Distributed Gaming Using J2ME

A Writing Project

Presented to

The Faculty Of Department Of Computer Science

San Jose State University

In Partial Fulfillment of the Requirement for the Degree

Master Of Science

By

Rekha Vaddepalli

November 2004

Rekha Vaddepalli

rekha_vad@yahoo.com

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Christopher Pollett

Dr. Mark Stamp

Dr. Jon Pearce

APPROVED FOR THE UNIVERSITY

Abstract

The aim of this project is to develop a distributed economics game in which there can be several players playing on Personal Digital Assistants and a single central Database server. Economics games are very popular and have been around for many years [NPP91]. They help in understanding economics concepts and in simulating real world's market scenarios.

The game is a state-based game composed of the following states - Selection of the plots, Configuration, Production and Auction. In the Auction stage, each player decides to be a buyer or a seller depending on the production he has made and there is a central server that coordinates all the activities of these players. The player who maintains the lead in the score is the winner.

Several properties of the game such as the maximum number of players in each game, initial sum of money the players possess, score, expenditure and production formulae are configurable at the central server. These configurable parameters help in creating different scenarios. Several kinds of experiments are done and the results are recorded. This game application can be used to demonstrate the way in which the supply and demand determine the equilibrium market quantity and market price.

Table of Figures and Tables

1. Introduction

The use of small handheld computing devices has been increasing in recent years as people have become more conscious of using the time that they are mobile. In tandem, mobile gaming on these handheld computers has also been on the increase in these years.

Games can be utilized not only for entertainment but also for the simulation and understanding of some difficult concepts from the real world. For example, understanding the Economics concepts can be easier if you simulate the economic scenarios as games. The advent of computers has made it easier for both Economics instructors and students. These economic scenarios can be thought of as games and can be played among different players distributed geographically. Our project specifically is a distributed, multiplayer mobile game application with a central server that aims in serving as a framework to create different economic scenarios that might be possible in the local markets.

Mobile game development is being done on many platforms. We have utilized Java 2 Micro Edition (J2ME) for our game application development. J2ME is a subset of Java language platform that is optimized for small devices such as mobile phones and Personal Digital Assistants. The two components of J2ME are configurations and profiles. A configuration consists of a low-level API and a virtual machine (VM) for small devices. The two common configurations are Connected Device Configuration (CDC) and Connected Limited Device Configuration (CLDC). A profile is a set of APIs built on top

a configuration. Examples of profiles are the Mobile Information Device Profile (MIDP), Foundation Profile etc.

This project is based on the MIDP profile over the CLDC configuration. MIDP's user interface components have been used for creating the interactive user interface for the game. The persistent store has been utilized for storing the player's details. Network Input/Output has been achieved using the Generic Connection Framework Component. The communication between the players and the central server is achieved through Java Servlets. The information on the server is stored in the relational database and the communication between the central server and the database is achieved through Java Database Connectivity (JDBC).

The next chapter in this report gives the details of the technologies mentioned above. The third chapter analyzes the requirements of the project. The fourth chapter discusses the design and implementation of the project. The fifth chapter lists the experiments done to test the validity of the game and their results.

2. Background and related work

This chapter discusses the specific concepts of the technologies used in this project. The concepts of Java 2 Micro Edition (J2ME), Java Servlets, Java Database Connectivity (JDBC), and Oracle database are now discussed.

2.1 Java 2 Micro Edition concepts

The main reason for using Java is portability. As the mobile devices have limited resources, J2ME, a reduced version of Java API was designed. J2ME is composed of configurations and profiles. The Java Community Process Program [JCP02] defines a **configuration** as the Java run-time environment and core classes that operate on each device. The Java Virtual Machine (JVM) for a particular small computing device is defined by the configuration. There are two configurations – CLDC (Connected Limited Device Configuration) and CDC (Connected Device Configuration). Usually, CLDC devices are low-end mobile devices and CDC devices are high-end devices.

A **profile** consists of the Java classes that facilitate implementation of features for a class of small computing devices. The different profiles available now for the devices that fall under the CDC configuration are the Foundation Profile, the Game Profile, the Personal Profile, the Personal Basis Profile, and the RMI Profile. The CLDC profiles available are the Mobile Information Device Profile, the PDA profile, and the Information Device Profile. Each profile has a different functionality. Our project utilizes

the Mobile Information Device Profile (MIDP) over the CLDC configuration. As CLDC devices are subsets of CDC devices, this application can also run on CDC devices.

2.1.1 MIDP Applications (MIDlets)

MIDP applications are called MIDlets. There is another step after compilation called pre-verification in the case of MIDlets, as the memory on the small devices is scarce. The bytecode verification is divided into two pieces. Somewhere off the device, a preverify step is performed. The device itself is only required to do a lightweight second verification step before loading classes.

MIDP applications are deployed as MIDlet suites. The class files are packaged in a Java Archive (JAR). Every JAR file includes a manifest file, META-INF\MANIFEST.MF that describes the contents of the archive. The manifest file must contain important information needed for the MIDP runtime environment like the MIDlet's class name, and the versions of CLDC and MIDP that the MIDlet expects. Another file that is needed for the MIDlet deployment is the application descriptor file. This file is outside the JAR and enables the application management software to learn about a midlet without installing it.

The manifest file of the MIDlet suite in our project is

```
MIDlet-1: Acquire, Acquire.png, Acquire.acquireMIDlet
MIDlet-Name: acquire.acquireMIDlet
MIDlet-Vendor: Rekha
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

The application descriptor file of the MIDlet suite in our project is

Table 2: Application Descriptor file

```
MIDlet-1: Acquire, Acquire.png, acquire.acquireMIDlet
MIDlet-Jar-Size: 100
MIDlet-Jar-URL: Acquire.jar
MIDlet-Name: acquire.acquireMIDlet
MIDlet-Vendor: Rekha
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

MIDlet Lifecycle

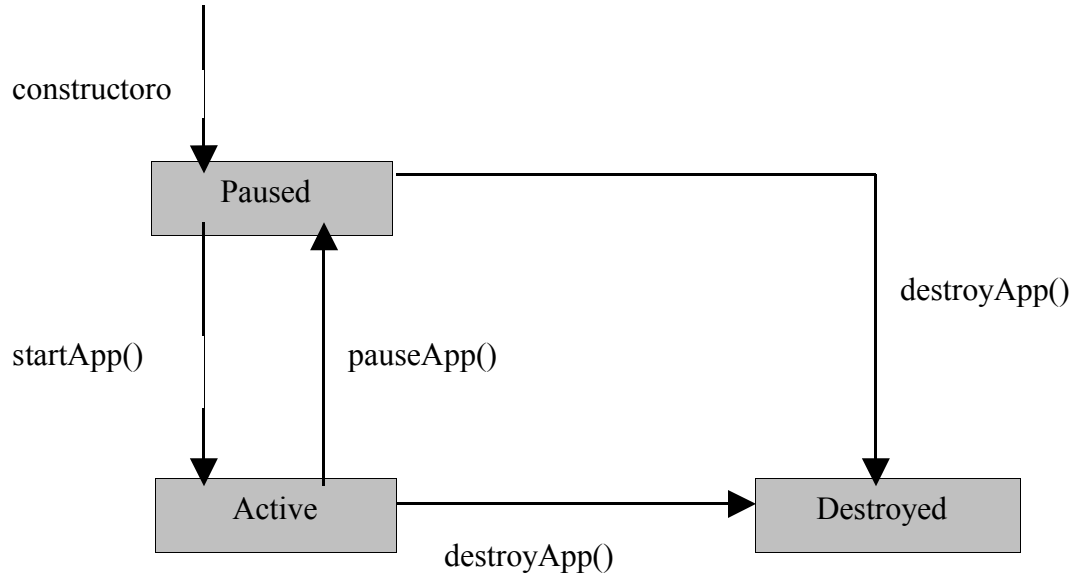


Figure 1: MIDlet Lifecycle

When the MIDlet is about to run, an instance is created. The MIDlet's constructor is run and the MIDlet is in the paused state. Next, the MIDlet enters the Active state after the application manager calls startApp(). While the MIDlet is active, the application manager can suspend its execution by calling pauseApp(). This puts the MIDlet back in the Paused state. A MIDlet can place itself in the Paused state by calling notifyPaused(). The application manager can terminate the execution of the MIDlet by calling destroyApp (), at which point the MIDlet is destroyed. A MIDlet can destroy itself by calling notifyDestroyed(). A sample code snippet from our project is given below to illustrate the above.

```
public class acquireMIDlet extends MIDlet implements CommandListener{
   private Command CMD_START=
                                 new Command
                                 ("START",Command.SCREEN,1);
   private Command CMD_EXIT=
                                 new Command("EXIT",Command.EXIT,1);

   public void startApp() {
      instance = this;
      ui_form = new Form ("Acquire Game");
      ui_form.addCommand(CMD_START);
      ui_form.addCommand(CMD_EXIT);
      Display.getDisplay(this).setCurrent(ui_form);
      ui_form.setCommandListener(this) ;
   }

   public acquireMIDlet() { }

   public void pauseApp() { }

   public void destroyApp(boolean unconditional) { }

   public void commandAction(Command c, Displayable s) {
      if (c == CMD_START) {
         try{
            GPobj = new GetProperties();
            Display.getDisplay(this).setCurrent(GPobj);
         }catch(Exception e){
            e.printStackTrace();
         }
         GPobj.start();
      }else if(c == CMD_EXIT){
         destroyApp(false);
         notifyDestroyed();
      }
   }
```

Table 3: Code snippet to show the midlet lifecycle methods

MIDP contains user interface classes in the javax.microedition.lcdui and javax.microedition.lcdui.game packages. The device's display is represented by an instance of the Display class. A reference to the device's display can be attained through the getDisplay() method of Display class. Displayable instances are the ones that will be shown on the display. Using the Display's setCurrent() method, we can set what is to be shown. Displayable has two important subclasses- Screen and Canvas. Various high-level user interface classes like Alert, List, Form, and TextBox are derived from the Screen class. Whereas using the Canvas class, low-level user interfaces can be created.

Various user interface controls called items can be added to the forms. In MIDP 2.0, CustomItem class has been introduced using which one can design new items to be added just like the built-in items.

In our project, all the forms in the application are derived from the Form class and CustomItem class was utilized extensively to represent the game world and the plot areas. The CustomItem class was very useful as we can enable internal navigation.

2.1.3 Persistent Storage

Persistent storage is provided through the use of record stores. It is a small database with pieces of data called records. Record stores are represented by instances of javax.microedition.rms.RecordStore class. Within a MIDlet suite, record store names must be unique. Record Stores can be shared among the MIDlet suites.

To open a record store or to create a new record store, any of the available openRecordStore() methods can be used. To find out the names of all the record stores available in a MIDlet suite, listRecordStores() can be used. To delete the record store, deleteRecordStore() can be used.

In our project, the persistent store is utilized to store the intermediate values of the Configuration stage. In the configuration stage, the plot area selected in the Selection stage is configured to produce different kinds of products. As the plot area consists of nine plots, the configuration details of the plots already configured are stored in the record store and later retrieved and sent to the server, which calculates the production using the details.

2.1.5 Generic Connection Framework

The CLDC defines an extremely useful API for network connections called the generic connection framework. This API is contained in the javax.microedition.io package and based around the Connection interface.

To use this framework, we have to pass a connection string to one of Connector's static methods and get back some Connection implementation. Each type of connection has a different kind of URL as connection string. MIDP 1.0 had support for only one type of connection – Hyper Text Transfer Protocol (HTTP). MIDP 2.0 has support for HTTPS (secure HTTP) and several other types of connections.

In this project, we have used HTTP protocol for the communication between the mobile devices and the centralized server. The Generic Connection framework is used extensively as the server has a lot of functionality as very little work is done on the client's side, hence a lot of information exchange takes place between them.

2.2 Java Servlets

A servlet is a server-side software component, written in Java, that dynamically extends the functionality of a server. Servlets provide a framework for implementing the client-server applications. While working with mobile devices, the load needs to be shifted from the mobile device to the server and this can be accomplished by the usage of servlets.

2.2.1 Servlet Lifecycle

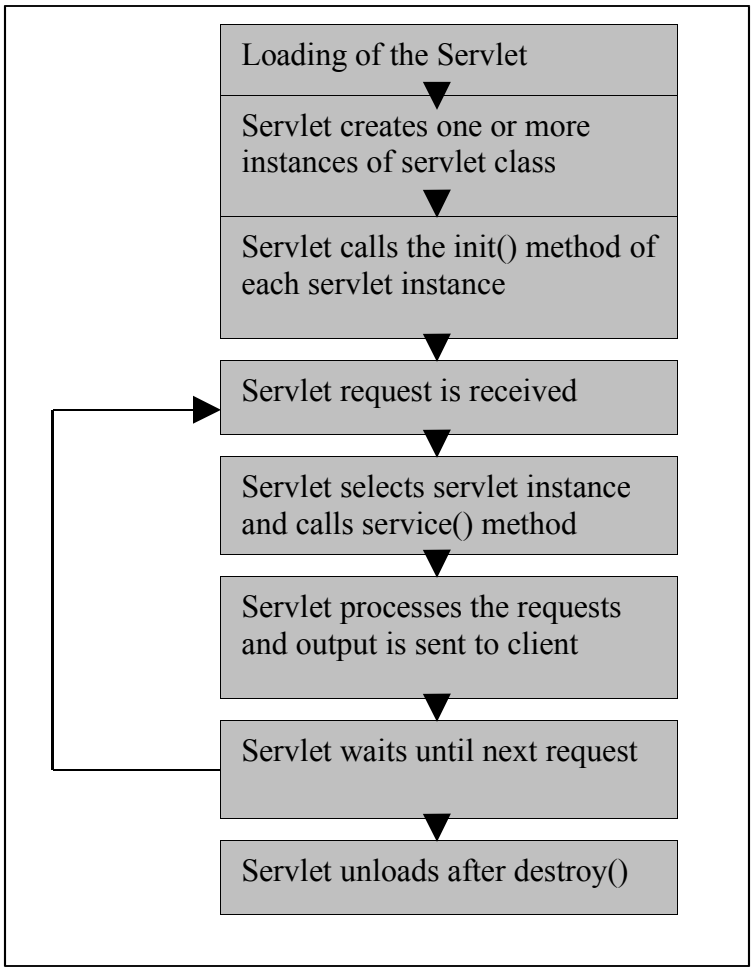


Figure 2: Servlet Lifecycle

2.3 JDBC

Java Database Connectivity (JDBC) is the API for accessing different relational databases in a uniform way. The primary function of the JDBC API is to allow the developer to issue SQL statements from Java and process the result in a consistent, database independent manner. It has a rich set of classes and interfaces that provide object-oriented access to any database.

