UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANTONIO IGOR MENDES DE OLIVEIRA

USO DE BLOCKCHAIN PARA MITIGAR O EFEITO CHICOTE EM CADEIA DE SUPRIMENTOS

RUSSAS

2018

ANTONIO IGOR MENDES DE OLIVEIRA

USO DE BLOCKCHAIN PARA MITIGAR O EFEITO CHICOTE EM CADEIA DE SUPRIMENTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Me. Filipe Maciel Roberto

Coorientador: Prof. Dr. Dmontier Pinheiro Aragão Júnior

RUSSAS

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

D32u De Oliveira, Antonio Igor Mendes.
Uso de blockchain para mitigar o efeito chicote em cadeia de suprimentos / Antonio Igor Mendes De Oliveira. – 2018.
77 f.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Ciência da Computação, Russas, 2018.
Orientação: Prof. Me. Filipe Maciel Roberto.
Coorientação: Prof. Dr. Dmontier Pinheiro Aragão Júnior.

1. Blockchain. 2. Cadeia de suprimentos. 3. Efeito Chicote. 4. Jogo da Cerveja. I. Título.
CDD 005

ANTONIO IGOR MENDES DE OLIVEIRA

USO DE BLOCKCHAIN PARA MITIGAR O EFEITO CHICOTE EM CADEIA DE SUPRIMENTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Me. Filipe Maciel Roberto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Dmontier Pinheiro Aragão Júnior (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Daiane de Oliveira Costa
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim. Mãe e pai, o cuidado de vocês e dedicação foi que deram, em alguns momentos, a esperança para seguir. A Bárbara, que durante toda a graduação me ajudou nos momentos bons e ruins, sendo alicerce para essa conquista. Aos colegas de turma que se tornaram grandes amigos e estiveram juntos comigo durante toda a jornada.

AGRADECIMENTOS

Ao Prof. Me. Filipe Roberto Maciel por me orientar neste trabalho, fornecendo material, ferramentas e norteando o desenvolvimento da pesquisa.

Ao Prof. Dr. Dmontier Pinheiro Aragão Júnior, por fornecer material e me orientar sobre cadeia de suprimentos e jogo da cerveja.

Aos alunos do GLOG (Grupo de estudos e pesquisa em Logística) orientado pelo Prof. Dr. Dmontier Pinheiro Aragão Júnior, que tanto participaram de um experimento com a ferramenta desenvolvida neste trabalho, como utilizaram a ferramenta em um experimento durante a semana das engenharias da UFC Campus Russas e sempre se mostraram solícitos durante o desenvolvimento do trabalho.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Ao Miron Vranjes, product manager do Facebook, criador da ferramenta Beer Distribution Game Simulator que serviu de base para a ferramenta criada neste trabalho.

Agradeço a todos os professores por me formar, não só apenas como profissional, mas como ser humano. E por todo o conhecimento transmitido ao longo de 4 anos.

“Se vi mais longe foi por estar de pé sobre os ombros de gigantes.”

(Isaac Newtom)

RESUMO

Este trabalho mostra uma maneira de como se pode utilizar blockchain para mitigar o efeito chicote em cadeia de suprimentos. O efeito chicote pode levar a um excesso de investimento em estoque, baixa customização de serviços, perda de vendas, inatividade do setor de transporte entre outros problemas. Fazendo-se necessário, uma maneira eficaz de mitigar o problema e suavizar os prejuízos no mercado. Para isso foi desenvolvido um simulador de jogo da cerveja que conta com o auxilio de blockchain para armazenar os dados e as transações durante o jogo. O simulador foi desenvolvido com dois modos, um na qual é possível recuperar dados da blockchain para auxiliar na previsão de demanda e o outro modo na qual não é possível recuperar dados da blockchain. Experimentos foram realizados com dois perfis de usuários, um perfil era de usuários que não possuíam experiência com jogo da cerveja e outro grupo com usuários que possuíam experiência com jogo da cerveja. Em cada experimento os usuários jogavam uma partida sem o auxílio da blockchain para realizar previsão de demanda e outra partida com o auxílio da blockchain. Os resultados mostraram que em 3 dos 4 grupos que usaram o simulador, houve uma diminuição nos problemas causados pelo efeito chicote, o que leva a conclusão que a blockchain é uma boa ferramenta para auxiliar no compartilhamento de informações em cadeias de suprimentos e ajudar a mitigar o efeito chicote.

Palavras-chave: Cadeia de suprimentos. Efeito Chicote. Blockchain. Jogo da Cerveja.

ABSTRACT

This paper shows a way of using blockchain to mitigate the bullwhip effect in the supply chain. The bullwhip effect can lead to excess inventory investment, low customization of services, loss of sales, downtime of the transportation industry among other problems. It is necessary, an effective way to mitigate the problem and smooth the losses in the market. For this was developed a beer game simulator with the help of blockchain to store the data and transactions during the game. The simulator was developed with two modes, one in which it is possible to recover data from the blockchain to help with demand forecasting and the other way in which blockchain data can not is possible to recover data. Experiments were carried out with two profiles of users, a profile was of users who had no experience with beer game and another group with users who had experience with beer game. In each experiment the users played a game without the aid of blockchain to carry out forecast of demand and another match with the aid of blockchain. The results showed that in 3 of the 4 groups that used the simulator, there was a decrease in the problems caused by the bullwhip effect, which leads to the conclusion that blockchain is a good tool to aid in the sharing of information in supply chains and to help mitigate the bullwhip effect.

Keywords: Supply Chain. Bullwhip Efect. Blockchain. Beer Game.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Um problema que pode causar grandes prejuízos na indústria é o chamado efeito chicote, que consiste em uma grande variabilidade nas encomendas de um determinado produto em relação as suas vendas a medida que se percorre a cadeia de suprimentos. Isso acontece principalmente devido a imprecisões durante o processo de previsão de demanda realizado pelos próprios membros da cadeia. Uma cadeia de suprimentos compreende os responsáveis por fabricar, distribuir e vender um determinado produto e geralmente é composta por fabricante, distribuidores, atacadistas e varejistas (LEE *et al.*, 1997).

Como exemplo do efeito chicote Lee *et al.* (1997) citam o caso em que executivos da *Procter & Gamble* examinaram os padrões das encomendas na cadeia de suprimentos da fralda *Pampers*. Eles viram que existia uma pequena flutuação nas encomendas dos varejistas, mas quando analisaram a nível de distribuição ficaram assustados com o grau de variabilidade das encomendas e quando analisaram as suas próprias encomendas de matéria-prima viram que a variação era ainda maior. E tudo isso traz uma dificuldade a mais no gerenciamento de produção e estoque.

Algumas pesquisas como a de Lee *et al.* (1997) sugerem maior troca de informações entre os membros da cadeia para mitigar o efeito chicote, assim, este trabalho propõe o uso da *blockchain* como ferramenta para auxiliar na comunicação, através de um simulador de jogo da cerveja. Jogo da cerveja é um RPG (*Role-playing game*) criado para ensinar estudantes e executivos a lidar com o efeito chicote em cadeias de suprimentos (STERMAN, 1992). Ele foi escolhido por representar bem o processo de compra e venda na cadeia dando um bom retorno sobre a eficácia da *blockchain* no combate ao efeito chicote.

Este trabalho tem por objetivo geral utilizar *blockchain* para diminuir o efeito chicote em cadeias de suprimentos, visando a troca de informações ao longo cadeia. Observando os seguintes objetivos específicos:

- Implementar um simulador *web* de jogo da cerveja, com e sem troca de informações.
- Aplicar o simulador em partidas com e sem troca de informações.
- Comparar os resultados nos dois modos do simulador, com e sem a *blockchain* sendo utilizada para auxiliar na comunicação.

