e-Game: A Generic Auction Platform Supporting Customizable Market Games

Maria Fasli
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ, UK
mfasli@essex.ac.uk

Michael Michalakopoulos
University of Essex
Wivenhoe Park, Colchester, CO4 3SQ, UK
mmichag@essex.ac.uk

Abstract

This paper presents the e-Game (electronic-Generic Auction Marketplace) platform. e-Game is a configurable on-line auction server that supports both human and software agents. A wide range of auction types are currently supported by e-Game. These are parameterized in order to allow users to tailor-make auctions according to their own needs. The main feature of e-Game is that it supports the design, development and execution of market game scenarios involving auctions analogous to those of the Trading Agent Competition (TAC) by third parties. Such game scenarios can be used for conducting research on markets, mechanism design and strategies as well as for educational purposes.

1 Introduction

With the advent of the internet there has been a mounting interest in deploying agent-based and multi-agent systems to fully automate commercial transactions. Semi-autonomous software agents are continuously running entities that can negotiate for goods and services on behalf of their users reflecting their preferences and perhaps negotiation strategies. Commercial transactions involve broadly speaking three main phases: firstly potential buyers and sellers must find each other or meet in a marketplace, secondly they need to negotiate the terms of the transaction and finally they execute the transaction and the goods/monetary resources change hands. Despite their increased popularity, the huge potential of agents and multi-agent systems in e-commerce hasn't been fully realized [10].

One of the most popular ways of negotiating for goods and services is via auctions [6, 8]. Auctions constitute a general class of negotiation protocols in which price is determined dynamically by the auction participants. The task of determining the value of a commodity is transferred from the vendor to the market, thus leading to a fairer allocation of resources based on who values them most. Auctions are perhaps better known as a means of selling works of art or antiques but with the growth of the internet electronic auctions have been used extensively to trade products and services ranging from holidays to computer spare parts [2]. Constructing trading agents to take part in auctions for a single good is relatively simple. However, in reality individual customers or businesses may have to negotiate for a bundle of perhaps interrelated goods being auctioned in different auctions following different rules. Developing agents that can participate in simultaneous auctions offering complementary and substitutable goods is a complex task. The successful performance of a trading agent does not only depend on its strategy but on the strategy of the other agents as well. Designing efficient and effective bidding strategies is difficult since real world data about trading agents is difficult to obtain.

One approach to this problem is to run simulated market scenarios and test one's bidding strategies. However, the results of such experiments conducted by a single person may lack the touch of realism, since the bidding strategies explored may not necessarily reflect diversity in reality. This is the central idea and major motivation behind the International Trading Agent Competition (TAC) [5] which features artificial trading agents competing against each other in market-based scenarios. Currently two scenarios are available and run as part of the competition: the Classic Travel Agent Scenario game and the latest Supply Chain Management game. Ideally, we would like open-source platforms where market-based scenarios could be implemented and researchers have the opportunity to build and test their trading agents against those of others. This is the role that we envisage e-Game will play. e-Game[1] is an on-line configurable auction platform that supports the development of customizable auction-based market scenarios by third parties. This paper discusses e-Game and its features. The structure of the paper is as follows. We start with the architecture and the most significant features of the system. Auctions as negotiation protocols, their support and implementation is discussed next. The following section discusses game development and the facilities provided by e-Game to

[1] http://cswww.essex.ac.uk/staff/mfasli/e-game.htm

Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)
0-7695-2101-0/04 $ 20.00 IEEE