2.4 Database

This project uses an Oracle Database on the server to store the information needed for processing the client's requests. In this project, most of the work is delegated to the server by the client devices. So, the server is required to store lot of information in the database. We have tested this project using MySQL also.

2.5 Related Work

To understand the above concepts and also to get a start in my project, we have done various short deliverables in the CS 297 course. We have designed and implemented a MIDlet that gets business card information from the user and stores the information in the Record Management System of the device to retrieve the information later. This deliverable helped in the understanding of the Persistent storage concepts.

The second deliverable was to develop an application that enables the transfer of business card information wirelessly between two devices. This was done using the Client Server architecture and JDBC support. In the third deliverable, we have programmed the setting up of a connection between a Pocket PC device and an HTTP Server. Finally, in the fourth deliverable, we implemented the setup of the connection between the HTTP Server and the Oracle database. The last two deliverables were very helpful because my project is based on the three-tier system with

the device as the first tier, the centralized server as the middle tier and the database as the third tier with all the communication taking place through the server alone.

3. Requirement Analysis

This chapter discusses the requirements for this project.

3.1 Purpose

The purpose of this project is to design and develop a multiplayer, distributed economics game that can be used to simulate local auction scenarios for J2ME-enabled wireless PDAs. The main idea for this project is to implement a distributed game that connects to a centralized database.

3.2 Scope

The scope of this project is to implement an economics game- *Acquire*, on wireless devices using J2ME, JDBC, Servlets and a relational database. The following section gives the overview of this game and describes the different phases of the game.

3.2.1 Game Overview

The Acquire game can be played between two or more players. This game borrows ideas from an old computer game called “Mule”. The player is presented with the game world that comprises of a number of plot areas. Each plot area is comprised of nine sub plots. Each plot has three properties

- Mine value: how favorable it is to mine on the plot
- Farm value: how favorable it is to farm on the plot
- Energy value: how favorable it is to produce energy on the plot

Rules of the Game

The rules of the game are as given below

1. Player needs to select a plot area comprising of nine sub plots
2. He needs to configure the plots to produce mines, food, or energy
3. If he satisfies the critical resources limit, he can go for another round of selection or else he has to enter auction.
4. He has to log out of the auction before entering the second round of auction though he can go for a second round of selection and configuration before logging out of the auction.

Game Play

The player can decide what he wishes to produce on each of the nine sub plots of his plot area. These configuration details are sent to the server, which calculates the score after each round using the formulae set by the properties file on the server. If the player's production satisfies the critical limit for all the products then he has a choice of going to the auction or another round of plot area selection. But if his production does not satisfy the critical resources limit, then he has to go the auction. The aim of the game is to maintain the highest score among all the players.

3.3 Game Application Perspective

Table 4: Game Application Perspective

J2ME Device	Game Server	Database
Launch Acquire Game and Register	Receive registration information	Store registration information
Login	Receive login information and check if the login information is correct	
Join already started game or start a new game	Receive the game number of the game that the player has joined and send him the game world associated with it. If a new game is started, create the game world	Update the database to store the game the player is associated with.
Go to Selection Phase: Selecting a plot area	Receive the selection and check to see if the selection is in conflict with others.	Record the selection and update the database
Go to Configuration Phase: configure the plots to produce the goods	Receive the configuration details and calculate production based on the formulae and send him the production amounts and the critical values.	Update the scores of the player.
If critical values are satisfied, go to Selection to select more plots or go to Auction to sell excess goods. If critical values are not satisfied, go to Auction phase and declare the roles If seller set the no of units to sell and selling price. While he is waiting for other players to declare, he can go to the Selection/ Configuration phases of the next round.	Receive the roles declaration and provide relevant information back to the player	Update the database to record the roles of the players.
If buyer, select the no of units to buy. If seller, while he is waiting, he can go to the Selection/Configuration phases of the next round	Receive the number of units to buy and do the trade.	Update the database and change the no of units and the rokda units(money units) of both the seller and buyer

3.4 User types and characteristics

The users of this game are assumed to have different levels of expertise when it comes to using the PDAs. Users need to get used to using the PDA's built-in keyboard, arrow keys and buttons of the PDA. Users of this application should know to install the application and to launch the game application from the program menu.

3.5 Operating Environment

This game can be played on J2ME enabled devices. Though the game application is written to run on CLDC devices, it can run on CDC devices as CLDC devices are a subset of CDC devices. The hardware platform, and the other software components are described in the table below:

Table 5: Operating Environment

Application	Operating Environment
Game Client	J2ME Wireless Toolkit, Windows CE/ME
Game Server	Java 1.4.1 or higher installed on Windows
Database	Oracle 9i / MySQL

3.6 External Interface Requirements

3.6.1 User Interfaces

The user interfaces are quite easily understandable with as little scrolling as possible. Usage of pictures and graphs has been done to display the information to the user to make the understanding easier. The users will take about five minutes to understand the game flow and to get adjusted with the commands on each form. Experienced PDA users can adjusted with the game in two minutes

3.6.2 Hardware interfaces

The hardware interface for the game will be any J2ME-enabled device (Palm, Pocket PC). The PDA device should have a wireless internet card as it is required to interact with the centralized server on another machine.

3.6.3 Software Interfaces

- **Game Client:** The client should have the MIDP device JVM for J2ME (ex: IBM's J9) installed on it. The client should also have an internet wireless card installed on it.
- **Game Server:** The operating system must have the JVM version 1.4.1 or greater installed.
- **Database:** The database can be any relational database. The ones we have tested our program with are Oracle database and MySQL.

3.6.4 Communication Interfaces

There are two communication interfaces involved in my project. The first is the communication interface that is between the game client and the game server. This communication is achieved through the generic connection framework of the CLDC API. The second communication interface is between the game server and the database. This is achieved through the Java Database Connectivity (JDBC).

In our project, the two players do not interact directly and the communication is done always through the centralized server.

3.7 Functional Requirements

The functional requirements for the game client, game server and the centralized database are explained below:

3.7.1 Functional requirements for game

Acquire game client should be able to perform the following functions:

Register: The new user should be able to register with the server.

Input: The player's username and password

Effect: The player will be informed as to whether the registration was successful

Login: If the user is already registered, he can enter his username/password to enter the game system.

Input: The player's username/password

Effect: Whether the player successfully logged in. If he is successful, he will be presented with the Games List i.e., the list of games that are already started in the system.

Join Game: This function allows the player to join the game that was already started by another player and with players already in it.

Input: The input for this function will be the game number of the game that the player wants to join

Output: The game world displayed to him graphically.

Start New Game: This function allows the player to start a new game.

Input: The input for this function will be the next game number

Output: The server creates the new game world and is displayed graphically to the player.

Select Plot Area: The player can go across the game world to select the plot area to see the properties of the plots and then select a plot.

Input: The input for this function will be the plot area's XY coordinates.

Output: The server checks if the selection is in conflict with another player's selection. If in conflict, returns a plot area's coordinates with nearest property values. If there is no conflict, assigns the plot to the player.

Zoom Plot: The user should be able to zoom the plot area to see the plot area's properties without selecting the plot area.

Input: The plot area's XY coordinates

Output: The zoomed plot and the property values of the sub plots in the plot area will be displayed to the user.

Configure Plot: This function allows the user to configure the plot to produce mines, food, or energy.

Input: The plot area's XY coordinates, production type and the time left for production (which will be different for each plot) on that plot.

Output: The configuration details will be stored in the device's Record Management System and will be passed onto the server after all the plots in the selected plot area have been configured. The server returns the production units of each of the products and the user will be informed in the form of a bar graph. The surplus will be displayed in blue color and the shortage will be displayed in the red color.

Configure All Plots: This function allows the user to configure all the plots at once as configuring each plot one by one takes time.

Input: The sub plots' XY coordinates, the time left for production (which will be same for all the plots) and what type of production is being done on each of the plots.

Output: The output is similar to the Configure Plot function.

Go to Auction: This function allows the player go to the Auction phase.

Input: nothing

Effect: The form wherein the user needs to declare the roles for each type of products (mines, food and energy) i.e., whether he is a seller, buyer or none. If declared to be a seller, he can set the number of units he is willing to sell and the selling price for each unit.

Transfer Units: This function allows the player who logged in more than one game to transfer the production units to this game.

Input: The number of units to transfer from one game to another.

Effect: The production units are transferred from one game to another.

Submit Declaration: This function allows the player to submit his declaration

Input: The player's login, game he is logged into and the role declarations will be sent to the server.

Output: The server records the declarations and sends relevant information back to the player.

Declaration Check: This function allows the user to wait for other players in the game to declare.

Input: The player's login, game he is logged into and his Wait and Poll status will be sent to the server.

Effect: The server consults the database and gives an update.

Go to Selection/Configuration From Declaration Step: This function allows the user to go to the Selection phase of the next round if the player is allowed to do so (the player will not be allowed if the he does not have the critical units of each of the product types). Go to Selection option will be available if the user has not finished the selection of the next phase. Go to Configuration option will be available if the user has finished the selection phase of the next round and his pending state is configuration phase.

Submit Buy Units: This function allows the player to select the number of units to buy from the seller.

Input: The number of units to buy from the seller and seller's login.

Effect: The server does the trade between the seller and the buyer. If there are multiple buyers for the same seller, the first buyer's request will be satisfied first and if there are more units to sell with the seller, the second buyer will be satisfied and so on. The number of units that have been traded will be displayed to both the buyer and seller.

Get Score: This function lets the player ask the server for his score

Input: The player's login and the game number

Output: The player's score and his standing within his game and also in all the games will be displayed to the client.

Buy From Store: This function allows the player to buy from the store.

Input: The player's login, game number, no of units to buy from the store

Output: The number of units traded with the store.

Sell to the store: This function allows the player to sell to the store if there are no other players who are willing to buy. As the store is not implemented as another type of player, the store's default buy prices and the maximum number of units it is willing to buy from the players will be set by the server in the java property file on the server.

Input: The player's login, game number, no of units to sell to the store and the buy price.

Output: The number of units traded with the store.

Log Out: The player is allowed to log out the system. He can again login the game he was in later on to continue from the point where he left.

Input: The player's login and the game number

Output: A status value to indicate whether the player logged out successfully.

3.7.2 Functional Requirements for the game server

The Acquire game server should be able to handle the following functions:

Create Game: This function occurs when the user requests to start a new game

Input: Start new game request and game number.

Output: The plots will be created with different and random property values for the Mining Value, Farming Value and Energy Value and each value with the range of [1,4].

Update On Selection: This function allows the server to update relevant tables of the database to show that the selection has been made.

Input: The X and Y coordinates of the plot area, player login name and game number.

Output: The updated plot's selection table

Update On Configuration: The expenditure and the net production is calculated and the game score table is updated**.**

Input: The net production of each of the product types.

Output: The updated game score table

Update On Trade: If the buyer has decided to sell from a particular seller, then the trade takes place.

Input: The number of units to buy, buyer login, seller login and the game number

Output: The updated game score table and the updated game trade table.

Update On Transfer of units: If the player likes to transfer the units from the other games he is involved in to this game.

Input: The number of units to transfer from other games to this game

Output: The updated game production tables in both the games.

Update Scores: The scores of the players will be updated whenever there has been a trade involving the player.

Input: The game number and player login

Output: The output of the function will be the updated scores list.

3.7.3 Functional Requirements for the database

The centralized database should perform the following functions:

Store Records: This function occurs when the server requests for storing new records

Input: The server's request

Output: The output of the function will be the storing of the records.

Retrieve Records: This function occurs when the server requests to retrieve the records

Input: The server's request

Output: The output for this function will be the retrieving of the specified records

Update Table: This function occurs when the server requests to update the existing table.

Input: The server's request.

Output: The output for this function will be the updated records in the database.

4. Design and Implementation

This chapter discusses the design and implementation phases of this project.

4.1 System Architecture

The main objective of the project is to implement a game with which some economics scenarios can be simulated. Though our aim is not to show some specific complex issues of economics, we want to show that it is possible to simulate the economic scenarios in the form of mobile games. Our project can be taken as a framework for the development of economics-style games. The game is made more playable with the usage of icons, graphs, and keeping of scores. The game we have implemented is a distributed, multiplayer game. The division of components has been done based upon the functional requirements of the game. The game is developed as a three-tier system with the game client/PDA as the first tier, the game server as the second tier and the database as the third tier.

Usage of Three-Tier Distribution Architecture Pattern

In our system, the front-end clients are the J2ME-enabled devices. The middle tier is the Game Server. The third tier is the database.

The Game Server is the essential centralized logical component required in the game application. It is the key component that is responsible for many tasks such as creation of the game world, joining the player to the existing game, conducting transactions between the players, retrieving the information from the database, updating the information in the database. The interaction between the game clients and the server is through HTTP through the use of generic connection framework. The interaction between the server and the database is achieved through the JDBC.

