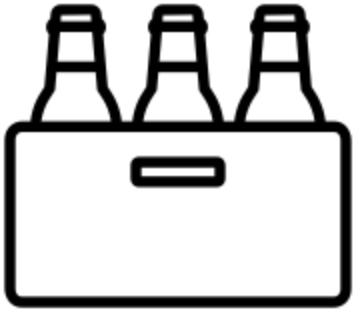


BEER GAME PROPOSAL

Software Engineering Assignment 1

Team 8

Dedaj Fjolla

Checiu Eliza

Spirkoski Kristijan

Soham Saha

Kandel Dipak

ABSTRACT

This document plays a key role in the design phase of the beer game development. A poorly elaborated proposal can lead to rework and loss of investment in production and postproduction phases. This document incorporates a common understanding of terms, quality assurance, decision making, definition of relations, boundaries; limitations and knowledge of elements which all put together create our interpretation of the beer game.

This proposal contains all our documentation and interpretation of the beer game, a guide to be used to implement this project. It contains general information explaining the rules of the games, the mechanics of the game, requirements and constraints, diagrams and schemes which best portray the information we are delivering.

INTRODUCTION

An important notion in supply chain management is the bullwhip effect, stating that demand variability increases when switching positions in the supply chain. Evidence suggests that the orders placed by a retailer tend to be much more variable than the customer demand seen by that retailer. This increase in variability causes a distortion in the pattern of orders received by distributors, manufacturers and suppliers. In order to better understand this effect, the beer game was created to clearly illustrate and present the bullwhip effect, in a practical interactive way. Versions of this game have already taken off starting from the original board-and-card version to computerized versions. Web-enabled versions give the user the possibility to have multiple players' games as well as a single player's games, online.

The game that has been developed as part of this paper is one in that direction.

KEY WORDS

Bullwhip effect, Beer Game, Object Oriented Design, UML

UNDERSTANDING OF THE GAME

The beer game is an online game intended for students to better understand how supply chain works in a real world situation, applying the theory learned in class and practicing decision making. Therefore it is a functional type of game. This project simulates the way beer is being ordered and delivered from production and factory, going to distributor, then wholesaler and then to retailer and consumer. Each of these stages can be played by the students who, order enough beer stock in the position they hold, taking into consideration shipment times, delays, backorders, inventory such that, every partner in this chain doesn't suffer any losses.

The objective of the game is to satisfy the demand of the customer, while keeping the cost low. There is a cost for holding inventory and a cost for not satisfying demand (backorder). The demand for the product remains until it is satisfied i.e. backorder persists until it is fulfilled. It creates the illusion of a real time supply chain, each could view only the downstream data and the upstream shipment they receive. (Figure 1)

GAME TERMINOLOGY

- Inventory: The amount of beer you have on stock after receiving the beer you ordered and after shipping the current delivery.

- Backlog: Demand you could not fulfill (= „negative“ inventory). Backlog cumulates as long as you cannot fulfill demand!

- Demand: Amount of beer you have to deliver this week (= amount of beer your successor ordered 2 weeks ago).

- Inc. Shipment: Amount of beer you receive this week (= amount of beer you ordered 4 weeks ago).

TECHNICALITIES OF THE GAMES

Accessing the game, a user can either be an instructor or a player, more details being provided below.

The game offers 4 positions: Factory, Distributor, Wholesaler, Retailer, each being a position a player can participate in. A game can be played with a minimum of 2 positions, Factory and Distributor, or 3 positions, adding either Wholesaler or Retailer, or all positions. The positions are set up by the instructor. The retailer gets orders from the customer', who is external to the game, therefore not an actual actor in the game.

There is a big game, which has multiple rounds – sessions - inside, which vary in number (maximum of 52 games), number set by the instructor, each round corresponding to a week.

An instructor can sign up/login with username (email in this case), password and his/her university/organization/institute credentials, then set up the number of game sessions and the settings of each session. The instructor receives a list with details for each player for each game session, which he/she can modify if needed. These could be printed by the instructor in a label format. After distributing these details to the players and giving the go, the game is played entirely by the students without any intervention by the instructor.

The player can only login in with the details given by the instructor. The details refer to a unique password, the number of the game session they are in, and the position they will hold for the game. In this way no player can view other partner's data. (Figure 2)

The system notes down all the details and information mentioned in the user's profile, which will be used throughout the tenure of the game. Any time the user inputs anything into the game, the system takes in those values and saves them in the corresponding databases.

In order to play, the player needs to access the website of the game, click on *Play the Game*, then find *Logistics@Jacobs_University* and write the login details given by the instructor.

The Login event should be a success. Only after the player/instructor has been successfully logged in will the system present a decision form to him/her to fill up with the following step - monitoring a game, changing settings, starting/continuing the game.

In a round, each player must fill all current orders, specify the amount of beer, place orders with his own supplier, and try to minimize back orders and inventory by keeping in mind the number of weeks time it takes to receive an order and the supply already available. (Figure 3)

After validating the login, the player is transferred to their game screen. The game screen consists of 4 quadrants, namely order input screen, past information over 10 weeks about their position, status information of the supply chain partners (other positions in your game) and plot and settings screen. (Figure 4)

First quadrant is the input screen of a player contains the week number the game reached; then a table containing: one column with Demand from X (X being the previous position - for Distributor it is Factory, for Wholesaler it is Distributor, for Retailer it is Wholesaler) with the number demanded; On Backorder and the number of negative inventory; then we have the sum of all that in Total Requirements. In the next column we have Beginning Inventory with the number of inventory we have in stock at the moment; Incoming Shipment with the number of stock on its way to be delivered to our position and the sum of all this stock under Total available. Then we have Units shipped this week to the next position in the chain and the number of inventory; Ending inventory and the number of beers we are left with after this session; Enter the number of units to be purchased from the previous position and a empty block in which the player can add the value he/she wants to contribute for this session. And a submit button, which sends the information forward so the game can continue. At the end of the game, this part will only contain a button where you can access the results of the game.

The second quadrant is the player's information for the past 10 weeks. It contains a table with all the details of a week: Week Number, Inventory/Backlog, Demand, Incoming shipment, Outgoing shipment. At the end of the game, this part will only continue a button where you can access the results of the game.

The third quadrant has the status of the other supply chain members of the game, showing if the other players have placed their orders or not in that week, this status being updated every 15 seconds. As soon as the players have submitted their answers, the entire page will be updated for the following week's orders and progress.

The last quadrant contains the features added in order to improve the user's learning experience in the game, giving a more clear understanding of how it all works: showing the plots of the demand, of the orders, of the inventory/backlog and plotting it all together for a bigger perspective. Moreover, we have some information about the game, such as holding cost, backorder cost, downstream player, upstream player, shipping delay and information delay.

