Fantasy VC

Documentation

Danny Pan | Timothy Lew | Aaron Schrock | Alan Coon | Jeffrey Chen

Table of Contents

High Level Requirements

Login GUI
Options to login as existing user, create a new user, or play as a guest user.
Introduction GUI
Options to start a new multiplayer lobby, join an existing multiplayer lobby, or play single player. Lightweight user profile where a profile picture and an alias can be set. Lobby buttons will be disabled for guest users.
Lobby GUI
A list of all the players in the lobby with a marker to show whether each is ready or not. Users will be able to name their firm at this step and click a ready button to lock in. Once everyone's ready we start bidding.
Company List GUI
All the companies will be listed in order of minimum asking price, and a random player will get to choose which company everyone bids on first. There will be a chat feature visible from this point onward. Companies will be color coded by their size.
Bidding GUI
A timer in the top corner will show how much time is left for players to bid on the currently displayed company. The company's name, logo, description, previous year's growth, equity offered, and minimum asking price will be listed. Each player will be able to bid as many times as they would like, as long as their bid is higher than the running current bid. Each successful bid will add three seconds to the countdown timer. The current bid and bidder's name will be displayed for entertainment purposes.
Time Lapse GUI
A graph with meaningless values for the players to enjoy. A tray will fill up gradually with events that affect prices of companies to simulate reality. The portion won't last more than ten or so seconds. The current year and quarter will be displayed.
Intermission GUI or Quarterly GUI
You see your overall earnings, and you have options to initiate a bidding round on a free agent company, sell one of your current companies (for 50% of your portion's worth), or initiate a trade. If someone tries to initiate a bidding round, each player can opt in or out. The bidding round would reuse the Bidding GUI previously discussed. Intermission ends when each player hits a ready button on his or her screen. We switch back between this GUI and the Time Lapse GUI until we have hit our maximum number of quarters (20).

Final Report GUI
Each player's liquid assets and portfolio evaluation will be displayed, and the winner will be announced by who has the greatest sum! If a player clicks okay he or she will be sent back to the Introduction GUI. The lobby that had just ended will be removed.

Technical Requirements

New additions to our high level ideas are included in **green**. Deleted text from the high level ideas document will be included in **red**. Unchanged text will be **black**.

Expected debugging time is included in every time estimate given. **Since we expect to be working with a front-end language which we have very little experience in, the hours to design GUIs will reflect the learning curve we will encounter.**

Hrs.	**API & Game Logic**	**Implementation Details**
6	Parse everything out of the database into usable objects.	Create a Java API that enables our web based client to communicate with our database and retrieve information.
4	Add new users and update a player's highest score.	Use Java to communicate with the server and add the new user and new information
8	Create functions to maintain the game state.	Use Java to run the actual game logic itself, calculating and determining the actual progression of the game.
4	Give the client side functionality to retrieve the information from the actual game logic.	Use JavaScript to create functions for the web client to call during gameplay.
6	Object oriented design and implementation for the game.	Design all the objects that will be necessary to run the game logic. This will range from User objects to Company objects. These will be written in Java.
Hrs.	**Backend**	**Implementation Details**
4	Build databases to save each user's information and the data for the companies.	We will use MySQL to manage our databases. We must create a table with 50 trendlines and over 20 data points (company trends). We must also create a table with all of the companies, their descriptions, and their icons (these actual companies will be randomly paired with the stats we create to ensure random gameplay). Lastly, we must create a table to store all of the users, their passwords, their icons, their total scores, and their highest scores, etc.

7	Create server thread to execute Ajax queries.	Create a server thread class that receives web client messages and maps them out to required functions. Then, the thread should push the information back out to all clients.
8	Synchronization of clients using a server thread.	Create a server thread and instantiate a new client thread for each person in the session pushing each thread to a list of threads. The server thread will simply forward every incoming message from all of the clients out to all of the clients.
6	Create a client thread object to execute functions from the web client and push to the master server thread.	The client thread will be an object that houses all the functions that our front end server will call. This serves as a way to execute Ajax queries from the client side and to synchronize the game by communicating with the master server thread object.
5	Establish a server.	This server will host the database as well as all of the actual game logic. It will interface with the client (browser), where it will relay any relevant information.
Hrs.	**Login GUI**	**Implementation Details**
5	Create options to login as an existing user, create a new user, or play as a guest user.	Provide text fields for the user to enter a username and a password, and create three buttons: one to login as an existing user, one to create a new user with the credentials from the text fields, and one to play as a guest user. We will also use Facebook's login API to implement this portion, so people can sign in using their Facebook accounts.
Hrs.	**Introduction GUI**	**Implementation Details**
6	Include options to start a new multiplayer lobby, join an existing multiplayer lobby, or play single player. Lobby buttons will be disabled for guest users.	These options are displayed as rectangular buttons across the screen. If a user clicks on an existing lobby they will go to the lobby GUI.

2	When single player is pressed go straight to the betting menu for single players.	Go to the single player GUI that was used for guest mode.
6	If the player clicks on 'create new lobby', go to a new GUI with the option to set the amount of players and to invite people to play..	Identify when new lobby button is clicked and switch the view to a new lobby GUI. Add a slider for the number of players and a create lobby button. When the 'create lobby button' is pressed, go to Lobby GUI.
6	When a new lobby is created, display it to the introduction GUI.	If a user creates a new lobby, an open lobby will be displayed on the introduction GUI for other users to see.
5	Create a lightweight user profile where a profile picture and an alias can be set.	A modify profile button can be clicked which will provide a small pop up window that allows the user to upload a PNG or JPG file as an avatar, and a text field to set an alias. There will be an apply button and a cancel button on the profile pop up.
Hrs.	**Lobby GUI**	**Implementation Details**
4	Create a list of all the players in the lobby with a marker to show whether each is ready or not. Users will be able to name their firm at this step and click a ready button to lock in. Once everyone's ready we start bidding.	We can store player objects in an ArrayList for easy access. Each player will have a data member of whether he or she is ready, which will be updated when he or she clicks the ready button. A text field will be displayed on the screen that allows the player to enter the name for his or her firm that will be displayed for the remainder of the game. It will be saved as a data member of the player object. The ready button cannot be clicked until there is text in the firm name field. This can be done using a DocumentListener, or an equivalent in another language. Each player's name, firm name, and ready status will be displayed in the GUI. Since the firm name is empty at first, the firm name labels will have empty Strings initially. We will make a small yellow dot icon that will be displayed for a player that has not clicked the ready button yet, and a small green dot icon that will be displayed

		once the player hits the ready button.
Hrs.	**Company List GUI**	**Implementation Details**
5	All the companies will be listed in order of minimum asking price, and a random player will get to choose which company everyone bids on first. There will be a chat feature visible from this point onward. Companies will be color coded by their size.	Before this screen loads, this is the point where we will grab information from the databases. We will grab both trendlines and company data, then randomly couple a trendline with a company. We will then accumulate each company into some sort of list data structure, and then we will iterate through the list to add each company to the GUI. Since we want the companies to be sorted by their sizes, we will code a comparator for company objects that can be used to sort the companies by their minimum asking price. Since we want each company representation to be clickable, we can present each as a button, which leads all of the players to the bidding round for that company when clicked. The implementation of the chat will have a multi-threaded design that incorporates a server thread (on the host's machine), and every person is represented by a client thread. The chat will be in the southern border of the GUI.
Hrs.	**Bidding GUI**	**Implementation Details**
8	A timer in the top corner will show how much time is left for players to bid on the currently displayed company. The company's name, logo, description, previous year's growth, equity offered, and minimum asking price will be listed. Each player will be able to bid as many times as they would like, as long as their bid is higher than the running current bid. Each successful bid will add three seconds to the countdown timer. The current highest bid and bidder's name will be displayed for entertainment purposes.	Add a running timer as a separate thread to show how much time is left for players to bid on the currently displayed company. The timer will be a label on the GUI. When the timer hits zero, we want to end the bidding on the current company, give the team with the highest bid possession of this company, and immediately bring up the bidding screen for the next company. To place a bid, there will be a text field with an increment button, a decrement button, and a submit button. The text field will automatically be populated with the current highest bid (so it will change as new bids are placed). The arrows can be used to add $1000 at a time, but the text

		field can also be directly typed into by the player for exact figures. The bid button will be disabled unless the player's bid is strictly greater than the current highest bid. The chat box will be on the GUI at the southern border of the frame.
7	Implement our bidding logic and functionality, and update the bid GUI appropriately.	Each player will be able to bid as many times as they would like, as long as their bid is higher than the current highest bid. Each successful bid will add three seconds to the countdown timer. The current bid and bidder's name will be displayed for entertainment purposes.
Hrs.	**Time Lapse GUI**	**Implementation Details**
5	A graph with meaningless values for the players to enjoy. A tray will fill up gradually with events that affect prices of the companies to simulate reality. The portion won't last more than ten or so seconds. The current year and quarter will be displayed.	The graphic will resemble a financial chart and have meaningless movement to give the player a feeling that calculations are being made and time is progressing. At the beginning of this screen, we will run an algorithm that will randomly decide which companies will sustain an event. An event can randomly be beneficial or bad for the investing company. Then a blurb will be randomly selected from a list of news headlines, and the company's name will be ad-libbed into the headline. An example is "[company]'s CEO caught in insider trading circle!" for a bad event. The event will then impact the company's evaluation at the quarterly report, on top of their pre-planned trend. Since we plan on making this a web-app, we will need to pull these blurbs from a database. A text area below the graphic will store all of the blurbs that get generated, all players will see the same text area, so players will know if another player's company has lost or increased in value. To keep track of the current year and quarter we will just have a pair of variables, one for the year and one for the financial quarter. The financial quarter will increment every cycle until it hits quarter 5, at which point it will reset to 1 and