The architecture of our game application is given below:

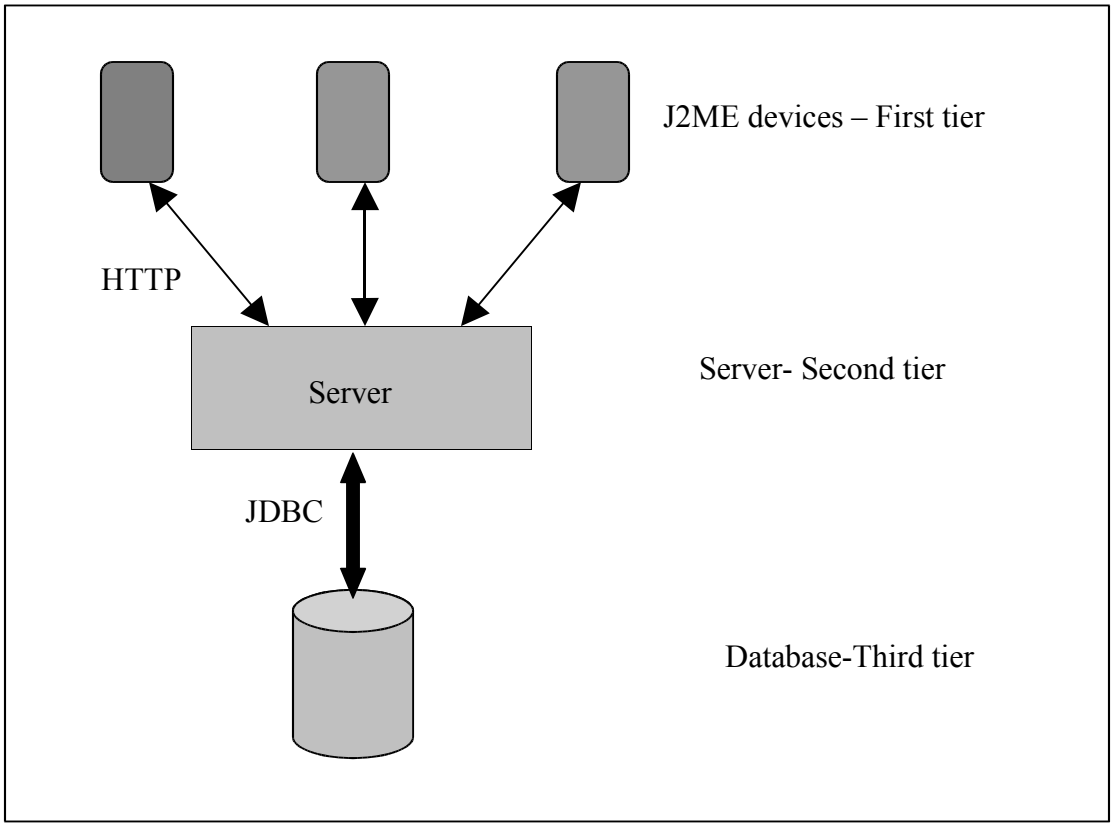


Figure 3: System Architecture

4.2 System Design

The design of the game application is discussed in this section.

4.2.1 Server

The game server is the central component of this system that does most of the work. In any mobile application, most of the functionality should be shifted to the server side as computing power of the hand held devices is generally weak. The creation of the game world, the calculation of scores etc. are the various functions done by the server as illustrated by the figure below:

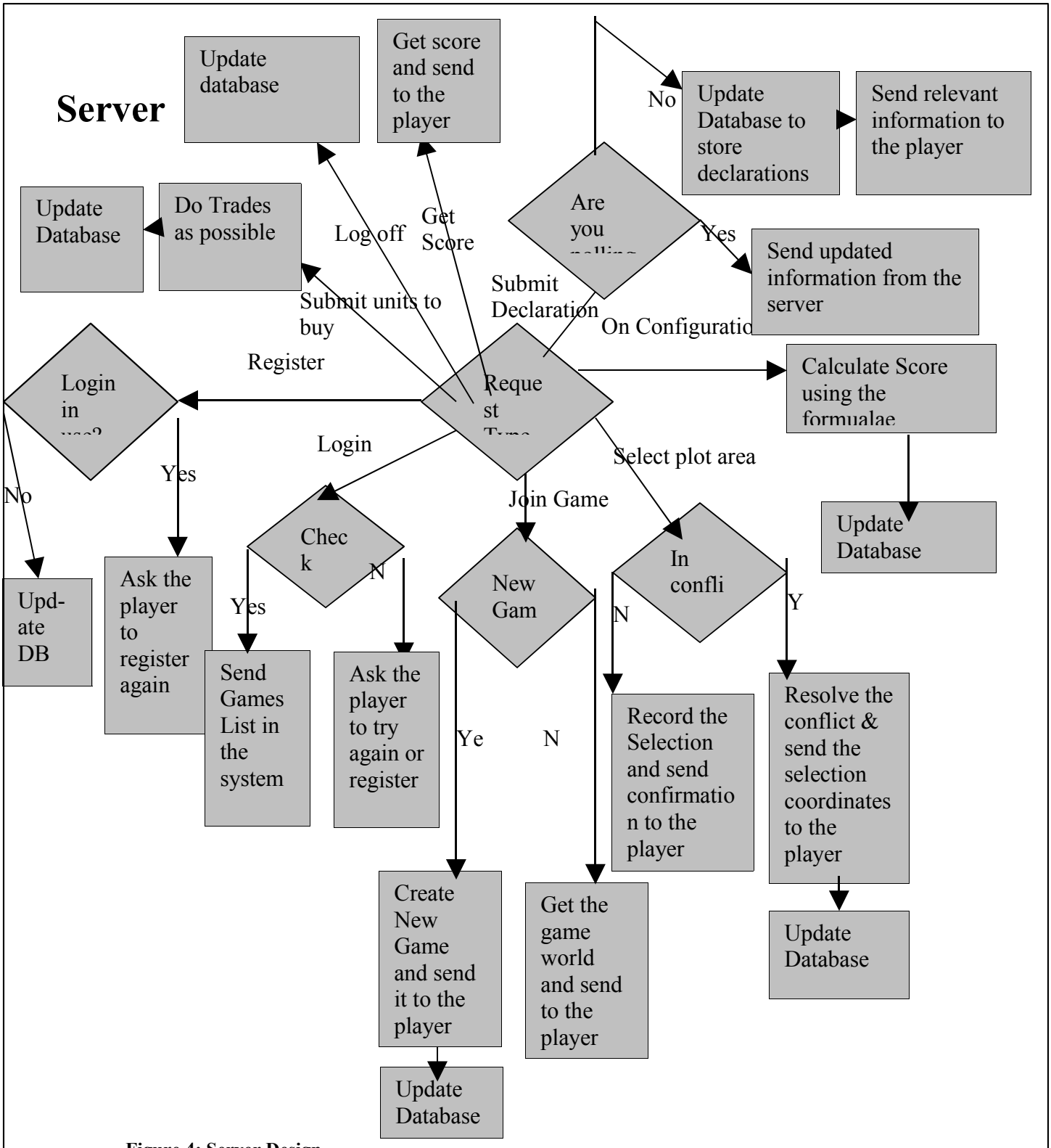


Figure 4: Server Design

4.2.1.1 Register

If the player is playing for the first time, he needs to register with the system. The player's registration information is sent to the server and the server checks to see if his login/password is unique in the system. If so, he is sent confirmation that he is registered and he can continue to the game by logging in using his newly created login/password. If not, the player will be asked to try registering again.

4.2.1.2 Login

If the player is already registered with the system, he can login using his login/password. The server validates the player and then checks to see if the player is already playing or not. If he is registered and not playing any other game, the server retrieves the list of all the games in the system and also the number of players in each of the games and sends this information back to the client. Based upon the number of players already in the game, an icon is allocated to the player and is sent back to the player.

4.2.1.3 Create Game world

Each game in the system has a different game world. All the players who have joined the same game will see the same game world. The game world is generated the following way: As this is a handheld application, the screen area is small, so the number of rows and columns of the plot areas is limited to 15, since it requires lot of scrolling of the screen. Each plot has three properties: Mine value, Farm value and Energy value. Each value is a random number between one and four. A value of four signifies that the plot is good for that type of production. After the creation of the game world, it is stored in the database.

If a player wishes to join already created game, his login name is associated with the appropriate game world and the game world is sent to him. If a player wishes to start a new game, a new game world is created and is sent to him.

4.2.1.4 Update on Selection

When the player sends the coordinates of the plot area that he has selected, the server checks to see if there is any conflict in the selection. If not, the server sends him the confirmation. Otherwise, the server gets another plot area with minimum difference in the average property values and allocates that plot area. The server then sends this plot area's coordinates to the player. In both the cases, the database is updated to reflect the selection.

4.2.1.5 Update on Configuration

When the player finishes the configuration of the plot area, the configuration information (the type of production on each of the plots, and the time for which he has done the production on each of the plots) is sent to the server. Then the server calculates the net production. The production and expenditure formulae are set in the properties file on the server. The expenditure can be in mine units (like machinery expenses), in food units (like food expenses), in energy units (the power expenses), in rokda units (which are the money units, like worker d land expenses). So expenditure units are subtracted from the production done and net production is calculaed. Some of the formulae can be formulated as follows:

Table 6:Sample Formulae

```
Machinery_Expenses  =  (Mine_Value*10) +(Farm_Value * 2) + (Energy_Value *2)
Production  =  (Type_Of_Product * Appropriate_Property_Value)
Score  =  (Mine_Units * 10) + (Food_Units * 2) + (Energy_Units * 3) +
          (No_Of_Successful_Trades * 3)
```

Changing the formulae results in different economic scenarios. After the calculation of the net production, the database is updated to reflect the change in the score.

4.2.1.6 Update on Role Declaration

If the player decides to go the auction, he will be presented with the role declaration form. He has to decide what he decides to be for each type of auction. Like he can be a buyer for the mine auction, seller for the food auction and none for the energy auction.

If he decides to be a seller, he also needs to enter number of units he wishes to sell and also set the selling price. When he submits the roles, the server updates its database and sends the player the relevant information. This relevant information might be things like if the player is a buyer, the server sends the seller logins and the number of units available to buy and vice versa. The player can also decide to sell to the store. If he wishes to sell to the store and the store still has not crossed the buy limit, the trade will be done immediately and the database is updated. The player can also poll for any change in the number of buyers and sellers in the game, then no database update is done but information is retrieved and sent to the player.

4.2.1.7 Transfer Units

In this game, the player can login in more than one game. Then, he has the option of transferring the production units from the other game to this game if he wishes to improve the score in this game. By this function, the concept of local auctions is realized in this project. When he chooses to transfer the production units, the production tables of both the games are adjusted to reflect the transfer.

4.2.1.8 Do Trade

In this game, the buyer initiates the trade by setting the number of units to buy from each seller and sends the information to the server. The server then does the trade immediately. The trade is done among the players based on timestamps of the requests that means the player who

responds faster has more probability of a successful trade. The database is updated upon the completion of the trade. The only case where a seller can initiate a trade is when he wishes to sell to the store. If the player wishes to stop trading with anybody, he can select the finish auction command. He is required to finish the previous auction before he starts the next round of auction.

4.2.1.9 Get Score

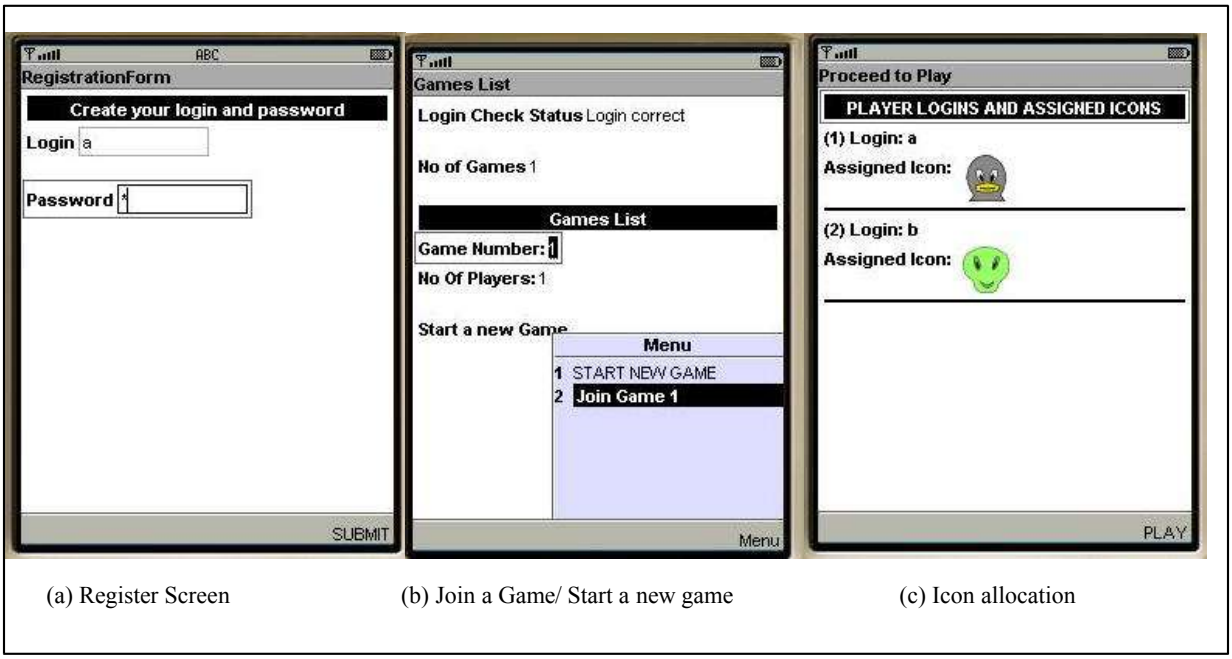
The score is calculated based upon the player's request. The formula for score calculation is also stored in the properties file. The sample score formula could be the total value of the products the player has in his possession. The current positions of the players are calculated within the game and also in all the games in the system.

4.2.1.10 Log off

The player can log out of the game to re-login later. When the player logs off, his score is retained in the system so that when the player logs in the same game again, he can start where he left off. This log off option is there only at the end of the auction stage as he can start from the selection stage when he logins back later.

4.2.2 "Acquire" Game

The invocation of this game starts with a login screen. If the player has already registered with the server, he can login using his username/password or else he can register himself with the server. After player has logged in, he will be presented with the Games List. The games list has different games being played in the system and the number of players in each game will be displayed. The player can slide down to the game he wants to join or else he can start a new game. For each new game, the server creates whole new world of plots. After he has joined the game, he will be assigned an icon.



(a) Register Screen (b) Join a Game/ Start a new game (c) Icon allocation

Figure 5: Register, Login and Icon Allocation Screens

Each player who logs in will have an initial number of mine units, food units, energy units, rokda units (or money units) to start the game with. After this, the game involves the following phases:

- Selection Phase
- Configuration & Production Phase
- Auction Phase

The Selection Phase

The central server sends the game world of the game to the player. The game world is displayed to the player as square shaped, colored plot areas. If any of the plot areas are already selected by other players in the game, they will be marked with the corresponding players' assigned icons. The icons were created using Inkscape's Scalable Vector Graphics Editor [OSVGE]. The colors of the plot areas are set according to the properties of the nine plots inside each plot area. The colors are set as the weighted sum of the three colors – red, green and blue signifying the three

properties – Mine Value, Farming Value and Energy Value respectively. These three values have a value range of [1,4].