Here the player is required to make decisions on how much to ship to the immediate downstream facility in the supply chain and how much to order from the immediate upstream facility in each period of operation. After the form is filled the player clicks on submit after which the system takes this information and updates the status of the player.

Depending on the commands the player choses, the system automatically informs him/her about the expected demand during lead-time, which will help the player to get a fair idea of how much s/he should order. The game is being updated every 15 seconds such that any progress is shown and the game can move forward.

If there are wrong values given, a negative value for the cost, the system sends an alert message. The player is required to enter the correct value otherwise s/he cannot advance in the game.

Some features of the games are graphing plots of individual game players, graph summary of the game, automatic cost calculations, instructor game monitoring (not control).

All the calculations mentioned previously are done by the system, using the parameters input from the start of the game.

The instructor could change the structure of individual games in the followings ways:

- Change the lead time of shipment and information sharing, 2week shipping and information delay, 1 week shipping and information delay and 1 week shipping and no information delay;
- Change the demand pattern for the game;
- Change the holding cost for the game;
- Change the backorder cost for the game;
- Remove Retailer;
- Remove Wholesaler;
- Remove both wholesaler and retailer

Below is a list of options and what they entail for the game and the mechanics of:

- Time delay between supply chain partners - players : By varying this option (0 weeks, 1 week and 2 weeks) we can change the shipping and information delay between the supply chain partners.
    - 2 weeks : In this case both shipping and information delay between supply chain partners is set at 2 weeks
    - 1 week : In this case both shipping and information delay between supply chain partners is set at 1 week.
    - 0 weeks (No delay) : In this case there is no information delay between

supply chain partners, but there is a 1 week shipping delay
    - Exception to all these settings
        - There is no shipping or information delay in case of the customer and the immediate supply chain partner
        - There is only one week production (i.e. shipping) delay between Brewery (Raw material source) and Factory, regardless of delay settings (0,1 or 2 weeks)
    - The information delay between factory and brewery is always 1 week, regardless of delay settings(1 week or 2 weeks)
- Is Wholesaler Present? By setting no to this option, you can remove the wholesaler in the supply chain setting.
- Is Retailer Present? By setting no to this option, you can remove the retailer in the supply chain setting. (Note: You can remove both Wholesaler and Retailer in a game, simply by changing this option - press update to reflect the change)
- Demand Pattern - There are 3 customer demand patterns for this game. Pattern 1 is the default pattern. Pattern 2 has a steep increase in demand before stabilizing. Pattern 3 gives the instructor the option to personalize the demand pattern by his/her own standards and goals.
- Holding cost, it could be set to any real number. Default setting is $0.50
- Backorder cost, it could be set to any real number. Default setting is $1
- Reset this game, by clicking this button, you reset the game to week 1 (starting week). The game is also reset to default settings, namely 2 week delay, along with wholesaler and retailer being present as supply chain partners with a holding cost of $0.5 and backorder cost of $1.
- Player pwds, You can change the passwords of individual supply chain partners using this option. (NOTE: The passwords are case-sensitive and contain no zeros or ones.)
- Update, by clicking this button you can record the changes you made to the individual games. You should click update every time you change the settings, else the changes will be not be reflected in the game
- Reset all the games, similar to the reset option of individual games, only difference it resets all the games to default settings
- Change the number of games, by clicking this button you add or remove the games for this account
- Print all pws - details for the players' login, this option will help you to print a copy of all the passwords for the players in a label format, which could be distributed to the class as flash cards
- Freeze or Freeze/Pause all the games The instructor could freeze/pause the game. This allows the instructor to carry out class discussions during the course of the game to clarify issues like backorders, orders not being shipped etc. After the discussion/recess the instructor could restart the game from where they left, by clicking the "Continue" or "Continue all the games" button.
- Consumer demand is determined by the computer