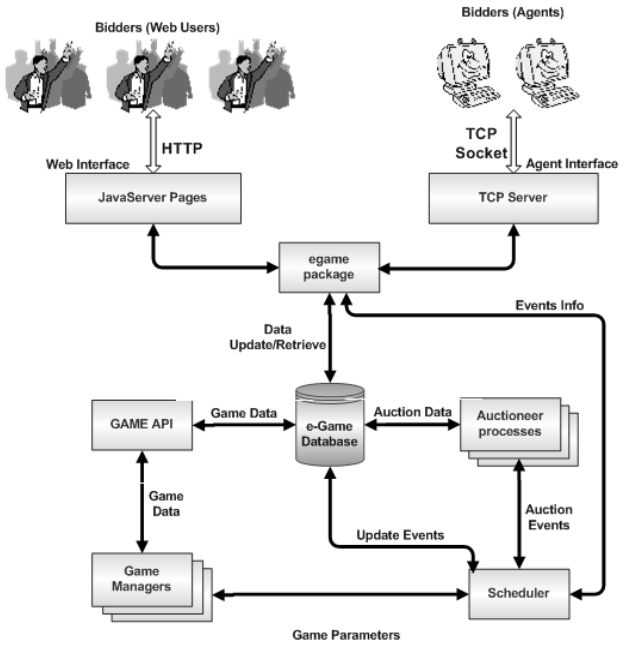

Figure 1. The architecture of e-Game.

aid this. A simple game and its development process is also presented. Finally, the paper ends with the conclusions and pointers to future work.

2 The e-Game Platform

e-Game (**e**lectronic **G**eneric **a**uction **m**arketplac**e**) is a configurable auction server in which participants may be humans or software agents. It is a generic platform that can be extended to support many types of auctions. Additionally, e-Game is designed to support auction games analogous to those found in the Trading Agent Competition [5] which can be designed and developed by third parties.

2.1 Architecture

e-Game is based on a modular architecture separating the interface layer from the data processing modules and the database (Figure 1). Its modularity and expandability are further strengthened by adopting the object-oriented paradigm for its components. The platform has been developed in JAVA [4] in order to gain the benefits of platform independence and transparency in networking communication and object handling. Some of the main components and features of the e-Game platform are described next.

2.1.1 Interfaces

e-Game is designed to allow for both web users and software agents to participate and as such has two interfaces to the outer world: the *Web* and the *Agent* interfaces similarly to [11]. The *Web* interface provides a series of functions such as registration, creation of new auctions using a variety of parameters (auction type, information revealing, clearing and closing mechanism), bid submission, search facilities according to a series of attributes (auctioned item, status of auction, user's participation in an auction) and viewing the user's profile (change personal information, view previous auction information).

The *Agent* interface provides agent developers the same functionality with the *Web* interface with respect to auctions plus an additional set of commands for participating in market games. Agents connect to the *Agent* interface using the TCP protocol and submit their commands using a simple string format. The majority of the functions that the web users and the agents perform are identical and are implemented in a single package which is shared between the two interfaces. The advantage of such an approach is that both components can be easily extended in the future with the minimum effort in terms of implementation and testing.

2.1.2 Scheduler

The *Scheduler* runs in parallel with the *Web* and the *Agent* interfaces and is responsible for starting up auctions, games and passing bids to the appropriate *Auctioneer* processes. The auctions can be scheduled by web users or by a specific *GameManager* handler according to a market scenario's rules, whereas games are scheduled by web users. The *Scheduler* provides a fault-tolerant behaviour, since if it is brought off-line for some reason, when it is restarted it can reconstruct its lists of events by looking into the database for pending events.

2.2 Auctions

Auctions constitute a general class of negotiation protocols in which price is determined dynamically by the auction participants. In auctions goods of undetermined quality can be sold. They can be used for the sale of a single item, or for multiple units of a homogeneous item as well as the sale of interrelated goods [1]. There are two main (self-) interested parties in an auction: The auctioneer who wants to sell goods at the highest possible price (or subcontract out tasks at the lowest possible price) who may be the owner of the good or service or a representative of the actual seller, and the bidders who want to buy goods at the lowest possible price (or get contracts at the highest possible price). Bidders bid in order to acquire an item for personal use, for resale or commercial use. The task of determining the value

of a commodity is transferred from the vendor to the market, thus leading to a fairer allocation of resources based on who values them most. The process of an auction involves the following typical stages:

• Registration. Buyers and sellers register with an auction house.

• Bidding Phase. The participants bid according to the rules of the particular auction protocol used. A bid indicates a bound on the bidders' willingness to buy or to sell a good.

• Bid Processing. The auctioneer checks the validity of a bid according to the rules of the auction used and updates its database (manual or electronic).