Levando em consideração a indústria, o efeito chicote pode levar a um excesso de investimento em estoque, baixa customização de serviços, perda de vendas, inatividade do setor de transporte entre outros problemas. Fazendo-se necessário, uma maneira eficaz de mitigar o

problema e suavizar os prejuízos no mercado (LEE *et al.*, 1997). Nesse contexto, a *blockchain* consegue englobar diversas entidades de forma distribuída a fim de automatizar as trocas de informações, além de fornecer consistência e uma potente camada de segurança a esses dados. Assim, O projeto pode auxiliar empresas de bem de consumo com grandes canais de distribuição que estão sujeitas ao efeito chicote, mostrando como ele pode ser combatido através do uso de *blockchain*.

Para o desenvolvimento do presente trabalho foram realizados uma pesquisa bibliográfica e um estudo de caso. A pesquisa bibliográfica baseou-se em publicações da área de logística principalmente relacionadas a cadeia de suprimentos e efeito chicote, como também material bibliográfico relacionado a *blockchain*. O estudo de caso foi desenvolvido criando um simulador de jogo da cerveja e posteriormente aplicando este simulador com alunos do curso de engenharia de produção.

A pesquisa começa abordando os principais conceitos e trabalhos relacionados na fundamentação teórica (capítulo 2), primeiramente sobre cadeia de suprimentos (seção 2.1) e em seguida sobre *blockchain* (seção 2.2). Depois o simulador de jogo da cerveja proposto neste trabalho é mostrado (capítulo 3). Em seguida é explicado o experimento realizado para validação da proposta (capítulo 4), bem como os resultados do experimento (capítulo 5). Depois é feita uma comparação com trabalhos relacionados (capítulo 6). Por fim, é feita uma conclusão do trabalho juntamente com sugestões de trabalhos futuros (capítulo 7).

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção explicará os principais conceitos abordados no trabalho, ela é dividida em duas subseções, uma que fala sobre os conceitos relacionados ao jogo da cerveja e outra sobre conceitos relacionados a *blockchain*. A primeira subseção inicia falando sobre cadeia de suprimentos em logística, efeito chicote, causas e soluções para o efeito chicote e por fim explicará o jogo da cerveja. Já a segunda subseção aborda *blockchain* e a importância dela para a pesquisa, base criptográfica, definição e algumas aplicações.

2.1 Cadeia de suprimentos e o Jogo da Cerveja

2.1.1 Cadeia de Suprimentos (Supply Chain)

Uma cadeia de suprimentos compreende todos os membros que participam da fabricação, transporte e venda de um determinado produto. Geralmente essa cadeia é composta por um fabricante que manufatura um produto, um ou mais distribuidores que têm o objetivo de levar o produto a diferentes regiões geográficas, um ou mais atacadistas que são empresas que compram e vendem grandes volumes do produto e um ou mais varejistas que são as empresas que vendem o produto ao consumidor final.

A cadeia é dividida em vários níveis, na qual o fabricante está no nível mais alto e o varejista no nível mais baixo.

O grande desafio em uma cadeia de suprimentos é gerenciar a quantidade de produto que entra e sai dos estoques. Comumente os membros da cadeia encomendam os produtos baseados em suas vendas, o que a primeira vista parece bem racional, mas frequentemente causam estoques excedentes e escassos ciclicamente em todos os pontos da cadeia. Assim, o grande objetivo da cadeia de suprimentos é fornecer uma quantidade de produtos correta no tempo correto (STERMAN, 2015).

Para uma cadeia de suprimentos conseguir ser eficiente, várias pesquisas tratam de um conceito importante que é o de Gerenciamento de Cadeia de Suprimentos. Que é frequentemente relacionado a ligação e sincronização de operações de negócios entre os membros da cadeia de suprimentos, como por exemplo, compartilhamento de informações, materiais, mão-de-obra e equipamento de capital. Um bom Gerenciamento de Cadeia de Suprimentos requer bastante co-operação e coordenação entre as companhias numa cadeia (SVENSSON,

2005).

2.1.2 Efeito chicote em cadeias de suprimentos

Lee *et al.* (1997) citam o caso em que executivos da *Procter & Gamble* examinaram os padrões das encomendas na cadeia de suprimentos da fralda *Pampers*. Eles viram que existia uma pequena flutuação nas vendas dos varejistas, mas quando analisaram a nível de distribuição ficaram assustados com o grau variabilidade das encomendas, e quando analisaram as suas próprias encomendas de matéria-prima viram que a variação era ainda maior. Eles chamaram esse fenômeno de efeito chicote.

O efeito chicote ocorre quando a variabilidade da ordem de demanda de um produto, é amplificada a medida que se sobe na cadeia de suprimentos (LEE *et al.*, 1997). O que pode levar a um excesso de investimento em estoque, baixa customização de serviços, perda de vendas, inatividade do setor de transporte entre outros problemas.

2.1.2.1 Principais causas

Sterman (1992) mostra que as principais causas do efeito chicote são comportamentos irracionais como a falta de conhecimento sobre a capacidade de estoque ou sobre a demanda dos membros abaixo na cadeia. Já o estudo de Lee *et al.* (1997), mostra que essas causas são consequências de comportamentos racionais dos membros da cadeia de suprimentos, como a frequente atualização da previsão de demanda, encomenda de grandes lotes, a flutuação de preços, racionamento e jogos de escassez.

2.1.2.1.1 Frequente atualização da previsão de demanda

Geralmente um membro da cadeia de suprimentos prevê sua demanda a partir das suas vendas para o membro do nível mais abaixo e assim faz sua encomenda ao membro do nível mais acima. Porém essa previsão não reflete exatamente a demanda do cliente, e se esse processo é feito diariamente, reajustes nessa demanda também são feitos diariamente, causando uma imprecisão entre a demanda prevista e a real. Agora pensando na previsão de demanda de um membro acima na cadeia que repita o mesmo processo do membro anterior, as suas previsões serão realizadas em cima de demandas que já possuem uma certa imprecisão entre a demanda prevista e a real, o que causa uma variação ainda maior. E essa variação aumenta na medida do

fluxo *upstream*, e se torna ainda maior quando o atraso de entrega de produtos é grande (LEE *et al.*, 1997).

2.1.2.1.2 Encomendas em grandes lotes

Comumente empresas encomendam grandes lotes de mercadoria muitas vezes influenciadas pelo alto custo de transporte, assim, encomendar um caminhão totalmente cheio acaba saindo mais barato do um parcialmente ocupado. Porém grandes lotes acabam influenciando negativamente no efeito chicote. Já que grandes lotes podem não refletir a demanda real do cliente passando um *feedback* errado ao fornecedor, além de aumentar o tempo entre as encomendas. O comportamento ideal para diminuir o efeito chicote é realizar encomendas pequenas ciclicamente (LEE *et al.*, 1997).

2.1.2.1.3 Flutuação de preços

Alguns fabricantes costumam realizar promoções, às vezes para melhorar as vendas ou em outros momentos para reduzir estoques, entre vários motivos. O que acontece, é que diante dos preços baixos os clientes fazem grandes compras e quando o preço volta ao normal elas diminuem, às vezes porque os estoques ainda estão grandes ou às vezes porque os clientes estão esperando novas promoções. E esse comportamento causa um *feedback* que não reflete a demanda real do mercado, causando grandes variações nas demandas da cadeia de suprimentos (LEE *et al.*, 1997).

2.1.2.1.4 Racionamento e jogos de escassez

Quando a quantidade de produtos disponíveis é menor que a demanda, empresas normalmente racionam a venda desses produtos e atendem apenas uma parte da encomenda do cliente. Sabendo da escassez do produto, clientes reagem com encomendas para vários fornecedores e maiores que sua necessidade, e quando uma encomenda é atendida, o cliente cancela as outras. Tudo isso passa uma impressão errada da demanda do mercado e por muitas vezes faz com que os fabricantes aumentem sua produção para atender todos os pedidos, mas depois têm um grande prejuízo já que os pedidos vão sendo cancelados. Esse comportamento também acontece quando por jogada de marketing, fornecedores antecipam uma possível escassez do produto, os chamados jogos de escassez (LEE *et al.*, 1997).