FIGURES

Figure 1: Supply Chain of the beer game

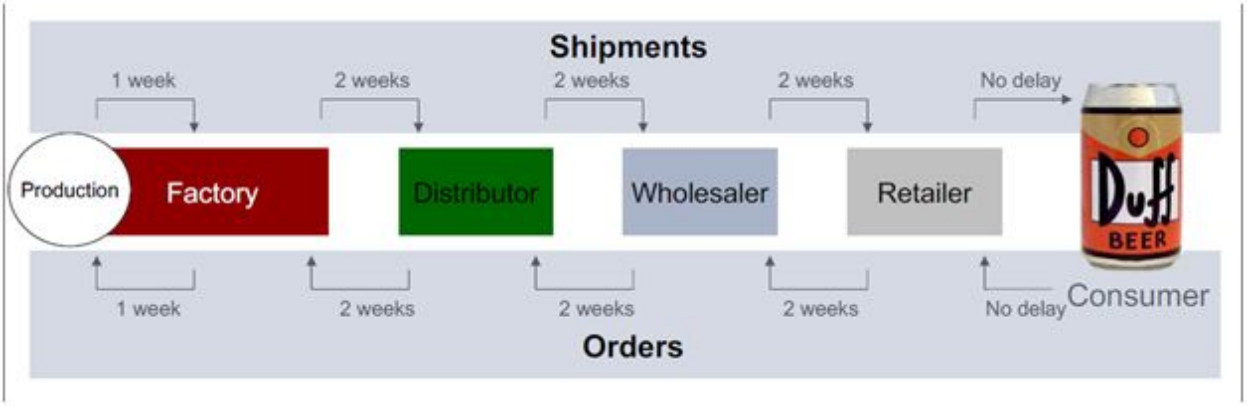


Figure 2: example of details for a player

Figure 3: sample look of the game when a player just logged in

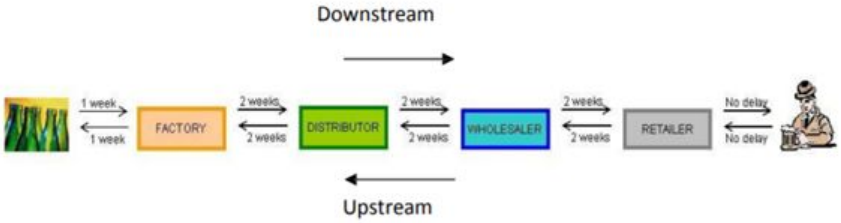
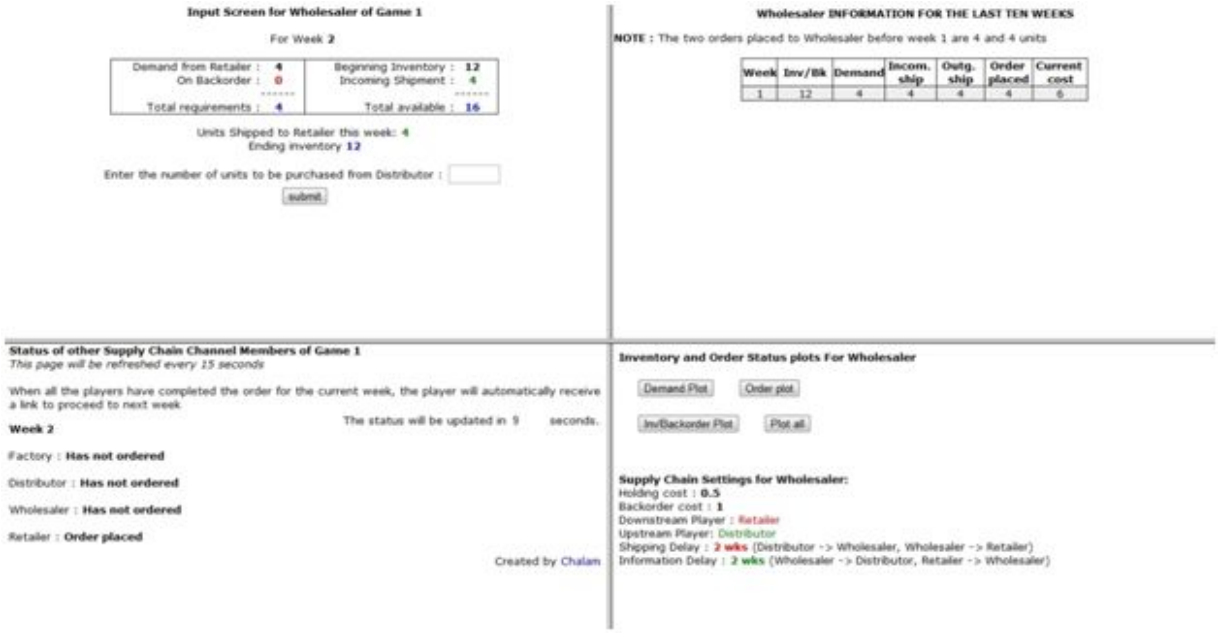


Figure 4: Example of a game screen of a player



SYSTEM DESIGN

For designing the system, we used Rational Rose to create UML diagrams which can be used in the implementation phase, further on. The diagrams presented were Class diagrams, Use Case diagrams, Activity diagrams and Sequence diagrams for the various uses and commands.

Diagram 1: Use Case Diagram of the Beer Game. The diagram represents what the instructor is able to do in the game and which functionalities have an impact on the player's behavior. Some of the base use cases are extended in the diagrams below this one.

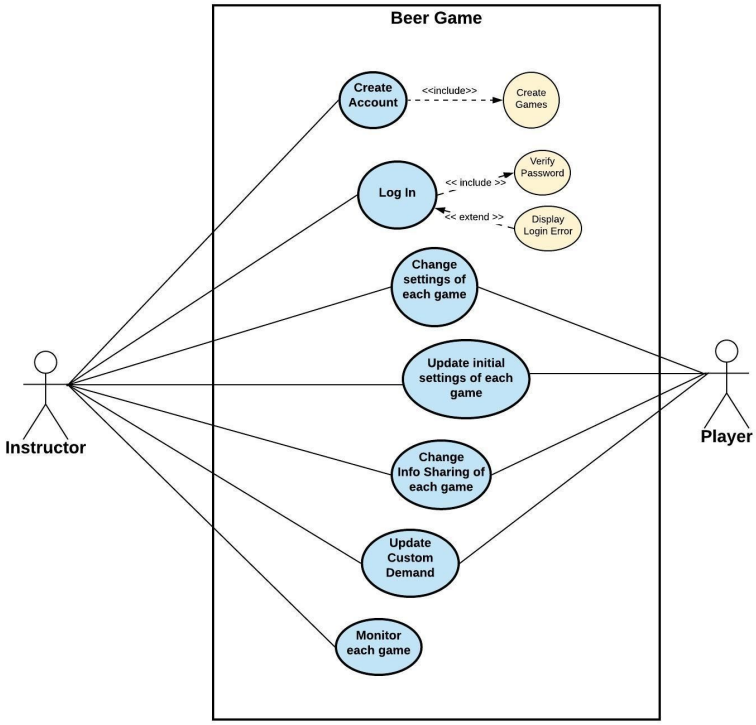


Diagram 2: Extended Use Case Diagram for changing the settings of each game. This diagram shows the settings that can be changed by the instructor for each individual game.

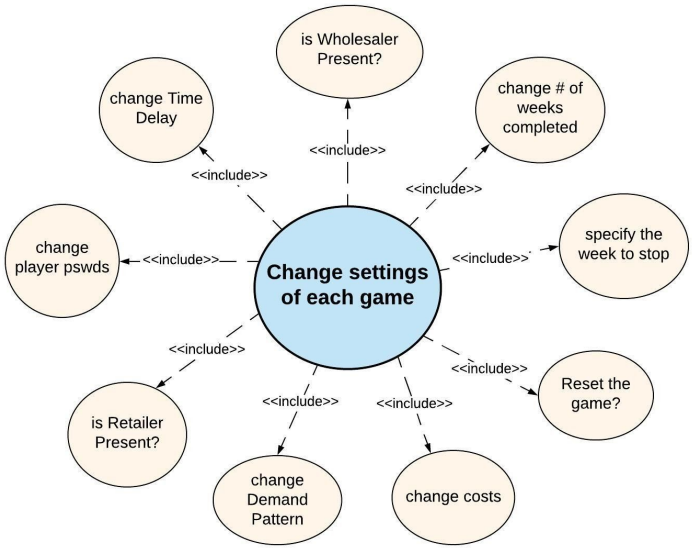


Diagram 3: Extended Use Case Diagram for updating the initial settings of each game. This diagram shows how the instructor can specify the initial values of the inventory, orders and shipments for each individual game and position of the player.