• Price quote generation. The auction house via the auctioneer or by other means may provide information about the status of the bids. A *bid quote* is the highest outstanding effective offer to buy, that is the current price that a bidder has to exceed in order to acquire the good. An *ask quote* is the lowest outstanding effective offer to sell, that is if a seller wants to trade, it will have to be willing to accept less than that price.

• Clearance (or Matching). Through clearance, buyers and sellers are matched and the transaction price is set. What is termed clearing price is the final transaction price.

• Transaction Phase. The transaction actually takes place: the buyer bidder pays and receives the good/service.

Traditional auction protocols are generally classified depending on whether multiple buyers and sellers are allowed (single/double), the bidders have information on each other's bids (open/closed), the flow of prices (ascending/descending) and how the bidders form their valuations (private/common/correlated value). Among the most well known auctions are the English, Dutch, FPSB, Vickrey and CDA. The English auction is an open-outcry and ascending-price auction which begins with the auctioneer announcing the lowest possible price (which can be a reserved price). Bidders are free to raise their bid and the auction proceeds to successively higher bids. When there are no more raises the winner of the auction is the bidder of the highest bid. The Dutch auction is an open and descending-price auction. The auctioneer announces a very high opening bid and then keeps lowering the price until a bidder accepts it. The FPSB (First-Price Sealed-Bid) and Vickrey (also known as uniform second-price sealed-bid) auctions have two distinctive phases: the bidding phase in which participants submit their bids, and the resolution phase in which the bids are opened and the winner is determined. The bid that each bidder submits in both auctions is sealed. In the FPSB auction the highest bidder wins and pays the amount of its bid, while in the Vickrey auction the higher bidder wins, but pays the second-highest bid. Finally, the CDA (Continuous Double Auction) is a general auction mechanism that is used in commodity and stock markets [1]. Multiple buyers and sellers can participate in such auctions and a number of clearing mechanisms can be used. For instance in the stock market clearing takes place as soon as bids are matched.

Auction Type	Sellers	Units	Buyers
English	1	1	Many
Vickrey	1	1	Many
FPSB	1	1	Many
Dutch	1	1	Many
Mth Price	1	Many	Many
Continuous Single	1	Many	Many
Double Auction	Many	Many	Many

Table 1. Auctions supported by e-Game.

2.2.1 Auction Support

Currently e-Game supports a wide range of auction types (Table 1). These basic types of auctions can be further refined by allowing the user to define additional parameters that include [11]:

• Price quote calculation: Upon arrival of new bids, at fixed periods or after a certain period of buy/sell inactivity.

• Auction closure: At a designated date and time or after a period of buy/sell inactivity.

• Intermediate clearing: Upon arrival of new bids, at fixed periods or after a certain period of buy/sell inactivity.

• Information revelation: Whether the price quotes and the closing time are revealed or not.

Such a parameterization can have an effect on the users' or agents' behaviour, since they modify the basic functionality of an auction as well as change the amount of information revealed regarding bids and closing time.

The actual auctions are carried out by the *Auctioneer* classes. Each *Auctioneer* is started by the *Scheduler* and implements a specific auction protocol. Therefore, there are as many *Auctioneer* classes, as the number of auction protocols supported by e-Game. Since all the *Auctioneer*s perform alike actions, they inherit the basic behaviour and state from a *BasicAuctioneer* class. Once an auction starts, the *BasicAuctioneer* class is responsible for generating the auction's events based on the choices the user made during the setup. At a given moment in time there may be a number of similar or different classes of active *Auctioneers* within the e-Game platform, each one handling a different auction.

2.2.2 Message Types

Agents connect to the *Agent* Interface using the TCP protocol and submit their commands using a simple string format which complies with the syntax of the HTTP query string: (command: parameter1=value1\&..parameterN=valueN). This facilitates participation in individual auctions as well as auction-based market exercises. Such commands allow

agents for instance, to retrieve information about an auction such as start up and closing time, submit bids to auctions or obtain information about the state of a submitted bid. The full set of commands is available for agent developers. For every command an agent submits, the e-Game platform returns an appropriate result together with a command status value that indicates the outcome of the command.

