Supply and Demand Simulation Software

Project For:

ECE 4220: Real-Time Embedded Computing

Dr. DeSouza

Project By:

Bernard Naumann

Abstract

This project is to create a game that teaches business students about supply and demand. This is a problem that is often described in the classroom, but not often demonstrated with hands on learning. Since a lot of students achieve a greater level of understanding by taking the hands on approach, this game would serve as a stepping stone to achieve that understanding. The idea is to split the students into two groups, buyers and sellers. The sellers will be selling a product of unknown value to the buyers. The winners of this game will be the seller who has the most cash and the buyer who has the most product at the end of a set time period. The key is that since sellers can post a sale at any price and buyers can offer any price, the numbers will meet very quickly in the middle and the winners will only win buy marginal amounts. This project will be implemented in C programming and use a network of computers able to send messages to the host computer, which will make the sale or offer known to the other students. In this configuration the master will be the host computer and the slave computers are the ones being used by the students playing the game. The master computer will be in charge of updating the list of offer and bids, keeping track of the game timer, and deciding the winner. This leaves the slave computers to only be in charge of taking user inputs and keeping track of the user's score. Each slave computer will use a socket to send messages to the host computer, while the host computer will use the socket to broadcast messages to all other computers. Multithreading will be used in conjunction with real time tasks to update and remove posts based on which offers/bids have been made or accepted and to keep track of the game timer.

Introduction/Problem Statement:

This project provides a unique tool to help teach supply and demand. The reason having a program like this is helpful to a student is that a lot of students simply learn better through hands on activities. This project caters to that methodology by teaching it in a fun and interactive way by allowing them to compete against their classmates in what they initially view as a game. While it is indeed a game of sorts, it is easy to realize after the game is over how the fundamental idea of supply and demand truly are applicable to most things in a marketplace. The idea is to split the students into two groups, buyers and sellers. The sellers will be given a specified amount of a product of unknown value and the buyers will be given a specified amount of money. The sellers will proceed to sell the product to the buyers until there is no more products to sell, no more money to buy with, or until the game timer runs out. The winners of this game will be the seller who has the most cash and the buyer who has the most products at the end of a set time period. The key is that since sellers can post a sale at any price and buyers can offer any price, the numbers will meet very quickly in the middle and the winners will only win by marginal amounts, thus showing the basic economic policy of supply and demand.

Development and Implementation:

This project will implement a C program on TS-7250 embedded system boards. The boards will be setup on a network and connected to computers that will display the information. The boards will communicate with each other through UDP networking settings. Since one of the features of this product is that it does not matter where the buyers, sellers, or even the host I located, they will have to determine their own IP dynamically. Sockets will be used by all computers to send and receive messages. All messages, both offers and acceptances, will be sent to the host computer to be processed. After being processed a message will be broadcasted by the host computer to either post or remove an offer from the other computers and inform the computer who put in the request that they got the offer. If multiple requests are made on the same offer then the first to make the request will win.

During the setup of the game all computers will need to be assigned to be a buyer or a seller. To implement this, a way of voting was chosen. After the host inputs the number of players participating in the game each player will then have a chance to vote on what role they would like to play. Since an equal number of buyers and sellers is desired the roles will be handed out on a first come first serve basis and if a role that has already been fulfilled by other computers is voted on by another player than that player will be automatically shifted to the other role. As computers vote on a role in the game they will also be sending their ip address so the host has a way of identifying that player. After receiving all the votes the host will then send messages out to all computers informing them of their roles and giving them an amount of units or cash based on that role. The player computers will check each message for the ip field to see if the message applies to them and if it does then the board will read that message and either become a buyer or a seller based on that information. After the host is done initializing all of the players it will send out a message informing everyone that the game is ready to begin.

One board will have to be the host, and this will require a different set of functions to handle the game. The first function of this board should be to setup a game timer. This timer should last for a reasonable amount of time to finish the game. This thread will be set up using pthreads and the made into a real time task. The real time task will be set to have a period of one second so that it can increment a counter. The counter is useful because the timer task is also in charge of letting the player boards know when it is time to update the available offers screen. Updates have to happen at an amount of time so that the pace of the game is not to slow and also ensure that the players have time to respond in time before the next update. It will also be the duty of the timer task to acknowledge when the allotted game time has expired and send a message to all boards indicating it is time to tally up the scores.