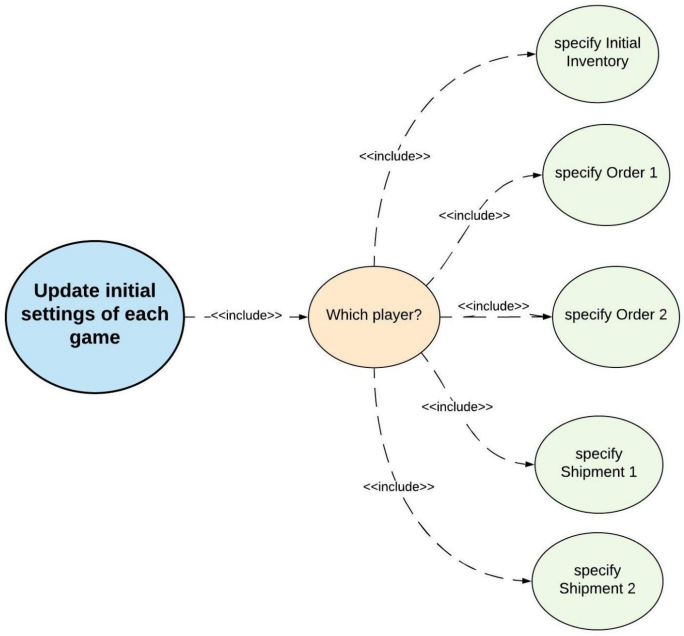


Diagram 4: Extended Use Case Diagram for updating the custom demand of each game for a maximum of 52 weeks. This diagram represents how the instructor can specify the value of the demand for each week consecutively.

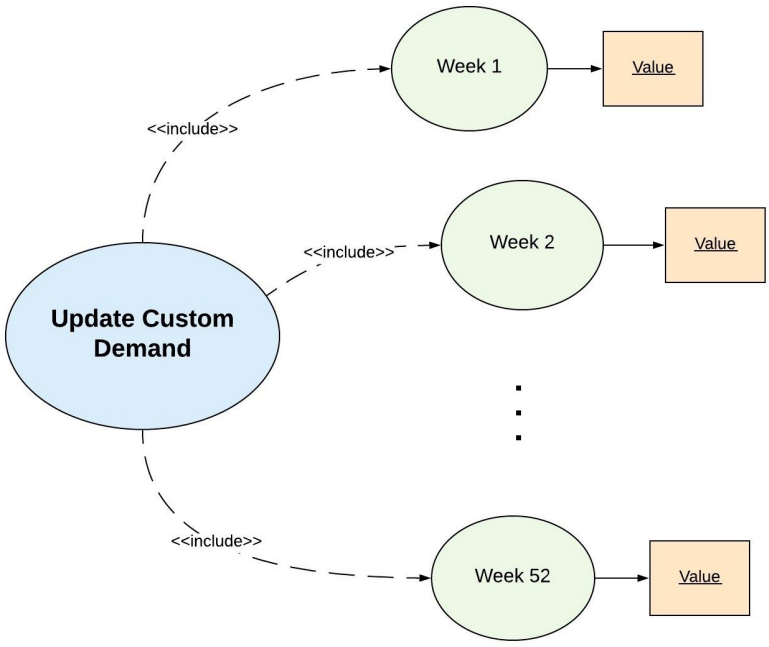


Diagram 5: Extended Use Case Diagram for changing the info sharing of each game. The diagram shows the possible options that are available to the instructor regarding the information shared between the players of a game.

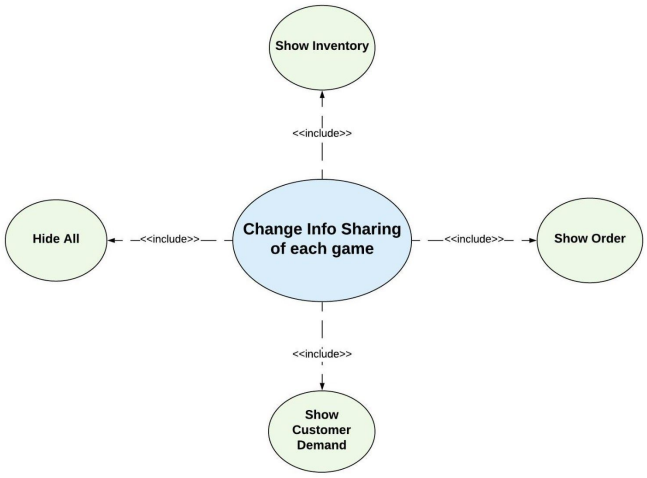


Diagram 6: Extended Use Case Diagram for monitoring each game. This diagram shows the available options to the instructor for the information he can access from the active games (including graphical plots) and the option to freeze or unfreeze games.

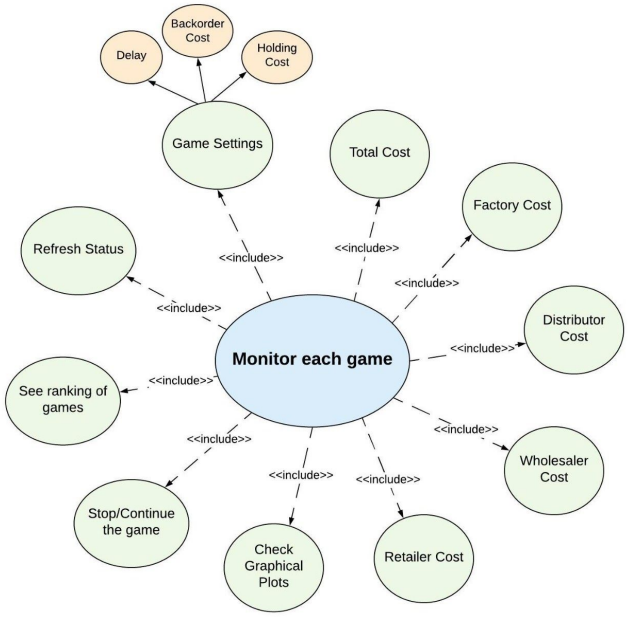


Diagram 7: Sequence Diagram for logging in as an instructor.

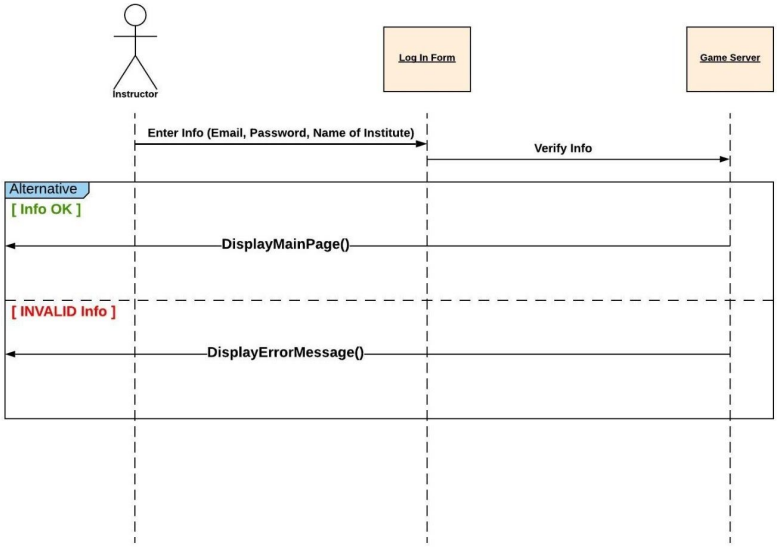


Diagram 8: Sequence Diagram for creating an account as an instructor.