		increment the year.
Hrs.	**Quarterly GUI**	**Implementation Details**
6	(We decided to remove Intermission GUI from the title of this portion for simplification.) You see your overall earnings, and you have options to initiate a bidding round on a free agent company, sell one of your current companies (for 50% of your portion's worth), or initiate a trade. If someone tries to initiate a bidding round, each player can opt in or out. The bidding round would reuse the Bidding GUI previously discussed. Intermission ends when each player hits a ready button on his or her screen. We switch back between this GUI and the Time Lapse GUI until we have hit our maximum number of quarters (20).	The GUI is divided into three columns. In the first column, from top to bottom, we first have a timer that shows how much time players have left to make their choices before the next Time Lapse period will automatically begin. We will be generous with time, but this is in place to keep the game moving if one player attempts to filibuster or stall the game. Beneath that will be a panel with each player's ready status, accompanied by one button labeled 'Ready'. If every player readies up before the timer goes down, we skip the rest of the timer and jump to the next Time Lapse period. Finally at the bottom of the left column is the market action tray. If another player proposes a bid on a free agent company, or requests a trade with you, a square with a description of the event will pop up in the tray. Clicking one of the squares that pops into the tray will lead you to a separate GUI. If you click on a bidding square for a free agent, the bid screen will show up, or if you click on a trade proposal someone else sent you, the game will display a popup that gives an overview of the trade and has an 'Accept' and 'Reject' button. The center column will contain three tables with vertical scroll bars. The top table is all of your companies, their current evaluations, and their percent change from last quarter. The middle table is similar but contains other players' companies. The bottom table is similar but only contains free agent companies. Each item in each table is hoverable, and will display a dropdown menu. Any of your companies will have the three options of 'sell' (your company will become a free agent and you will collect 50% of the value of your share in the company), 'stats' (view company information and statistics), and 'trade to' (which gives you another dropdown list of the other players in the game - selecting one of the players will open a GUI that allows you to send a trade offer). Any other

		player's companies will have the 'trade for' (opens the Trading GUI), and 'stats' options. Any free agent will have the 'initiate bid' option, which will push a bidding square to the market action tray. The bid will then commence in fifteen seconds or when everyone has clicked on the square. It is possible for a player to sit out or miss the bidding round if he or she ignores the square. We will reuse the Bidding GUI from earlier in the game. In the right column, the top half will have a list of the player's current standings, and the bottom half will have a news ticker. The news ticker will be scrollable, and will contain all of the news from the previous time lapse segment of the game, but as players act in the current state of the game their actions will be reported to the news ticker. For example if Player 1 sells his shares in Company A, it will be pushed to the news ticker for everyone to see. At the very south border of the screen, spanning across all three columns, is the chat-box. The current year and quarter will be unobtrusively posted in the very top right corner of the screen.
Hrs.	**Final Report GUI**	**Implementation Details**
6	Each player's liquid assets and portfolio evaluation will be displayed, and the winner will be announced by who has the greatest sum!	We begin the game in quarter 4 of year 2016, so the game will end in quarter 4 of year 2021. When we hit that date, for each player, we will sum up the evaluations of each of his or her companies, then add the player's cash data member to that number. The companies can be kept in an HashSet data member of the player object for easy removal, lookup, and insertion. A comparator can be used to sort the players in their ArrayList based on their total wealth, and the top player's firm name and username will be displayed in the center of the GUI, declared as the winner! The second, third, and fourth place contestants will be labeled below.
2	If a player clicks okay he or she will be sent back to the	A button at the southern border of the GUI will take each player back to the Introduction GUI

	Introduction GUI. When the 'Okay' button is clicked on this GUI, send the user back to the Introduction GUI.	independently (i.e. player 1 clicking the "return to introduction" button will not change player 2's screen).
2	The lobby that had just ended will be removed.	Remove the lobby for the current game from the set containing all active lobbies and update the Lobby GUI so that it reflects this change.
Hrs.	**Others**	**Implementation Details**
8	Create overall website design and theme.	Create a unified CSS for the entire website so that there is a consistent look and feel to every page in terms of design. Make sure that the CSS used is compatible across multiple browsers.
5	Design logos and descriptions for all of the possible companies.	The available companies in the game need to be represented in some form. Each company will have its own logo as well as a description about them.
4	Collect data points that will be the basis for trends companies can follow.	Aggregate and quantify past performances of companies to put in the database. This data can then be used to simulate the performance of in game companies.
5	Chat at the bottom of the screen after the game has begun	We will have a multi-threaded design that incorporates a server thread (on the host's machine) and every person is represented by a client thread. The chat will in the southern border of the GUI.

Total hours: **155 estimated**

Detailed Design Document

Hardware Requirements:

- Windows: Windows 10, Windows 8 (Desktop), Windows 7, Windows Vista SP2
- RAM: 128MB
- Disk Space: Recommended 512MB
- Processor: Minimum Pentium 2 266MHz processor
- Mac: OS X: Intel-based Mac running Mac OS X 10.8.3+, 10.9+
- Linux: Oracle Linux 5.5+ Oracle Linux 6.x (32-bit), 6.x (64-bit) Oracle Linux 7.x (64-bit) Red Hat Enterprise Linux 5.5+ (32-bit), 6.x (64-bit) Ubuntu Linux 12.04 LTS, 13.x

Software Requirements:

- Internet Browser: Mozilla Firefox or Google Chrome
- Internet Connection

Database Schema:

Users
• userID (int, *primary key)* • Username (varchar[25]) • Password (varchar[25]) • Profile Picture (varchar[100]) • Email (varchar[100]) • Bio/Team Description (varchar[140]) • gamesPlayed (int) • totalMoney (int)

Users - Table that holds all of the relevant user information.

Companies
• companyID (int, *primary key)* • tierValue (int) • Icon (varchar[100]) • Name (varchar[50]) • Description (varchar[140]) • askingPrice (int) • startingValue(int)

Companies - Table that holds all of the user facing companies. This will just be what the user sees, however each company will get matched with a trend for the actual gameplay.

Class Diagram:

Lobby
- List<Game> : availableGames - LobbyClientListener + login(String username, String password) : User + createGame(User) : Game + inviteFriend(User) : void + joinGame(Game, User) : void

Lobby - Hosts information of all available games that the user can currently join. Also has options to create a new game and invite players in.

Game
- List<User> : users - List<Company> : companies - Int : currentQuarter + addUser(User) : void + getWinner() : int

Game - Maintains all of the current players inside of the instance of the game. Has a list of of companies that are available. Also keeps track of the game state.

User
- List<Company> : ownedCompanies - String : Name - String : Image - String : Description - Int : Current amount of money + getName() : String + getImage() : String + getDescription() : String + getMoney() : int + setMoney(int) : void + setDescription(String) : void + setImage(String) : void + setName(String) : void + updateCompanyValuation(int) : void + addCompany(Company) : void + deleteCompany(Company) : void

User - Created whenever a new player joins a game. Maintains a list of current companies owned and how the user is doing in the game overall.

Company
- String : Image - String : Name - String : Description - Int : startingPrice - Int : askingPrice - Int : currentWorth - Int : tierLevel + getName() : Sting + setName(String) : void + getImage() : String + setImage(String) : void + getDescription() : String + setDescription(String) : void + getAskingPrice() : int + updateAskingPrice(int) : void + getStartingPrice() : int + setStartingPrice(int) : void + getTier() : int + setTier(int) : void + getCurrentWorth() : int + updateCurrentWorth(int) : void + performRandomEvent() : void

Company - Each company is matched with one of the trends from the database. It will also maintain its current worth. The company either belongs to the Game or a specific User.

Server
- ServerSocket : ss - ServerLobbyListener : sll - Server() - listenForConnections() : void + sendLobby() : void

Server - Responsible for creating ServerLobbyListener and sends the lobby through.

ServerLobbyListener
- ServerSocket : ss - List<ServerGameListener> : serverGameListeners - List<ServerLobbyCommunicator> : serverLobbyCommunicators - removeServerLobbyCommunicator() : void - run() : void - sendLobby() : void

- Lobby : lobby0

ServerLobbyListener - Listens for incoming connections on the Socket. Once a connection is made, it will create a new socket and pass it to the ServerLobbyCommunicator. It will also create a ServerGameListener if a new game is created. The SeverLobbyListener will maintain all the ServerGameListeners.

ServerLobbyCommunicator
- ServerSocket : ss - ObjectInputStream : ois - ObjectOutputStream : oos - run() : void - ServerGameListener : sgl

ServerLobbyCommunicator - Holds a connection with a single client. Responsible for sending the current lobby state to the Client.

ServerGameListener
- ServerSocket : ss - List<ServerGameCommunicator> : serverGameCommunicators - Sever - removeServerGameCommunicator() - run() : void - sendGame(Game) : void - Game : fantasyVC

ServerGameListener - Receives a Socket from the ServerLobbyListener. Once a connection is made, it will pass it to the ServerGameCommunicator. It will maintain all the ServerGameCommunicator.

ServerGameCommunicator
- ServerSocket : ss - ObjectInputStream : ois - ObjectOutputStream : oos - run() : void - ServerGameListener : sgl

ServerGameCommunicator - Holds a connection with a single client. Responsible for sending the current game state to the Client.

LobbyClientListener
- ServerSocket : ss

- ObjectInputStream : ois
- ObjectOutputStream : oos
- GameClientListener : gcl
- run() : void
- sendUpdate(Object) : void
- Lobby : l1
- initializeVariables() : boolean
- LobbyClientListener(Socket)

LobbyClientListener - Listens to the Server that the client is connected to waitings for updates. Once information is received, the current Lobby GUI will be updated accordingly. Sends out information to the server when a new game is created and added to the lobby.

GameClientListener

- ServerSocket : ss
- ObjectInputStream : ois
- ObjectOutputStream : oos
- Game : vcFantasy
- run() : void
- sendGame(Object) : void
- initializeVariables() : boolean
- GameClientListener(Socket)

GameClientListener - listens for Ajax commands from the web client and pushes the associated message to the ServerClientListener. The ServerClientListener should execute the commands to update the gamestate and then the game state is pushed back to client from the server. The GameClientListener will continuously poll for inputs from the webclient.