Another task running concurrently with the timer task will also be created and implemented in real time. This task will run more frequently as it will be in charge of accepting all user inputs. It will also be programmed so that this task runs as many times as it is called until the game timer sends a message that the game is to end. This task is essentially a conditional statement with several cases. The first case, as just discussed, is that if the end of game message is received then the pthread should exit and run no more. Another case would be if a player had submitted a valid response. In this case the host computer should add it to the list of offers in the appropriate location based on whether it was a seller or buyer who made the offer. Since there is a limit on how many offers will be stored by the host, it will also check if an offer is being removed from the list. If there is indeed an older but still valid offer being removed than the player that posted that offer is notified and the funds held back because of that offer are made available again. The new offer will then be included in the next update sent to the players. The last case would be if a player had accepted an offer. In this case the offer needs to be checked to see if it is still available since updates are only given to the players periodically and another player might have previously accepted. If that offer has indeed already been accepted previously the player should be notified and no change to their funds should occur. If, however, the offer is still on the table then the player should be notified that they received the offer and then the offer should be removed from the list so that it is not included in any further updates.

The last thing the host computer is in charge of is determining the winners for each role. This will be done at the end of the game after the timer task has finished running and all threads are closed. The host sends a request out to all other players for their player status and then sorts them based on role into a player array. The array is then looked at and a message is sent listing the buyer with the most units, using cash as a tiebreaker as the winner of that role all players. After the buyer winner is decided the array is looked at again and a similar message is sent listing the seller with the most cash, using units as a tiebreaker as the winner of that role. After this final task the host exits and the game is over.

On the player's boards a different program is implemented. As discussed earlier the first thing the player program does is find its ip dynamically and then send that along with its newly found ip address to the host computer. Upon receiving back the player's actual role, the program waits for the host to send a message to begin the game. A screen is then printed showing first the information about the player at that computer immediately followed by offers available by the opposite role that are ready to be accepted. Next, the offers of that player's competitors are printed out so that the player gains insight into what a good offer would be. Finally, a list of options available to the user is printed out. These options are to make an offer, accept an offer, or to simply wait and do nothing until the next refresh occurs and more offers are available.

For the first option, making an offer, the player will be guided through a short process asking them how many units and how much cash they are looking to do business with. These values are then checked to ensure that the player has sufficient funds to back up said offer. If the player does not have enough units or cash, depending on the role, to support the deal then they are asked to reenter parts of data that are out of their range. If, on the other hand, the offer is valid then it is sent to the host to be included in the next update. The program then waits for a refresh from the host timer task and then updates the display with new offers.

The next option, accepting an offer, is similar to that of making an offer. The player is asked to choose from the list of offers and then the offer data is compared to what the player has to see if the player has sufficient funds to accept said offer. If the user does not then they are sent to wait for a refresh message from the host timer task to display the offers screen again. If, however, the offer is acceptable by the player a message is sent to the host computer requesting the offer for itself. The player computer then waits for a confirmation message from the host to know if they got the deal or not. If the offer is unavailable then no player data is changed and a refresh is awaited. If the offer is claimed by the player then the appropriate funds are deducted or allocated and once again, a refresh is awaited.

The only other option available to be received by the player is the end of game option, in which the player will halt everything and send a message to the host with information about what address the message is from, what role that player fulfilled, and the amount of cash and units that player has. The player program will then wait until the host is done processing all player scores. Once the host is done determining the winners the data will be received by each player and printed out on the screen. This message will detail the winning player's cash and units for comparison to the player's own statistics. After the game is over the players will be told what the starting units were and what the starting cash was for each player and inferences could be made about the distribution and evening out of funds.

Relevance to Class:

This project is relatable to what we've done in class because it encompasses things that we have done in every lab and combines them together. One of the first things we learned about was multithreading, which is used in this project multiple times. Multithreading is necessary in this project because the computers need to be checking for multiple things at any given time. One of the biggest advantages of using multithreading, even on the single core processor on the TS-7250 board, is its ability to keep the board responsive to inputs while running other tasks. The concurrent execution of updating game clock, player screen refreshes, and checking if the user is ready to accept an offer or post one of their own is absolutely necessary in a game like this. Implementing this code on the boards in lab is ideal because embedded boards generally have more support for multithreading because they have decreased thread-switch time.