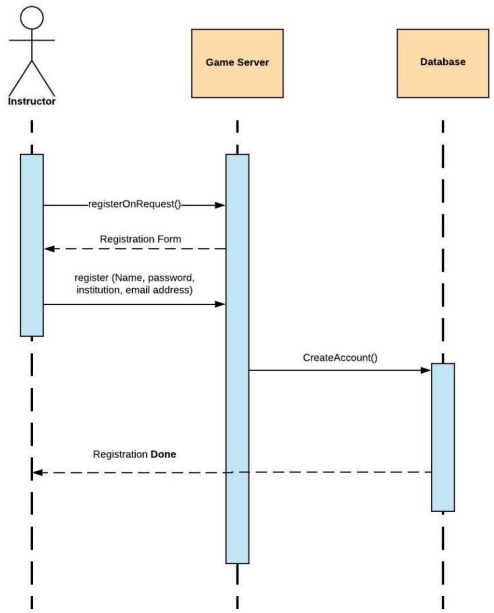


Diagram 9: Sequence Diagram of the instructor for changing the settings of each game.

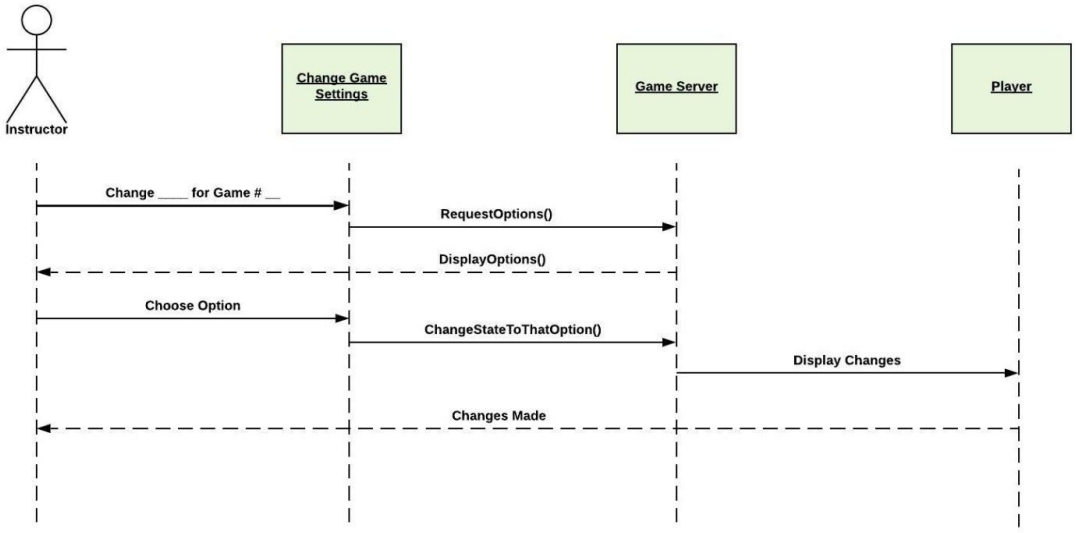


Diagram 10: Sequence Diagram of the instructor for updating the initial game settings.

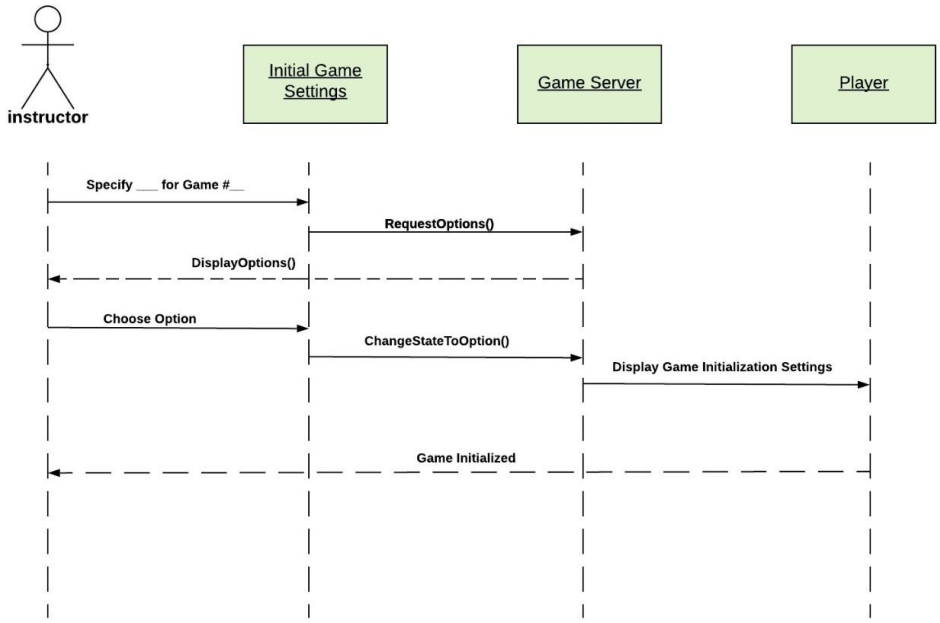


Diagram 11: Sequence Diagram of the instructor for updating the custom demand.

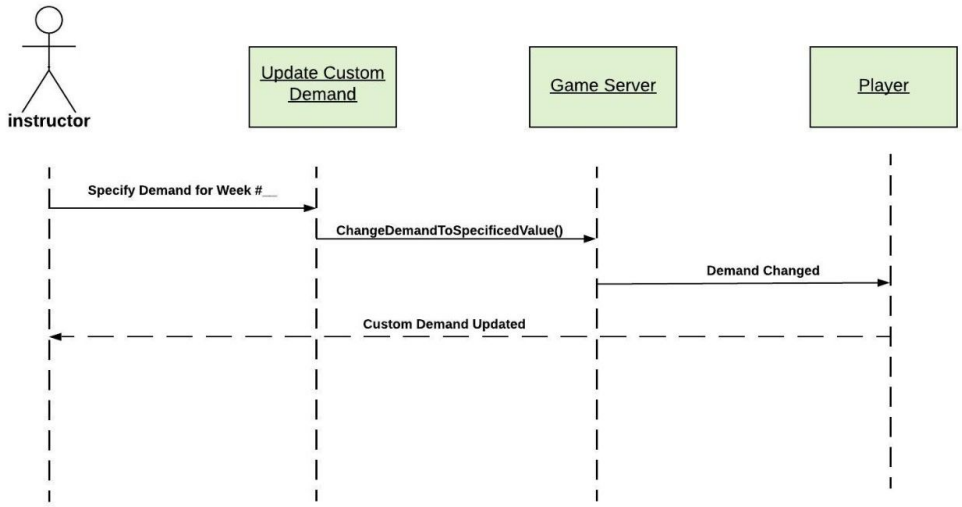


Diagram 12: Sequence Diagram of the instructor for changing the info shared between players.