2.1.2.2 Blockchain e o efeito chicote

Uma maior troca de informações pode ajudar bastante a combater os problemas apresentados na seção anterior. Com uma rede *blockchain* seria possível saber exatamente o fluxo de produtos em cada membro da cadeia de suprimentos, assim ela seria uma ótima ferramenta para auxiliar a previsão de demanda, como também ajudaria um membro a saber se um grande lote encomendado reflete realmente as vendas daquele cliente ou se é apenas para reduzir gastos com transporte ou uma reação a uma flutuação de preços. Outra aplicação seria confirmar a escassez de um produto e evitar os problemas que os jogos de escassez podem trazer.

2.1.3 Jogo da cerveja (Beer Game)

Uma das melhores formas de se entender como funciona o efeito chicote na indústria é através de um jogo de simulação chamado Jogo da Cerveja, que têm o objetivo de representar a produção e distribuição de cerveja (STERMAN, 1992).

O jogo foi resultado da pesquisa sobre dinâmica industrial de Jay Forrester na Sloan School of Management do MIT nos anos 60 e de lá pra cá já foi jogado por estudantes de graduação, gerentes de empresas, engenheiros de produção e funcionários públicos (STERMAN, 1992).

2.1.3.1 Início

O jogo possui várias versões, na utilizada neste trabalho é possível ter de 4 até centenas de jogadores, podendo ser jogado individualmente, ou por equipes. Cada jogador ou equipe representando um dos 4 quatro membros da cadeia de suprimentos: fabricante, distribuidor, atacadista e varejista.

2.1.3.2 Funcionamento do jogo

Durante o jogo, o varejista faz encomendas ao atacadista e fornece caixas de cerveja ao consumidor final a uma demanda aleatória. O atacadista supre as necessidades do varejista e encomenda caixas de cerveja ao distribuidor. O distribuidor fornece cerveja ao atacadista e encomenda ao fabricante. E o fabricante fornece ao distribuidor e fabrica cerveja.

A ordem cronológica do jogo é realizada por semanas. A cada semana cada jogador entrega caixas ao seu cliente e faz uma encomenda ao seu fornecedor. Todas as transações são

feitas em quantidade de caixas de cerveja.

Para poder simular o *lead time* que acontece no mundo real, os pedidos possuem um atraso de duas semana até chegar ao fornecedor, e as entregas possuem um atraso de transporte de duas semanas até chegar ao cliente.

Em cada semana é calculado um custo de manutenção de estoque, um custo por demanda não atendida e um custo por unidades em transporte. Os custos e o total de semanas são definidos previamente. Ao final do número de semanas pré-estabelecido quem tiver acumulado o menor custo total vence o jogo.

Os jogadores não podem conversar entre si, portanto não podem coordenar suas ações e somente o varejista sabe a demanda do consumidor final.

2.1.3.3 Resultados Frequentes

Sterman (1992) mostra que os jogos diferem quantitativamente, mas apresentam os mesmos padrões de comportamento.

Geralmente os níveis de estoque vão baixando em sequência do varejista ao fabricante. Daí os jogadores aumentam suas encomendas e os estoques acabam esvaziando, o que gera encomendas e atrasos de entrega ainda maiores. As fábricas aumentam suas produções drasticamente, os pedidos são atendidos e gera um novo acúmulo de estoque. Outro ciclo é iniciado.

Sterman (1992) cita três padrões de comportamento:

- Oscilação: Encomendas e estoques apresentam grandes flutuações de amplitude, com um período médio de cerca de 20 semanas.
- Amplificação: A amplitude e a variação dos pedidos aumentam constantemente do cliente para o varejista até a fábrica. A taxa máxima de pedidos na fábrica é, em média, mais que o dobro da taxa de pedidos no varejo.
- Atraso de fase: A taxa de pedido demora mais a atingir o máximo à medida que se move do varejista para a fábrica.

2.2 Blockchain

2.2.1 Base Criptográfica

A criptografia compreende todo um conjunto de técnicas e princípios que visam proteger dados e informações. Segundo o dicionário Dicio criptografia é "conjunto de regras e técnicas utilizado para cifrar, para codificar a escrita, transformando-a num tipo de código incompreensível para quem não está autorizado a ter acesso ao seu conteúdo"(CRIPTOGRAFIA, 2018). Ou seja, se duas pessoas trocam mensagens, a criptografia tem o papel de modificar a mensagem de tal forma que somente os envolvidos na comunicação consigam entender a mensagem. Já para Lindell e Katz (2014) criptografia é o estudo de técnicas matemáticas para assegurar informações digitais, sistemas e computação distribuída de ataques adversários.

Dentre as técnicas criptográficas, duas bastante poderosas são importantes para este estudo, funções *hash* e ponteiros *hash*, pois juntas, elas constituem a base da *blockchain*.

2.2.1.1 Funções Hash

Uma função *hash* segundo Narayanan *et al.* (2016) é uma função matemática que possui as seguintes propriedades:

- A entrada pode ser de qualquer tamanho.
- A função irá produzir uma saída de tamanho fixo.
- Ela é eficientemente computável. Intuitivamente isso significa que, para uma determinada *string* de entrada, é possível descobrir a saída da função *hash* em uma quantidade razoável de tempo. Mais tecnicamente, computar o *hash* de uma *string* de *n-bits* deve ter um tempo de execução O(n).

Mas para esta pesquisa, uma função *hash* geral como descrito acima não ajuda muito, para o objetivo desta pesquisa precisa-se de uma função *hash* criptográfica, que além das três propriedades existentes na função hash geral, ela possui três propriedades adicionais (NARAYANAN *et al.*, 2016).

- Resistência à colisão.
- *Hiding*.
- Amigável a desafios.

Resistência à colisão é a propriedade que uma função *hash* possui de dado duas entradas diferentes, ser impossível encontrar uma saída igual. É provado, que em função *hash* em

que a entrada pode ser de qualquer tamanho e a saída possui tamanho fixo, existem duas entradas que resultam em uma mesma saída. Mas apesar de existir, princípios estatísticos nos garantem que dependendo do tamanho da saída, é praticamente impossível acharmos duas entradas que produzam a mesma saída e é aí que reside a resistência a colisão (NARAYANAN *et al.*, 2016).

Por um princípio de contagem é fácil perceber que em uma função *hash*, existe mais de uma entrada que gera uma mesma saída, pois como a entrada de uma função hash pode ser de qualquer tamanho e a saída possui um tamanho fixo, é fácil perceber que o domínio da função (espaço de entrada) é bem maior que o da imagem (espaço de saída). Portanto é possível uma mesma saída ser gerada por duas entradas diferentes (NARAYANAN *et al.*, 2016).

Narayanan *et al.* (2016), Formaliza resistência a colisão como a seguir:

Uma função *hash* H() é dita ser resistente à colisão se for impossível encontrar dois valores, x e y, de modo que x y, então H (x) = H (y).

Hiding é a propriedade que garante que dado o *hash* de saída, não é possível encontrar a *string* de entrada.

Em algumas funções *hash*, especialmente as que possuem um conjunto de entrada bastante pequeno, é possível descobrir qual entrada originou uma determinada saída. Um exemplo disso é a criptografia de um lançamento de dados. Depois de alguns lançamentos é possível deduzir qual face saiu apenas olhando a saída da função *hash*.

Narayanan *et al.* (2016), formaliza *hiding* como a seguir:

Uma função *hash* H() tem a propriedade *hiding*, se: quando um valor secreto r é escolhido a partir de uma distribuição de probabilidade que possui alta entropia, então, dado H (r || x), não é possível encontrar x.

Onde || significa concatenação, e alta entropia significa que o conjunto de informação é bem abrangente.

Já uma função *hash* é dita amigável a desafios quando a partir de uma entrada parcial se queira chegar a um valor de saída alvo, é muito difícil encontrar outro valor que atinja esse alvo (NARAYANAN *et al.*, 2016).

Esse é um conceito bastante complexo e é utilizado na mineração de bitcoins.