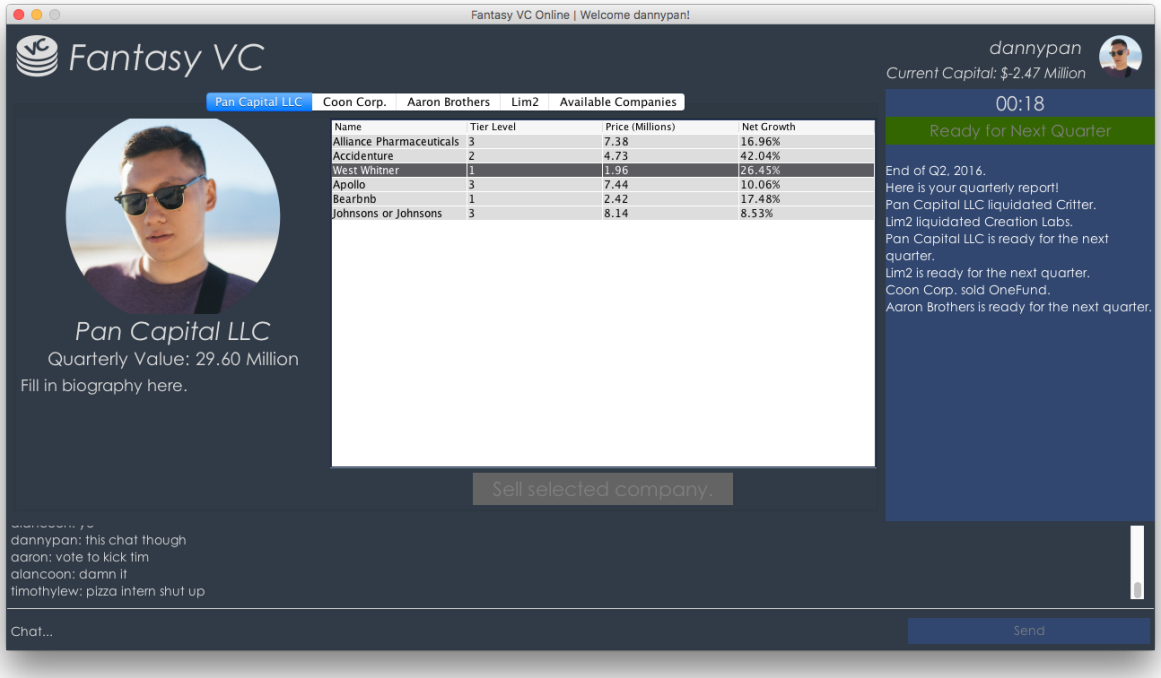
Quarterly GUI:

The *Quarterly GUI* is where players are allowed to see their overall earnings, the status of their and other players' companies, bid on free agents, make trades with other players, and sell companies from their own portfolio. The bottom sixth of the screen is the chatbox. For the upper portion of the screen we half two partitions. The right hand quarter, in the top third is a panel holding a timer which shows the amount of time left in the *Quarterly GUI* view before the next *Timelapse*, a ready button, and a list of all of the players with their ready statuses. If all of the players ready before the timer has reached zero, then the game will progress to the next *Timelapse* anyway. Below that is the player's *Feed*, a list of elements that consists of abbreviated bid listings and trade offers.

The majority of the screen will be the *Grid View Pane*, which is a tabbed pane, each user has a tab, and the last tab is the free agents tab. Clicking on a tab of a player will show you his or her

avatar, username, firm name, and a portfolio table. You can click on available companies to buy and sell in real time between quarters.

We switch back between this GUI and the *Time Lapse GUI* until we have hit our maximum number of quarters.

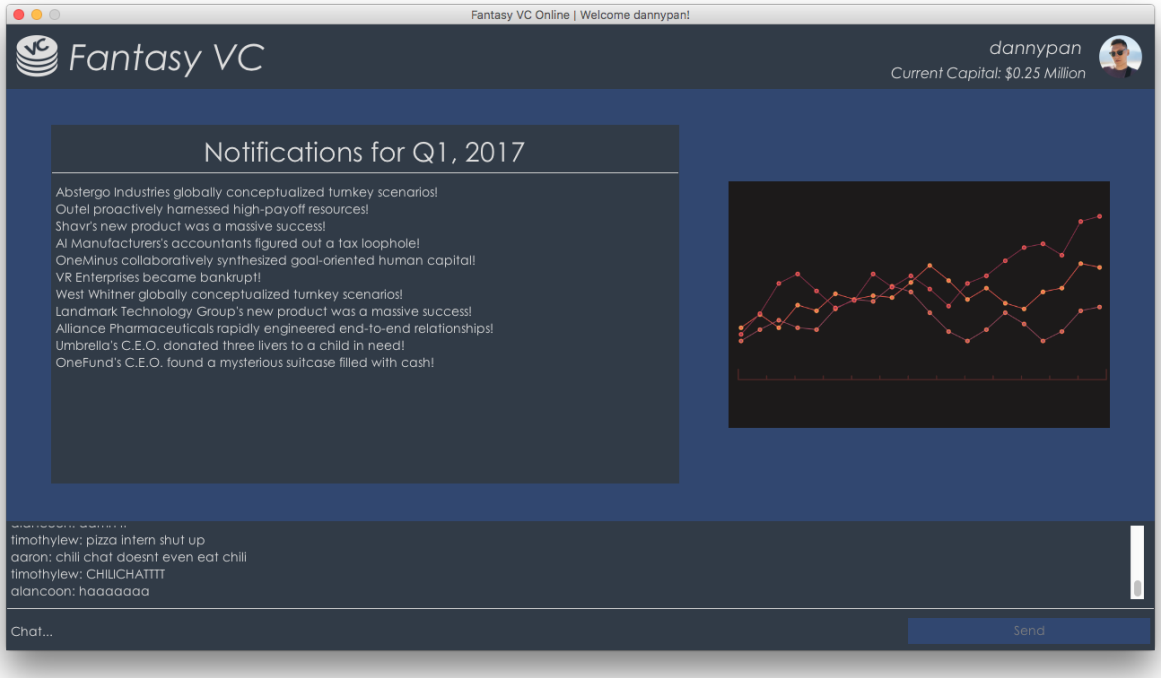


Timelapse GUI

The *Timelapse GUI* will be an intermediate GUI that displays action events and notification during gameplay. The goal is to simulate passing of time and to build anticipation for the quarterly reports. This interface is mainly utilized for presentation of information and will have minimal user interaction. We will have a text field at the bottom of the page for the chat box. A panel on the right side will hold a graphic that will show a graph animation. A text area on the left will be a tray that will have random event messages pushed to it, which we will decide using an algorithm we have outlined above.

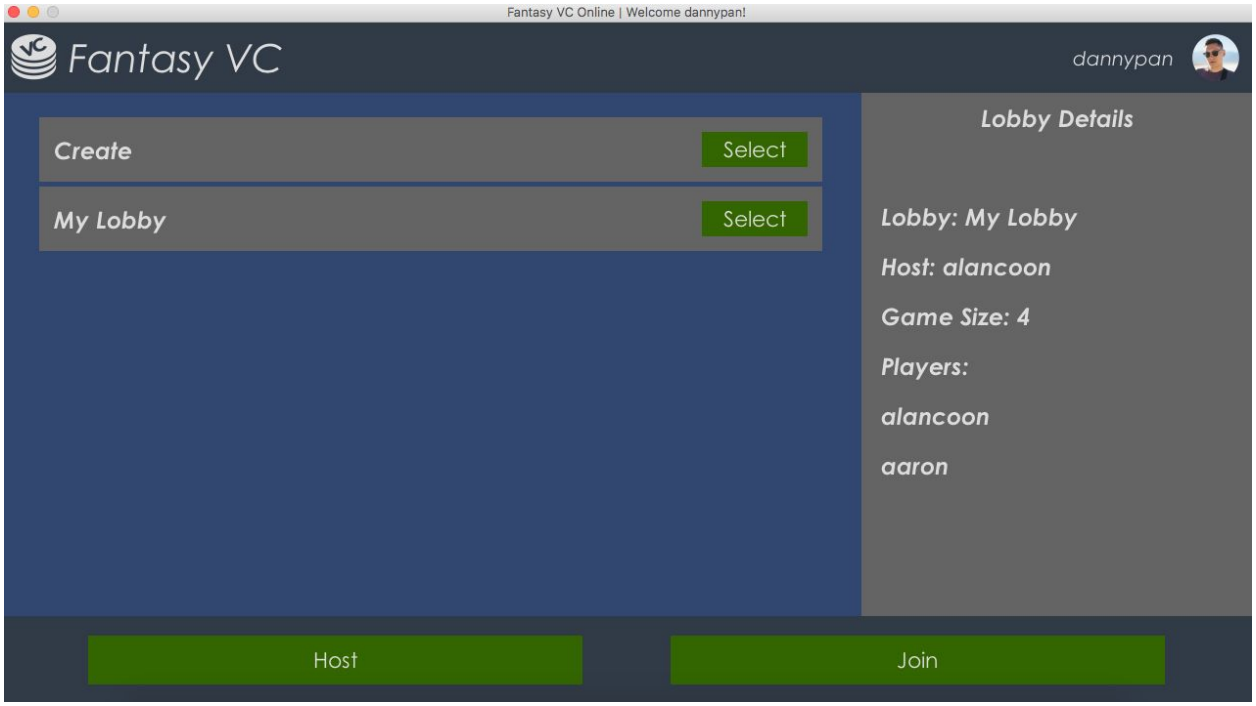
Components
• Text Field on bottom to display chat messages. • Text Field on bottom to enter a new chat message. • Button next to Text Field on bottom to submit new chat message.

- Panel on left side above Text Fields to display notifications.
- Label on the top left side for the notification banner
- Panel on right side to hold a graphic that will show a graph animation.
- Label on the top right side for the notification banner



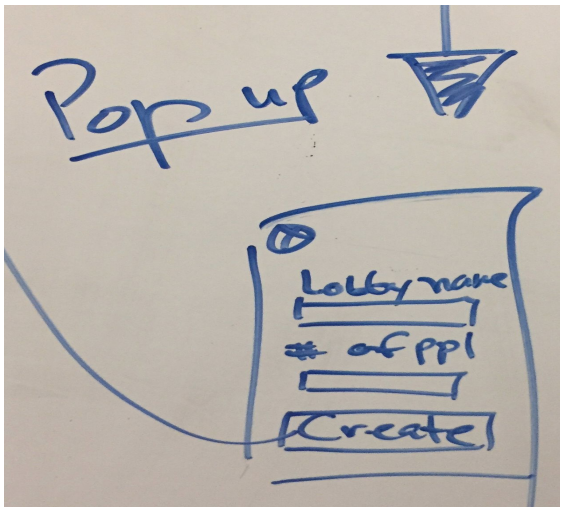
Introduction GUI

Directly after the login screen, the user will enter the *Introduction GUI* where there is a list of all the current lobbies in the left panel. The list will update in realtime as lobbies are created and closed. Lobby details of selected lobby are shown on the right panel. Options to host, join, and play single player are listed as buttons on the bottom. If the user is a guest, this introduction window will be skipped and the client automatically goes to the *Lobby GUI*. The create lobby button will spawn a modal window that allows the user to enter lobby settings.