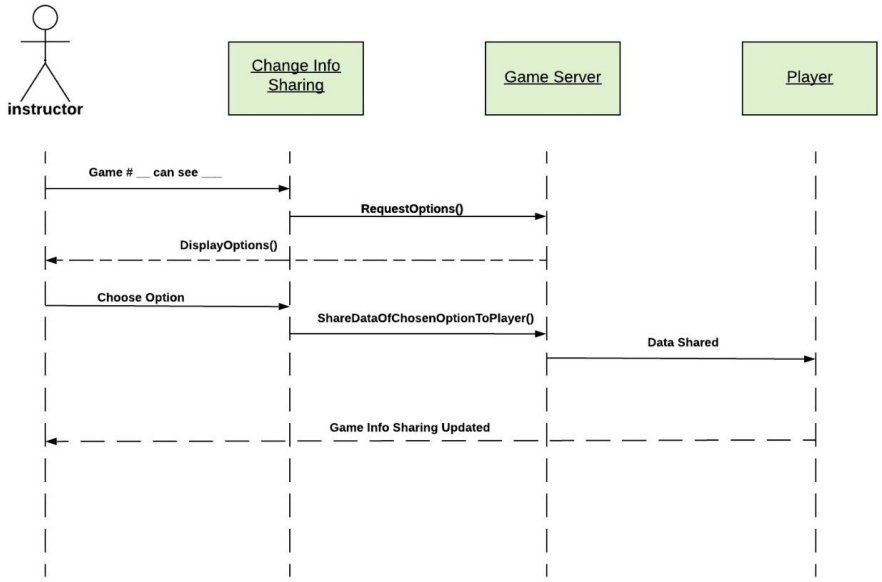


Diagram 13: Sequence Diagram of the instructor for monitoring each game.

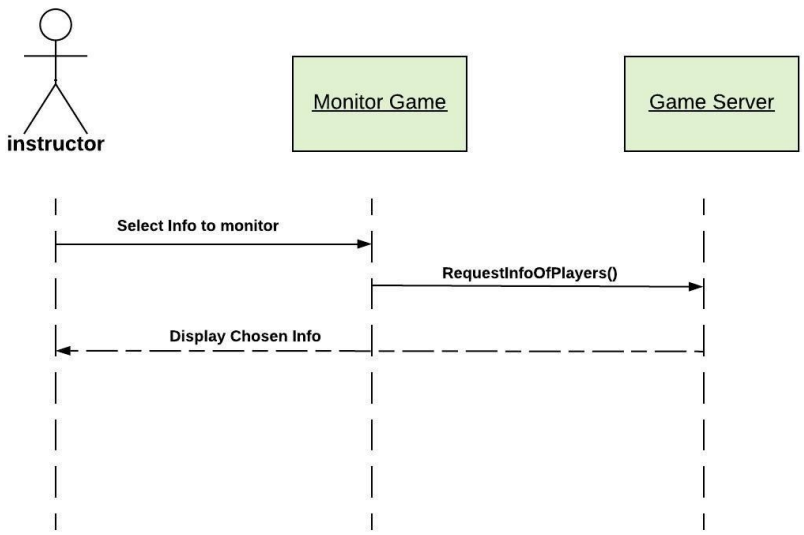


Diagram 14: Instructor Activity Diagram, this diagram describes the behavior of the instructor in the game.

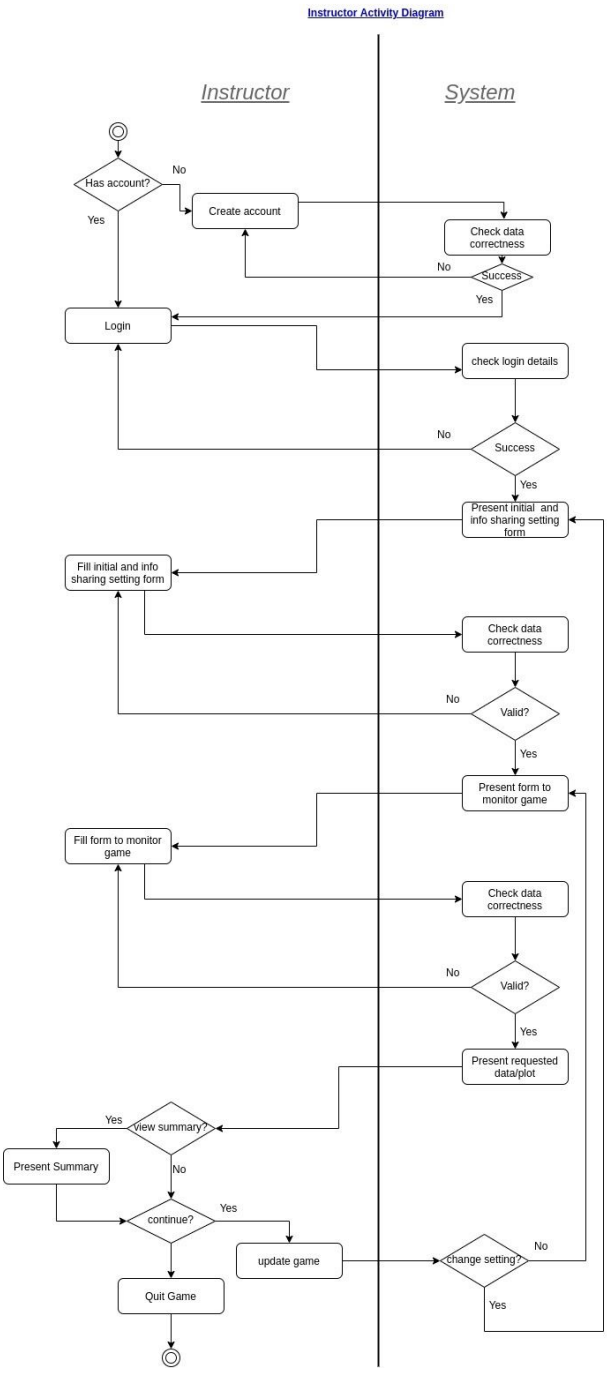


Diagram 15: Player Activity Diagram, this diagram describes the behavior of the players in the game.