Narayanan *et al.* (2016), formaliza o conceito de ser amigável a desafios como a seguir:

Uma função *hash* H() é dita ser amigável a desafios se para cada possível valor de saída de n-bits y, e se k for escolhido de uma distribuição com alta entropia, então não é possível

encontrar x de forma que H(k x) = y. Em tempo significativamente inferior a 2n.

Se a função *hash* não for especificada, a padrão a ser utilizada é a Sha-256 de (PUB, 2012)

2.2.1.2 Ponteiros Hash

São estruturas de dados que possuem um ponteiro que indica a posição onde uma informação está armazenada e um código *hash* do conteúdo dessa informação. O ponteiro é utilizado para recuperar os dados enquanto o *hash* fornece uma maneira de verificar a integridade destes dados (NARAYANAN *et al.*, 2016).

Ponteiros *hash* são úteis para a criação de qualquer outra estrutura de dados mais complexa, como listas encadeadas, filas, árvores binárias e etc. Com o diferencial de fornecer uma segurança a mais contra corrupção dos dados.

Esse tipo de ponteiros é a base para a construção de uma *blockchain*.

2.2.2 Definição de Blockchain

Blockchain é uma estrutura de dados que consiste em uma lista encadeada (CORMEN *et al.*, 2009) através de ponteiros *hash*. São blocos de conteúdo que possuem um ponteiro para o bloco anterior bem como um hash deste bloco anterior. A cabeça consiste em um ponteiro hash que aponta para o bloco mais recente. E no início é criado um bloco Gênesis, que marca o início da cadeia de blocos (NARAYANAN *et al.*, 2016).

Sua grande vantagem é a resistência à fraude. Pois como cada bloco possui um *hash* do bloco anterior, é possível verificar se o conteúdo de um bloco foi alterado. Então se algum atacante quiser fraudar qualquer bloco da cadeia, ele terá que alterar todos os outros blocos, e isso é uma forte camada de segurança que pode ser aplicado em várias áreas do conhecimento (NARAYANAN *et al.*, 2016).

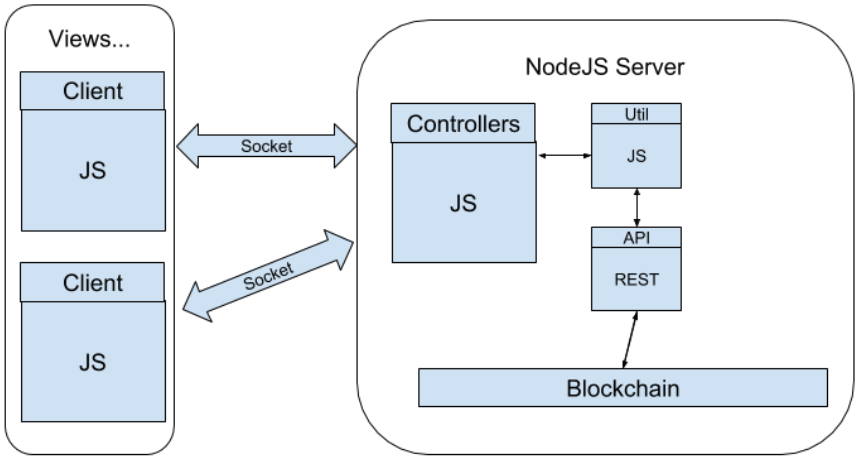
A *Blockchain* foi criada para servir como base do *Bitcoin* (NAKAMOTO, 2008), por garantir uma forte resistência a fraude. E o *Bitcoin* é o grande exemplo de uso da *Blockchain* hoje em dia, apesar que desde sua criação estejam se desassociando e a *Blockchain* sendo usada para vários outros propósitos.

3 LEDGER BEER GAME: UM SIMULADOR DE JOGO DA CERVEJA

3.1 Arquitetura

O simulador proposto neste trabalho está esquematizado na figura 1.

Figura 1 – Arquitetura do simulador



Fonte: Autor

O simulador foi desenvolvido no modelo arquitetural MVC (Model-view-controller) e possui os seguintes componentes:

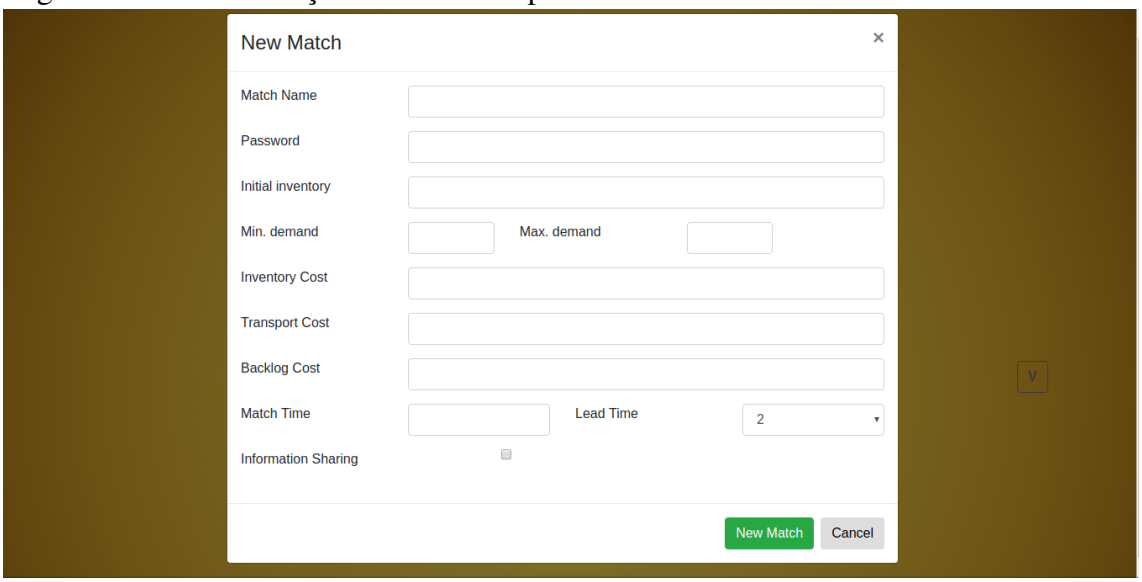
- Servidor: Servidor NodeJS responsável por abrigar a aplicação javascript.
- Clientes: Usuários que utilizam o simulador.
- Views: Camada da aplicação que é mostrada aos clientes, as telas do jogo.
- Controllers: Classes e métodos javascript que trocam mensagens com os clientes e utilizam a classe Util para se comunicar com a *blockchain*.
- Util: Classe que serve como ponte entre os controlerrs e a *blockchain*, quando um método é chamado é feita uma requisição a API REST.
- API REST: Interface usada para comunicação com a *blockchain*. Todas as operações realizadas na *blockchain* são feitas através da API.
- Sockets: Canais de comunicação entre os clientes e o servidor.
- Blockchain: Usada para armazenar todas as informações do jogo. Partidas, usuários,

produtos e transações são armazenadas nela e podem ser recuperadas.