2.2.3 Auction Implementation

The implementation of all auctions is based on the Mth and M+1st price clearing rules. The clearing rules for the auctions are just special cases of the application of the Mth and M+1st clearing rules. However, much depends on the initial set up performed by the user. For instance to achieve a chronological order clearing in an auction one can set the intermediate clearing periods to be as soon as a bid is received and thus the parameterized auctioneer will attempt to perform intermediate clearings upon the arrival of a new bid. As in classical chronological matching, if a portion of the bid cannot transact, then it remains as a standing offer to buy (or sell). The implementation of the Mth and M+1st clearing rules uses ordered lists.

3 Game Development

Apart from the support for agents and web users regarding auction operations, e-Game's main feature is that it supports the design, development and execution of different market scenarios or games that involve auctions analogous to [5]. Since the e-Game platform allows for web users as well as agents to participate in auctions, it is feasible for everyone to participate in a market game as well. This may sometimes be desirable when comparisons between humans and software agents need to be drawn with respect to laying out a strategy to maximize utility.

Prior to designing a new game, a developer should have a good understanding of the auction types supported by e-Game. The web site offers a glossary that describes the different types of auctions and their setup parameters. If the participants of a game are web users, the web site already provides the functionality to participate in the auctions involved in the game. However in most cases, one is interested in designing a game that will be played by software agents. When designing a new game, the developer should have in mind the following key points:

• Realism: The more realistic the auction scenario is, the more comprehensible it will be to the people who wish to develop agents to participate in it. Ideally, the auction-based market scenario should describe a realistic situation, which may be encountered as part of everyday life (holiday planning, buying interrelated goods, scheduling of resources).

• Efficient use of e-Game: Ideally, the game should not be based on a single type of auction: the game developer has a wide range of auctions to choose from. The usage of different types of auctions raises more challenges for the participants, who will have to lay out different strategies depending on the auction type and the amount of information that is revealed at run time.

• Fairness with regards to specifications: In an auction game different agents with most probably different strategies will try to accomplish a certain goal. Though the goal will be similar in concept for every agent (for example, assemble one computer by combining different resources), the achievement of individual goals will give different utilities for each agent. The developer should consider specifications that give everyone the opportunity to come up with the maximum utility.

• Fairness with regards to agent design: Agents with a more appropriate bidding strategy should manage to achieve a better outcome, than agents with "naive" and "simplistic" strategies. When "naive" strategies actually represent corresponding actions in real life (for which people can actually reason for), this prerequisite does not hold. However, the purpose of introducing this key point is that there should not be any flaws in the market game that would allow non-realistic strategies to outperform rational ones.

• Computational efficiency: The handler of an auction game should have the ability to operate in a real-time manner. Normally, there should not be any algorithms that take a long time to run, especially in critical time points. It is acceptable to have this kind of algorithms at the point where the game ends and scores have to be calculated.

It is the developer's responsibility to ensure that a game is realistic, fair and non-trivial. Transparency in e-Game is enforced by publishing the logfiles and outcome of auctions.

3.1 Game Implementation

In terms of implementation, game development is simplified by extending the *GenericGame* class as provided by e-Game. As a consequence, developers are expected to be familiar with the Java programming language and provide the implementation for a number of processes in the form of methods that are part of a game. In general a game involves the following phases:

• Set up. In this initial phase of the game the developer sets up the game environment and specifies the auctions that are to be run and the various parameters for these. The *startupGame* method needs to be implemented to provide this information.

• Generation of parameters. Since a marketplace is simulated in a game, agents that will be participating in this will have some objective such as for instance to acquire goods on behalf of some clients while at the same time max-

imizing their profit. Client preferences, initial endowments, other information regarding the state of the market, scheduled auctions, availability of goods and other resources and any other data the developer wants to communicate to each agent in the beginning of the game can be done through the *generateParameters* method. The agents are responsible for retrieving this information and then using it in their strategy.