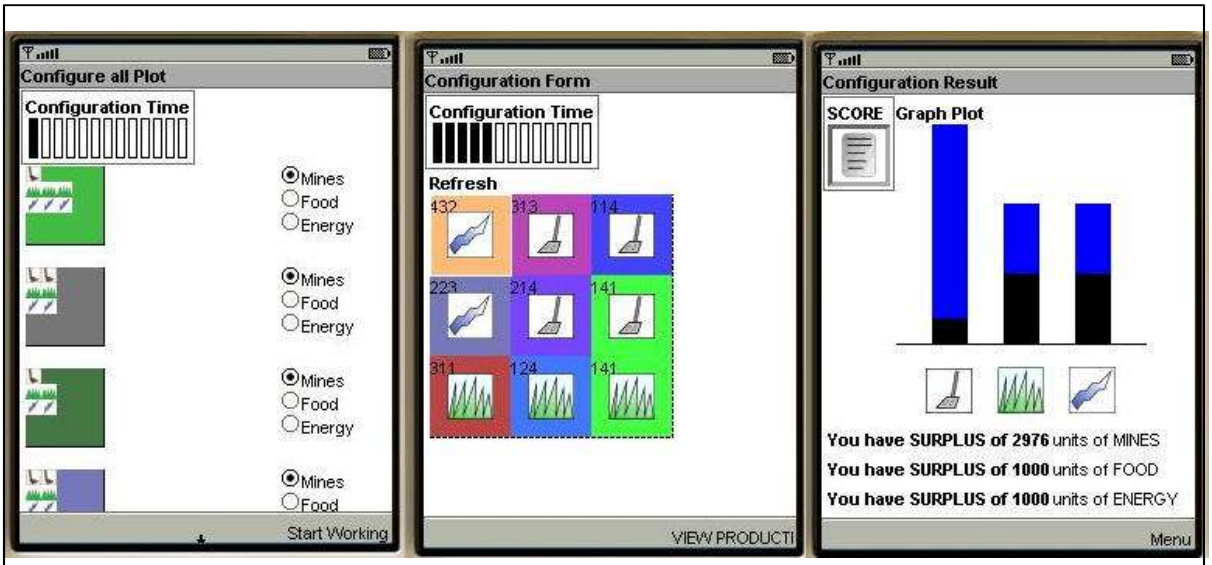
(a) Make Selection Screen (b) Zoom of the plot area (c) Selection Result Screen

Figure 6:Selection, Zoomed plot area and Selection Result Screens

There are five plot areas in a row and there are five rows. Each plot area can be further zoomed in to display the nine plots in each plot area. The player can go across the plot areas, zoom in and select the plot area he likes. The selection step is timed i.e., there is a particular amount of time (set by the server) within which he has to make a decision. The time gauge on the top of the screen shows the time elapsed. After making the selection decision, he sends his selection to the server. The server checks to see if there is any conflict with other players' selections. If there is a conflict, then server assigns the player a plot area that has nearest property values as the plot the player originally selected. The server then sends back the result of the selection phase.

The Configuration and Production Phases

Once the selection step has been completed, the user proceeds to the Configuration Phase. This is the phase in which the user configures the sub plots of the plot area that he selected. He can configure each sub plot differently. He can decide this based on the properties of the plot or the shortage of products. The configuration phase is also timed. There are two ways in which the plots can be configured- one plot after another plot or all the plots together. In the one-after-another way, the user can select each subplot in his plot area to configure and then go back to the plot area and configure other plots similarly. The time remaining after he has configured a particular plot is the production time for that particular plot. For each subplot, the type of production as configured by the player and the time left after he configured will be recorded and stored in the record management store of the device.



(a) Configure Screen (b) Configuration Done Screen (c) Result of production

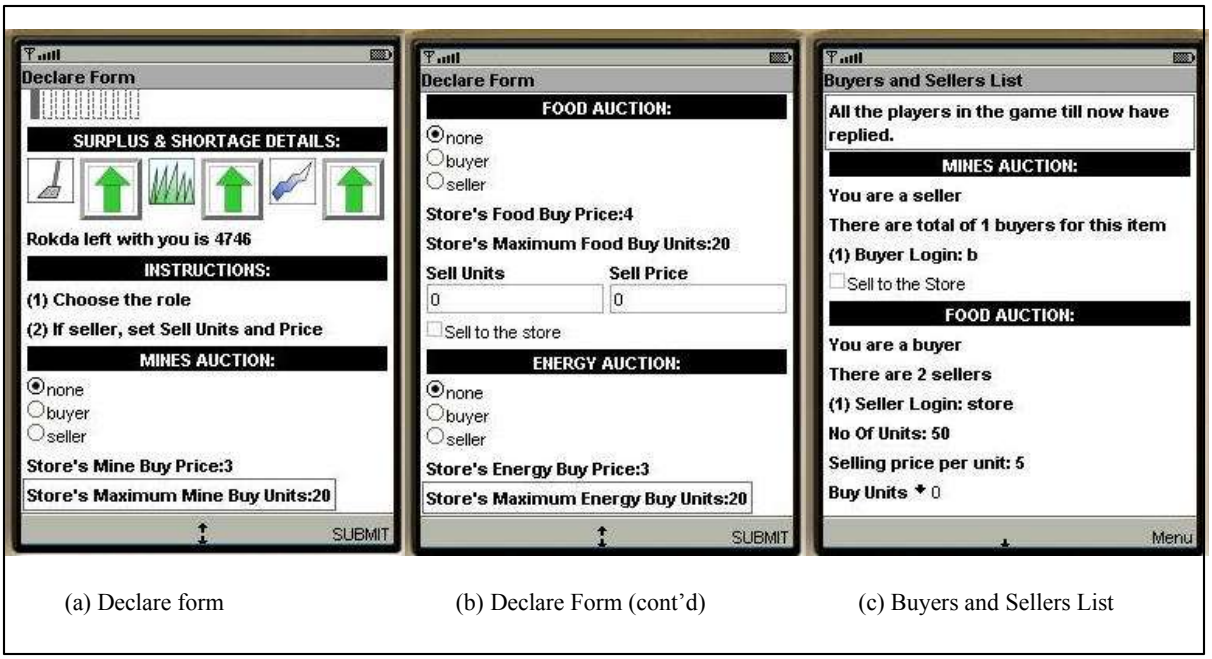
Figure 7: Configuration Screen and Production Result Screens

In the all-at-once way, he can configure all the sub plots at once in a single form. In this way, as all the plots are configured at once, the production time for all the sub plots will be the same and it will be the time left after configuring divided by nine as there are nine plots in each plot area.

The configuration details that are recorded will be sent to the server. The server calculates the production using the configuration details and the production formulae in its properties file. The server updates the score and sends back the production information and the critical units of each of the products to the client. The production information is displayed as a bar graph, with blue showing the surplus if he has any. If the client has more units than the critical units for each product type, he can go for the next round i.e., to the Selection phase to select another plot area and configure it. He can also go to Auction Phase to trade some of his excess stuff. But if he has less than critical units for any product type, he must go to the auction phase to gather the critical resources first.

The Auction Phase

If the user selects to go to the Auction Phase, he will be led to a Declaration Form where the user needs to declare what role he would play for each type of product auction i.e., Mine Auction, Food Auction and Energy Auction. The roles he can choose from are buyer, seller or none. He will also be provided with the option of selling to the store. The store's maximum buy price and also the maximum number of units the store can buy in the game will be displayed to him. But this buy price usually would be very low. The user will not profit much if he chooses to sell to the store. But if he needs money/rokda units to proceed to the next round and there are no other buyers for that type or no buyer chooses to buy from him, then it is the only option. If the player decides to be the seller, then he has to set the selling price and the number of units he wishes to sell. Then he can submit his declaration and wait for other players to declare or go to the Selection phase.



(a) Declare form (b) Declare Form (cont'd) (c) Buyers and Sellers List

Figure 8: Role Declaration and Buyers & Sellers List Screens

After the role declaration submission, the user gets the relevant information from the server i.e., if the player is the seller, he gets the information of whether any players have declared themselves as buyers and if the player is the buyer, he gets the information of the sellers for that type of product and gets the information of how many units they are selling and at what price. Then the player can choose the number of units he wants to buy from each seller from the drop-down list displayed for each seller. When the player submits this decision, the server will do the trade immediately and the players' scores will be updated.

4.3 Local Auctions

4.3.1 Concept of Local Auctions

The local auctions are the type of markets in which only specific groups of people are allowed to trade with each other. In other words, every player is not allowed to interact with every other player in the system directly. This concept is illustrated by the figure below:

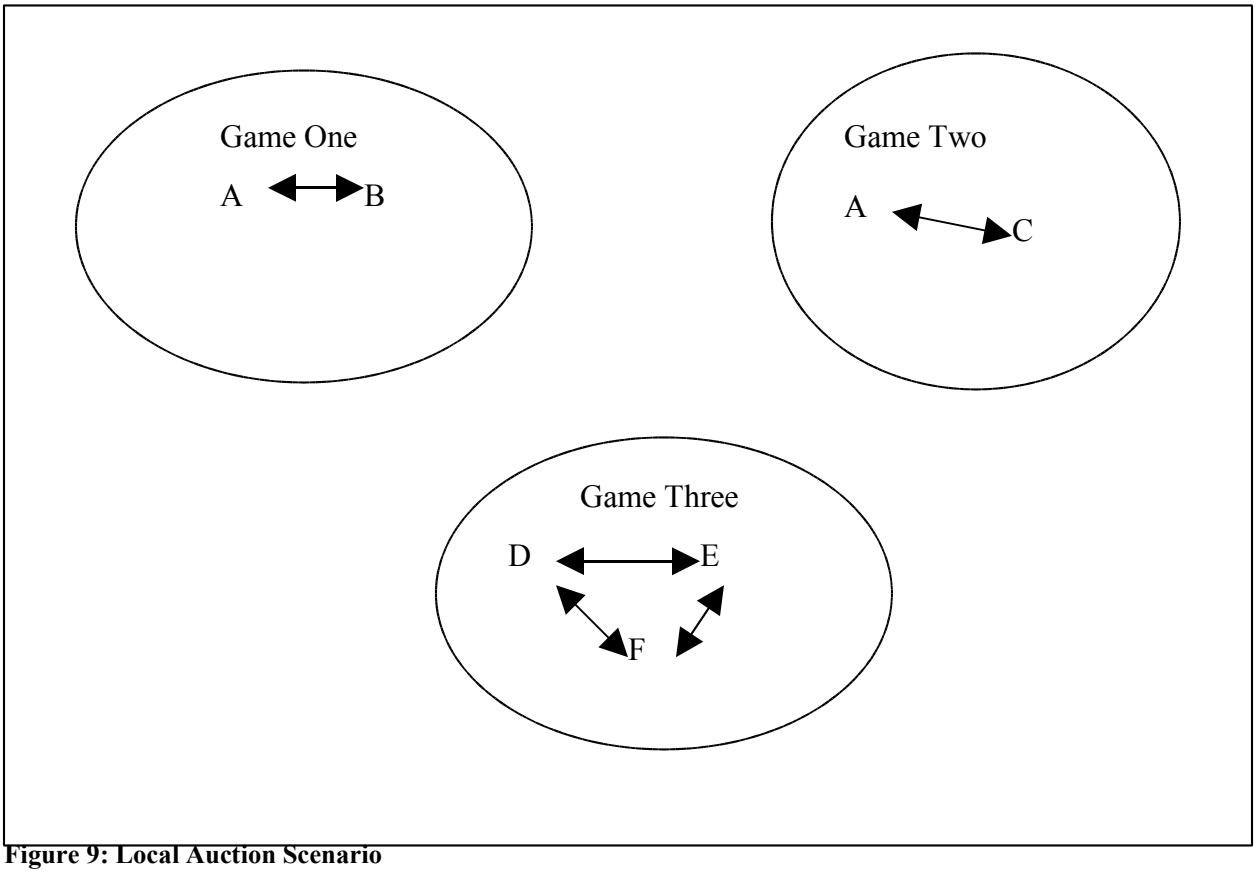


Figure 9: Local Auction Scenario

In the above figure, there are three games being played in the system. Player A and Player B are involved in the Game One, Player A and Player C are involved in the Game Two and Player D, Player E and Player F are involved in the Game Three respectively. The arrow marks indicate the possibility of trades between them.

The Player A can interact with both the Player B and Player C. But the Players B and C cannot interact with each other.

4.3.2 Realization Of Local Auctions

In our project, any player is allowed to login in more than one game. Then, at the time of auction, he will have the option of transferring his production units in the other games to this game. By this way he can improve the score in the present game. In this way, the different games can be visualized as different auctions and by allowing the transfer of units from one auction to another; the concept of local auctions is realized.

4.4 Implementation

4.4.1 Acquire Game Implementation

The Acquire game is implemented using the following design patterns.

4.4.1.1 State pattern

The State pattern is an extension of the Strategy design pattern. This pattern is used when the behavior needs to be changed based on the state the system is in.