3.2 O jogo

O jogo se inicia com a definição de alguns parâmetros que conduzirão a partida, eles estão na figura 2:

Figura 2 – Tela de criação de uma nova partida

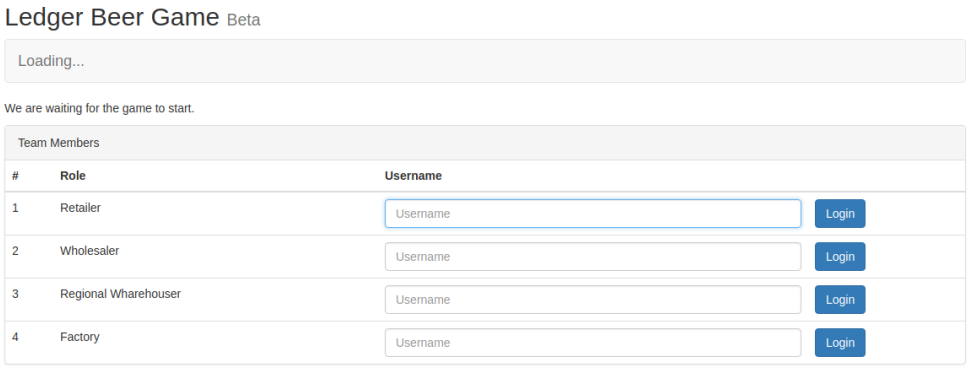


Fonte: Autor

- Initial inventory: Quantas caixas de cerveja cada membro vai ter no seu estoque inicial;
- Min. demand e max. demand: A partir da terceira semana a demanda do consumidor final será aleatória entre um intervalo definido no início do jogo;
- Inventory cost: O custo por semana de cada caixa em estoque.
- Transport cost: O custo por semana de cada caixa em transporte;
- Backlog cost: O custo por semana de cada caixa que não foi atendida no pedido do cliente. Sempre que o cliente fizer o pedido, por motivos de estoque, o fornecedor pode ou não atender todo o pedido, assim cada caixa que faltar será contabilizado um custo pelos transtornos gerados;
- Lead time: Quantas semanas um pedido passará em transporte;
- Match time: Quantas semanas durará o jogo. Após a definição dos parâmetros e todos os jogadores se conectarem ao jogo, a partida é definitivamente iniciada.

Após a criação da partida, os jogadores devem escolher uma posição na cadeia de suprimentos e *logar* na partida. A tela de *login* está na figura 3.

Figura 3 – Tela de criação de uma nova partida



Fonte: Autor

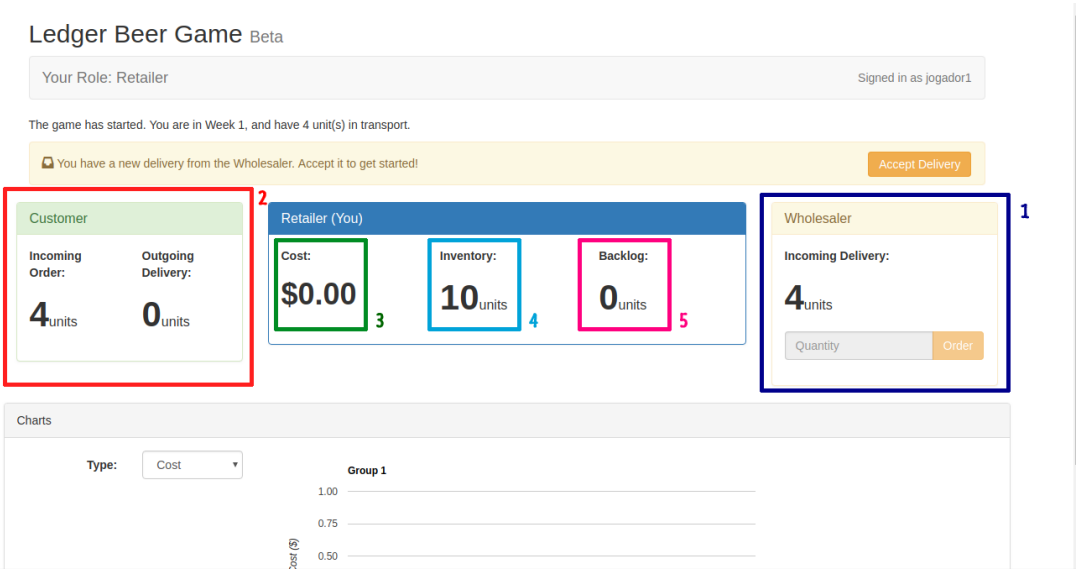
Cada jogador deve inserir um nome de usuário no campo de entrada respectivo ao papel escolhido por ele. Após apertar o botão de login, o sistema verifica se já existe alguém inserido naquela posição, e caso o papel esteja livre, um usuário é criado na *blockchain* e associado àquela posição na cadeia de suprimentos.

Depois que todas as posições da cadeia de suprimentos possuem um jogador associado, o administrador da partida pode iniciá-la. O jogo é dividido em semanas, e para cada semana, há uma sequência de passos do jogador que consiste:

- Receber entrega
- Atender pedido
- Enviar pedido
- Esperar os outros jogadores terminarem seus turnos

Na figura 4 é mostrada a tela do usuário durante a partida. O processo se inicia com o jogador recebendo uma entrega que chega do seu fornecedor, a quantidade de unidades que estão sendo recebidas é mostrada no quadro 1, após isso, os produtos que estão chegando são adicionados ao estoque, quadro 4. Em seguida o pedido do cliente, que é mostrado no quadro 2, deve ser atendido, a quantidade solicitada é retirada do estoque e colocada em transporte e caso o estoque não seja suficiente para atender o pedido, a quantidade deficitária é adicionada ao *backlog* (pedido em atraso, que é indicado no quadro 5). Após o envio dos produtos é hora do jogador fazer um pedido ao fornecedor, com o pedido enviado o jogo atualiza os custos, que é mostrado no quadro 3, o cálculo é feito baseado no estoque, *backlog*, e a quantidade de produtos

Figura 4 – Tela do usuário em um jogo



Fonte: Autor

em transporte e levando em consideração os custos informados na criação da partida. Após esse processo o jogador aguarda todos os outros terminarem seus turnos, após todos concluírem, uma nova semana é iniciada.

Essas atividades são realizadas até que atinja o número de semanas definido no início do jogo, então o jogador com o menor custo total vence a partida.

O jogo possui um sistema com *lead time* de quatro semanas, que é um atraso até o fornecedor atender um pedido bem como um atraso até a mercadoria chegar no cliente, ou seja, o pedido feito por um jogador na semana 1 apenas chegará ao fornecedor na semana 3, e os produtos enviados pelo fornecedor na semana 3 chegará no cliente na semana 5. Para assim, simular os atrasos de transporte e pedido em uma cadeia de suprimentos.

3.3 Blockchain e o jogo

Todo o jogo foi modelado para a *blockchain*, cada partida, produto e pedido são objetos que podem ser registrados nela. E cada usuário é um participante que pode realizar transações, ou seja, usar contratos inteligentes, como por exemplo, realizar um pedido a um fornecedor ou enviar produtos para um cliente. Toda modelagem da *blockchain* utilizada neste trabalho está contida no apêndice B.

Como todas as informações são inseridas na *blockchain*, qualquer um dos jogadores terá acesso as informações dos outros independentemente da posição que eles ocupam na cadeia de suprimentos. Custo, estoque, e pedidos são as informações disponíveis para consulta. Assim,

a *blockchain* é uma ferramenta de comunicação entre os jogadores e auxílio na previsão de demanda.

As tecnologias usadas para criação da *blockchain* foram o *Hyperledger Fabric* e o *Hyperledger Composer*.

3.3.1 Hyperledger Fabric

O *Fabric* é um *framework open source* que permite o desenvolvimento de redes *Blockchain* permissivas, ou seja, que permitem o controle dos usuários. Ele possui componentes como mecanismos de consenso, registro de banco de dados e controle de membros (MAHESHWARI, 2018).

Uma rede criada no *Hyperledger Fabric* possui os seguintes componentes (MAHESHWARI, 2018):

- Assets: Que são qualquer objeto que possua algum valor. Eles possuem basicamente um estado, que é os valores atuais dos seus atributos, e um proprietário que é o dono do asset.
- Shared Ledger. Armazena os estados e os proprietários dos assets e possui dois componentes:
  - World State. Que funciona como um banco de dados armazenando o estado do sitema.
  - Blockchain. Que armazena um log de todas as transações realizadas.
- Smart Contract. São chamados de chaincode e são métodos que são invocados quando se quer acessar o Ledger.
- Peer Nodes. São nós na rede que executam chaincodes, acessam o ledger, validam transações e servem como interface para outras aplicações.
- Channel. É um conjunto de peers que permitem criar transações separadas que não são acessíveis a outros peers.
- Organizations. Uma rede no Hyperledger Fabric é um conjunto de organizações que possuem vários peers, que contribuem para a manutenção da rede e fornecem certificados digitais para que a rede possa ser acessada.
- Membership Services Provider (MSP). É um serviço que funciona como uma autoridade certificadora. Sem os certificados, nenhuma entidade pode executar transações na rede e é isso que a torna permissiva.
- Ordering Server. Serviço que garante a entrega das transações a todos os nós da rede.

