238409776f,1.160964362f,1.088807074f,1.0
21556864f,0.958860025f,0.900388511f},
{0.117947322f,2.14253412f,7.415349514f,14.59938519f,21.97969651f,28.47358834f,33.61
297435f,37.31715662f,39.70433024f,40.97397649f,41.3426713f,41.01284867f,40.16005228
f,38.92984556f,37.43936074f,35.78076989f,34.02527756f,32.22696176f,30.42617856f,28.6
5244542f,26.92681534f,25.26379555f,23.67287799f,22.15974849f,20.72723499f,19.376046
55f,18.10534605f,16.91319185f,15.79687635f,14.75318405f,13.77858681f,12.86939036f,1
2.02184304f,11.2322153f,10.49685687f,9.812236578f,9.174969212f,8.581832332f,8.02977
5658f,7.515924893f,7.03758145f,6.592219236f,6.177479355f,5.791163407f,5.431225877f,
5.095766004f,4.783019419f,4.491349751f},
{0.201678793f,3.66353122f,12.67954811f,24.96357138f,37.58320748f,48.68714987f,57.47
501509f,63.80881734f,67.89065902f,70.06163433f,70.69206767f,70.12810209f,68.6699007
2f,66.56636328f,64.01777485f,61.18174097f,58.18001469f,55.10506433f,52.02589494f,48.
99297861f,46.04231397f,43.19870702f,40.47838807f,37.89107937f,35.44161644f,33.13121
167f,30.95843369f,28.91996248f,27.01116829f,25.22655292f,23.56008357f,22.00544342f,
20.55621746f,19.20602853f,17.94863497f,16.77799887f,15.68833179f,14.67412368f,13.73
015885f,12.85152252f,12.03360036f,11.27207299f,10.56290692f,9.90234309f,9.28688387f
,8.713279135f,8.178511979f,7.679784362f}};
```

```
public static float pper[][]=
{{0.012561727f,0.228186016f,0.789755946f,1.554876307f,2.340900586f,3.032518651f,3.5
7987797f,3.974383984f,4.228624807f,4.363845766f,4.403112819f,4.367985764f,4.2771605
09f,4.146139972f,3.987399073f,3.8107544f,3.623789442f,3.432263663f,3.240475098f,3.05
1567442f,2.8677829f,2.690666533f,2.521229259f,2.360076636f,2.207509849f,2.063604413
f,1.928271173f,1.801303339f,1.682412543f,1.571256326f,1.46745893f,1.370626906f,1.280
360691f,1.196263077f,1.117945299f,1.045031279f,0.97716048f,0.913989705f,0.855194089
f,0.800467512f,0.749522566f,0.702090216f,0.657919232f,0.616775478f,0.5784411f,0.5427
13664f,0.509405257f,0.478341603f},
{0.012561727f,0.228186016f,0.789755946f,1.554876307f,2.340900586f,3.032518651f,3.57
987797f,3.974383984f,4.228624807f,4.363845766f,4.403112819f,4.367985764f,4.27716050
9f,4.146139972f,3.987399073f,3.8107544f,3.623789442f,3.432263663f,3.240475098f,3.051
567442f,2.8677829f,2.690666533f,2.521229259f,2.360076636f,2.207509849f,2.063604413f
,1.928271173f,1.801303339f,1.682412543f,1.571256326f,1.46745893f,1.370626906f,1.2803
60691f,1.196263077f,1.117945299f,1.045031279f,0.97716048f,0.913989705f,0.855194089f
,0.800467512f,0.749522566f,0.702090216f,0.657919232f,0.616775478f,0.5784411f,0.54271
3664f,0.509405257f,0.478341603f},
{0.012561727f,0.228186016f,0.789755946f,1.554876307f,2.340900586f,3.032518651f,3.57
987797f,3.974383984f,4.228624807f,4.363845766f,4.403112819f,4.367985764f,4.27716050
9f,4.146139972f,3.987399073f,3.8107544f,3.623789442f,3.432263663f,3.240475098f,3.051
567442f,2.8677829f,2.690666533f,2.521229259f,2.360076636f,2.207509849f,2.063604413f
,1.928271173f,1.801303339f,1.682412543f,1.571256326f,1.46745893f,1.370626906f,1.2803
60691f,1.196263077f,1.117945299f,1.045031279f,0.97716048f,0.913989705f,0.855194089f
,0.800467512f,0.749522566f,0.702090216f,0.657919232f,0.616775478f,0.5784411f,0.54271
3664f,0.509405257f,0.478341603f}};
```

6. Global class