The use of real time is also necessary for this project. Although used sparingly in this project, the project would have failed without it. There is no better way to implement an accurate game timer than I can think of than setting up a periodic task. Also, the setting up of real time tasks ensures that offers for all players are updated at the same time. This is important because if players updated randomly then some users would be able to see some offers quicker and thus gain an unfair advantage over the other players. This would largely obscure the supply and demand curve and defeat the purpose of this lesson.

Another main subject that this project is very intensive on is network communication. This was also discussed in lecture several times and was included in a lab about communicating through sockets. The project relies heavily on this portion of the class as it is a game with multiple players, each one on a different computer. Without sockets this project would cease to exist because you would just be staring at a blank screen. This part of the project is also ideal for the TS-7250 boards in lab as they are already connected on a network and can communicate with one another by basic socket implementation.

Results:

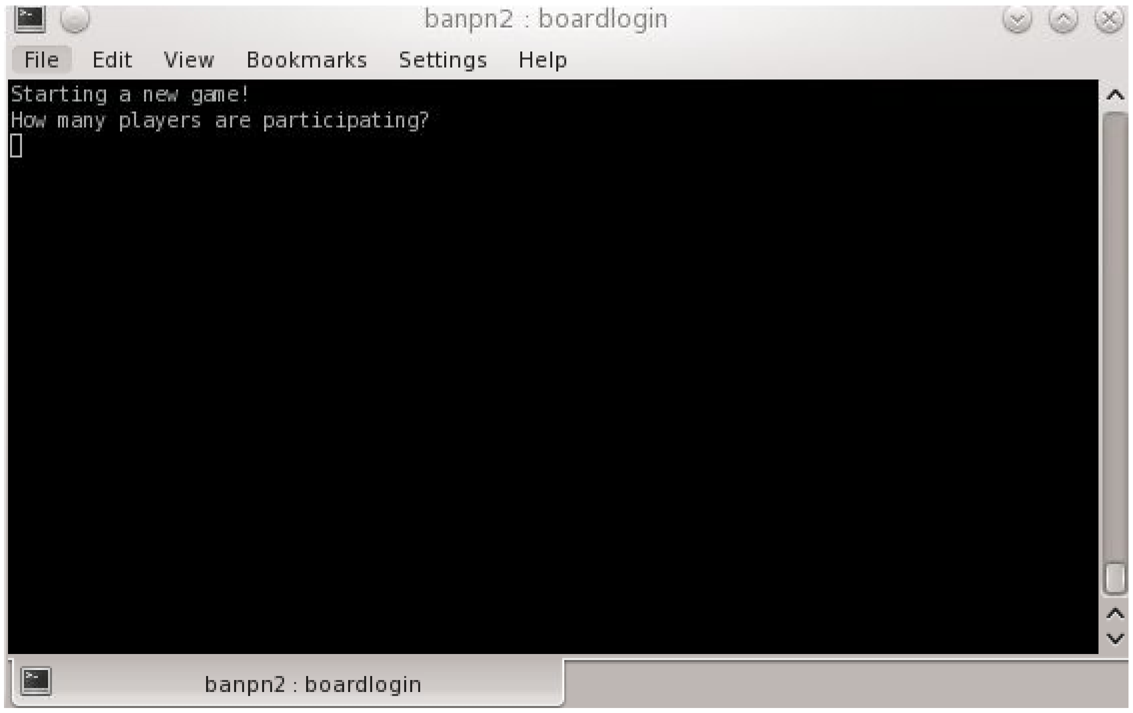
Overall, the program is a successful demonstration of why supply and demand is a relevant policy. The figures described in appendix A show snapshots of the different stages in the game proving that successful runs have been made and it is in working order. The problem this program was to solve was proving the laws of supply and demand. The reason this program works in showing that is largely due to the fact that the players can see all the offers on the table including offers for them and offers made by their peers. This leads to a competitive market where sellers are trying to beat other sellers and buyers are trying to beat out other buyers. On the counter though, the goal as far as the players see the game is also to have the best score at the end, which in turn leads to the players not wanting to sacrifice too much for the sake of getting the offer.