3.3.2 Hyperledger Composer

O *Composer* é um conjunto de ferramentas que auxiliam na criação e no gerenciamento de aplicações que utilizam o *Hyperledger Fabric*. Sua principal vantagem é ocultar a complexidade do desenvolvimento dessas aplicações, além de oferecer ferramentas para modelagem das tais. Como uma linguagem orientada a objetos (*Composer Modelling Language*) para modelar toda a aplicação, e oferecer suporte para o uso de javascript no desenvolvimento das transações.

Para gerar uma definição de uma *Business Network*, o *Composer* precisa de quatro tipos de arquivos (MAHESHWARI, 2018):

- Network Model File (.cto). Arquivo de modelagem da rede, possui todos os assets, participantes e as informações de entrada de uma transação.
- JavaScript file (.js). Nele contém as funções executadas quando uma transação é invocada.
- ACL file (.acl). Contém as permissões de acesso para cada asset ou transação.
- Query file (.qry). Arquivo que contém as querys de busca que podem ser chamadas pelas aplicações.

Usando estes arquivos, o *Composer* gera um *Business Network Archive* (.bna) que contém as definições e as transações executáveis de uma *business network*. O arquivo .bna pode ser exportado e implantado no *Hyperledger Fabric* (MAHESHWARI, 2018).

Outra ferramenta bastante útil do *Composer* é o *Composer Rest Server*, que a gera uma API REST que funicona como uma interface entre a aplicação e a rede.

4 PROCEDIMENTO METODOLÓGICO

Para implementar e testar a *blockchain* como ferramenta de combate ao efeito chicote, foi construído um simulador de jogo da cerveja com duas configurações, uma sem *blockchain* como fonte de informação e outra com *blockchain* auxiliando os jogadores. Tudo isso seguindo os seguintes procedimentos.

4.1 Pesquisa Teórica

Primeiramente foi realizada uma pesquisa teórica, na qual foram lidos em um primeiro momento trabalhos relacionados a cadeia de suprimentos e efeito chicote para uma melhor compreensão do problema abordado. Em seguida, simuladores de jogo da cerveja foram analisados com o intuito de entender o funcionamento básico deste tipo de *software*. Por fim, a *blockchain* foi estudada, seu funcionamento, aplicações e ferramentas atuais de implementação foram pesquisadas para definir como aplicar esta tecnologia ao jogo da cerveja.

4.2 Projeto do simulador

O simulador foi projetado usando a técnica de projeto de sistemas, criação de diagramas (BEZERRA, 2015). Foram construídos diagramas de caso de uso, atividades, classes e um que representa a arquitetura do sistema, todos eles com o intuito de auxiliar na tarefa do desenvolvimento da aplicação e são mostrados no apêndice A. Os requisitos do simulador foram obtidos através da leitura de artigos e da utilização de outros *softwares* parecidos.

4.3 Implementação do simulador

Depois de o simulador ter sido projetado, foi iniciado a sua implementação. Ele foi desenvolvido para a plataforma web utilizando a linguagem de programação JavaScript, um servidor NodeJS, um banco de dados MogoDB e uma API REST. A ferramenta Docker também foi utilizada para a criação da rede , nela, *conteirners* abrigavam os nós necessários para que a *blockchain* pudesse funcionar.

A Blockchain foi construída utilizando as ferramentas Hyperledger Fabric e Hyperledger Composer. O Composer foi utilizado para modelagem da Blockchain e o Fabric para criação dos nós necessários para a manutenção da blockchain.

4.4 Aplicação o simulador

O simulador foi utilizado por alunos do curso de Engenharia de Produção em dois momentos diferentes, o primeiro durante a semana das engenharias da UFC Campus Russas, na qual alunos sem experiência com jogo da cerveja utilizaram o simulador. No segundo momento, os alunos do GLOG (Grupo de Estudo e Pesquisa em Logística) utilizaram a ferramenta e diferentemente do primeiro grupo, eles possuíam experiência com jogo da cerveja. Em cada experimento, os alunos foram divididos em dois grupos de 4 pessoas, e cada pessoa representou um membro da cadeia de suprimentos em uma partida. Em um primeiro momento, com parâmetros bem definidos, foi realizada uma partida no simulador sem a troca de informações entre os jogadores. Em um segundo momento, com as mesmas equipes e os mesmos parâmetros da partida anterior, foi realizada uma outra partida, desta vez com cada jogador tendo na tela do simulador informações sobre as ações dos outros jogadores.

5 RESULTADOS

Os experimentos com o simulador foram divididos em dois momentos, em um primeiro momento o jogo foi usado por alunos sem experiência em jogo da cerveja, ou seja, alunos que nunca tiveram jogado. Em um segundo momento quem participou dos experimentos foram alunos do GLOG (Grupo de Estudos e Pesquisa em Logística) que já tinham usado o jogo da cerveja diversas vezes. Cada experimento foi dividido em duas partes, na primeira os usuários jogaram sem compartilhamento de informações, e na segunda com compartilhamento de informações e os usuários podendo acessar os dados gravados na *blockchain.*

Os resultados das duas vezes que o simulador foi aplicado estão nas seções seguintes.

5.1 Usuários sem experiência com jogo da cerveja

Para o experimento com usuários sem experiência, estiveram presentes 8 voluntários na qual foram divididos em 2 cadeias que utilizaram o simulador simultaneamente. Houve premiação para a cadeia que tivesse o menor custo acumulado ao final do experimento.

5.1.1 Sem compartilhamento de informações

A primeira aplicação do jogo se deu sem uso da *blockchain* para compartilhamento de informações. Por problemas técnicos do jogo, a cadeia 1 conseguiu jogar apenas 6 semanas. Já a cadeia 2 jogou por 12 semanas.

Para análise, foram consideradas semanalmente quantidade de estoque, pedidos realizados, pedidos recebidos, demanda não atendida, quantidade de produtos em transporte, custos semanais e custo total acumulado.

Os custos eram de $1 por unidade em transporte, $2 por unidade em estoque e $3 por unidade não atendida. Os gráficos obtidos nesta aplicação são mostrados nas figuras 5 e 6.

Na cadeia 1 devido ao pouco tempo de jogo, não se dá pra verificar muito bem o efeito chicote, há apenas uma variabilidade nos estoques da fábrica e do varejista.