```
public class Glob
{  public static int[][] p;
   public static int[][] tr;
   public static int[][] b;
   public static float[][][] rfq;
   public static float[][][][] drfq;
   public static int[][] wins;    // to save the winner suppliers submitted their bids
   public static int[][][] wind; // to save the winner distributor in each location
   public static float[][] winbids;        // to save the bid value of winner supplier
   public static float[][][] winbidd;//to save the bid value of winner distributor in each
location
   public static int[][][] nb;
   public static int[] Aliveman;
   public static int[] Aliveagent;
   public static int[][] Alivedist;
   public static int[][] acts;
   public static int[][][] actd;
   public static int[] mborn;
   public static int[] aborn;
   public static int[][] dborn;

Glob()
{
   p= new int[G.bid+2][G.man];
   tr= new int[G.man][G.spacealloc];
   b= new int[G.man][G.spacealloc];
   rfq= new float[G.man][G.spacealloc][G.agent];
   drfq= new float[G.man][G.spacealloc][G.Region][G.distributor];
   wins= new int[G.man][G.spacealloc];       // to save the winner suppliers submitted their
bids
   wind= new int[G.Region][G.man][G.spacealloc];  // to save the winner distributor in each
location
   winbids= new float[G.man][G.spacealloc];           // to save the bid value of winner
supplier
   winbidd= new float[G.Region][G.man][G.spacealloc]; //to save the bid value of winner
distributor in each location
   nb= new int[G.Region][G.man][5*G.spacealloc];
   Aliveman= new int[G.man];
   Aliveagent= new int[G.agent];
   Alivedist= new int[G.Region][G.distributor];
   acts= new int[G.agent][G.man];
   actd= new int[G.Region][G.distributor][G.man];
```

```
    mborn= new int[G.man];
    aborn= new int[G.agent];
    dborn= new int[G.Region][G.distributor];
}

  public synchronized void modifymborn(int temp08,int temp18)    // index, value(0,1)
 { mborn[temp08]=temp18;
 }
  public synchronized void modifyaborn(int temp09,int temp19)    // index, value(0,1)
 { aborn[temp09]=temp19;
 }
  public synchronized void modifydborn(int temp10,int temp20,int temp30)    // index,
value(0,1)
 { dborn[temp10][temp20]=temp30;
 }

 public synchronized void modifyacts(int temp06,int temp16,int temp26)    // index,
value(0,1)
 {  acts[temp06][temp16]= temp26;
 }

 public synchronized void modifyactd(int temp07,int temp17,int temp27, int temp37)    //
index, value(0,1)
 {
  actd[temp07][temp17][temp27]= temp37;
  }

public synchronized void modifynb(int temp05,int temp15,int temp25, int temp35)    // index,
value(0,1)
 {
  nb[temp05][temp15][temp25]=temp35;
  }

 public synchronized void modifyalives(int temprs,int tempr1s)    // index, value(0,1)
 {  Aliveagent[temprs]=tempr1s;
  }

 public synchronized void modifyalived(int location,int tempr2d, int tempr1d)
 {
    Alivedist[location][tempr2d]=tempr1d;
  }
 public synchronized void modifyalivem(int temprm,int temp1rm)
 {
   Aliveman[temprm]=temp1rm;
  }

public synchronized void modifyrfq(int tempf,int temp1f,int temp2f, float temp3f)
 {
   rfq[tempf][temp1f][temp2f]= temp3f;
   }

public synchronized void modifydrfq(int temp0df,int temp10df,int temp20df, int
temp30df,float temp40df)
 {
    drfq[temp0df][temp10df][temp20df][temp30df]=temp40df;
   }

public synchronized void modifyproduct(int tempp,int temp1p,int temp2p)
 {
    p[tempp][temp1p]= temp2p;
   }

public synchronized void modifyyear(int tempy,int temp1y, int temp2y)
 {
    tr[tempy][temp1y]=temp2y;
   }

public synchronized void modifybase(int tempb,int temp1b,int temp2b)
 {
  b[tempb][temp1b] =temp2b;
  }

public synchronized void modifywins(int temp04,int temp14,int temp24)
 {
   wins[temp04][temp14]= temp24;
  }

public synchronized void modifywind(int temp03,int temp13, int temp23, int temp33)
 {
   wind[temp03][temp13][temp23]= temp33;
   }
```

```
public synchronized void modifywinbids(int temp01,int temp11,float temp21)
 {
   winbids[temp01][temp11]= temp21;
   }

public synchronized void modifywinbidd(int temp02,int temp12, int temp22,float temp24)
 {
   winbidd[temp02][temp12][temp22]= temp24;
   }
}
```

7. Statistics class