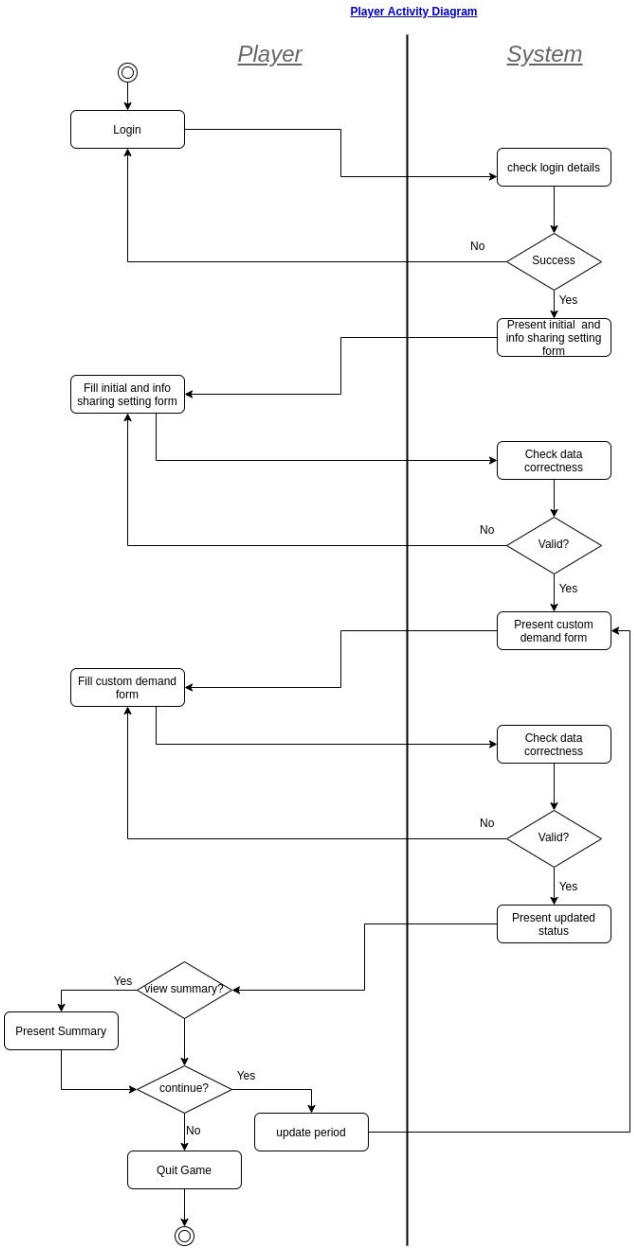
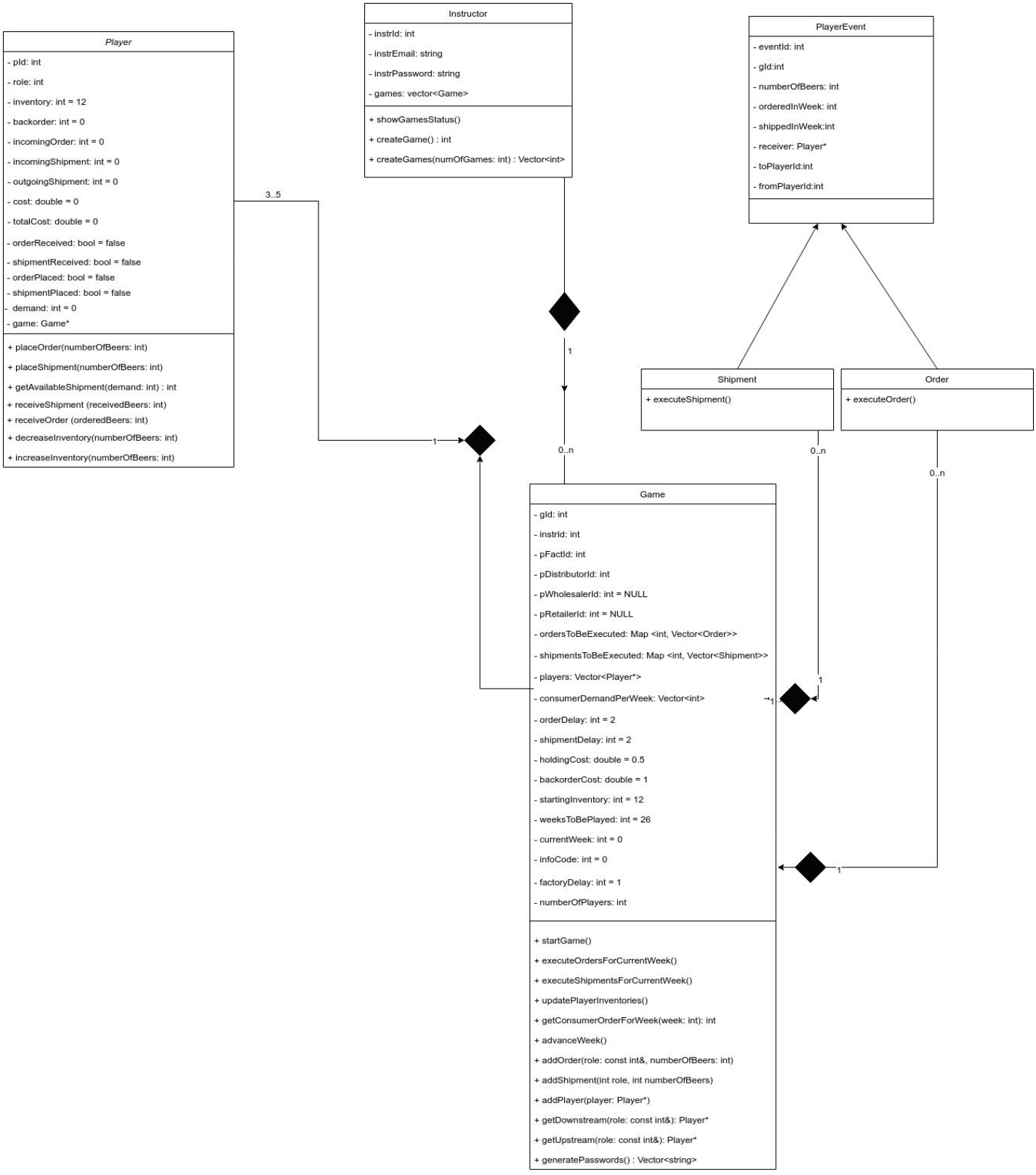


Diagram 16: Class diagram. A visualization of how each object is structured. A Game is created by the Instructor object, which has his own credentials. The Game consists of the settings which are set, and can be changed by the Instructor. The Player class is abstract for the roles Factory, Distributor, Wholesaler and Retailer. A game must contain one Factory, and at least one more Player. At most, there can be four players. Every week, a player creates an Order, which is saved by the Game instance, and executed on the specified week.