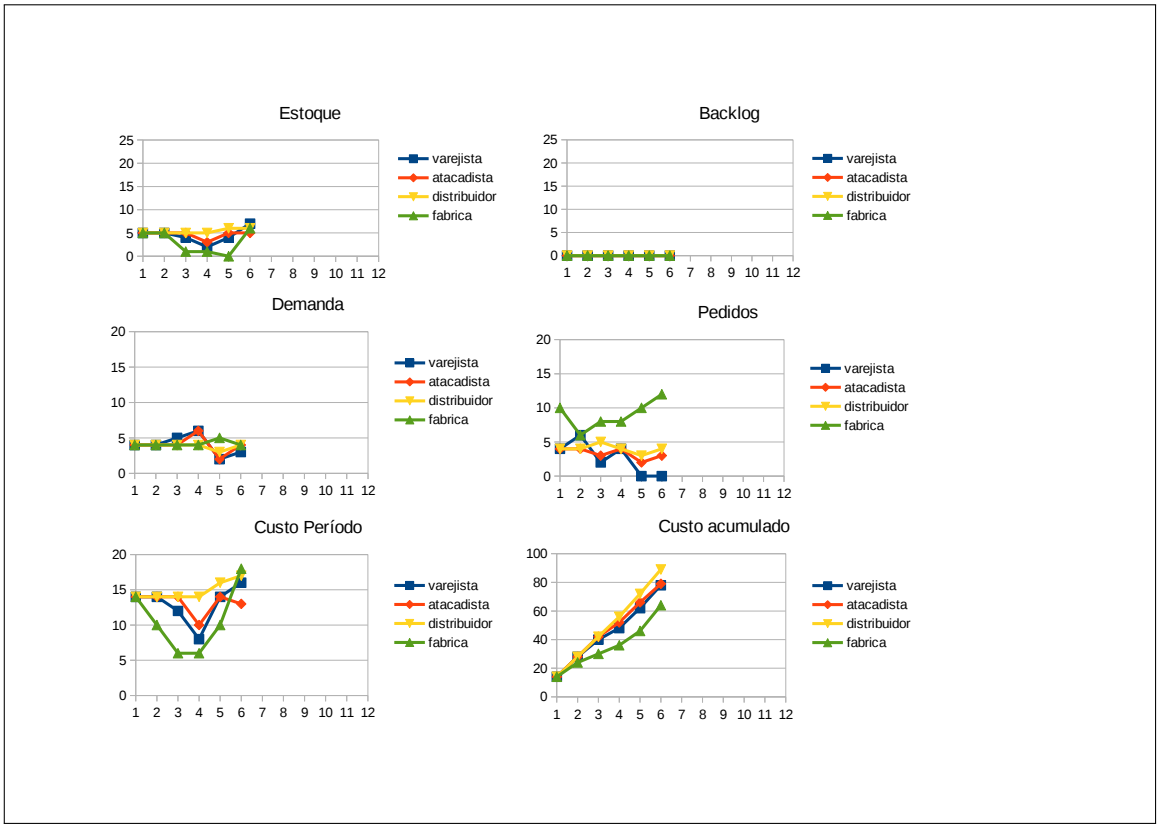
Já na cadeia 2 é bem nítido os problemas do efeito chicote ao longo da cadeia inteira, o estoque do varejista saiu de 5, ficou variando entre 0 e -3 e terminou devendo -8 ao cliente final. O atacadista sofreu com atrasos durante todo o jogo chegando a ter -12 de *backlog*. O distribuidor conseguiu manter um bom nível de estoque durante quase toda partida porém no fim terminou com atraso de 9. O efeito chicote foi mais nítido na fábrica que chegou a ter -7 em

estoque e terminou com 24 unidades, uma grande variação.

Com relação aos pedidos na cadeia 2, houve uma grande variabilidade no varejista e na fábrica, os pedidos do varejista variaram entre 0 e 12 e os da fábrica entre 0 e 15. Já os pedidos do atacadista variaram entre 2 e 8 enquanto os do distribuidor entre 0 e 8.

Nessa aplicação foi verificado que os custos não estavam condizendo bem com a realidade, a punição de $2 para produtos no estoque e apenas $3 para cada produto não atendido fez com que quem tivesse produtos no estoque obtivessem mais custos que quem tivesse poucos produtos em atraso, mas na vida real, o atraso é bem mais caro que os custos de estoque. E tudo isso acabou fazendo com que quem tivesse o melhor rendimento no jogo não terminasse vencedor.

Figura 5 – Resultados obtidos na aplicação com usuários sem experiência e sem compartilhamento de informações na cadeia 1

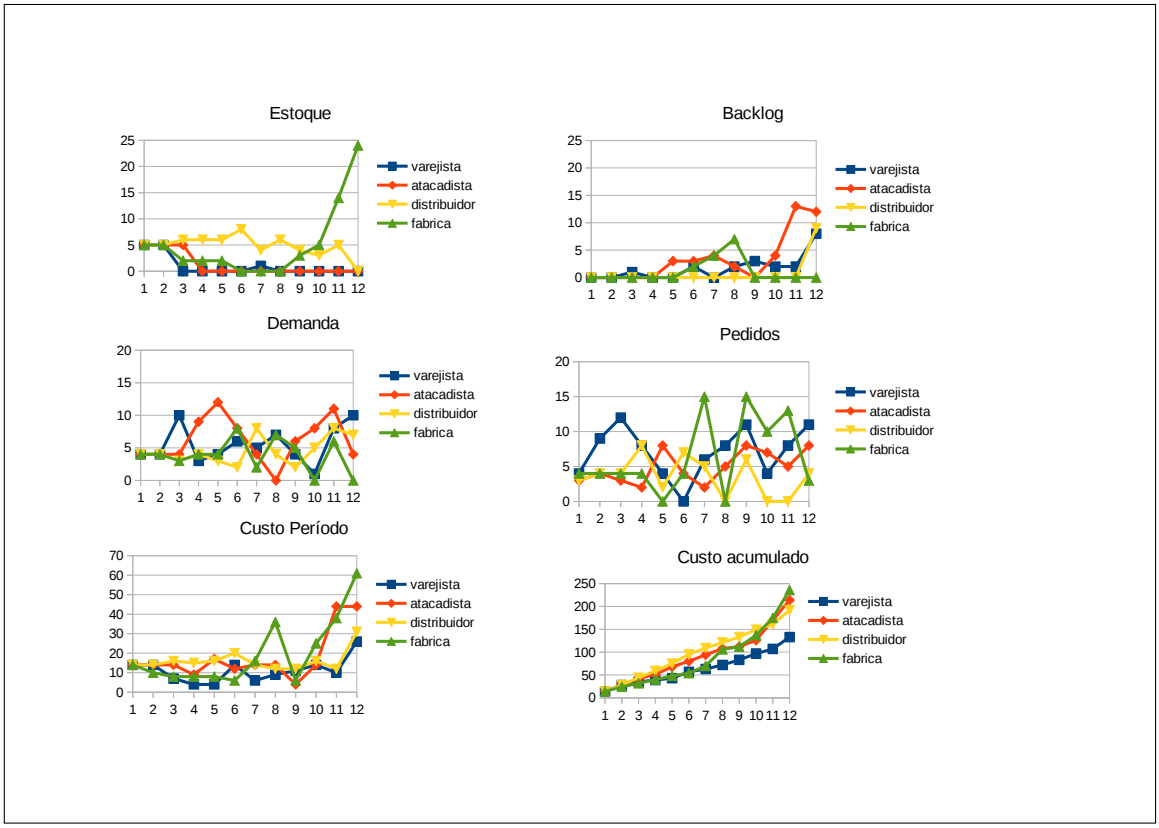


Fonte: o autor.

5.1.2 Com compartilhamento de informações

Na segunda aplicação os usuários além das informações retiradas da *blockchain*, os usuários podiam conversar entre si e definir a melhor estratégia para prever a demanda do cliente

Figura 6 – Resultados obtidos na aplicação com usuários sem experiência e sem compartilhamento de informações na cadeia 2



Fonte: o autor.

final e fazer com que os pedidos e os produtos caminhem na cadeia de suprimentos com o menor custo possível.

Os custos nesta aplicação foram de $1 para os produtos em transporte, $3 para os produtos em estoque, $6 para os pedidos não atendidos.

Para análise, foram consideradas semanalmente quantidade de estoque, pedidos realizados, pedidos recebidos, demanda não atendida, quantidade de produtos em transporte, custos semanais e custo total acumulado. Os resultados obtidos são mostrados nas figuras 7 e 8

Na cadeia 1 mesmo com o compartilhamento de informações foi possível perceber o efeito chicote. O estoque do varejista começou em um nível bom e terminou alto. Já os outros 3 membros da cadeia tiveram estoques negativos no começo e terminaram com bastante excedente, o atacadista saiu de um estoque de -2 e terminou com 17, o distribuidor chegou a ter estoque de -7 e terminou com 11 e a fábrica chegou a ter um estoque de -11 e terminou com estoque de 19. No caso da cadeia 1 é bem nítido como o efeito chicote se propaga ao longo da cadeia de suprimentos, já que a variação de estoques foi maior a medida que se percorre a cadeia.