```
public class Stat{
   public synchronized static float Round(float Rval)
  {
     float p = (float)Math.pow(10,1);
     Rval = Rval * p;
     float tmp = Math.round(Rval);
     return (float)tmp/p;
  }
  public synchronized static float sum(float[] a)
   {
     float add = 0;
     int size = a.length;
     for (int i=0; i < size; ++i)
            add = add+a[i];
     return add;
}
 public synchronized static int sumi(int[] a)
{
     int add = 0;
     int size = a.length;
     for (int i=0; i < size; ++i)
            add = add+a[i];
     return add;
}

public static final float mean( float[] a1,float[] a2 )
{
     float size = sum(a2);
     float sum = sum(a1);
     return sum/size;
}

public synchronized static float minimum( float[] a )
{
    float minimum = Float.MAX_VALUE;
    int size = a.length;
    for (int i=0; i < size; ++i)
    if (a[i] < minimum && a[i]!=0)
                    minimum = a[i];
    return minimum;
}
```

```
public synchronized static float stdev(float[] a1, float[] a2)
{
        float n = sum(a2);
        if (n < 2)
           return 0;
        else{
                float ave = mean(a1,a2);
                float v= 0;
                for (int i=0; i < a1.length; ++i)
                {    if(a2[i]==1)
                        {float s = a1[i] - ave;
                         v = v + s*s;
                        }
                }
                v = (v/(n-1));
                return (float)Math.sqrt(v);
        }
}

public synchronized static float maximum( float[] a )
{
        float maximum = -Float.MAX_VALUE;
        int size = a.length;
        for (int i=0; i < size; ++i)
          if (a[i] > maximum)
                maximum = a[i];
        return maximum;
}

public static float scorer(int ty)
   {
           float minim=0;
           float maxim=0;
           float mode=0;
           float h1=0;
           float v1;
           float v2;
           float tempor;
           float r= (float)Math.random();
           if(ty==0)                // 0= Distributor
           { minim=2.0f;
             maxim=7.76f;
```

```
            mode =3.1f;
            h1= 4.0f;
           }
           if(ty==1)                   // 1= manufacturer
           { minim=4.2f;
             maxim=9.0f;
             mode =7.63f;
             h1= 5.0f;
           }
           if(ty==2)                   // 2= Supplier
           { minim=3.723f;
             maxim=8.523f;
             mode =7.152f;
             h1= 5.0f;
           }
           float area1 = (float)(0.5*(mode-minim)*h1);
           float area2 = (float)(0.5*(maxim-mode)*h1);
           float ratio= area1/(area1+area2);
           if(r<ratio)
           { tempor= r*(mode-minim)*(maxim-minim);
             v1= (float)Math.sqrt(tempor)+ minim ;
           }
           else
           {tempor= (1-r)*(maxim-mode)*(maxim-minim);
            v1 = maxim- (float)Math.sqrt(tempor);
           }
        v2= (float)Math.exp(v1);
        return v2;
}
}
```

8. Semaphore class(Q)

```
class Q {
 private int signalsup = 0;
 private int signalssup = 0;
 private int signals1sup = 0;
 private int signals2sup = 0;
 private int signals3sup = 0;
 private int signaldis = 0;
 private int signalsdis = 0;
 private int signals1dis = 0;
 private int signals2dis = 0;
 private int signals3dis = 0;
 private int signalman = 0;
 private int signalsman = 0;
 private int signals1man = 0;
 private int signals2man = 0;
 private int signals3man = 0;
 private int signal = 0;
 private int signals = 0;
 private int signals1 = 0;
 private int signals2 = 0;
 private int signals3 = 0;
 public int sig = 0;
 Glob g;
 Q(Glob g)
   {this.g =g;
   }

synchronized void put(int x,int ind, int s) {
this.signal++;
if(s==0)
 this.signalsup++;
if(s==1)
 this.signaldis++;
if(s==2)
 this.signalman++;

if( (this.signal<
(Stat.sumi(g.Aliveman)+Stat.sumi(g.Aliveagent)+Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Aliv
edist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.Alivedist[3]))) || (this.signalsup<
(Stat.sumi(g.Aliveagent)))|| (this.signalman< (Stat.sumi(g.Aliveman)))||(this.signaldis<
(Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.
Alivedist[3]))))
```