• Execution. In this phase the actual auctions that are part of the game run, agents are allowed to submit bids according to the auctions rules and information is revealed according to the parameters specified in the *startupGame* method. e-Game runs the auctions and handles the messages that need to be exchanged with the agents. The agents are responsible for retrieving information on the status of their bids and making decisions accordingly during this phase of the game.

• Score calculation. An agent's success is usually measured in terms of a utility function. This obviously depends on the strategy that an agent is using, however the utility function that measures how successful an agent is needs to be provided by the developer. The *calculateScores* method allows the developer to look into the auctions' results and calculate the utility for each individual agent.

• Close down. This is the final phase of a game and it is automatically called at the end. This also allows for any necessary clean up code to be performed.

Game developers do not have the ability to directly call methods from the other platform's components or gain direct access to the e-Game database. They can only use the methods that the *GenericGame* class offers them which allow them to perform a wide range of tasks such as:

• Retrieve the IDs of the participants in the current game.

• Schedule any type of auction according to the rules of the game (for example, set a reserved price or choose not to reveal the closing time).

• Associate certain generated parameters with an agent.

• Retrieve information for the auctions associated with this game, such as their IDs, closing time or current winning bids (quantity and price) and their owners.

• Associate a score with a certain agent.

• Submit bids for a certain auction, for example when there is a Descending Price auction and the *GameManager* is actually the Seller.

The developer can also provide two more classes when developing a new game:

• A sample agent that other users can extend according to their needs and that will also be used as an agent that will fill one of the available player slots when starting a game.

• An applet that can be loaded at runtime, so users can monitor the progress of a game.

The decision whether to design and implement a sample Agent and an applet is left to the developer. Once the game and the applet have been developed, it is very easy to integrate it with the e-Game platform. The developer simply needs to provide the class file, together with an XML file that describes general properties of the game such as the name of the game, the name of the implemented classes (game, applet), a short description for the game, its duration in seconds, the number of allowed participants and the resources (items) that this game will use.

Information is parsed by the *GameImporter* application and is stored in the database. Web users can browse the web site and by following the link "Agent Games" they can view all the installed types of games, together with appropriate links to schedule new instances of games, monitor the progress of a current game and view previous scores. When a new game is scheduled, the *Scheduler* receives the corresponding event and at the appropriate time, loads the user defined *GameManager* class. Since the name of the user defined class is not known until runtime, the *Scheduler* uses Java's Reflection mechanism to dynamically load it. At the end of a game, users can view their score and the resources, they managed to obtain, as well as the initial parameters.

3.2 The Computer Market Game

To provide the reader with a feeling of the capabilities of the e-Game platform regarding game development, this section discusses the implementation of a simple game. In the Computer Market Game (CMG), which lasts nine minutes, each of the six agents participating is a supplier agent whose task is to assemble PCs for its five clients. For simplicity, there are only three types of parts that are necessary to make up a properly working PC: a motherboard, a case and a monitor. There are three types of motherboards, each one bundled with CPUs of 1.0 GHz (MB1), 1.5 GHz (MB2) and 2.0 GHz (MB3). Cases come in two different types, one packed with a DVD player (C1) and another one with a DVD/RW (C2) combo drive. For the purpose of this game there is only one monitor type.

At the beginning of the game the agents receive their clients' preferences in terms of a bonus value for upgrading to a better motherboard (MB2 or MB3) and a better case (C2). So for instance, in a particular game instance an agent participating in the game may receive the preferences illustrated in Table 2. e-Game generates values in the following ranges for these preferences MB2 = [100..150], MB3=[150..200], C2=[200..300]. Hence, in the above example client 2 of agent 1 has a bonus of 130 for obtaining a 1.5 GHz motherboard and a bonus of 200 for upgrading to a 2.0 GHz motherboard. For upgrading to the better case with the DVD/RW drive the customer is offering 300.

Each component is available in a limited quantity and is traded in different auctions. Table 3 summarizes the quantities of the items and the respective auctions. Figure 2 illustrates how the auctions for the various items are scheduled

Agent	Client	MB2	MB3	C2
1	1	110	150	220
1	2	130	200	300
1	3	120	170	250
1	4	150	184	296
1	5	100	156	201

Table 2. Example of client preferences.