Usage in this project:

The state pattern is put to use in this project in the following way. This game application is essentially a state-based game. The player can be in any of the possible states like Selection, Configuration or Auction, the State pattern is used to decide the actions that are possible for the player from that particular state. Like for example, if the player is in the Auction state, he will be presented with appropriate actions for that state. And also, if at all the player has gone back to Selection State while he is waiting for more players to declare as buyers/sellers, his state is changed to that Selection State and his current state is marked as pending so that he can return back to the state later. The states in the game are illustrated in the figure below:

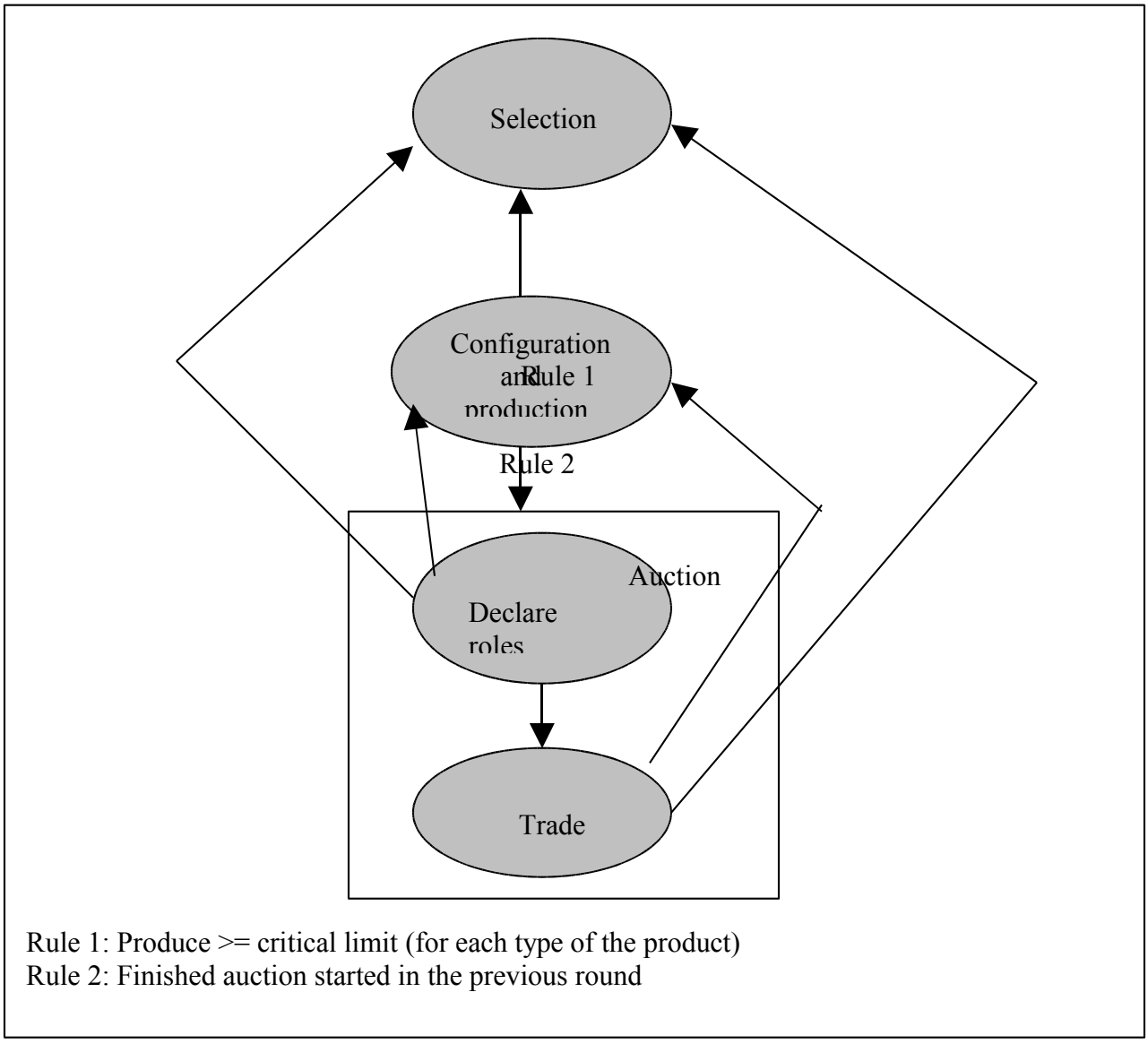


Figure 10: State Diagram

As shown above, the player after production can go to Selection state (if his production is more than the set critical limit) or to the auction state. If the player wants to wait for more players to declare as buyers or sellers before buying units, he can go to the selection state in which case Declare state will be marked as the pending state. And then if from the selection state he reached

configuration state and wishes to check on the buyers/sellers who declared later on, the configuration state will be marked as pending and declare state as the current state. The other rule listed in the diagram is this: the player cannot enter the auction before finishing off the auction step of the previous round.

4.4.1.2 Notification pattern

The Notification pattern used in the server is passive notification. Passive notification is simplest form of notification.

Usage in the project:

Every event that is taking place at the server is not relayed to all the players in the game in this form of notification. For example, if Player A has finished his selection and Player B needs to make the selection, he gets the plot with the selection by Player A marked. If in the mean time a Player C also needs to make the selection before even Player B makes his selection, the Player C gets the plot with only Player A's selection marked. Player B and Player C make selections and send to the server and if at all there is a conflict among them, server resolves it and assigns a different plot to one of them with the closest property values to the original plot selected. Player A does not know of Player B and Player C's selections as he can continue with his configuration and production stages until Player A reaches the selection stage of the next round.

There is a constant interaction between the player's device and the server as all the game information is stored at the server itself. Only while the configuration is being done, the information is stored in the device's record management system. Other than that no other information is stored in RMS because device's RMS is quite small. A class that involves interaction with the server implements the HttpConnectionHelper.Callback interface in addition to the Runnable and CommandListener interfaces. HttpConnectionHelper is a helper class to handle

any redirections in the web address. Using prepareRequest() method of the HttpConnectionHelper, the request can be modified before it is actually made.

4.4.1.3 Feedback pattern

The Feedback Pattern is used to keep the user updated of the server process. A separate thread is used to make the HTTP connection to keep the user interface active and responsive.

Usage in this project:

The reason to use this pattern is to prevent the player from repeatedly pressing the 'submit' button on the PDA. If there is no change in the user interface after the player clicks the 'Submit' button, the player is going to have no idea as to whether the server is invoked. So, appropriate messages are displayed on the screen for the player to have an idea as to what the current state is like "Obtaining the Http Connection object" and " Connecting to Server". For example, Registration Status Form displays the status to the user after the user submits the Submit button on the registration form.

4.4.1.4 Model View Controller Design pattern

In this pattern, the model component is responsible for the storage of application data. The view component and the controller component do not need to know about the model. The controller component is responsible for handling user input and user commands. The view component is the user interface components like forms.

Usage in this project:

The controller component is played by the AcquireMIDlet in our project. The view components are the various forms like RegistrationForm, LoginForm, SelectionForm, ConfigurationForm, DeclareBuyerSellerForm, BuyersSellersListForm etc. The model components are classes that hold the data like LoginInfo, Selection, ConfigurationDetails, and GameLand etc.

4.4.2 Server Implementation

The server consists of these servlets:

Registration: The Registration servlet gets login and password from the player. The servlet checks to see if the login and password are unique in the system using checkLoginPassword() and if the login/password pair validates, updatePlayersTable() updates the database.

GamesList: The GamesList servlet gets the login/password as input. The servlet checks to see if the player is already playing using checkLoginPassword() and this function returns status of the player. If the player is not playing, getGamesList() gets the games list (the number of games, number of players in each game) and will be sent to the player.

Game: The Game servlet gets the login, game number as the input. If the player starts a new game, the received game number is 0. The servlet checks to see if the player started the new game or he joined already created game. If it is a new game, joinNewGame() initiates the store using a Java properties file. The assignIcons() assigns a unique icon within the game to the player. The player receives the game number and icon he is associated with.

LandScape: The LandScape servlet gets the game number, login from the player. The servlet checks to see if the game is already started. If not, createNewGame() creates a new game world taking number of rows and number of columns as input. The new game world is sent a string back to the player.

PlotSelection: The PlotSelection servlet gets the game number, login, plot selection coordinates from the player. If the Selection timer is up before the player selects a plot, the selection coordinates are set to invalid values and sent to the server. The server checks to see if the selected coordinates are valid and then isInConflict() checks if the player's selection is in conflict. If in conflict resolveConflict() resolves the conflict. It then sends back the selection result.

Production: The Production servlet gets the game number, login, plot selection coordinates, the types of production as an array, and the production time array. The calculateNetProduction() calculates the net production, updates the database and sends back the production details to the player. The parsing of the formulae in the properties file is done using JEP (Java Mathematical Expression Parser) from Singular Systems available for download from [JEP].

RoleDeclaration: The RoleDeclaration servlet gets the game number, login, auction role for each product type, if seller for any product, the selling prices and the sell units and whether ready to sell to the store. If ready to sell to the store, doTradeAsSeller() does the trade immediately and updates the database. The player will be sent back the information like if he was a seller and wished to sell to the store, how many units were traded will be sent back to him. If he was a buyer, the number of sellers and their sell units and selling prices will be sent to him**.**

Trade: The Trade servlet gets the game number, login, units to buy, seller logins, whether he wishes to sell to the store, and whether he wishes to finish the auction from the player. The doTradeAsBuyer() does the buyer-initiated trade and database is updated. If a seller wishes to sell to the store, that trade done using doTradeAsSeller(). And the number of units traded as the result of auction is sent to the player. If the player wishes to finish the auction, he will be removed from the Sellers table in the database so that no other players will be able to see this player as a seller from then on.

Score: This Score servlet receives the game number, login as the input from the player. The number of mine units, farm units, energy units, and rokda units are updated regularly whenever any trades are done. The calculateScore() uses these values and the formula from the properties file to calculate the scores of the players as a single unit. Then positions are calculated and returned to the player.

5. Experiments and Results

The purpose of developing this game application was to be able to test certain scenarios. Though not to delve into Economics concepts completely, the idea is to show that certain kinds of tests can be done which can be useful in understanding how the local markets work.

In this game application, the different games in the system can be thought of as different local markets. And each local market has a central store that can buy a certain fixed number of items of each type in the market but it usually buys the products at the rock bottom price. So, players will be usually interested in selling to the store only when no other players are willing to buy from him. But it might be the only option, when he requires rokda/money to buy other goods with the money he gets from selling his surplus goods.

The store also sells goods at fixed prices that are usually higher than what other players would sell for. So, players would try to buy from other sellers first. But sometimes, buying from the store might be the only choice.

The system is designed so that all the production and expenses formulae are placed in the Java properties file at the server where the production is calculated based upon the user configuration details (like what the user decided to produce on each of the plots he selected). So, for each of the test cases, the appropriate formulae in the properties file have been changed to reflect that particular scenario.

We have designed some scenarios to see how our system would work in each of the scenarios. Many other scenarios also are possible in addition to the ones below.

5.1 Economic test cases

The test cases that we have designed are:

1. Realization of local auctions scenario.
2. Making two of the products easy to produce and the other product tougher to produce. Also the product tougher to produce is very costly at the store.
3. If you decide to produce one of the products the player profits more.
4. Using real world cases like for example, worker expenses are more for mining than for farming, land expenses are more for mining land than for farming etc.
5. Changing the number of players in all the three test cases above.

5.1.1. Realization of local auctions scenario

Description

This test case is intended to show how the local auctions scenario is realized.

Results

(a)With three players

Player A starts Game 1 joined by Player B later. Player C starts another game. The Player A decides to login into the game started by Player B too.

The Production values after a round in the Game 1 are given below:

Table 7: Production Values of Game 1

	Mines	Food	Energy	Rokda	Score
A	798	868	638	1860	11874
B	218	474	1106	1886	9296

The Production values after a round in the Game 2 are given below:

Table 8: Production Values in Game Two

	Mines	Food	Energy	Rokda	Score
C	864	304	640	1876	9972
A	348	564	354	1868	7280

As seen from the above tables, the Player A is leading in the Game 1 but not in the Game 2, so he wishes to transfer his production units from Game 1 to Game 2 in order to improve his score in the Game Two. He transfers 100 units of each product from Game 1 to Game 2. As a result of this, his score improves from 7280 to 9886 and becomes comparable to that of Player C.

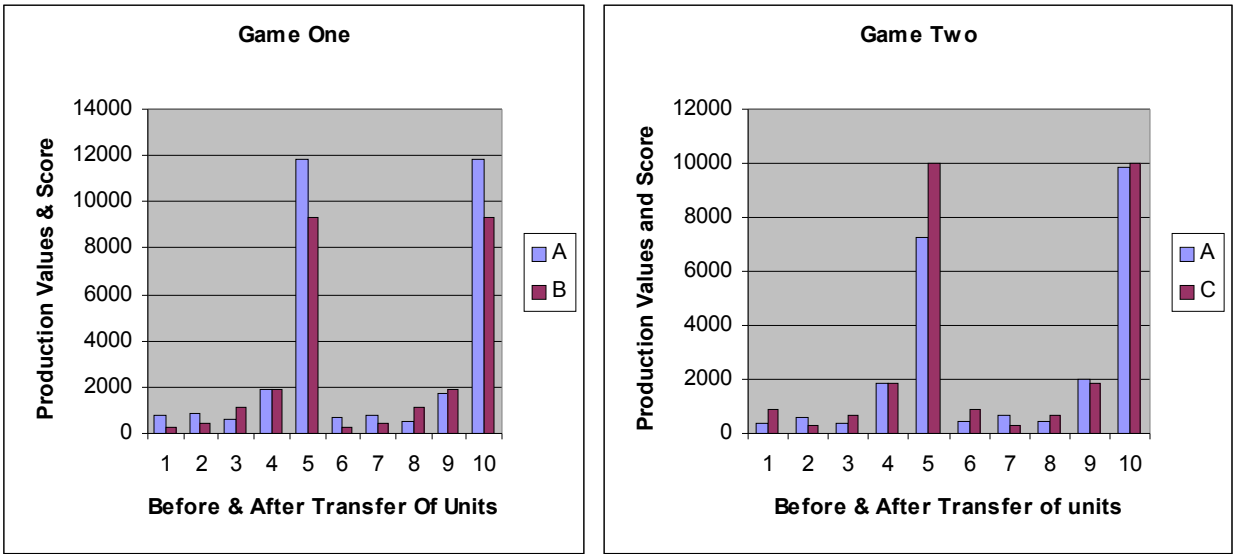


Figure 11: Local Market Scenario Realization

Conclusion

By the above results, it can be understood that the Player A could transfer the surplus goods from one local auction to another as a result of which he could improve the score in the second game.

5.1.2 Test Case One

Description

Making two products easy to produce and the other product tougher to produce. Also the product tougher to produce is sold at a very high price at the store.