Fantasy VC Online | Welcome dannypan!
Fantasy VC
dannypan
Create
Select
My Lobby
Select
Lobby Details
Lobby: My Lobby
Host: alancoon
Game Size: 4
Players:
alancoon
aaron
Host
Join

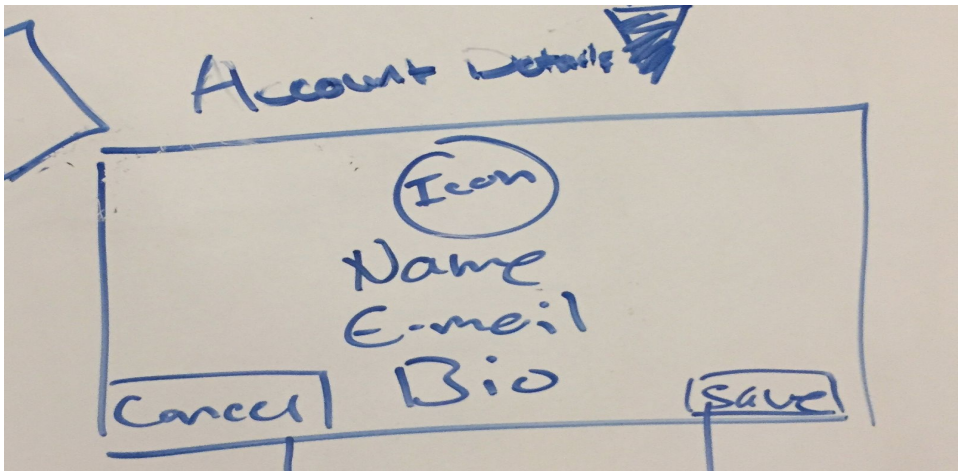
Host Game Pop-up

This modal window will pop up when the user selects create a lobby from the intro GUI. Once the user enters all the information, they will click the create button on the modal window to close the modal window and transition to the lobby GUI. This will create a new lobby instance from the user's client and it will push the update to the server.



Account GUI

The user's avatar will be displayed at the top as an image, three editable text fields will be listed below, each one labeled with their purpose (name, email, and bio). There will be a cancel button on bottom left corner to return to the previous screen without modifying your account information. Save button on bottom right corner to overwrite your profile with changes made to the individual text fields. For example, if I only write a new email address in the email text field then press the save button, only my email address will be updated.



Login GUI

Label for game name at the top that reads: “Fantasy Venture Capital!”. Editable text field under the label for the player to enter his or her username, the text field will have suggestion “ghost text” that reads “Username”. Another editable text field under the username field for password entry will have ghost text that reads “Password”. In the southern panel of the screen will be three buttons. The login and create account buttons are side by side, and the guest button is below them. The login button only activates when both the username and the password text fields have non-whitespace text in them.

Clicking the login button will pull the username and password strings from the text fields and reference our database that holds all of our users. If a user with the specified username exists in the database, we will hash the inputted password and check if that user’s stored password hash is a match. If it is, we progress into the *Introduction GUI*. If it isn’t, we display a label in red text telling the user that his or her inputted password was incorrect, and we erase whatever text was inputted into the password text field.

The create account button will operate similarly, and will only activate if both the username and password fields have non-whitespace text. We will query the database to see if a user exists with the given username. If the username exists, we display a label in red text prompting the user to pick a different name, and both the username and password fields will be cleared. If there is no such user with the specified username, we will execute an update to the database, inserting the user as another entry in our table of users. We will use a hash function to store the user’s password. The game will then progress the player to the *Introduction GUI*.

The guest button is active at all times (not dependant on the text fields). It will bring the player to the *Lobby GUI*, where the player can specify a firm name, then click the ready button to progress to the game. However, since the player does not have an account, he or she will not be able to play multiplayer and the invite feature will be disabled.


Fantasy Venture Capital
VC
Fantasy VC
Password
Login
Create Account
Continue as Guest

Lobby GUI

List of all of the users in the left panel. Right panel lets you set firm name, invite people, ready up (by clicking ready button), and leave the lobby (by pressing the leave button). Chat box shown at the bottom for the first time.

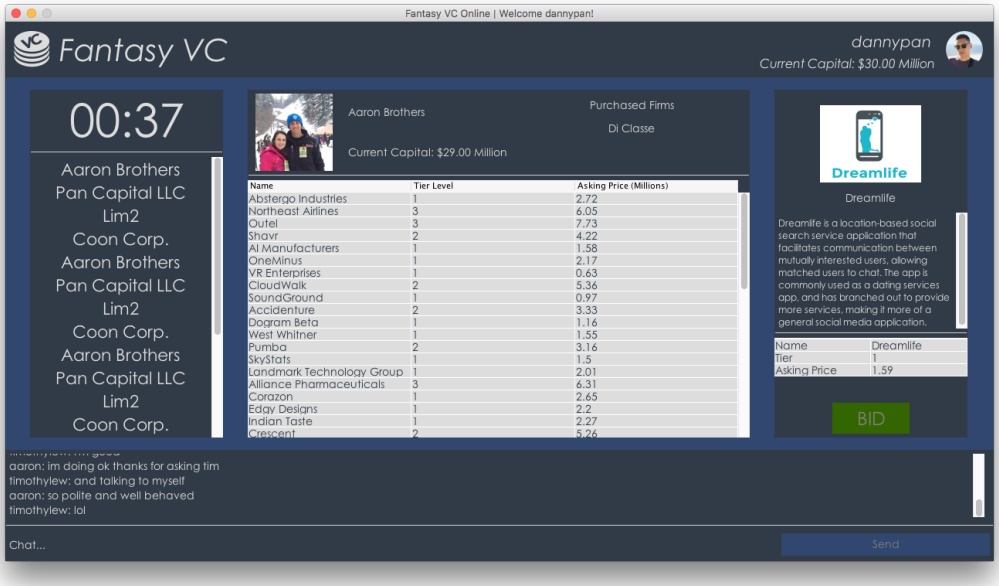
Components
• Label for each available slot of the game   ○ If a player joins the label switches from “empty” to their team name • Check mark image appears next to a user when they have clicked ready • TextArea at the bottom that contains a chat box • User information on the panel on the right where the user can enter their firm name, click ready, enter a username to invite someone, or leave • Firm Name text field • Invite User text field • Ready button that is enabled when a firm name has been entered • Invite button that is enabled when a username is entered into the invite user text field • Leave button that takes the user back to the intro GUI

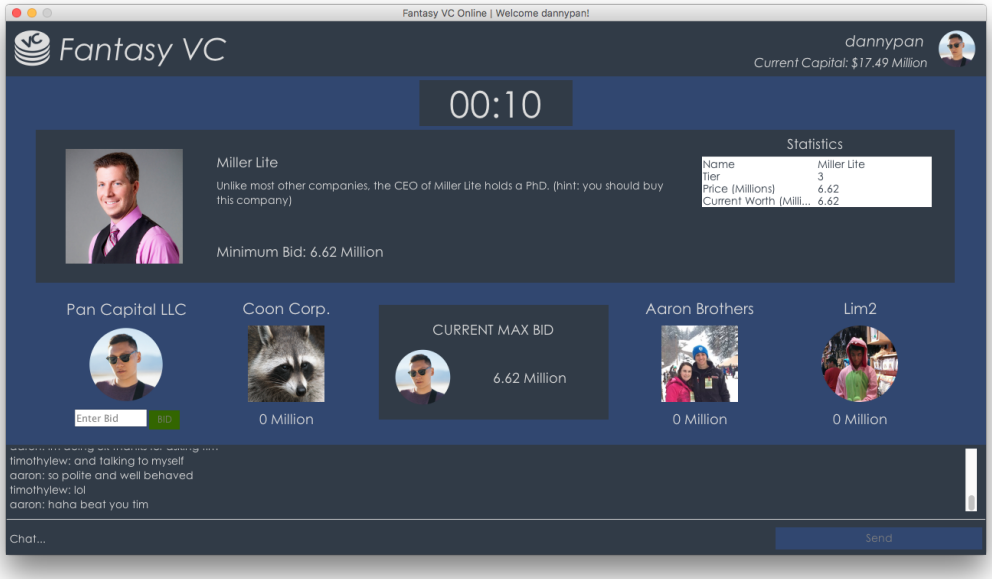
Auction GUI

The *Auction GUI* will use a card layout design that switches between the *Company List Pane* and the *Interactive Auction Pane*. The chat box will remain at the bottom of the screen.

The *Company List Pane* will have a timer on the upper left panel and a queue of players underneath it. We will populate the queue by pushing each player clockwise for five cycles, so each player gets to pick a company five times. The timer is to keep the game moving if a player is taking too much time to decide, if the timer reaches zero the game will automatically move to the *Interactive Auction Pane* with the highest value company still on the list. The company list in the central pane will be a table that contains small previews of each company – logo, name, size, and minimum bidding price. If the user clicks on a company, the right panel will update the descriptive statistics to reflect the data of the selected company. The logo and description will be shown with a table of important statistics and facts – projected growth, current valuation, offered equity, company type, company size, asking investment. At the bottom of the panel is the bid button that takes all of the players to the *Interactive Auction Pane* to bid on the selected company. The bid button is only enabled for the player who is at the top of the queue. We then pop the queue after the bid button is pressed, so the next time we finish a bid, the next clockwise player will be allowed to pick the team.

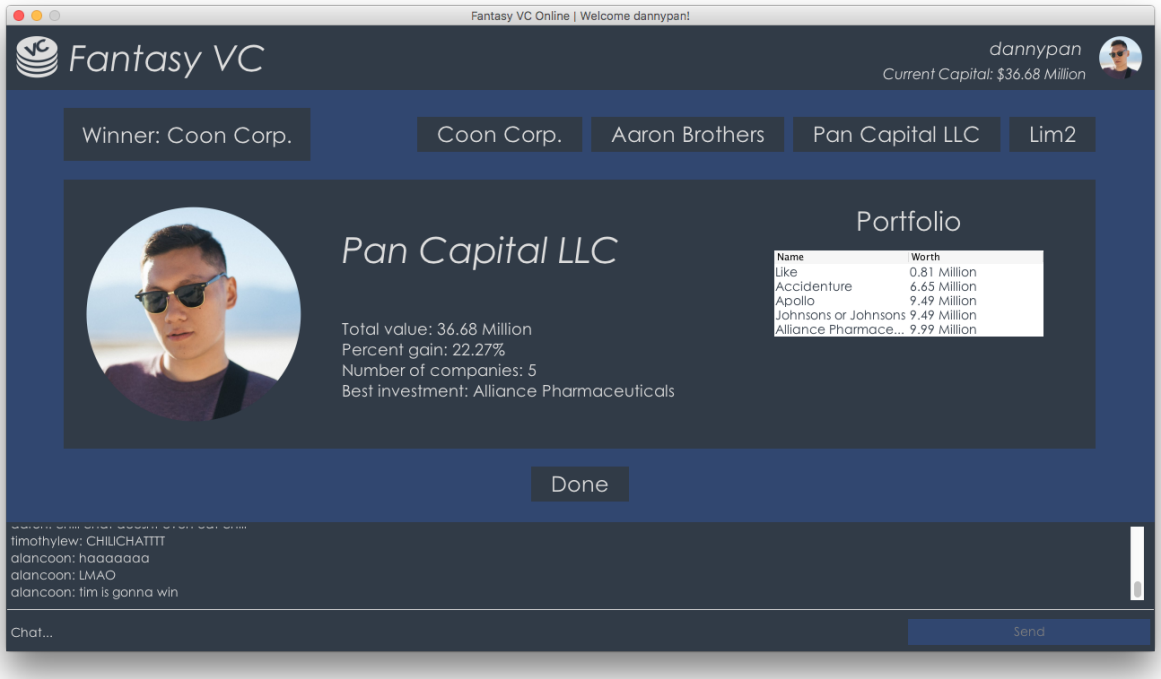
The *Interactive Auction Pane* appears once the current company picker has selected a company from the company list to bid upon. The timer in the top left corner continually counts down, while the company logo, name, description, minimum bid price, equity offered, and some statistics are shown so players can make an informed decision on how much to bid. An updating label will show the current top bidder and his or her bid. A text field and a submit button will allow the user to place his or her own bet. A label will show how much money is left in your own bank account, so you know the amount of money left you can spend on bidding. Clicking the submit button will send the bet to the server, which will determine whether the entered bet is greater than the current highest bet. If it is greater than the current highest bet, the game will update the highest bet labels and add a second or two to the timer, so players cannot place bets at the very last moment. When the timer reaches zero, the highest bidder will have the company added to his or her portfolio, and their bid will be deducted from his or her cash on hand. Then we return to the *Company List Pane*, unless the queue is empty, in which case we will jump to the *Timelapse GUI*.