Component	Quantity	Auction
MB1	17	Mth Price
MB2	8	Mth Price
MB3	5	Mth Price
C1	20	Mth Price
C2	10	Mth Price
M	30	Continuous single seller

Table 3. Component availability and auctions.

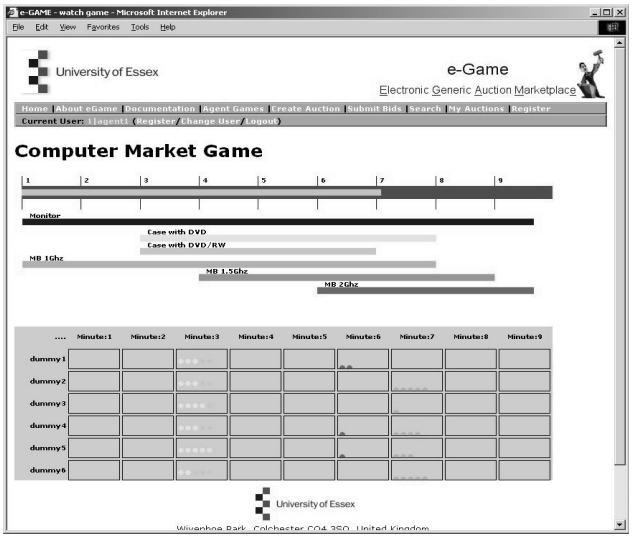

Figure 3. The CMG applet.

during the nine minutes of the game. The dotted part of the lines indicate that the auctions may close anytime in that period, but the exact closing time is not revealed to the agents.

An agent's success in this game depends on the satisfaction of its clients. An individual client's utility (CU) is:

CU = 1000 + Motherboard Bonus + Case Bonus

For each client that has been allocated a completed PC the agent gets 1000 monetary units plus any bonus for upgrading to a better motherboard or case. If no completed PC is allocated to a client, then this utility is 0. An agent's utility function (AU) is the sum of all the individual client utilities minus the expenses incurred:

AU = (CU1+CU2+CU3+CU4+CU5) - Expenses

The agent's strategy should be focused on providing a PC to each one of its clients or to as many as possible, while at the same time trying to minimize costs. There are obvious interdependencies between goods, as a fully assembled PC requires three components. In creating a strategy for this game, one has to take into account that the availability of goods is limited, prices in auctions may vary, auctions close

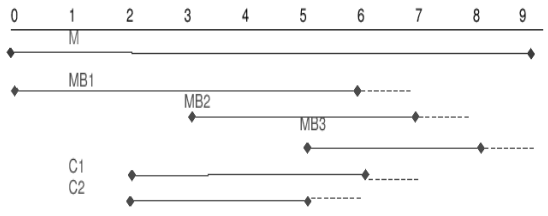

Figure 2. Auction schedule in CMG.

at different times and therefore one may have to switch to a different auction if they fail to acquire a certain good, and customers give different bonuses for upgrading to a better specification. Moreover, an agent's success, does not only depend on its own strategy, but that of the other agents too. However, an agent does not have any means of knowing the identity of the other players or monitoring their bids.

In order to implement the above scenario the developer would simply provide an XML file describing the general properties of the game (names of game and applet classes, number of players, duration of the game, auctioned resources), together with the appropriate classes. The specific *GameManager* would simply schedule the auctions at the beginning of the game by implementing the method *startupGame* and using the *scheduleAuction* methods. The next step would be to generate the parameters that each agent receives by implementing the method *generateParameters*. The utility of each agent would be calculated at the end of the game by examining the winning bids of each auction. Finally, any language resources that the developer used would be freed-up in the *closeGame* method. In addition, the developer may choose to write an applet so that web users can view the progress of their agent. Such an applet for this simple game is illustrated in Figure 3.

4 Conclusions

e-Game is a generic auction platform that allows web users as well as agents to participate in auctions and auction-based market games. Its main characteristic is that it was designed having in mind future expansions regarding support for new auctions and games. In order to face

the challenge of future expansions, the e-Game platform was built using the Object-Oriented paradigm and distributing the business logic (interfaces, packages, *Auctioneers*, *GameManagers*) among various components.