```
            {
            try {
               if(s==0 ||s==1)
               wait();
               } catch(InterruptedException e) {
          System.out.println("InterruptedException caught");
         }
      }
   else
   {
   notifyAll();
   this.signal=0;
   this.signalsup=0;
   this.signaldis=0;
   this.signalman=0;
   }
}
synchronized void determinewinner(int s) {
this.signals++;
if(s==0)
 this.signalssup++;
if(s==1)
 this.signalsdis++;
if(s==2)
 this.signalsman++;
//System.out.println("Starting new round Total signal"+this.signals+" Total required
"+(Stat.sumi(g.Aliveman)+Stat.sumi(g.Aliveagent)+Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Al
ivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.Alivedist[3]))+"Total signal
supplier"+this.signalssup+" Total required sup"+Stat.sumi(g.Aliveagent)+"Total signal
manufacturer"+this.signalsman+" Total required man"+Stat.sumi(g.Aliveman)+"Total signal
distreibutot"+this.signalsdis+" Total required
distr"+(Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.su
mi(g.Alivedist[3])));
//System.out.println("Starting new round Total signal supplier"+this.signals3sup+" Total
required sup"+Stat.sumi(g.Aliveagent));
//System.out.println("Starting new round Total signal manufacturer"+this.signals3man+"
Total required man"+Stat.sumi(g.Aliveman));
//System.out.println("Starting new round Total signal distreibutot"+this.signals3dis+" Total
required
distr"+Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.su
mi(g.Alivedist[3]));
```

```
if( (this.signals<
(Stat.sumi(g.Aliveman)+Stat.sumi(g.Aliveagent)+Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Aliv
edist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.Alivedist[3]))) || (this.signalssup<
(Stat.sumi(g.Aliveagent)))|| (this.signalsman< (Stat.sumi(g.Aliveman)))||(this.signalsdis<
(Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.
Alivedist[3]))))
            {
            try {
               if(s==2)
           wait();
               } catch(InterruptedException e) {
           System.out.println("InterruptedException caught");
          }
     }
   else
   {
   notifyAll();
   this.signals=0;
   this.signalssup=0;
   this.signalsdis=0;
   this.signalsman=0;
   }
}
synchronized void determinewinner1(int s) {
this.signals1++;
if(s==0)
 this.signals1sup++;
if(s==1)
 this.signals1dis++;
if(s==2)
 this.signals1man++;
if( (this.signals1<
(Stat.sumi(g.Aliveman)+Stat.sumi(g.Aliveagent)+Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Aliv
edist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.Alivedist[3]))) || (this.signals1sup<
(Stat.sumi(g.Aliveagent)))|| (this.signals1man< (Stat.sumi(g.Aliveman)))||(this.signals1dis<
(Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.
Alivedist[3]))))
            {
            try {
           wait();
               } catch(InterruptedException e) {
           System.out.println("InterruptedException caught");
          }
```

```
      }
   else
   {
   notifyAll();
   this.signals1=0;
   this.signals1sup=0;
   this.signals1dis=0;
   this.signals1man=0;
   }
}


synchronized void determinewinner2(int s) {
//this.signals2++;
if(s==0)
 this.signals2sup++;
if(s==1)
 this.signals2dis++;
if(s==2)
 this.signals2man++;
if(s==3)
 this.signals2++;

if( (this.signals2< 1) || (this.signals2sup< (Stat.sumi(g.Aliveagent)))|| (this.signals2man<
(Stat.sumi(g.Aliveman)))||(this.signals2dis<
(Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.
Alivedist[3]))))
            {
            try {
                   wait();
               } catch(InterruptedException e) {
           System.out.println("InterruptedException caught");
          }
      }
   else
   {
   notifyAll();
   this.signals2=0;
   this.signals2sup=0;
   this.signals2dis=0;
   this.signals2man=0;
   }
}
```

```
synchronized void nextround(int s) {
//this.signals3++;
if(s==0)
 this.signals3sup++;
if(s==1)
 this.signals3dis++;
if(s==2)
 this.signals3man++;
if(s==3)
 this.signals3++;
// System.out.println("Next round Starting new round Total signal"+this.signals3+" Total
required
"+(Stat.sumi(g.Aliveman)+Stat.sumi(g.Aliveagent)+Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Al
ivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.sumi(g.Alivedist[3]))+"Total signal
supplier"+this.signals3sup+" Total required sup"+Stat.sumi(g.Aliveagent)+"Total signal
manufacturer"+this.signals3man+" Total required man"+Stat.sumi(g.Aliveman)+"Total
signal distreibutot"+this.signals3dis+" Total required
distr"+(Stat.sumi(g.Alivedist[0])+Stat.sumi(g.Alivedist[1])+Stat.sumi(g.Alivedist[2])+Stat.su
mi(g.Alivedist[3])));
if(this.signals3< 1)
{
            try {
              if(s==0 || s==1 ||s==2)
           wait();
               } catch(InterruptedException e) {
           System.out.println("InterruptedException caught");
          }
      }
   else
   {
   notifyAll();
   this.signals3=0;
   this.signals3sup=0;
   this.signals3dis=0;
   this.signals3man=0;
   }
}
}
```

9. Shared variable class between manufacturers(Qm)