Fantasy VC Online | Welcome dannypan!
Fantasy VC
dannypan
Current Capital: $30.00 Million
00:37
Aaron Brothers
Pan Capital LLC
Lim2
Coon Corp.
Aaron Brothers
Pan Capital LLC
Lim2
Coon Corp.
Aaron Brothers
Pan Capital LLC
Lim2
Coon Corp.
Aaron Brothers
Current Capital: $29.00 Million
Purchased Firms
Di Classe
Name | Tier Level | Asking Price (Millions)
Abstergo Industries | 1 | 2.72
Northeast Airlines | 3 | 6.05
Outel | 3 | 7.73
Shavr | 2 | 4.22
AI Manufacturers | 1 | 1.58
OneMinus | 1 | 2.17
VR Enterprises | 1 | 0.63
CloudWalk | 2 | 5.36
SoundGround | 1 | 0.97
Accidenture | 2 | 3.33
Dogram Beta | 1 | 1.16
West Whitner | 1 | 1.55
Pumba | 2 | 3.16
SkyStats | 1 | 1.5
Landmark Technology Group | 1 | 2.01
Alliance Pharmaceuticals | 3 | 6.31
Corazon | 1 | 2.65
Edgy Designs | 1 | 2.2
Indian Taste | 1 | 2.27
Crescent | 2 | 5.26
Dreamlife
Dreamlife
Dreamlife is a location-based social search service application that facilitates communication between mutually interested users, allowing matched users to chat. The app is commonly used as a dating services app, and has branched out to provide more services, making it more of a general social media application.
Name | Dreamlife
Tier | 1
Asking Price | 1.59
BID
aaron: im doing ok thanks for asking tim
timothylew: and talking to myself
aaron: so polite and well behaved
timothylew: lol
Chat...
Send

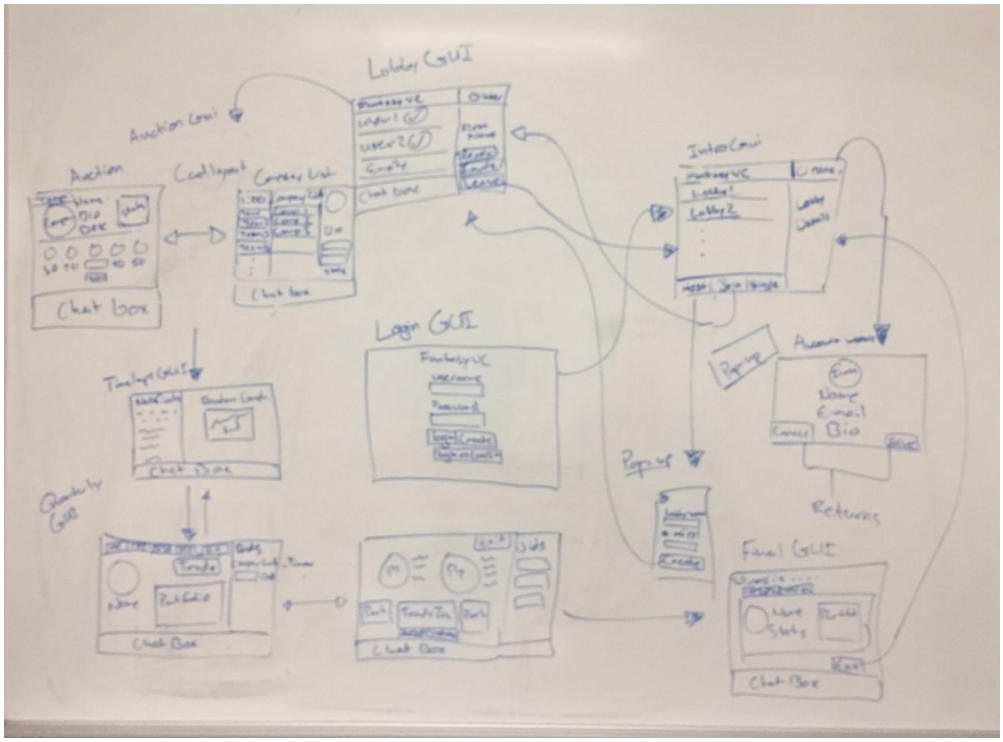

Fantasy VC Online | Welcome dannypan!
Fantasy VC
dannypan
Current Capital: $17.49 Million
00:10
Miller Lite
Unlike most other companies, the CEO of Miller Lite holds a PhD. (hint: you should buy this company)
Minimum Bid: 6.62 Million
Statistics
Name | Miller Lite
Tier | 3
Price (Millions) | 6.62
Current Worth (Milli... | 6.62
Pan Capital LLC
Enter Bid
BID
Coon Corp.
0 Million
CURRENT MAX BID
6.62 Million
Aaron Brothers
0 Million
Lim2
0 Million
timothylew: and talking to myself
aaron: so polite and well behaved
timothylew: lol
aaron: haha beat you tim
Chat...
Send

Final GUI

Top left section will show a label that identifies the winner. Underneath is a tabbed pane, with each tab corresponding to a player, in order of his or her final standings. Clicking on a firm's tab will present the player's firm name, avatar, final portfolio value and accumulated cash, and portfolio as a table. An exit button will individually take players back to the *Introduction GUI*. The chat box at the bottom.