The aim of the e-Game project is to offer the infrastructure for running complex market-based scenarios and conducting experiments with different bidding strategies. As such, this paper serves as an invitation to agent researchers and developers to use the e-Game platform to develop and test their ideas. The most important feature of e-Game and what distinguishes it as a platform from other efforts such as [9, 7, 11, 5, 3] is that it provides independent developers the facilities to design, implement and run auction-based market scenarios. Unlike [3] for instance, e-Game does not simply provide the facilities for scheduling, running or conducting experiments on the use of strategies in standalone auctions. Individual users/developers can write their own *GameManager* and define complete market scenarios together with accompanying applets and sample agents that can be inserted and run on top of the e-Game infrastructure. The development of these modules is completely separate from the rest of the system. By making use of existing classes and documentation, users can develop new games quickly. The process of integrating user-developed scenarios is fully automated. This is feasible by introducing appropriate built-in classes for each module separately. By making use of Java's Reflection it is then possible to look into a newly introduced class and access its methods and data members. How game development is aided through e-Game is illustrated in the simple game discussed in the previous section.

Moreover, e-Game provides the facility of having humans play against agents. This may be useful in designing strategies and examining strategy efficiency and complexity. One could study the benefits from using agents in complex markets in a systematic way. For example, in a simple game-scenario, a human may do better than an agent, but it would be interesting to pit humans against software agents and study what happens when the scenario gets more complicated (penalty points, allocation problems etc.).

Apart from using the platform for designing and running complex market experiments, it can also be used in teaching issues in negotiation protocols, auctions and strategies. Students can have a hands-on experience with a number of negotiation protocols and put into practice the principles taught. Currently e-Game is used in a graduate course on agent technology for e-commerce and forms the basis of an assignment in which students have to implement agents for the CMG game. It has also served as the basis for a 3rd year undergraduate project for building another marketplace.

e-Game has been extensively tested by individual users scheduling and bidding in a variety of auctions simultaneously. Two servers are currently in operation and so far more than 1000 instances of the CMG game in which six agents are connected have been scheduled and run without any problems. The individual auctions that e-Game can run do not have limits to the number of bids or bidders that they can accept. The TCPServer could practically support many agents, which could be participating in different games at the same time.

The database is used to model the auction space (parameters such as auction type, startup,closing time, clearing rules), as well as the user data (registration, user bids, auctions the user created). It also stores information regarding the game (which game started which auctions, ids of players, scores). It provides fault tolerance, in that the events queue can be reconstructed if there is a problem with the auctioneers or the scheduler.

A possible future extension includes replacing the communication and bidding languages with a standard ACL language (FIPA ACL). We believe that such an action contributes to agents' standardization and establishment of automated electronic markets. e-Game could also be extended so that the *Agent Interface* provides the same functionality as the *Web Interface*. Therefore, software agents could potentially set up auctions, and search for auctions. For instance, an agent could search for auctions selling specific items and then decide on whether or not to participate as well as its strategy. Although this may not be necessary at the current stage it is certainly feasible.

References

[1] Auctions. http://www.agorics.com/Library/auctions.html.

[2] Ebay. http://www.ebay.com/.

[3] JASA - Java Auction Simulator API. http://www.csc.liv.ac.uk/ sphelps/jasa/.

[4] Java technology. http://www.java.sun.com.

[5] Trading Agent Competition. http://www.sics.se/tac/.

[6] Y. Bakos. The emerging role of electronic marketplaces on the internet. *Communications of the ACM*, 41(8):35–42, 1998.

[7] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, 1996.

[8] M. Kumar and S. Feldman. Internet auctions. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, pages 49–60, 1998.

[9] M. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski. Magma: An agent-based virtual market for electronic commerce. *Journal of Applied Artificial Intelligence*, 6, 1997.

[10] N. Vulkan. Economic implications of agent technology and e-commerce. *The Economic Journal*, 453:67–90, 1999.

[11] P. Wurman, M. Wellman, and W. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the Autonomous Agents Conference*, pages 301–308, 1998.