```
class Qm {
 public float[][] score;
 public float[][] Highperf;
 public float[][] Highperfp;
 public float[] performance;
 public float[] performancep;
 public float[] cmsc;
 public int[] m1rank;
 public int[] m2rank;
 public int[] rank1;            // Array to store the rank of each manufacturer which is
determined based on performance
 public int[] rank2;
 public int[] rank;
 public int[] rankp;
 public int median=0;
 public int medianp=0;
 public float[] hpct;            // Array to store the High performance count(= sum of variable
Highperf over the last 8 rounds
public float[] hpor;           // Array to store the high performance odd's ratio for each
manufacturer
public float[] lhpor;          // Array to store the logarithm of high performance odd's ratio
public float totalsc= 0;
public int m1max;
public int m2max;
public int[] merge;
Glob g;

Qm(Glob g)
   { score= new float[G.man][G.percmp];
      Highperf= new float[G.man][G.percmp];
      Highperfp= new float[G.man][G.percmp];
      performance= new float[G.man];
      performancep= new float[G.man];
      cmsc= new float[G.man];
      rank1= new int[G.man];            // Array to store the rank of each manufacturer which is
determined based on performance
     rank2= new int[G.man];
     rank= new int[G.man];
     rankp= new int[G.man];
     median=0;
     medianp=0;
```

```
    hpct= new float[G.man];                // Array to store the High performance count(= sum of
variable Highperf over the last 8 rounds
    hpor= new float[G.man];            // Array to store the high performance odd's ratio for
each manufacturer
    lhpor= new float[G.man];           // Array to store the logarithm of high performance odd's
ratio
    totalsc= 0;
    this.g =g;
    m1max=0;
    m2max=0;
    // merge= new int[G.man];
   }
synchronized void updatemergerinf(int ind, float[] sc,float perf,float perfp)
{// merge[ind]=0;
  performance[ind]=perf;
  performancep[ind]=perfp;
  for(int i=0;i<G.percmp;i++)
   {score[ind][i]=sc[i];
    }
}

// Determine the rank of manufacturers based on their performance and score
synchronized void estrelativeperformance()
{
  int m =0;
  int mp =0;
  m1max=0;
  m1rank= new int[Stat.sumi(g.Aliveman)];
  m2rank= new int[Stat.sumi(g.Aliveman)];
  for(int r2=0;r2<G.man;r2++)
  {  if(g.Aliveman[r2]==1)
      { rank1[r2]=0;
        rank2[r2]=0;
        rankp[r2]=0;
        if(g.Aliveman[r2]==1)
          {         for(int r22=0;r22<G.man;r22++)
                     { if(g.Aliveman[r22]==1 && r22!=r2)
                              {if(performance[r22] >performance[r2])
                                        { rank1[r2]= rank1[r2]+1;
                                        if(m< rank1[r2])
                                          m=rank1[r2];
                                        }
                                 if(performancep[r22] >performancep[r2])
                                        { rankp[r2]= rankp[r2]+1;
                                          if(mp< rankp[r2])
                                          mp=rankp[r2];
                                        }

                                if(score[r22][G.percmp-1]>score[r2][G.percmp-1])
                                        { rank2[r2]= rank2[r2]+1;
                                        }
                              }
                     }
                    cmsc[r2]= (0.3f)*rank1[r2]+ (0.7f)*rank2[r2];
                     m1rank[rank1[r2]]=r2;
                     m1max=m;
               }
  }
  }
  m2max=0;
  for(int r2=0;r2<G.man;r2++)
  {
             if(g.Aliveman[r2]==1)
             {  rank[r2]=0;
                 for(int r22=0;r22<G.man;r22++)
                     { if(g.Aliveman[r22]==1 && r22!=r2)
                              {
                                if(cmsc[r22]<cmsc[r2])
                                        { rank[r2]= rank[r2]+1;
                                        }
                              }
                     }
                     if(rank[r2]>m2max)
                     {m2max= rank[r2];
                     }
                     m2rank[rank[r2]]= r2;
             }
  }
median = (int)(m/2);
medianp = (int)(mp/2);
}
// Determine the parameters to be used during mergers and acquisition
synchronized void estrelativeperformance2()
{
totalsc=0;
```

```
for(int r2=0;r2<G.man;r2++)
{
if(g.Aliveman[r2]==1)
{
 for(int r1=0;r1<(G.percmp-1);r1++)
   {
    Highperf[r2][r1]=Highperf[r2][r1+1];
    Highperfp[r2][r1]=Highperfp[r2][r1+1];
    }
            if(rank1[r2]<median)
               Highperf[r2][G.percmp-1]=1;
            else Highperf[r2][G.percmp-1]=0;

             if(rankp[r2]<medianp)
               Highperfp[r2][G.percmp-1]=1;
            else Highperfp[r2][G.percmp-1]=0;
               hpct[r2]=0;
               hpct[r2]= Stat.sum(Highperf[r2])+Stat.sum(Highperfp[r2]);
            if(hpct[r2] > 0)
               {     if(hpct[r2] <2*G.percmp)
                           { float t3=hpct[r2];
                             float t4= (2*G.percmp-hpct[r2]);
                             hpor[r2]= (float)Math.pow((t3/t4),2);
                            }
                          else hpor[r2]=10;
               }
               else
               {
                 hpor[r2]=1;
               }
             lhpor[r2] = (float)Math.log(hpor[r2]);
             String str="";
            for(int temp=0;temp<G.man;temp++)
            {str=str+" "+lhpor[temp];
            }
           totalsc= totalsc+score[r2][G.percmp-1];
   }
}
}
}
```

10. Shared variable class between suppliers(Qs)