Wireframe

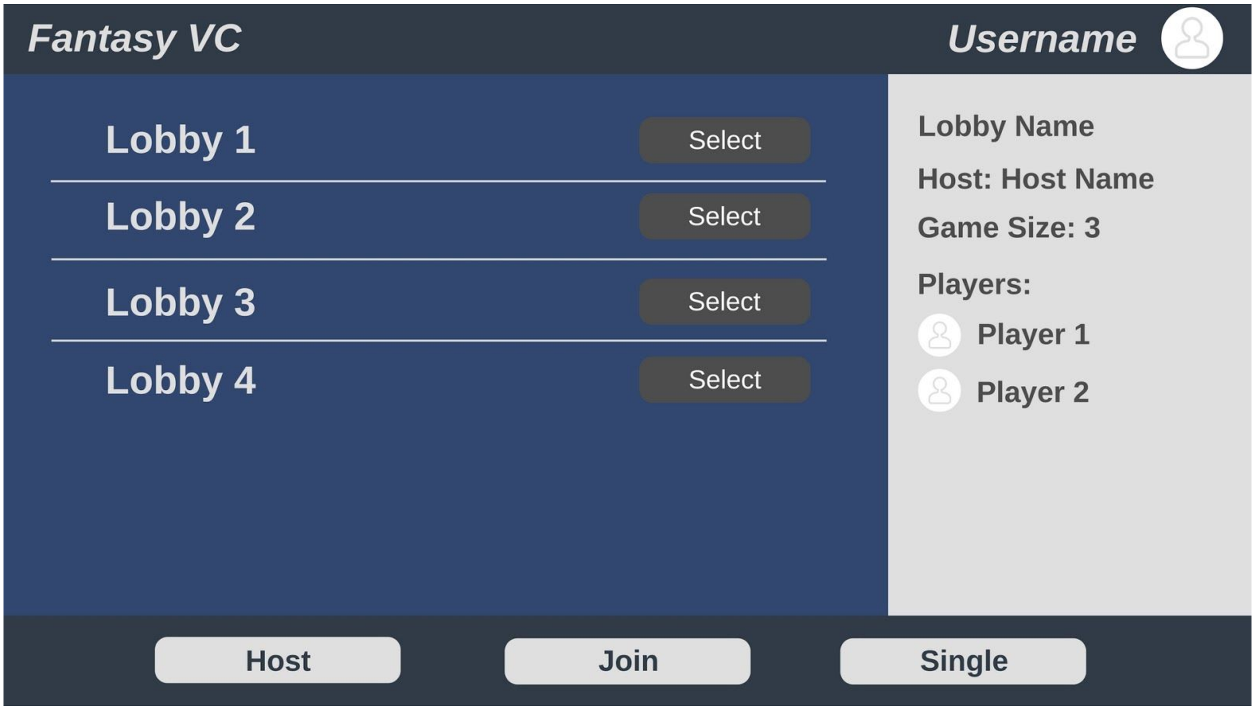

Lobby GUI
Auction GUI
Auction
Chat box
Login GUI
Final GUI
Chat Box

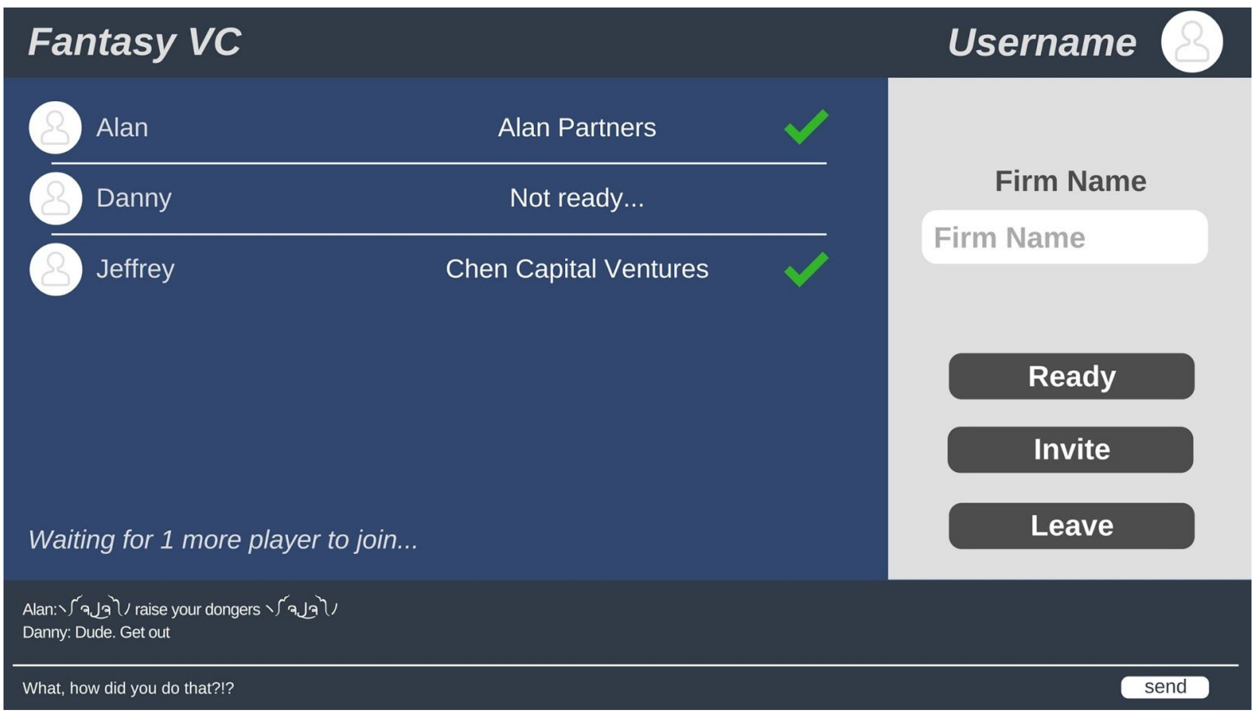
Testing Document

General

Test #	00
Test Description	If a player disconnects or quits during the game, the event should be gracefully handled by returning all players to the *Introduction GUI*. The lobby will be destroyed and no longer visible from the *Introduction GUI*.
Steps To Run Test	1. Start a lobby 2. Fill the lobby and begin the game 3. Have a player leave
Expected Result	A pop up appears on each player's window notifying him or her that a player has left the game, and that the program will be taking him or her back to the *Introduction GUI* now.
Actual Result	A confirmation window pops up, clicking or closing the window will lead the player back to the *Introduction GUI*.

Test #	01
Test Description	If a player disconnects or quits during the lobby stage, the event should be gracefully handled by decrementing the number of players in the lobby, and removing the player's listing from the container of players in the lobby.
Steps To Run Test	1. Start a lobby 2. Have a player join the lobby 3. Have the player ready up 4. Have the same player leave the lobby
Expected Result	Despite the player having readied up, the lobby should revert to a state as if the player had never joined, after said player leaves. Whether or not the player had readied up should not affect whether the other players will be able to begin the game.
Actual Result	The player's listing was removed from the list of players, and whether or not the player had readied up does not affect the others' ability to begin the game.


Fantasy VC
Username
Lobby 1
Select
Lobby 2
Select
Lobby 3
Select
Lobby 4
Select
Lobby Name
Host: Host Name
Game Size: 3
Players:
Player 1
Player 2
Host
Join
Single


Fantasy VC
Username
Alan
Alan Partners
Danny
Not ready...
Jeffrey
Chen Capital Ventures
Waiting for 1 more player to join...
Firm Name
Firm Name
Ready
Invite
Leave
Alan: ヽ༼ຈل͜ຈ༽ﾉ raise your dongers ヽ༼ຈل͜ຈ༽ﾉ
Danny: Dude. Get out
What, how did you do that?!?
send

Login GUI

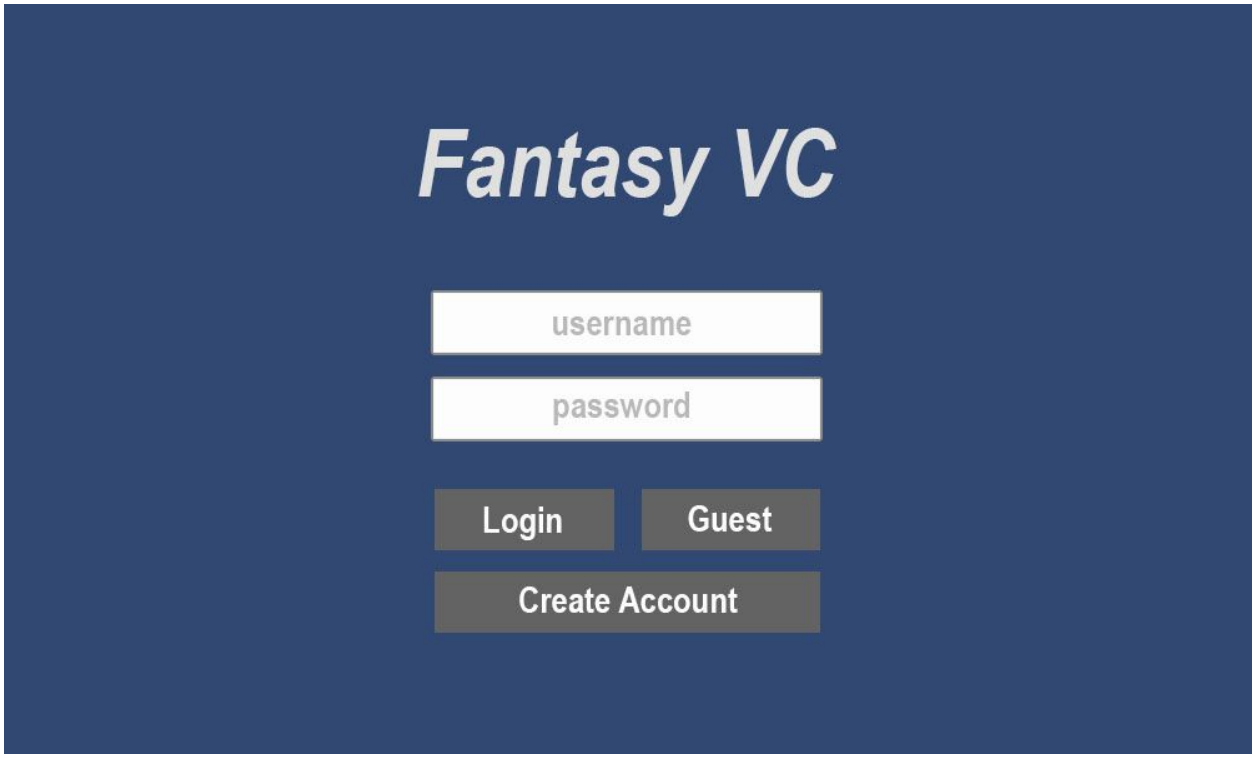
Test #	02
Test Description	Player should only be able to log in if both the username field and the password field have text in them. Whitespace usernames are not allowed, however whitespace passwords are allowed.
Steps To Run Test	4. Run the Main 5. Write text (non-whitespace) in the username field 6. Write text in the password field 7. Attempt to log in
Expected Result	Log in button should remain disabled until both fields have text in them.
Actual Result	The login button and the create account button are enabled once the username field contains non-whitespace characters and the password field contains text.

Test #	03
Test Description	Player should only be able to login using a username and password combination that exists in the SQL server.
Steps To Run Test	1. Enter a username that does not exist 2. Enter text in the password box 3. Click the login button
Expected Result	An alert label should show, telling the player that the specified username and password combination does not exist.
Actual Result	The alert pops up on the *Login GUI*, telling the player that the entered combination is not valid.

Test #	04
Test Description	Player should only be able to create an account with a specified username that does not already exist in the database.
Steps To Run Test	1. Enter a username that is known to already exist in the database 2. Enter text in the password box 3. Click the create account button
Expected Result	An alert label should show, telling the player that the specified username

	already is in use, and cannot be used in the creation of a new account.
Actual Result	The alert pops up on the *Login GUI*, telling the player that the entered combination is not valid.

Test #	05
Test Description	Player should only be able to login with the correct password, given a username and password combination that exists in the database.
Steps To Run Test	1. Enter a username that is known to already exist in the database 2. Enter a password that is not associated with the username 3. Click the login button
Expected Result	An alert label should show, telling the player that the password is incorrect.
Actual Result	The alert pops up on the *Login GUI*, telling the player that the entered password is not valid.