NOTE: For all the functions attributes you are supposed to implement setters and

getters, which will also be tested.

CONCLUSION

The purpose of the game, regardless of its exact implementation and/or its other variations, is the same and can be noted as to promote knowledge, increase skills, support teamwork, reinforce instructions, broaden perspectives, integrate functional viewpoints and improve decision-making.

All the information and details needed to pass to the next stage of implementation and development can be found in this proposal, in clear statements alongside diagrams to better illustrate our ideas.

REFERENCES

- https://www.draw.io/ for the creation of the diagrams
- https://www.lucidchart.com/ for the creation of the diagrams
- http://scgames.bauer.uh.edu/index.htm
- "Object-oriented design and implementation of a web-enabled beer game for illustrating the bullwhip effect in supply chains", Article in International Journal of Technology Management, august 2004, Research Gate
- http://www.faculty.jacobs-university.de/pbaumann/iu-bremen.de_pbaumann/Courses/SoftwareEngineering/index.php
- "Beer Distribution Game", http://www.pom.edu/beer
- "Object oriented modeling for decision support in supply chain networks", S. Biswas and Y. Narahari, European Journal of Operations Research, 153-3, (2003), 704-726. 18.
- "UML Resource Centre", http://www.rational.com/uml/index.jsp
- https://www.supplychain-academy.net/understanding-the-bullwhip-effect-in-supply-chains/

Updates

First Update - 06.03 (First Sprint)

In this sprint there are a number of updates regarding the uml class diagram:

- In class **Player**:
  - *Cost* attribute will be a *double*
  - the following methods will be added, in order to improve code usability:
    - void *setBackorder*(int backorder)
    - void *setCost*(int cost)
    - void *setInventory*(int inventory)
    - void setDemand(int demand)

        - int getDemand()
        - void setOrder(int order)
        - int getOrder()
    - Role attribute can become of type int and to each role will be assigned a specific digit, as follows:
        - 1 represents the Factory
        - 2 represents the Distributor
        - 3 represents the Wholesaler
        - 4 represents the Retailer
    - Add the following attributes:
        - int *demand*
        - int *order*

- In class **Instructor**, the following methods will be added:
    - void *setInstrEmail*(string instrEmail)
    - string *getInstrEmail*()
    - void *setInstrPassword*(string instrPassword)
    - string *getInstrPassword*()
    - void *setInstrID*(int instrID)
    - int *getInstrID*()

- In class **Game**:
    - *advance Week()* methods will be implemented only once
    - the following methods will be added:
        - int *getWeeksToBePlayed*()
        - int *getInfoCode*()
        - int *getStartingInventory*()
        - double *getBackorderCost*()
        - double *getHoldingCost*()
        - vector<int> *getOrdersToBeExecuted*()
        - void setCurrentWeek(int week)
        - Int getCurrentWeek()
    - The *holdingCost* and *backorderCost* should be *double* and accordingly, also the methods: setBackorderCost and setHoldingCost will take double parameters

Please note that the UML class diagram has been updated accordingly.

Second Update - 14.03 (Second Sprint)

- `shipmentDelay` and `orderDelay` moved from class `Player` to class `Game`
- `numberOfBeers` attribute added to class `Order`
- `ordersToBeExecuted` function modified in class `Game` to `map <int,`

`vector<Order>>`. `Int` denotes week number. The `vector` stores the Orders for that specific week. The position in the vector determines which type of Player the Order is for. For example, the first element of the vector will be the order for Factory, the second element would be the order for Distributor and so on.
- Setters and getters have been removed from the class diagram for better clarity. You should still implement them for the attributes of each class. If you're confused about what name to use for the setters and getters, check the provided test cases.
- In class `Player,` the attribute `order` has been renamed to `orderPlaced` due to conflict with the function of the same name.

Third Update - 24.03 (Sprint 3)

Player class:

- Added attributes `incomingOrder, incomingShipment, outgoingShipment, totalCost`.
- Added attributes that are boolean values: `orderReceived, shipmentReceived, orderPlaced, shipmentPlaced`.
- Modified attribute `orderPlaced` from `int` to `bool`.
- Modified methods `order(numberOfBeers:int, from:Player)` and `ship(numberOfBeers:int, to:Player)` to `placeOrder(numberOfBeers:int)` and `placeShipment(numberOfBeers:int)`.
- Added method `receiveOrder(int)` and `receiveShipment(int)`.
- Added method `getAvailableShipment(demand:int) : int`. According to the demand, inventory is adjusted and the number of beers available for shipping is returned.

Instructor class:

- Renamed `instrlEmail` to `instrEmail`. This was a typo in the class diagram.
- Added attribute `games:vector<Game>`. This vector stores the games that have been created by the Instructor.

Factory class:

- This class has been removed.

Game class:

- Removed attribute `orderTimeDelay`. This was redundant since `orderDelay` was already in this class.
- Added attributes `factoryDelay:int = 1, numberOfPlayers, shipmentsToBeExecuted:Map<int, Vector<Shipment>>`.
- Renamed attribute `demandPerWeek` to `consumerDemandPerWeek` for better clarity.

- Added attribute player: `Vector<Player*>`. This stores the `Player` objects participating in that particular game.
- Added method `startGame`.
- Added method `executeShipmentsForCurrentWeek`.
- Added method `getConsumerDemandForWeek(week: int)`. This returns the consumer demand according to the week from the vector, `consumerDemandPerWeek`.
- Added method `getDownstream(role:const int&): Player*`. This returns the `Player` object that is next in the "downstream" for a certain role.
- Added method `getUpstream(role:const int&): Player*`. This returns the `Player` object that is next in the "upstream" for a certain role.
- Modified method `addOrder(order:Order)` to `addOrder(role: const int&, numberOfBeers: int)`. An object of the class `Order` is created and put in the map, `ordersToBeExecuted`, according to the role.
- Added method `addShipment(int role, int numberOfBeers)`. An object of the class `Shipment` is created and put in the map, `shipmentsToBeExecuted`, according to the role.
- Added method `addPlayer(player: Player*)`. Here, objects of the class `Player` are added to the vector `players`.

Order class (renamed to PlayerEvent):

- `Order` class has been renamed to `PlayerEvent`.
- Added two derived classes, `Shipment` and `Order`.
- Added method, `executeOrder` for class `Order` and `executeShipment` for class `Shipment`.
- Added attribute `receiver:Player*`. This attribute indicates the `Player` object that receives the Order or the Shipment.