The one shortcoming of this project would be the lack of integration of hardware interrupts. The user inputs to decide if the player would like to make an offer, accept an offer, or do nothing can interfere with the updating of that specific board if a decision is not made by the next refresh cycle. This could be solved by using interrupts so that instead of waiting on a scanf statement to finish running, the program could continue to cycle and if the user did decide they would like to make or accept an offer they could push a button and an interrupt would be triggered and the service routine could handle the offer making and accepting process instead of the player program.

Conclusion:

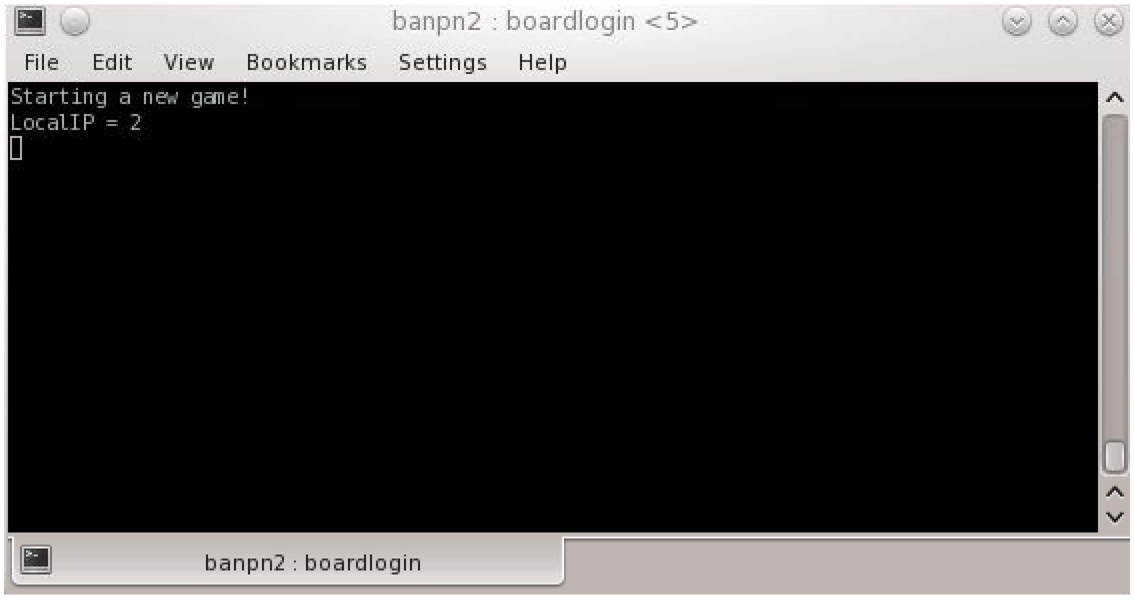
Learning by doing is the way a lot of students learn best, and in business classes there are not a lot of opportunities to do that early on. This program gives an opportunity for students to learn by interacting earlier on in the learning process rather than trying to absorb everything through power point presentations and lectures. In the future I think that this project could use a face lift and put into a GUI. For the time in which this project was done though, without prior knowledge of GUI programming, I did not feel as though I could accomplish this, but do believe that it would help the players to more easily interact with the finished product.

Appendix A



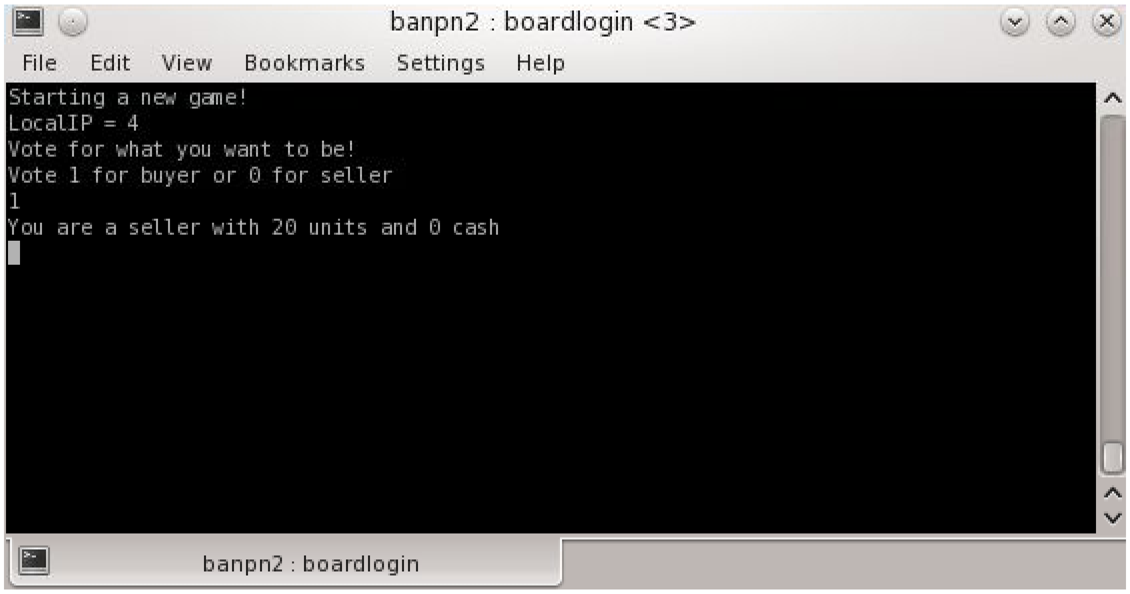
```
Starting a new game!
How many players are participating?
```