```
class Qs {
   public float[][] score;
   public float[][] Highperf;
   public float[] performance;
   public float[] cmsc;
   public int[] s1rank;
   public int[] s2rank;
   public int[] rank1;            // Array to store the rank of each manufacturer which is
determined based on performance
   public int[] rank2;
   public int[] rank;
   public int smedian=0;
   public float[] hpct;              // Array to store the High performance count(= sum of variable
Highperf over the last 8 rounds
   public float[] hpor;          // Array to store the high performance odd's ratio for each
manufacturer
   public float[] lhpor;         // Array to store the logarithm of high performance odd's ratio
   public float totalsc= 0;
   public int s1max;
   public int s2max;
   Glob g;
   Qs(Glob g)
   { score= new float[G.agent][G.percmp];
  Highperf= new float[G.agent][G.percmp];
  performance= new float[G.agent];
  cmsc= new float[G.agent];
  rank1= new int[G.agent];            // Array to store the rank of each manufacturer which is
determined based on performance
  rank2= new int[G.agent];
  rank= new int[G.agent];
  smedian=0;
  hpct= new float[G.agent];                // Array to store the High performance count(= sum of
variable Highperf over the last 8 rounds
  hpor= new float[G.agent];           // Array to store the high performance odd's ratio for each
manufacturer
  lhpor= new float[G.agent];          // Array to store the logarithm of high performance odd's
ratio
  totalsc= 0;
  this.g =g;
  s1max=0;
  s2max=0;
   }
```

```
synchronized void updatemergerinf(int ind, float[] sc,float perf)
{
performance[ind]=perf;
for(int i=0;i<G.percmp;i++)
{score[ind][i]=sc[i];
}
}
// Determine the rank of suppliers based on their performance and score
synchronized void estrelativeperformance()
{ int m =0;
   s1max=0;
   s1rank= new int[Stat.sumi(g.Aliveagent)];
   s2rank= new int[Stat.sumi(g.Aliveagent)];
   for(int r2=0;r2<G.agent;r2++)
  {  if(g.Aliveagent[r2]==1)
    { rank1[r2]=0;
       rank2[r2]=0;
            if(g.Aliveagent[r2]==1)
            {       for(int r22=0;r22<G.agent;r22++)
                     { if(g.Aliveagent[r22]==1 && r22!=r2)
                               {if(performance[r22] >performance[r2])
                                           { rank1[r2]= rank1[r2]+1;
                                           if(m< rank1[r2])
                                              m=rank1[r2];
                                           }
                              if(score[r22][G.percmp-1]>score[r2][G.percmp-1])
                                           { rank2[r2]= rank2[r2]+1;
                                           }
                               }
                     }
                    cmsc[r2]= (0.3f)*rank1[r2]+ (0.7f)*rank2[r2];
                    s1rank[rank1[r2]]=r2;
                    s1max=m;
            }
  }
  }
  s2max=0;
  for(int r2=0;r2<G.agent;r2++)
  {
            if(g.Aliveagent[r2]==1)
            {  rank[r2]=0;
                for(int r22=0;r22<G.agent;r22++)
                     { if(g.Aliveagent[r22]==1 && r22!=r2)
```

```
                               {
                              if(cmsc[r22]<cmsc[r2])
                                       { rank[r2]= rank[r2]+1;
                                       }
                               }
                        }
                        if(rank[r2]>s2max)
                        {s2max= rank[r2];
                        }
                        s2rank[rank[r2]]= r2;
                }
  }
smedian = (int)(m/2);
String tempstring="s1rank :   ";
for(int tempc=0;tempc<s1rank.length;tempc++)
{tempstring=tempstring+"\t"+s1rank[tempc];
}
//System.out.println(tempstring);
tempstring="s2rank :   ";
for(int tempc=0;tempc<s2rank.length;tempc++)
{tempstring=tempstring+"\t"+s2rank[tempc];
}
//System.out.println(tempstring);
}
// Determine the parameters to be used during mergers and acquisition
synchronized void estrelativeperformance2()
{
totalsc=0;
 for(int r2=0;r2<G.agent;r2++)
 {
  for(int r1=0;r1<(G.percmp-1);r1++)
    {
     Highperf[r2][r1]=Highperf[r2][r1+1];
     }
           if(rank1[r2]<smedian)
              Highperf[r2][G.percmp-1]=1;
        else Highperf[r2][G.percmp-1]=0;
              hpct[r2]=0;
              hpct[r2]= Stat.sum(Highperf[r2]);
        if(hpct[r2] > 0)
              {     if(hpct[r2] < G.percmp)
                         { float t3=hpct[r2];
                           float t4= (G.percmp-hpct[r2]);
```