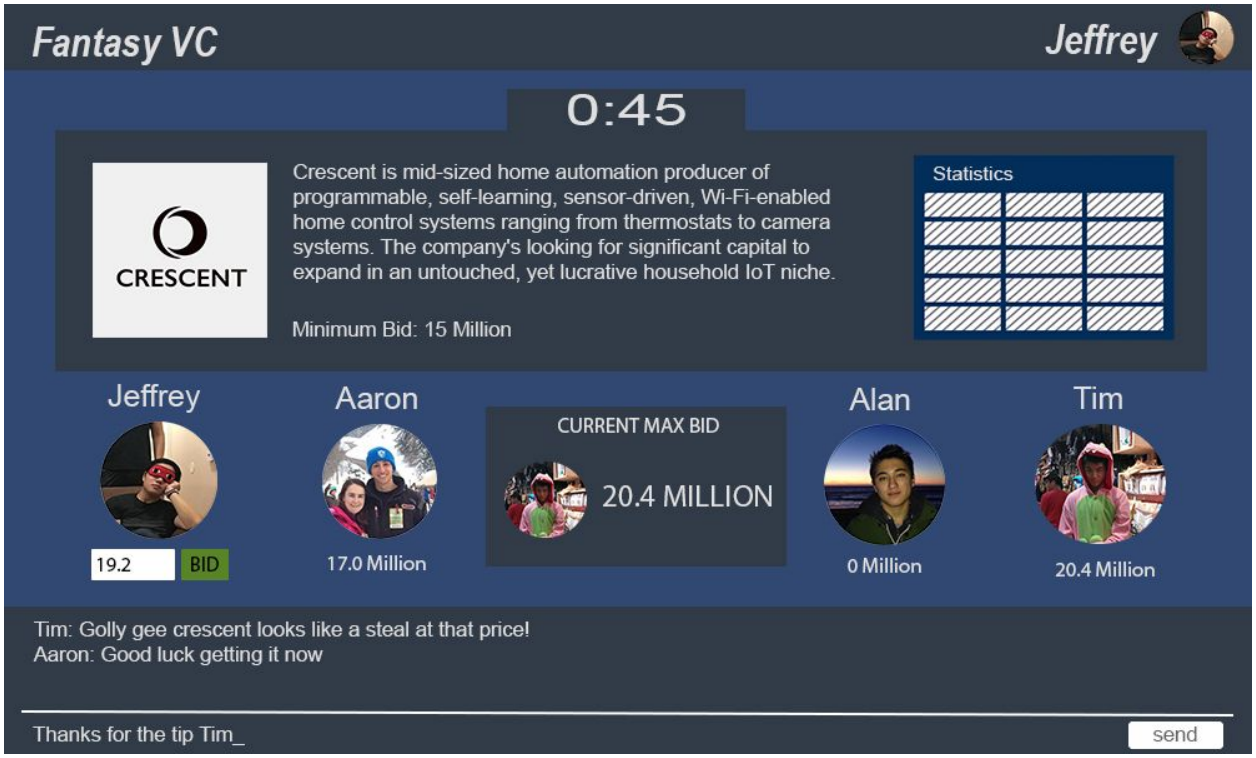
Auction GUI

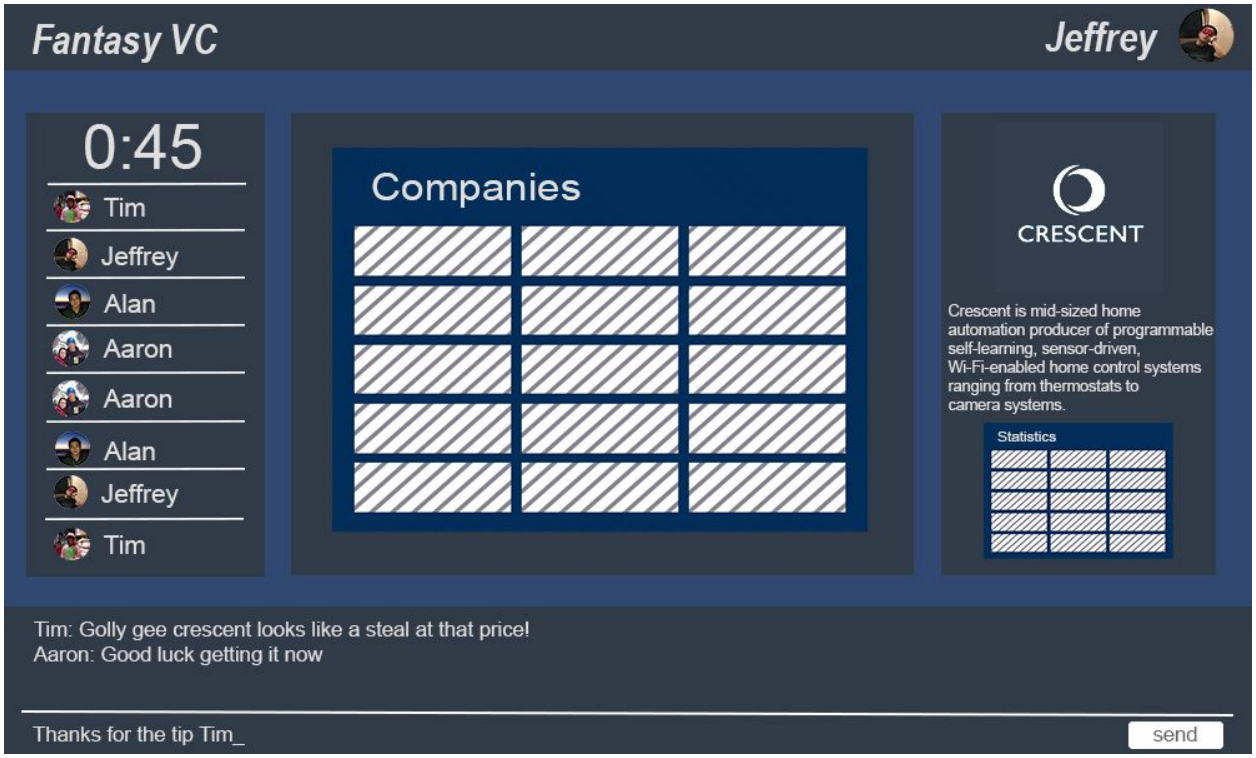
Test #	06
Test Description	If nobody places a bid on the current company at auction, the company should be added to the free agent list, and grayed out on the *Company List Pane*.
Steps To Run Test	1. Select a company to bid upon in the *Company List Pane* 2. After arriving at the *Interactive Auction Pane*, nobody bids on the current company, letting the timer at the top run out of time
Expected Result	Each player's screen should revert back to the *Company List Pane*, with the company grayed out because it had already been bidded upon. When we begin the game, the company should be in the free agents list.
Actual Result	Company is grayed out and the company is on the free agents list.

Test #	07
Test Description	We need to test the case that somebody wins the bid on a company.
Steps To Run Test	1. Begin a game and reach the auction rounds 2. Select a company to bid upon 3. As many of the players can bid as much as they wish on the listed company
Expected Result	If somebody wins the bid, the company should be added to his or her company list or inventory, and grayed out on the *Company List Pane*. The amount they paid for the company should be deducted from his or her bank account.
Actual Result	*Company List Pane* has been updated, and the player's new amount of cash is here-on-forth reflected.

Test #	08
Test Description	A player attempts to place a bid that is lower than the current highest bid, or the minimum asking price.
Steps To Run Test	1. Begin a game and reach the auction rounds 2. Select a company to bid upon 3. Have a player bid an amount lower than the minimum asking

	price 4. Let as many of the players validly bid as much as they wish on the listed company 5. Have a player bid an amount lower than the current highest bid
Expected Result	A label should be shown that the player's bid is invalid if a player attempts to place a bid on steps 3 and 5.
Actual Result	The label was successfully shown, telling the player that his or her bid was too low to be placed.


Fantasy VC
Jeffrey
0:45
CRESCENT
Crescent is mid-sized home automation producer of programmable, self-learning, sensor-driven, Wi-Fi-enabled home control systems ranging from thermostats to camera systems. The company's looking for significant capital to expand in an untouched, yet lucrative household IoT niche.
Minimum Bid: 15 Million
Statistics
Jeffrey
Aaron
CURRENT MAX BID
20.4 MILLION
Alan
Tim
19.2
BID
17.0 Million
0 Million
20.4 Million
Tim: Golly gee crescent looks like a steal at that price!
Aaron: Good luck getting it now
Thanks for the tip Tim_
send


Fantasy VC
Jeffrey
0:45
Tim
Jeffrey
Alan
Aaron
Aaron
Alan
Jeffrey
Tim
Companies
CRESCENT
Crescent is mid-sized home automation producer of programmable self-learning, sensor-driven, Wi-Fi-enabled home control systems ranging from thermostats to camera systems.
Statistics
Tim: Golly gee crescent looks like a steal at that price!
Aaron: Good luck getting it now
Thanks for the tip Tim_
send

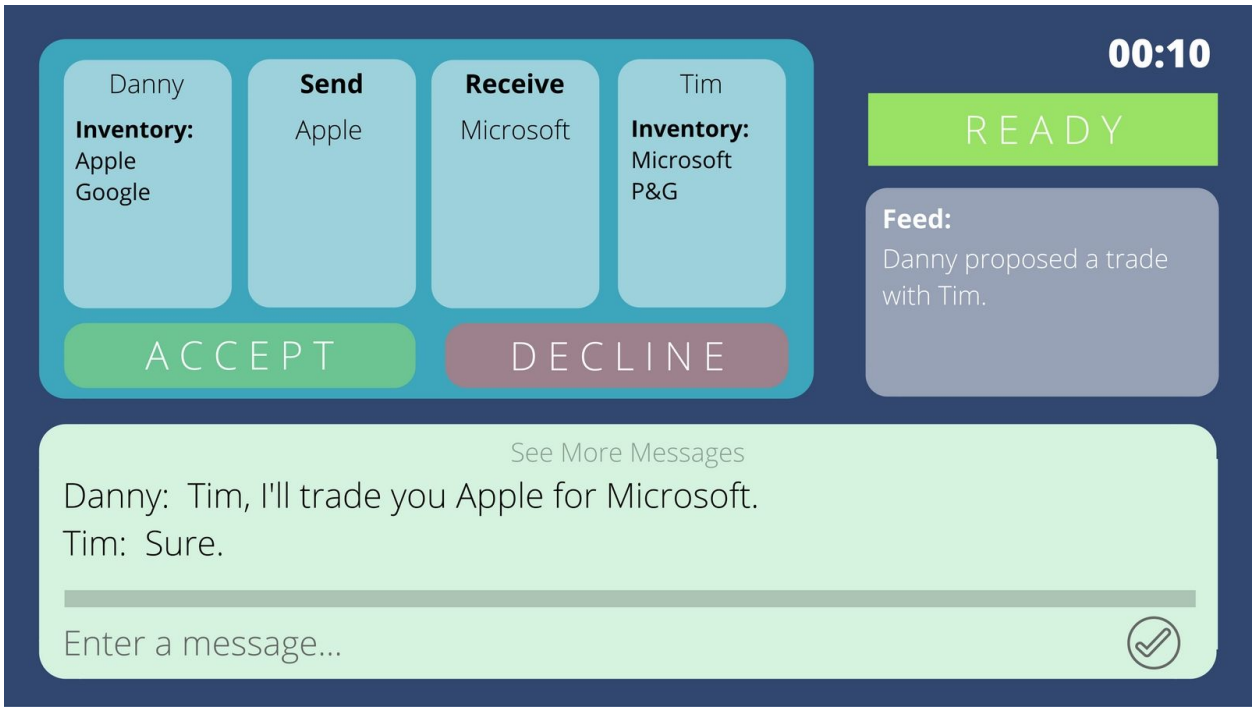
Quarterly GUI

Test #	09
Test Description	Player should only be able to offer a trade if the *Trade Space* in the *Trade View Pane* is not null.
Steps To Run Test	1. Initiate trade with a player 2. Do not add any items to the *Trade Space* 3. Attempt to offer the trade to the other person
Expected Result	The trade offer button should not be enabled while the *Trade Space* is null.
Actual Result	Send trade request button is disabled.