Figure 1: The initial host board screen after just running the program. After waiting for all boards to connect the host then enters the number of players to trigger all the boards to start the game.



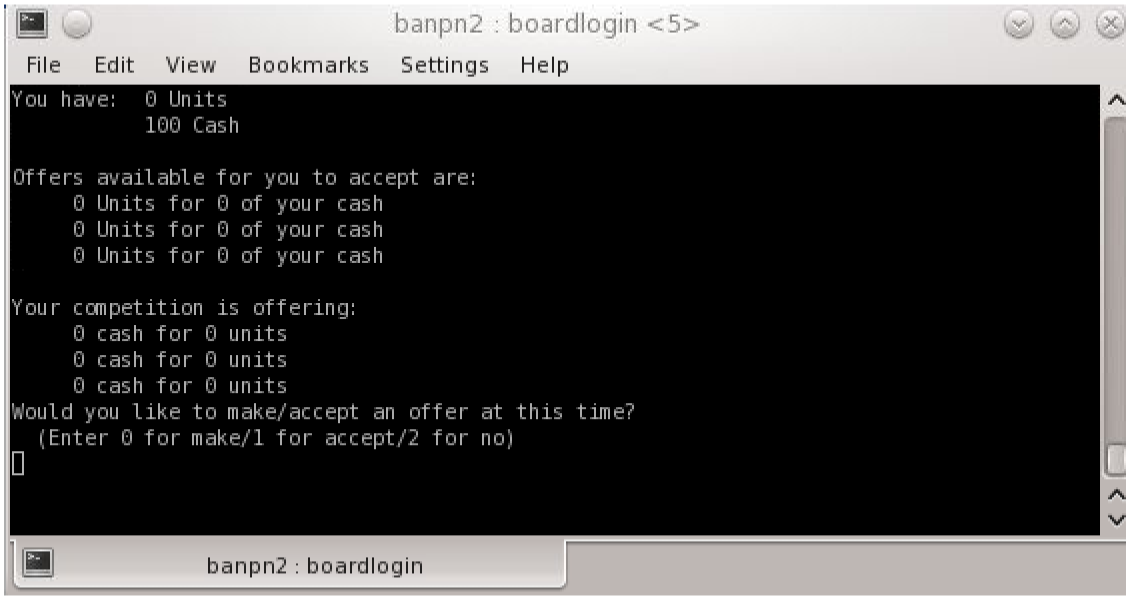
```
Starting a new game!
LocalIP = 2
```

Figure 2: The initial player board screen after just running the program. Host has not entered number of players yet.