The product that is tougher to produce is Mines. To understand the way the mines are made tough to produce is by changing some formulae in the properties file. The way we have done is though the production formula remains constant for all the types of products, the mine expenditure formula (which is represented by MachineryExpenditure which is calculated in mine units) is modified to increase its value and is subtracted from the mine production to get the net mine production. The other two products, Food and Energy are relatively easier to produce.

a) With Two Players

Results

The results of this test case are as given below:

Table 9: Test Case One-2 players: Score Values

Round	Player A	Player B
One	3115	2425
Two	2960	2430
Three	1900	1639

As we can see from the above table, Player A leads through all the three rounds

Average Mine Value in the three rounds is given below:

Table 10: Test Case One-2 players: Average Mine Values

Round	Player A	Player B
One	3	2
Two	3	2
Three	3	3

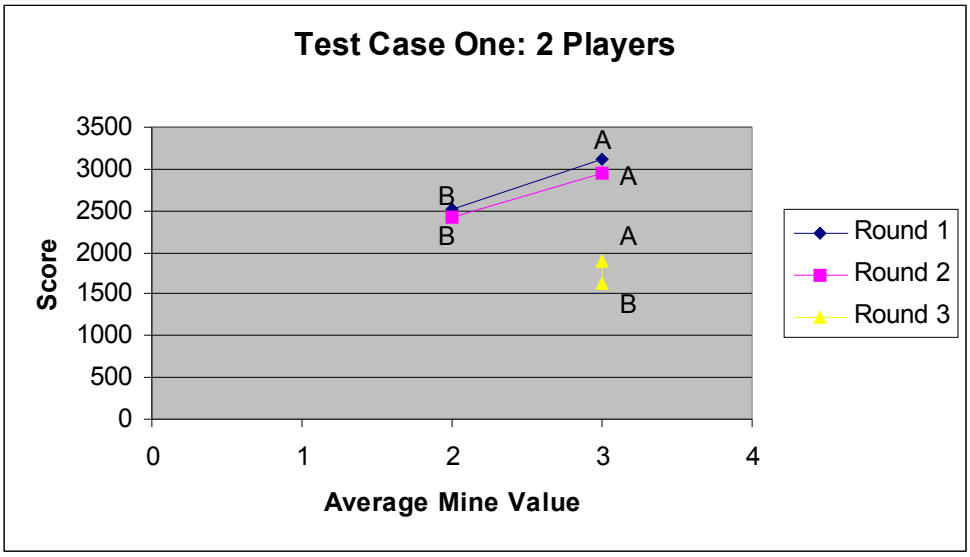


Figure 12: Test Case One- 2players: Average Mine Value vs Score

In the selection phase, Player A's strategy was selecting the plot area that was favorable for mine production. From the graph above, we can see that Player A selected the plot area whose average mine value was 3 (in the range of 1 to 4) as opposed to Player B who selected the plot area with average mine value as 2.

Round 3

So effectively, Player A has produced mines in more of the plots and so he gets ahead in the mine production leading from the first round. It is only in the third round, that the player B selects the plot area with the average mine value as 3.

By each round, the scores are decreasing because the players A and B have not traded for two rounds and are using up the initial units provided to them when they started or joined a game.

By the end of second round, A realizes that there is only other player so he trades his mines with the store so that the other player will not have a chance to sell to the store. The store can only buy certain number of units from the players. But in this round, Player B made a good selection of the plot area and his mine production increases and so the gap between Player A and Player B decreases. The score is also dependent on the average selling price at which the goods are sold. Initially this is very high as the system takes the selling price set by the central store but when the trades are done, the average selling price tends to go down as the players do not sell at a very high price. That is the reason why the score value has come down from round 2 to round 3.

As the mine price is high at the store, the average selling price for mines increases which plays an important role in increasing the score value.

Conclusion:

From the above results, the strategies to be used when playing in the above scenario can be drawn as given below.

Strategy 1: Grasp the chance of selling your surplus to the store

When there are only two players in the game, it becomes very important as to who grasps the chance of selling to the central common store as there won't be any other chance for the other player to sell his surplus goods if the only other player is not buying. The number of times the player successfully traded with anybody will also be counted towards the score value as given by

Score = (mu*msp)+(fu*fsp)+(eu*esp)+ru+(nooftimestraded*50)

where mu represents the mine units,
msp represents the average selling price for mines in the game,
fu represents the food units,
fsp represents the average selling price for food in the game,
eu represents the energy units,
esp represents the average selling price for energy in the game,
ru represents the rokda units , and
nooftimestraded is the number of time the player traded in the game.

Table 11: Score Formula

So, effectively the player A successfully traded 3 times with the store whereas player B traded once successfully as given below.

Table 12: Test Case One-2 players: Trade Table

Buyer login	Seller login	No of units	Product type
store	a	20	Mine
store	a	20	Food
store	a	10	Energy
store	b	0	Food
store	b	10	Energy

Strategy 2: Select plot area that is favorable for the products that are tougher to produce.

In this particular scenario, the plot selection played an important role as it was obvious that whoever had a good plot area for mine production profited.

(b) With four players

Results

The play is divided into rounds based on the number of times the players go through the whole process, from the selection stage to the auction stage.

Round 1:

The following are the average property values for the plot area selected by each player in the round one. Average Mine Value of 4 signifies that it is very good for mining and so on.

Table 13: Test Case One-4 players(Round 1): Average property values

	Average Mine Value	Average Farm Value	Average Energy Value
Player a	2	2	2
Player b	3	2	2
Player c	2	2	1
Player d	2	1	1

In this round, Player B got the best plot as the average mine value of his plot area is 3 and is higher than any of the others' plot areas. In this particular play, producing mine units contributes a lot to the score. Quite expectedly, Player B produces more mines than anybody else in this round.

Table 14: Test Case One-4 players(Round 1): Production values for before and after Auction

	Mines	Food	Energy	Rokda	Score	Mines (After Auction)	Food (After Auction)	Energy (After Auction)	Rokda (After Auction)	Score (After Auction)
A	77	180	190	864	4902	98	180	190	801	3779
B	322	106	150	852	7408(h)	322	116	111	900	4344(h)
C	132	139	51	896	4264	117	149	51	901	3050
D	234	101	27	898	5198	228	114	66	747	3357

In this round, the Player B configures wisely as he got the plot area suitable for mining and he produces 322 units which is much higher than the mine produce of any other players. But, in the auction, he was not successful in selling any of his mine surplus as he sets a higher price compared to Player C or Player D. Player A who produces the least number of mine units wishes to buy and he trades with Player C and Player D. Player B sells 39 units to the Player D. In this scenario, as the number of times the player successfully trades in an auction also contributes to the score as given by the score formula, the Player B's score falls as he was not able to make a successful trade with other players. He keeps on leading though, mainly because of the mine production he had made.

Round 2:

Table 15: Test Case One-4 players(Round 2): Average Property Values

	Average Mine Value	Average Farm Value	Average Energy Value
Player A	2	2	2
Player B	2	2	3
Player C	2	2	2
Player D	2	2	2

In this round, the Player B, who was leading in the previous round has not selected a favorable plot for mines but one for energy. So, he tries to improve his energy units as we can see in the table below.

Table 16: Test Case One –4 players (Round 2): Production Values

	Mines	Food	Energy	Rokda	Score	Mines (After Auction)	Food (After Auction)	Energy (After Auction)	Rokda (After Auction)	Score (After Auction)
A	118	246	263	667	6391	160	246	205	731	3460
B	378	230	219	809	9693(h)					
C	210	201	83	753	5965	208	201	113	699	3115
D	212	137	127	623	5703	172	137	155	607	2825

The Player B chooses not go in for auction, so the trades take place between Player A and Player C, Player C and Player D, Player C and Player A, and Player D and Player A. The Player D in order to buy energy units so that he can satisfy the critical energy units for the next round, he sells his mines to Player A for a low price so he loses his third position to the Player C who was previously in the last position. Choosing not to go to the auction is both negative and positive. It is negative in the sense that number of times you trade counts towards the score and you can only trade by entering the auction. It is positive as the player can go ahead and select a better plot before others come to that round and improve his score. The player should only do this when his score is much ahead of the other players.

Round 3:

Table 17: Test Case One-4 players(Round 3): Average property values

	Average Mine Value	Average Farm Value	Average Energy Value
A	2	2	2
B	2	2	1
C	3	2	2
D	2	2	2

In this round, the Player B gets the plot area with average value of 2 for the properties, but the player B still leads because of the score he had made in the previous rounds.

Table 18: Test case One - four players(Round 3): Production values

	Mines	Food	Energy	Rokda	Score	Mines (After Auction)	Food (After Auction)	Energy (After Auction)	Rokda (After Auction)	Score (After Auction)
A	229	309	272	603	8457	209	309	272	643	8581
B	491	262	231	689	11507(h)	491	262	231	689	11507(h)
C	330	293	162	543	6123	330	243	194	465	8217
D	242	229	165	467	5351	242	279	183	345	7429

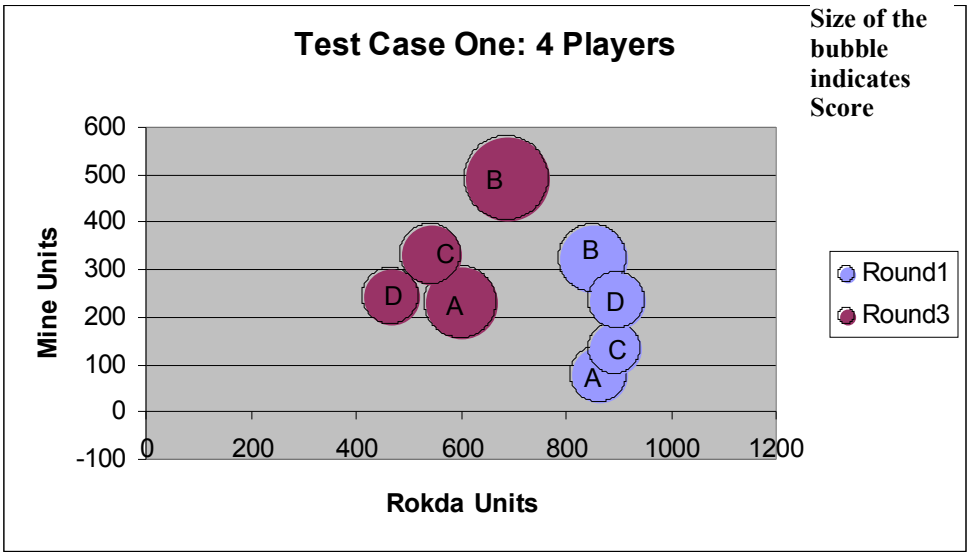


Figure 13: (Test Case One-4 players) Relationship between Mine Units, Rokda Units & Score

Only Round 1 and Round 3 values are plotted in the above graph to preserve clarity. In this scenario, the score represented by the size of the bubbles is directly proportional to both the mine production and also the rokda units as shown above. Only Player A's score is an exception in the sense that his high score value can be attributed to the good energy production he made.

In Rounds 1 and 2, Player B has been leading at the first position, Player A at the second place, Player D in the third position and Player C in the last position, but in the last round, Player C makes good progress as he makes a good trade with Player D in food units and thus his overall score improves and becomes comparable to that of Player A and he stands a chance of moving up to the second place in the fourth round.

Conclusion:

In this four-player game, the strategies to win differ slightly from that of a two-player game

Strategy 1: Make a good selection in the beginning

This strategy holds good in this game also. If you make a good selection and make good production of all the three products, it will not be necessary for you to trade any of the products for a very low price. Like in the demo above, Player B has made a good selection and was even able to skip the auction step without affecting his score much. On the other hand, the Player D had to sell his surplus at a very low price thus affecting his score adversely.

Strategy 2: Make good decisions and at the right time

Keep track of the relative scores and try to move up tackling just the one above you. Try to get better deals than the player just above you. Like in the demo above, the Player C has gone from position 4 to position 3 as Player D had made a bad decision to sell his surplus mines at a very low price.

Strategy 3: Players should be able to understand how the score is being calculated

By looking at the scores, the player should be able to understand quickly by the end of first round, so that he can make better decisions in selecting the plot the next time. This strategy is applicable in the real world too.

5.1.3 Test Case Two

Description

Just deciding to produce a certain product will contribute more to the score.

Deciding to mine will contribute more to the score. In this test case, just deciding to mine will give better score whatever the plot might be good for. For this, the number of times the player decides to mine is stored and this value is utilized in calculating the score.

The formula used to calculate score is given below:

```
Score=mine_times*10+farm_times*2+energy_times*2
```

a) With two players

Results

Round 1:

The following table gives the average mine value, average farm value and average energy value of the plot areas selected by Player A and Player B respectively.

Table 19: Test Case Two-2 players (Round 1): Average Property Values

	Average Mine Value	Average Farm Value	Average Energy Value
A	2	2	1
B	2	2	3

The following table gives the number of times the player selected to mine, farm and produce energy respectively.

Table 20: Test Case Two- 2 players (Round 1): Configuration Counts

	Mining Count	Farming Count	Energy Count
A	2	3	4
B	8	1	0

From the table above, we can see that B has configured the plot area so that it produces mines in most of the plots. So, effectively, in this scenario, the player B will have an advantage in the score comparison. But it is important to notice that the score calculation and the production calculation are different here. In order to go to the next round of selection without going to auction, the player should satisfy the critical resources limits.

Table 21: Test Case Two – 2 players(Round 1): Production values

	Mines	Food	Energy	Rokda	Score	Mines (After Auction)	Food (After Auction)	Energy (After Auction)	Rokda (After Auction)	Score (After Auction)
A	374	378	597	1768	34	462	378	567	1682	34
B	1984	386	14	1590	82	1856	386	44	1756	82

Both Players A and B needed to go to the auction as Player A was short of mines and food and Player B was short of food and energy. The score value does not change, as it is not dependent on the product units in this scenario.

Round 2:

The average property values are given below.

Table 22: Test Case Two - 2 players (Round 2): Average Property Values

	Average Mine Value	Average Farm Value	Average Energy Value
A	2	2	2
B	1	2	2

The number of times player selected to mine, farm or produce energy is given below:

Table 23: Test Case Two - 2 players (Round 2): Configuration Counts

	Mining count	Farming count	Energy count
A	9	0	0
B	4	2	3

In this round, Player A chooses to mine in all the plots of his plot area effectively increasing his score and getting a lead. Player B chooses to improve his food units and energy units instead. But though in this round, Player A has a lead, Player B can take the lead in the next round as he has surplus in all the products and so he can skip the auction step and go for another selection and configuration.