Test #	10
Test Description	If a trade transaction occurs, the new owners of the companies traded should be reflected on everyone's *Grid View Pane*.
Steps To Run Test	1. Have two players conduct a trade of a company for cash 2. Check the *Grid View Pane*, inspect both players' tabs
Expected Result	The company now shows up in the new owner's portfolio, and the cash is now reflected in the other player's portfolio.
Actual Result	Both the company and cash have successfully been transferred.

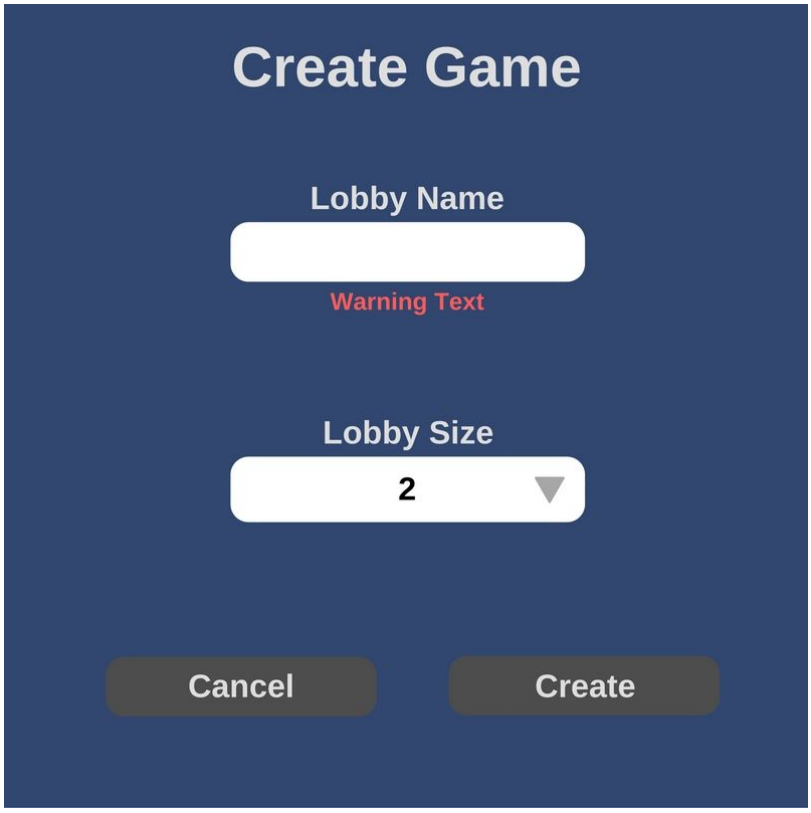
Test #	11
Test Description	If a free agent company is selected for a bid, an abbreviated bid listing should be pushed to everybody's *Feed*. The bid listing should have a submittable button that corresponds to a text field in which to place your bid. Since we are utilizing the same logic that has been already tested in the *Auction GUI* test cases, we do not need to retest those.
Steps To Run Test	1. Begin a game 2. Play the game until reaching a *Quarterly GUI* 3. Access the free agent tab and select to bid upon a company 4. Utilize the abbreviated bid listing to place a bid on the company
Expected Result	The bid listing should show up on every player's *Feed*, bids should be placeable.

Actual Result	The bid listing successfully pops up on every player's *Feed* and bids are placeable.



Create Lobby Pop-up

Test #	12
Test Description	Player should only be able create a lobby using a lobby name that is not currently in use.
Steps To Run Test	1. Start a lobby using an arbitrary name 2. On another instance of the game, attempt to form another lobby with an identical name
Expected Result	A label should alert the second instance's player that the name he or she is attempting to use for the lobby is already in use, and that another name should be picked.
Actual Result	The label appears and informs the player that lobbies cannot have redundant names. The lobby name field is cleared to prevent spamming.



Final GUI

Test #	13
Test Description	Clicking the okay button on the *Final GUI* should individually bring a player back to the *Introduction GUI* (without affecting any other players), and update the player's entry on the database by updating his or her total games played and lifetime cash earned. To the lifetime cash earned, we will add the difference between his or her end game net worth, and the start game allowance.
Steps To Run Test	1. Complete a multiplayer game 2. Have one player in the game press the okay button to return to the *Introduction GUI*
Expected Result	Only the player that clicks the button should be sent to the *Introduction GUI*. Now go open MySQLWorkbench and check to see which values have been updated as a result of the game that had just ended. Only the values for the only player to press the okay button should be different than they were at the beginning of the game.
Actual Result	The SQL database has been updated for the specific person that pressed the okay button, and his GUI has been returned to the *Introduction GUI*.

User Unit Tests

Test #	14
Test Description	Unit test that tests the User.addCompany(Company) function of the User.
Test Cases	1. Adds a valid company when ownedCompanies is empty. 2. Adds a valid company when ownedCompanies has ten item in it. 3. Adds a valid company when ownedCompanies has 50 items in it 4. Adds a null company
Expected Result	1. The list ownedCompanies should now have 1 company in it (the one that was added). 2. The list ownedCompanies should now have 11 companies and the one at index 10 should be the new one. 3. The list ownedCompanies should now have 51 companies and the new one should be at index 50. 4. Nothing should happen. The list ownedCompanies should have the same size that it did before addCompany was called.
Actual Result	1. ownedCompanies has 1 company in it (the one that was added). 2. ownedCompanies has 11 companies in it and the one at index 10 is the one we recently added. 3. ownedCompanies has 51 companies in it and the one at index 50 is the one we recently added. 4. Nothing happened. The null company was not inserted. ownedCompanies has the same size that it did before addCompany was called

Test #	15
Test Description	Unit Test that tests the User.deleteCompany(Company) function of the User.
Test Cases	1. Deletes a Company when there ownedCompanies is empty. 2. Calls deleteCompany(Company) and inputs a Company that does not exist in ownedCompanies. 3. Deletes the only company in the ownedCompanies list. 4. Deletes the first company in the ownedCompanies list. 5. Deletes the last company in the ownedCompanies list. 6. Deletes a company in the middle of the ownedCompanies list.
Expected Result	1. Nothing should happen. The ownedCompanies list should still remain empty. 2. Nothing should happen. The ownedCompanies list should remain the same size. 3. The only company in ownedCompanies should be deleted and the size should be zero. 4. The first company in ownedCompanies is deleted. The size should be one less than before and all of the companies should move up one index in the list. 5. The last company in ownedCompanies is deleted. The size should be one less than before. 6. The size of ownedCompanies is one less than before and the indices are changed accordingly.
Actual Result	1. Nothing happens. ownedCompanies is still empty. 2. Nothing happens. ownedCompanies list is the same size as before. 3. The only company in ownedCompanies was deleted and

	the size is now zero. 4. The size is now one less than before. All of the indices were moved up one. 5. The size is now one less than before and the last company was deleted. 6. The size of ownedCompanies is now one less than before and the indices are changed accordingly.

Test #	16
Test Description	Unit Test that tests the setter and getter functions of the User class. These functions include User.getName(), User.getImage(), User.getDescription(), User.getMoney(), and their corresponding setter functions.
Test Cases	1. Calls setName(String) and then getName() and checks that we get the desired String. 2. Calls setImage(String) and then getImage() and checks that we get the desired String. Also creates an Icon using the getImage() function as input and checks that it displays the desired image. 3. Calls setDescription(String) and then getDescription and checks that we get the desired String. 4. Calls setMoney(int) and then getMoney() and checks that we get the desired int.
Expected Result	1. The String returned by getName() should be the same String we inputted into setName(String) 2. The String returned by getImage() should be the same String we inputted into setImage(String). Also the icon created by using the getImage() function as the file path should be the desired image. 3. The String returned by getDescription() should be the same

	String inputted into setDescription(String). 4. The int returned by getMoney() should be the same as the int inputted into setMoney(int)
Actual Result	1. The String returned by getName() is the same String we inputted into setName(String) 2. The String returned by getImage() is the same String we inputted into setImage(String). Also the icon created by using the getImage() function as the file path is the desired image. 3. The String returned by getDescription() is the same String inputted into setDescription(String). 4. The int returned by getMoney() is the same as the int inputted into setMoney(int)

Company Unit Tests

Test #	17
Test Description	Unit Test that tests the setter and getter functions of the Company class. These functions include Company.getName(), Company.getImage(), Company.getDescription(), Company.getCurrentWorth(), Company.getAskingPrice(), Company.getStartingPrice(), Company.getTier(), and the corresponding setter functions.
Test Cases	1. Calls setName(String) and then getName() and checks that we get the desired String. 2. Calls setImage(String) and then getImage() and checks that we get the desired String. Also creates an Icon using the getImage() function as input and checks that it displays the desired image.

	3. Calls setDescription(String) and then getDescription and checks that we get the desired String. 4. Calls updateCurrentWorth(int) and then getCurrentWorth() and checks that we get the desired int. 5. Calls updateAskingPrice(int) and then getAskingPrice() and checks that we get the desired int. 6. Calls setStartingPrice(int) and then getStartingPrice() and checks that we get the desired int. 7. Calls setTier(int) and then getTier() and checks that we get the desired int.
Expected Result	1. The String returned by getName() should be the same String we inputted into setName(String) 2. The String returned by getImage() should be the same String we inputted into setImage(String). Also the icon created by using the getImage() function as the file path should be the desired image. 3. The String returned by getDescription() should be the same String inputted into setDescription(String). 4. The int returned by getCurrentWorth() should be the same as the int inputted into updateCurrentWorth(int) 5. The int returned by getAskingPrice() should be the same as the int inputted into updateAskingPrice(int). 6. The int returned by getStartingPrice() should be the same as the int inputted into updateStartingPrice(int) 7. The int returned by getTier() should be the same as the int inputted into setTier(int).
Actual Result	1. The String returned by getName() is the same String we inputted into setName(String) 2. The String returned by getImage() is the same String we inputted into

	setImage(String). Also the icon created by using the getImage() function as the file path is the desired image. 3. The String returned by getDescription() is the same String inputted into setDescription(String). 4. The int returned by getCurrentWorth() is the same as the int inputted into updateCurrentWorth(int) 5. The int returned by getAskingPrice() is the same as the int inputted into updateAskingPrice(int). 6. The int returned by getStartingPrice() is the same as the int inputted into updateStartingPrice(int) 7. The int returned by getTier() is the same as the int inputted into setTier(int).

Deployment

The application will be an executable JAR file. To deploy the application, simply double click on the executable jar file or use the command line: java -jar <myjarfile.jar>. The file is availible for download at jeffreychen.space

Alternatively, if a user has a copy of the actual project files, he or she can open Eclipse and import the CSCI201_FinalProject_venture.zip file into Eclipse. This generates a project with a src directory. Enter the src directory to find a main file, and click the green arrow (run button) to execute the program.