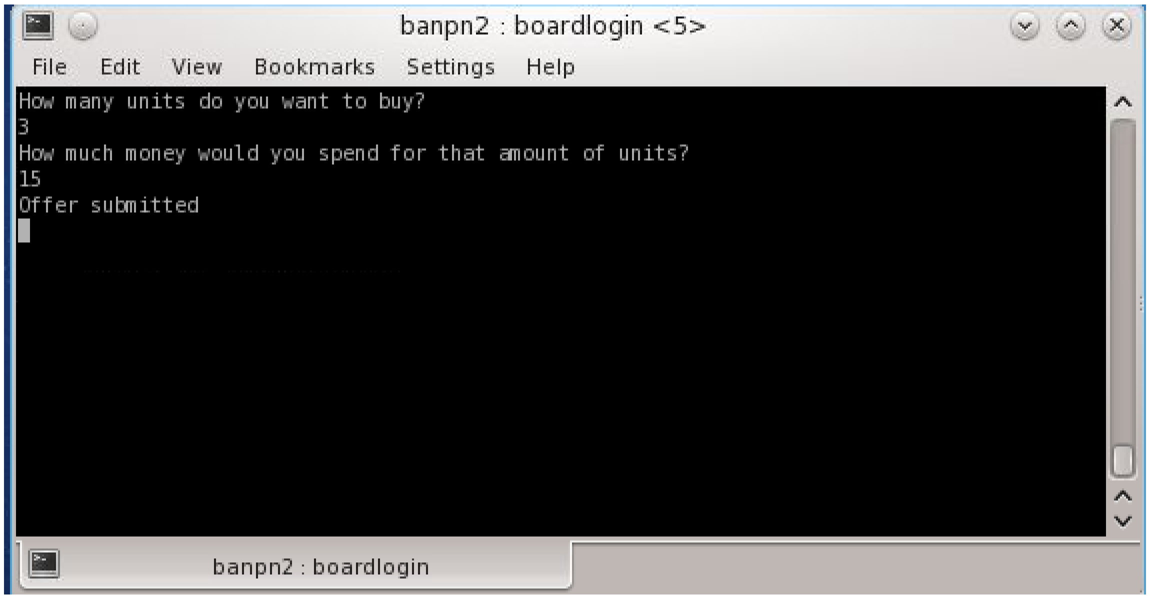
```
Starting a new game!
LocalIP = 4
Vote for what you want to be!
Vote 1 for buyer or 0 for seller
1
You are a seller with 20 units and 0 cash
```

Figure 3: Player board after host has entered number of players. Lines 3 and 4 are printed and then the program pauses to accept user input before continuing. The last line is only printed after the host board relays what the player's role will be. This is an example of the player voting for one role but that role already being fulfilled by too many other boards.



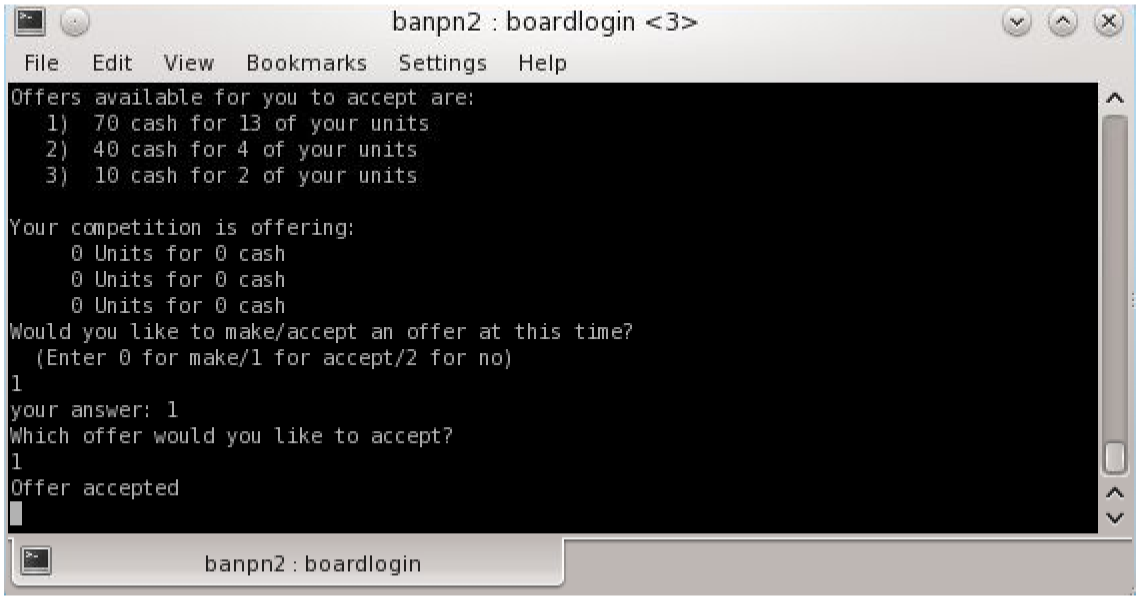
```
You have:  0 Units
           100 Cash

Offers available for you to accept are:
     0 Units for 0 of your cash
     0 Units for 0 of your cash
     0 Units for 0 of your cash

Your competition is offering:
     0 cash for 0 units
     0 cash for 0 units
     0 cash for 0 units
Would you like to make/accept an offer at this time?
  (Enter 0 for make/1 for accept/2 for no)
```