Table 24: Test Case Two - 2 players (Round 2) : Production Values

	Mines	Food	Energy	Rokda	Score
A	2398	349	515	1317	124
B	2163	608	705	1543	116

Conclusion

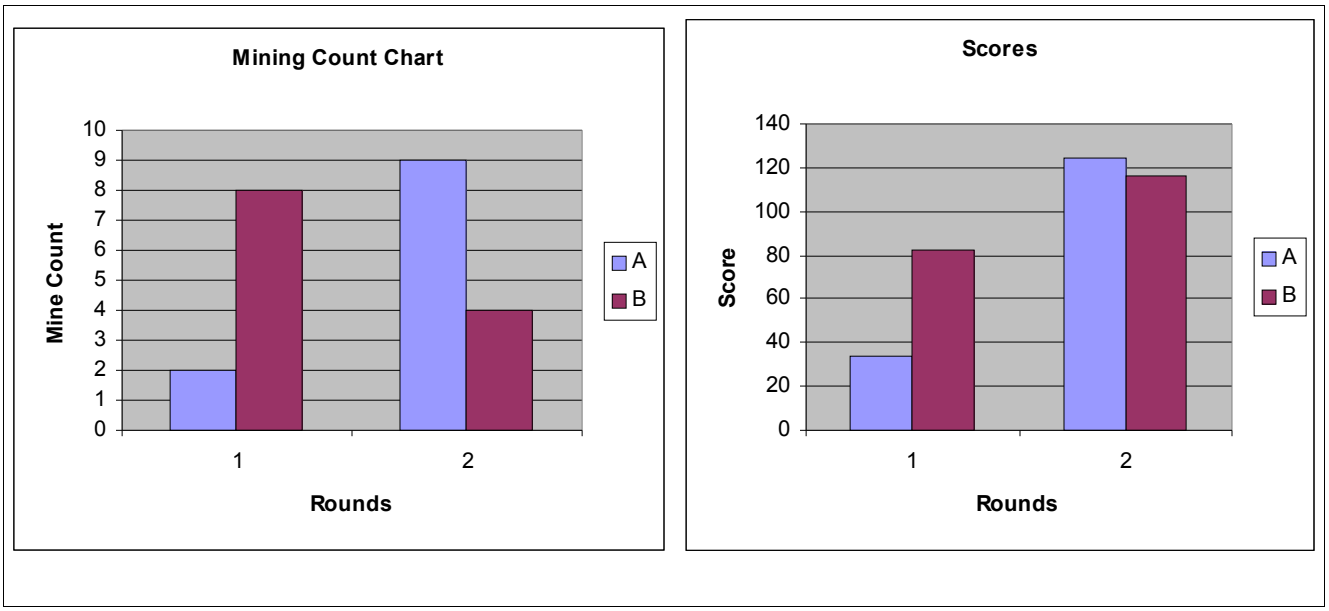


Figure 14: Test Case Two - 2 Players: (Mining count and score charts)

In this type of scenario, just deciding to mine in as many plots as possible gets you the lead in the game. But if the player initially concentrates on not getting a shortage in any of the products and then from the next round onwards, decides to mine in most of the plots would be a better idea to maintain the lead. Otherwise, if in the first round itself, the player decides to mine in most of the plots and gets a shortage in the other products, then he has to spend time in auction, while the other player can increase his score by going for the next rounds of selection and configuration. This point can be illustrated by the graph above, Player A has a lead in the second round because he has decided to mine in most of the plots but Player B is not far behind and he also has surplus in all the products, so in the third round if there was one, the player A will surely take the lead while the player A goes to auction.

(b) With four players

Results

Round 1:

Table 25: Test Case Two-4 players (Round 1): Average property values

	Average Mine Value	Average Farm Value	Average Energy Value
A	2	2	2
B	3	3	2
C	1	2	2
D	2	2	3

Table 26: Test Case Two-4players(Round 1) :Configuration counts

	Mining count	Farming count	Energy count
A	9	0	0
B	3	3	3
C	3	2	4
D	2	3	4

As you can see from the above tables, Player A has selected to mine in most of the plots and he is leading in the score as shown below. Only player B gets a surplus in all the goods and he need not go to the auction step whereas all the players are required to go to the auction step.

Table 27: Test Case Two-4players(Round 1): Production values

	Mines	Food	Energy	Rokda	Score
A	1670	28	10	1568	90
B	511	594	841	1708	42
C	390	340	806	1766	42
D	249	904	830	1738	34

So, the player B goes for another three rounds of selection/configuration and gets a lead in the score. He increases his score from 42 to 100 to 190 and then to 280. Only at this stage does his farm units decrease to 562 because of the expenditures involved in production stage. So, then he decides to go to auction to make up for his resources. The scores at the end of the round are as follows:

Table 28: Test Case Two-4players: Scores

Player	Score
A	90
B	280
C	42
D	34

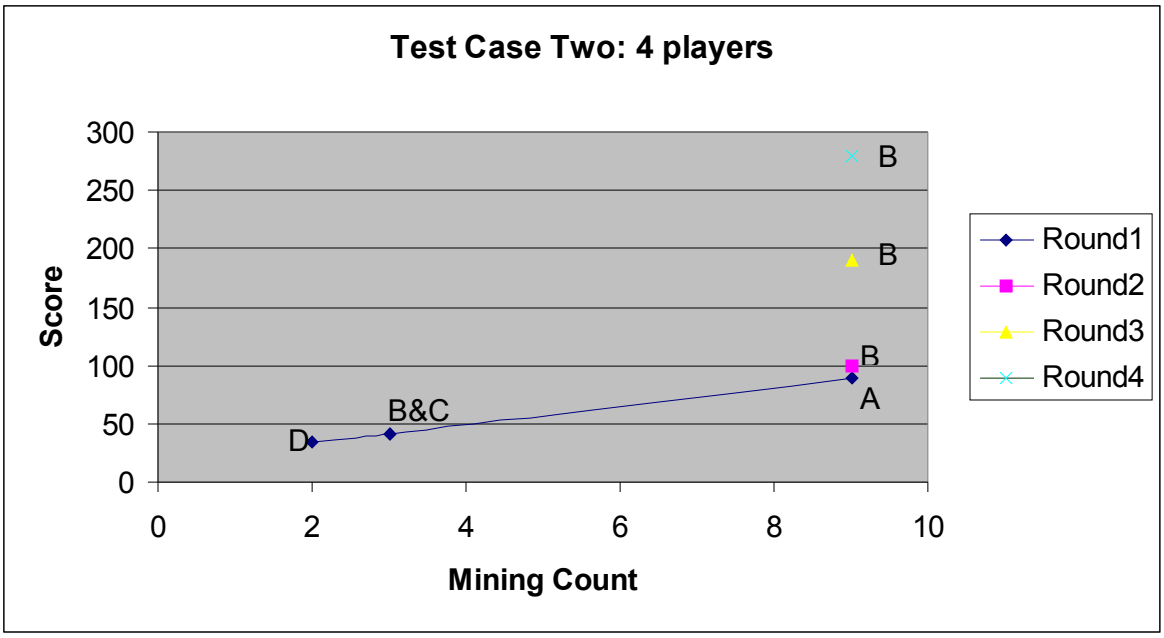


Figure 15: Test Case Two-4 players (Mining Count vs Score)

Conclusion

The Player B gets the lead by following these strategies in this scenario.

Strategy 1: Try to get a surplus in all the products initially instead of concentrating on leading in the score value.

Initially, try to get a surplus in all the products so that you need not go to auction and waste time, as the score formula does not depend on product units or the number of times you have traded. In this particular demo, Player A straight away started with producing mines alone, thereby getting an initial lead but fell short of food and energy units; whereas, Player B gets a surplus in all types of products and then later on produces only mines and maintains the lead for many rounds.

Strategy 2: Go to selection/ configuration stage as many times as possible

Go to the configuration stage as many times as possible as the score only depends on the number of plots you have decided to produce mines.

5.1.4 Test Case Three

Description

This test case is a scenario based on real world concepts. The worker expenses are higher for producing mines than for farming or energy production. The worker expenses are less for food production than for the production of mines or energy.

The way in which this scenario is set up is by changing some formulae in the properties file. Production is directly proportional to the appropriate property value for the plot and time for which production has been done on that plot. Each formula produces the corresponding product.

```
ProductionFormulaForMining=(10*m*time)
ProductionFormulaForFarming=(10*f*time)
ProductionFormulaForEnergy=(10*t*time)
```

In the real world, the worker expenses are more for Mining compared to farming or for producing energy. This is realized using the formulae given below. These expenses are subtracted from the rokda units (monetary units). Also the land expenses are more for mining land than for the farming or for producing energy.

E **With two players**

```
WorkerExpensesForMining=(3*(m+f+t))
WorkerExpensesForFarming=(1*(m+f+t))
WorkerExpensesForEnergy=(2*(m+f+t))
LandExpensesForMining=(3*(m+f+t))
LandExpensesForFarming=(1*(m+f+t))
LandExpensesForEnergy=(2*(m+f+t))
```

Results

Round 1:

Table 29: Test Case Three-2 players(Round 1): Average Property values

	Average Mine Value	Average Farm Value	Average Energy Value
A	3	3	2
B	2	2	2

The Player A gets exceptionally good plot that is both good for mining and farming whereas the Player B gets an average piece of plot area. Production values after round one are given below.

Table 30: Test Case Three-2 players(Round 1): Production values

	Mines	Food	Energy	Rokda	Score
A	806	1082	302	1698	20830(h)
B	366	720	508	1776	15260

Player A quite expectedly does well in the production of mines and food but not in energy. His rokda units are lesser than that of Player B as the Player A has produced mines in more plots than Player B and worker expenses are more for mining than for the other production. But mine production helps in increasing the score as the mines are sold at a higher price than the other goods.

The production values after auction are given below:

Table 31: Test Case Three-2 players(Round 1): Production values

	Mines	Food	Energy	Rokda	Score
A	766	1042	312	1660	20316(h)
B	406	720	517	1818	15620

The trades that took place are given below:

Table 32: Test case three - 2 players (Round 1):Trade table

Buyer	Seller	No: of units	Product Type
Store	A	40	Food
a	Store	10	Energy
b	A	40	Mine
b	Store	40	Energy

After the auction, Player B seems to have improved and Player A seems to have gone below his own score as Player B gets the good trades. In fact, Player A sets a low price for his surplus and Player B gets those goods and so effectively increases his score.

Round 2:

Table 33: Test Case Three-2 players(Round 2): Average property values

	Average Mine Value	Average Farm Value	Average Energy Value
A	2	1	2
B	2	1	3

The production values after production in the round 2 are as follows:

Table 34: Test Case Three- 2 players(Round 2): Production values

	Mines	Food	Energy	Rokda	Score
A	1210	1484	582	1620	30254(h)
B	796	1162	859	1416	25548

Player A leads after the round two. In this round, both the players get equally favorable plots for mining and farming. So, A maintains the lead. Player B improves his energy score and is in better position than the Player A.

Conclusion

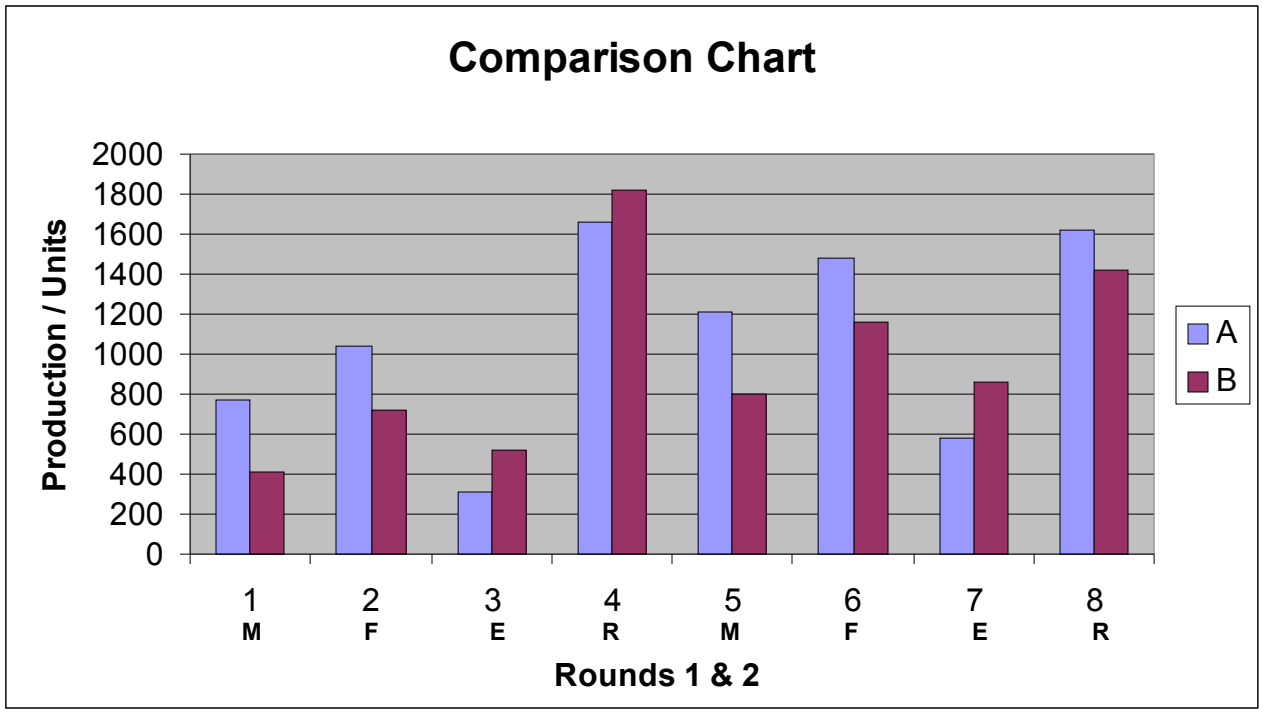


Figure 16: Test Case Three-2 players: Comparison of production values

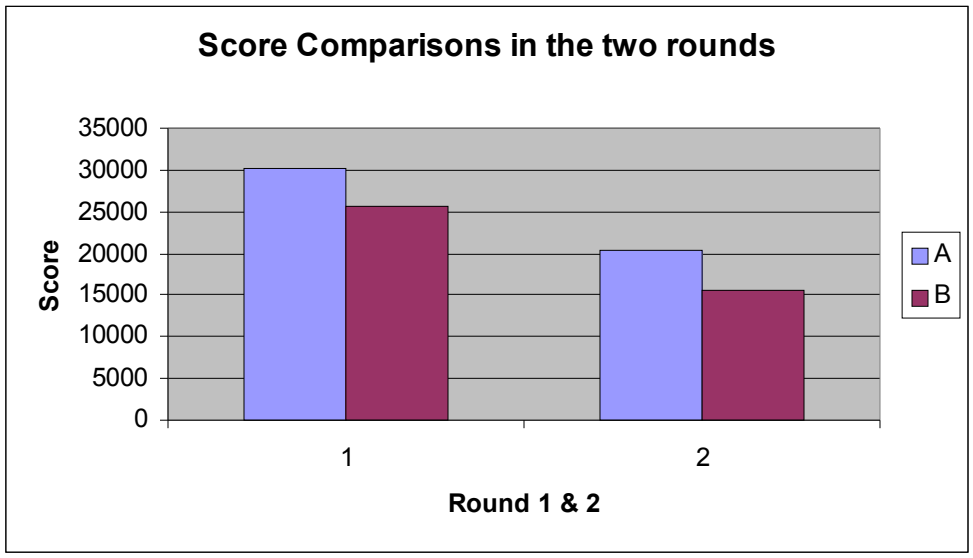


Figure 17: Test Case Three -2 players: Score Comparison

As seen from above two charts, the Player A has been leading in the score value over the two rounds. We can notice from the graphs that he has concentrated on food production as in this

particular scenario the worker expenses and land expenses are lesser for farming than for mining or energy production. That is the reason his rokda score is more than the Player B.