```
                    hpor[r2]= (float)Math.pow((t3/t4),2);
                   }
                 else hpor[r2]=10;
          }
          else
          {
            hpor[r2]=1;
          }
        lhpor[r2] = (float)Math.log(hpor[r2]);
        String str="";
        for(int temp=0;temp<G.agent;temp++)
        {str=str+" "+lhpor[temp];
        }

       if(g.Aliveagent[r2]==1)
       totalsc= totalsc+score[r2][G.percmp-1];
  }
}
}
```

11. Shared variable class between distributors(Qd)

```
class Qd {
   private int signalsup = 0;
   private int signaldis = 0;
   private int signalman = 0;
   private int signal = 0;
   public float[][][] score;
   public float[][][] Highperf;
   public float[][] performance;
   public float[][] cmsc;
   public int[][] d1rank;
   public int[][] d2rank;
   public int[][] rank1;              // Array to store the rank of each manufacturer which is
determined based on performance
   public int[][] rank2;
   public int[][] rank;
   public int[] dmedian;
   public float[][] hpct;                // Array to store the High performance count(= sum of
variable Highperf over the last 8 rounds
   public float[][] hpor;             // Array to store the high performance odd's ratio for each
manufacturer
   public float[][] lhpor;            // Array to store the logarithm of high performance odd's ratio
   public float[] totalsc;
   public int[] locmax1;
   public int[] locmax2;
   Glob g;

   Qd(Glob g)
   {this.g =g;
    score= new float[G.Region][G.distributor][G.percmp];
    Highperf= new float[G.Region][G.distributor][G.percmp];
    performance= new float[G.Region][G.distributor];
    cmsc= new float[G.Region][G.distributor];
    rank1= new int[G.Region][G.distributor];              // Array to store the rank of each
manufacturer which is determined based on performance
   rank2= new int[G.Region][G.distributor];
   rank= new int[G.Region][G.distributor];
   dmedian= new int[G.Region];
   hpct= new float[G.Region][G.distributor];                // Array to store the High performance
count(= sum of variable Highperf over the last 8 rounds
   hpor= new float[G.Region][G.distributor];             // Array to store the high performance
odd's ratio for each manufacturer
```

```
  lhpor= new float[G.Region][G.distributor];           // Array to store the logarithm of high
performance odd's ratio
  totalsc= new float[G.Region];
  locmax1=  new int[G.Region];
  locmax2=  new int[G.Region];
  }

synchronized void updatemergerinf(int loc, int ind, float[] sc,float perf)
{
performance[loc][ind]=perf;
for(int i=0;i<G.percmp;i++)
{score[loc][ind][i]=sc[i];
}
}

// Determine the rank of distributors based on their performance and score
synchronized void estrelativeperformance()
{   d1rank= new int[G.Region][G.distributor];
    d2rank= new int[G.Region][G.distributor];
  for(int loc1=0;loc1<G.Region;loc1++)
  { int m =0;
   locmax1[loc1]=0;
  for(int r2=0;r2<G.distributor;r2++)
   {
    if(g.Alivedist[loc1][r2]==1)
    { rank1[loc1][r2]=0;
      rank2[loc1][r2]=0;
            if(g.Alivedist[loc1][r2]==1)
            {       for(int r22=0;r22<G.distributor;r22++)
                     { if(g.Alivedist[loc1][r22]==1 && r22!=r2)
                               {//System.out.println("Performance of distributor"+ r22+" in location
"+loc1+" is "+performance[loc1][r22]);
                                if(performance[loc1][r22] >performance[loc1][r2])
                                         { rank1[loc1][r2]= rank1[loc1][r2]+1;
                                         if(m< rank1[loc1][r2])
                                            m=rank1[loc1][r2];
                                         }
                              if(score[loc1][r22][G.percmp-1]>score[loc1][r2][G.percmp-1])
                                         { rank2[loc1][r2]= rank2[loc1][r2]+1;