Figure 4: This is the general screen that comes after every refresh. For this run of the program 3 buyer and 3 seller offers are allowed to be active at a given time. After printing this screen the program waits for player input.



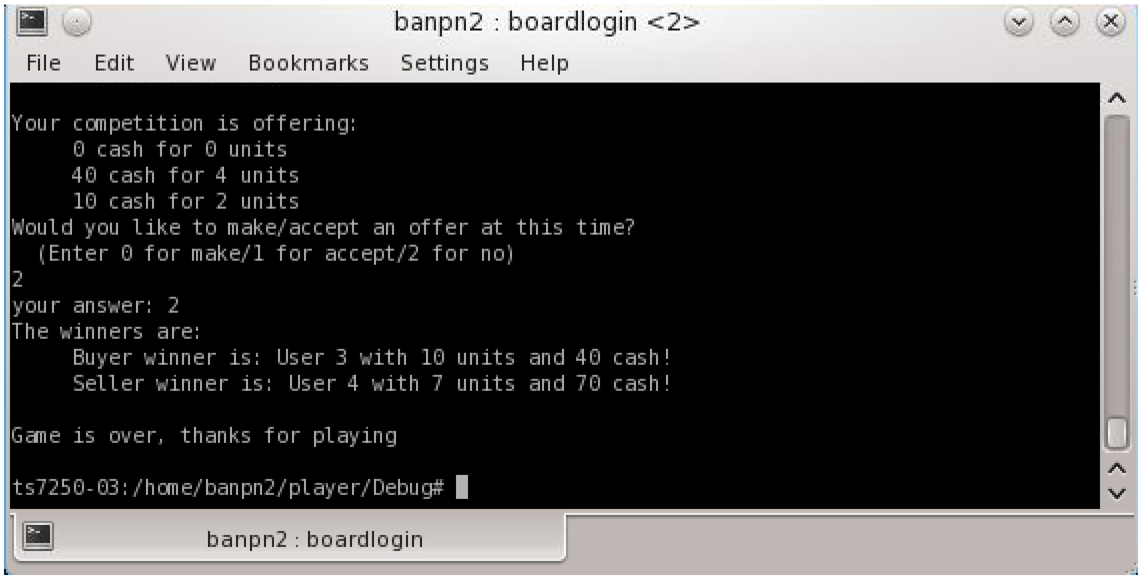
```
How many units do you want to buy?
3
How much money would you spend for that amount of units?
15
Offer submitted
```

Figure 5: Upon choosing the make offer option, the user is brought to this series of questions where they specify what they want to put up, followed by how much they want for it. The last line shows host confirmation of the offer.



```
Offers available for you to accept are:
   1)  70 cash for 13 of your units
   2)  40 cash for 4 of your units
   3)  10 cash for 2 of your units

Your competition is offering:
     0 Units for 0 cash
     0 Units for 0 cash
     0 Units for 0 cash
Would you like to make/accept an offer at this time?
  (Enter 0 for make/1 for accept/2 for no)
1
your answer: 1
Which offer would you like to accept?
1
Offer accepted
```

Figure 6: This figure shows the player screen upon choosing the accept offer option. The player inputs the number cooresponding to the offer they like. The last line is a confirmation that the player did indeed win that offer.



```
Your competition is offering:
     0 cash for 0 units
     40 cash for 4 units
     10 cash for 2 units
Would you like to make/accept an offer at this time?
  (Enter 0 for make/1 for accept/2 for no)
2
your answer: 2
The winners are:
     Buyer winner is: User 3 with 10 units and 40 cash!
     Seller winner is: User 4 with 7 units and 70 cash!

Game is over, thanks for playing

ts7250-03:/home/banpn2/player/Debug#
```

Figure 7: This image shows the final screen of the game where the winners are displayed on the players screens