In this scenario, the player who has a good selection and also the player who balances the configuration of the plots is in a position to lead. He has to take care that he is not mining in the plot that is not suitable for mining as the worker expenses, land expenses would increase and his rokda score would go down.

b) With four players

Results

Table 35: Test Case Three -4 players(Round 1): Average property values

	Average mine value	Average Farm value	Average Energy value
A	1	2	2
B	3	1	1
C	2	2	2
D	2	2	2

Table 36: Test Case Three- 4 players: Configuration Counts

	Mining count	Farming count	Energy count
A	3	4	2
B	5	1	3
C	3	3	3
D	6	2	1

Player A has mined in three of the plots though his average mine value was only 2. That was the reason for the increase in expenditure (as mines expenditure is inversely proportional to the mine value) and decrease in the net production. Player B does well in both the mines and energy production though faring badly in the food production as predicted by the average property values. Player C does well in all the three types of production and hence leads. Player D makes a bad choice of producing mines in six of the nine plots though his average mine value is only 2.

Table 37: Test Case Three - 4 players: Production values

	Mines	Food	Energy	Rokda	Score	Mines (After Auction)	Food (After Auction)	Energy (After Auction)	Rokda (After Auction)	Score (After Auction)
A	270	470	251	1740	5974	320	503	261	1600	5820
B	1290	82	677	1714	11200	1290	109	673	1672	11762
C	850	730	709	1722	11728	850	630	659	2022	10999
D	972	372	285	1678	9166	922	412	329	1560	9321

Trades that took place are as given below:

Table 38: Test Case Three - 4 players: Trade table

Buyer Login	Seller Login	No Of Units	Product Type
D	C	40	Food
D	C	44	Energy
A	D	50	Mines
A	C	33	Food
A	B	4	Energy
A	C	6	Energy
B	C	27	Food

In the auction, Player B makes good deals, he buys food units from Player C at a low cost of 2 and sells energy units at a high price of 3; whereas, Player C makes a bad choice of selling his surplus at low price. Thus, Player B begins to lead in the game after the auction.

Round 2:

Table 39: Test Case Three-4 players(Round 2): Average Property Values

	Average mine value	Average farm value	Average Energy value
A	2	3	2
B	2	2	2
C	2	3	2
D	2	2	2

Table 40: Test Case Three -4 players: Configuration counts

	Mining count	Farming count	Energy count
A	3	3	3
B	6	1	2
C	4	3	2
D	3	3	3

Player B decides to mine in most of the plots in the second round as he got a good number of mines and mines contribute more to the score as it is sold at a high price at the store and hence leads. All the players' scores fall after auction as the score formula uses average selling price of the product at which it is sold at the auction. In this particular demo, the mines selling price has decreased from 5 to 2 as the players have sold mines at a low price.

Table 41: Test Case Three –4 players: Production Values

	Mines	Food	Energy	Rokda	Score	Mines (After Auction)	Food (After Auction)	Energy (After Auction)	Rokda (After Auction)	Score (After Auction)
A	910	1263	928	1302	13342	870	1248	888	1492	11394
B	2435	299	989	1284	19762	2435	335	989	1080	14926
C	1415	1162	953	1732	16577	1415	1162	953	1732	13747
D	1422	916	642	1326	14538	1422	899	642	1360	11679

Trades that took place in this round are given below:

Table 42: Test Case Three -4 players: Trade Table

Buyer Login	Seller Login	No Of Units	Product Type
B	D	17	Food
B	Store	4	Food
B	A	15	Food
Store	A	40	Mines
Store	A	40	Energy

Conclusion

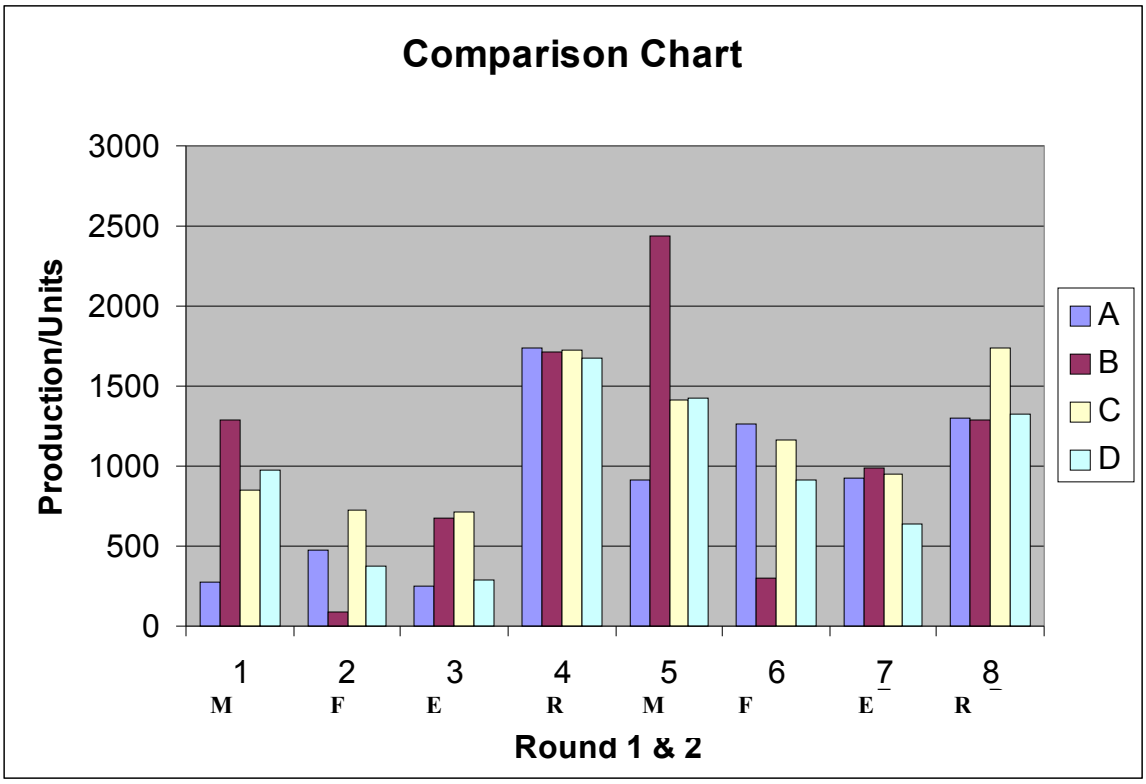


Figure 18: Test Case Three - 4players): Comparison of production units

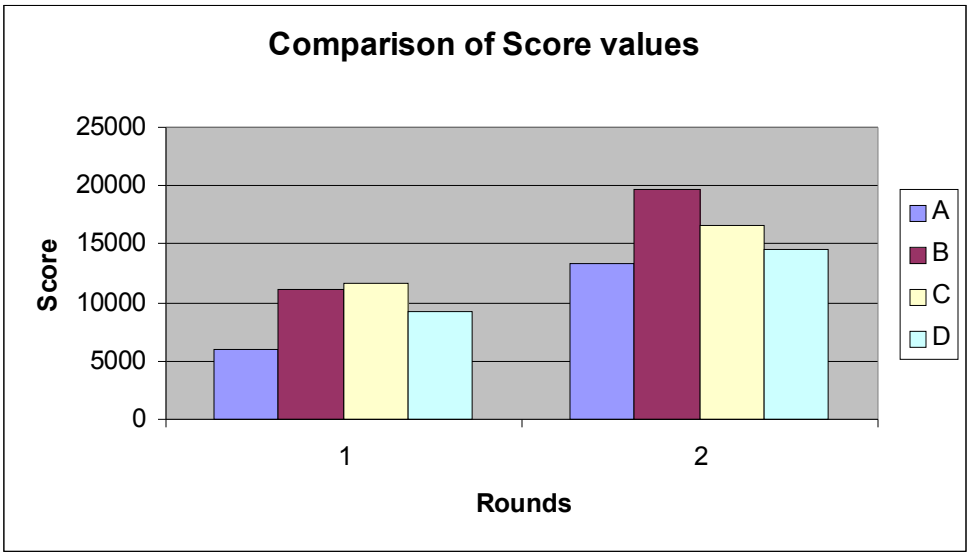


Figure 19: Test Case Three –4 players: Comparison of Score values

From the above two charts, it is evident that the Player B's lead is due to his good mine production and also the maintenance of good rokda/money units. In this particular scenario, the rokda units can be maintained only when configuration of plots is done wisely. The strategies to follow to maintain a lead in this scenario are given below:

Strategy 1: Make a good selection.

This strategy works in this scenario also as if you select a good plot area then you will be in a better place whatever you wish to produce.

Strategy 2: Produce depending on the property values of your plot area

Do not produce a type of product just because it has more value. Here, player A had made that mistake of producing mines just because mines have more value at the store.

5.2 Usability test cases

The usability of this game application based on the following criteria:

1. Understanding of the game
2. User Interface

The players we have selected fall under 3 categories.

1. Experienced players: These players have lot of experience in using the PDAs and they own a PDA themselves and play a lot of games.
2. Not-so-experienced: These players have lot of experience using computers in general but not particularly PDAs

Table 43: Usability tests

	Experienced	**Not-so-experienced**
Understanding the game	Could easily understand the game	Required effort to explain the game.
User Interface	Could easily understand the user interface.	Took time to get used to the user interface and to understand the soft buttons and the Select button on the pocket PC.

6.Conclusion

As discussed in the previous chapters, we have implemented a framework for economics games in this project. This project can be utilized to conduct some kind of experiments by creating different economic scenarios that may affect the way the buyers and sellers behave in any local auction. There can be multiple games with multiple players at any time in this distributed system and each game can be thought of as a separate local auction.

The game application is designed as a three-tier distributed system and the user interface is implemented using the model-view-controller design pattern, the smooth flow of the different states is implemented using state-based design pattern. The features that are employed in the development of the application are the User interface classes, Generic Connection Framework, and Persistent Storage. Some new classes that are introduced in MIDP 2.0 like CustomItem class have been utilized.

In most of the mobile applications that are client-server based, the server plays an important role as mobile devices have less computing power. In this project also, most of the functionality is shifted to the server.

All the communication takes place with the server in the middle. The communication with server is achieved through the JDBC and Servlets.

As this game is developed using Java, it can be played on any mobile device that supports J2ME that can be Palm devices, Pocket PCs, and many other cell phones.

Future Enhancements

This project uses a trivial login/password pair for the logging in the game. In the future, a better security scheme can be plugged in for increasing the game security. This game application uses only three types of products. Other types of products can be added and user interface can be improved further. There are different types of auctions. The one that is implemented here is the buyer-initiated auction. We can further extend the project to implement the other kinds of auctions. More economic experiments using the existing program could be carried out.

7. References

[SR02] Srikanth Raju (2002). XML and Java language programming for wireless devices: a tutorial. Retrieved May 21, 2003, from http://www.sun.com/developers/evangcentral/totallytech/j2me.html

[SM0503] Sun Microsystems Inc. Programming strategies for small devices. Retrieved May 21, 2003, from http://wireless.java.sun.com/midp/chapters/j2megiguere/chap3.pdf

[K02] Kim Topley (2002). J2ME in a Nutshell: A Desktop Quick Reference. (1st ed.). O'reilly Publications.

[J03] James Keogh (2003). J2ME: The Complete Reference.(1st Ed). Tata McGraw-Hill.

[P02] Paul Tremblett (2002). Instant Wireless Java with J2ME.(1st Ed). McGraw-Hill.

[K02] Jonathan Knudsen (2002). Wireless Java: Developing with J2ME (2nd Ed). An Apress book.

[CA03] Cynthia Bloch, Annette Wagner (2003). MIDP 2.0 Style Guide (1st Ed). Addisson-Wesley Professional.

[VM02] Victor Votsch, and Mark Walter. (March 02). http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb_buswp.pdf. Retrieved Dec 01, 2003.

[K95] J. Kagel. Auctions: A survey of experimental research. In J. Kagel and A. Roth, editors, Handbook of Experimental Game Theory. Princeton University Press, Princeton, 1995.

[NPP91] Nelson, Paul S. and Paul W. Grimes. "Supply and Demand Analysis: Using Markets Created in the Classroom." Journal of Education for Business, 66(6), July/August 1991, pp. 370-373.

[JEP] Java Mathematical Expression Parser v2.3.0 from Singular Systems

http://www.singularsys.com/jep/

[JCP02] Java Community Program (2002). http://www.jcp.org

[OSVGE] Open Source Vector Graphics Editor – Inkscape. http://www.inkscape.org