                                         }
                               }
                     }
```

```
                    cmsc[loc1][r2]= (0.3f)*rank1[loc1][r2]+ (0.7f)*rank2[loc1][r2];
                    d1rank[loc1][rank1[loc1][r2]]=r2;
                    locmax1[loc1]=m;
            }
      }
      else{rank1[loc1][r2]=100;
          rank2[loc1][r2]=100;
      }
  }
  locmax2[loc1]=0;
  for(int r2=0;r2<G.distributor;r2++)
  {
            if(g.Alivedist[loc1][r2]==1)
            {  rank[loc1][r2]=0;
                for(int r22=0;r22<G.distributor;r22++)
                     { if(g.Alivedist[loc1][r22]==1 && r22!=r2)
                              {
                             if(cmsc[loc1][r22]<cmsc[loc1][r2])
                                        { rank[loc1][r2]= rank[loc1][r2]+1;
                                        }
                              }
                     }
                     if(rank[loc1][r2]> locmax2[loc1])
                     {locmax2[loc1]= rank[loc1][r2];
                     }
                     d2rank[loc1][rank[loc1][r2]]= r2;
            }
  }
dmedian[loc1] = (int)(m/2);
}
}

// Determine the parameters to be used during mergers and acquisition
synchronized void estrelativeperformance2()
{
for(int loc1=0;loc1<G.Region;loc1++)
{totalsc[loc1]=0;
for(int r2=0;r2<G.distributor;r2++)
 {if(g.Alivedist[loc1][r2]==1)
  {
for(int r1=0;r1<(G.percmp-1);r1++)
    {
     Highperf[loc1][r2][r1]=Highperf[loc1][r2][r1+1];
    }

            if(rank1[loc1][r2]<dmedian[loc1])
               Highperf[loc1][r2][G.percmp-1]=1;
            else Highperf[loc1][r2][G.percmp-1]=0;
            hpct[loc1][r2]=0;
            hpct[loc1][r2]= Stat.sum(Highperf[loc1][r2]);
            if(hpct[loc1][r2] > 0)
               {     if(hpct[loc1][r2] < G.percmp)
                          { float t3=hpct[loc1][r2];
                            float t4= (G.percmp-hpct[loc1][r2]);
                            hpor[loc1][r2]= (float)Math.pow((t3/t4),2);
                           }
                         else hpor[loc1][r2]=10;
               }
            else
               {
                 hpor[loc1][r2]=1;
               }
            lhpor[loc1][r2] = (float)Math.log(hpor[loc1][r2]);
           if(g.Alivedist[loc1][r2]==1)
            totalsc[loc1]= totalsc[loc1]+score[loc1][r2][G.percmp-1];
  }
  }
}
}
}
```

12. Key generator class

```
public class keygen {
  int win=0;
  int loc=0;
  int sim=0;
  int j=0;
  int h=0;
  int r1=0;
  int temp=0;
  String s1;
  String s2;
  String s3;
  String s4;
  String s5;
  String s6;
  String s7;
  keygen(){};
  public synchronized String Keygenerator(int sim,int j,int h,int r1,int temp,int win,int loc)

   {        String key= "";
            s1="";
            if(sim<9)
            {s1= ""+"00"+""+(sim+1)+"";
            }else{ if(sim<99)
                       {s1= ""+"0"+""+(sim+1)+"";
                       }else{ s1= ""+(sim+1)+"";
                             }
            }

            s2="";
            if(j<9)
            {s2= ""+"00"+""+(j+1)+"";
            }else{ if(j<99)
                       {s2= ""+"0"+""+(j+1)+"";
                       }else{ s2= ""+(j+1)+"";
                             }
            }

            s3="";
            if(h<9)
            {s3= ""+"00"+""+(h+1)+"";
            }else{ if(h<99)
                       {s3= ""+"0"+""+(h+1)+"";
```

```
                  }else{ s3= ""+(h+1)+"";
                       }
            }

            s4="";
            if(r1<9)
            {s4= ""+"00"+""+(r1+1)+"";
            }else{ if(r1<99)
                  {s4= ""+"0"+""+(r1+1)+"";
                  }else{ s4= ""+(r1+1)+"";
                       }
            }
            s5="";
            if(temp==1)
                  s5="0D0";
            if(temp==2)
                  s5="0S0";
            if(temp==3)
                  s5="0M0";

            s6="";
            if(win<9)
            {s6= ""+"00"+""+(win+1)+"";
            }else{ if(win<99)
                  {s6= ""+"0"+""+(win+1)+"";
                  }else{ s6= ""+(win+1)+"";
                       }
            }
            s7= "";
            if(temp==1)
            {if(loc<9)
            {s7=""+"0"+""+(loc+1)+"";
            }else{s7=""+(loc+1)+"";
            }
            }else{s7="00";
            }
          key= s1+""+s2+""+s3+""+s4+""+s5+""+s6+""+s7;
          return(key);
    }
    }
```