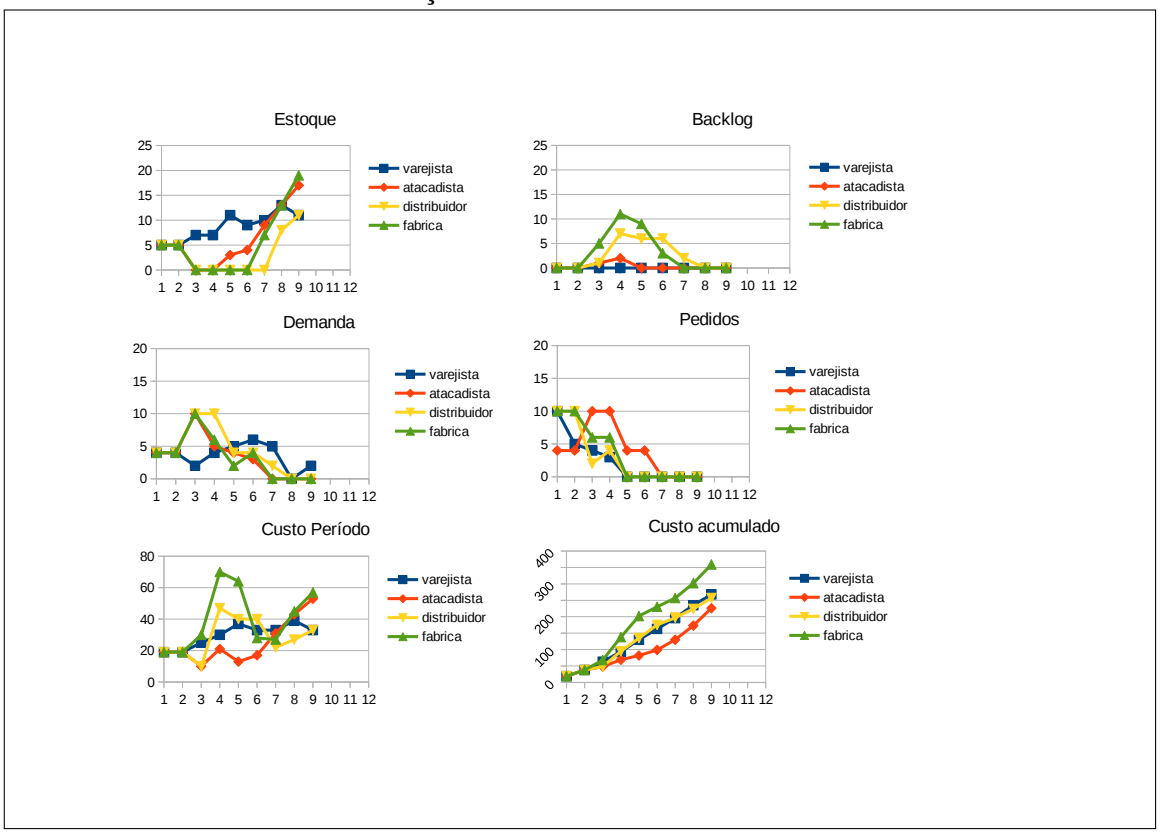
Em relação aos pedidos na cadeia 1, houve um padrão em que todos pediram grandes

quantidades nas primeiras semanas e foram diminuindo a medida que o jogo ia caminhando, sendo 0 nas últimas semanas, esse comportamento se deve ao caráter competitivo do jogo, pois como os jogadores sabiam que o jogo terminaria na semana 9, eles decidiram não mais fazer pedido para não aumentar o custo dos produtos em trânsito.

Com os custos mais realistas que na primeira aplicação, foi percebido que quem teve melhor atuação conseguiu ter menos custos, assim, o atacadista teve menor custo acumulado seguido de perto do varejista e do distribuidor. Já a fábrica como foi a que mais sofreu com o efeito chicote, também obteve um custo bem maior que os outros integrantes da cadeia.

Na cadeia 2, com exceção da fábrica que chegou a ter estoque -4 e terminou com 20 unidades, os outros conseguiram manter os níveis de estoque com uma variação boa. O varejista manteve o variando entre 1 e 8 até a última semana na qual ele ficou devendo 3 produtos. Já o atacadista e o distribuidor conseguiram manter o estoque entre 2 e 10.

Figura 7 – Resultados obtidos na aplicação com usuários sem experiência e com compartilhamento de informações na cadeia 1

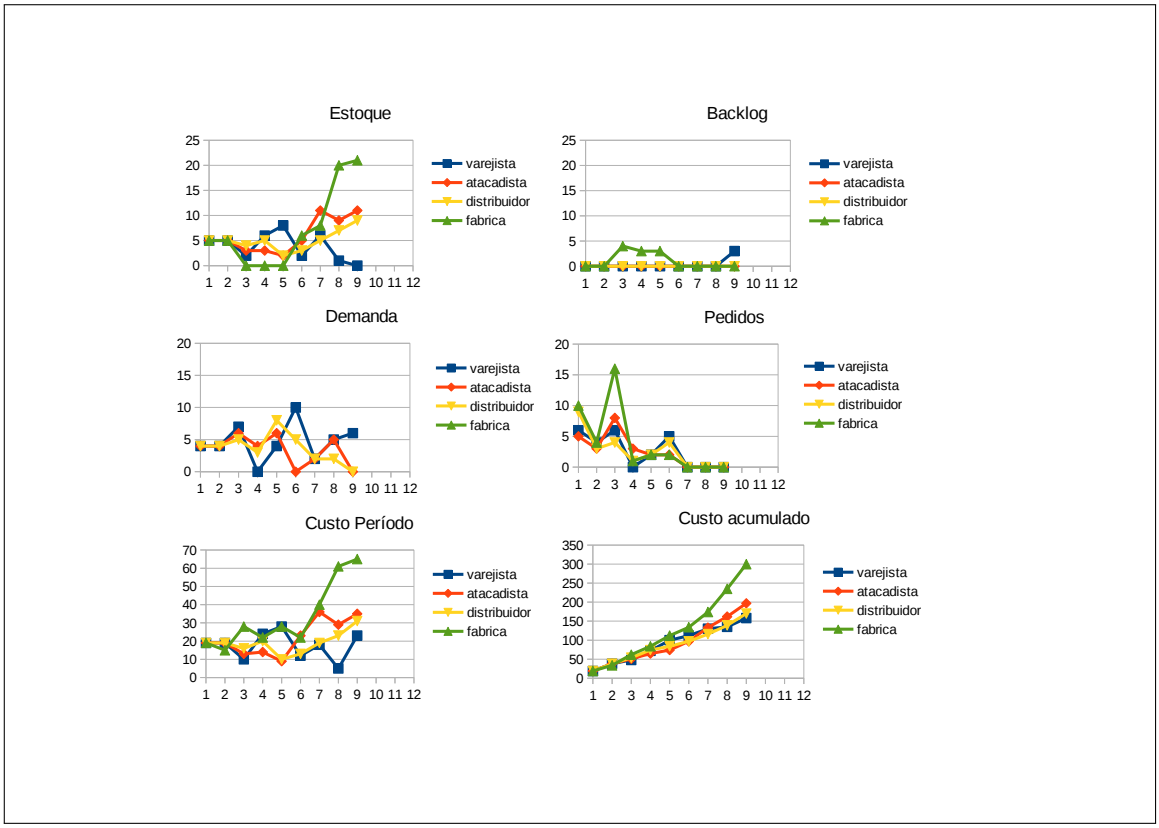


Fonte: o autor.

Com relação aos pedidos, mais uma vez a fábrica destoou dos demais membros, enquanto os pedidos da fábrica variaram entre 0 e 16, os pedidos dos outros membros variaram entre 0 e 10.

Como era de se esperar devido ao desempenho ruim da fábrica, ela também foi a que obteve mais custos, superando os demais em mais de $100. O varejista obteve o menor custo, com os outros dois membro próximos.

Figura 8 – Resultados obtidos na aplicação com usuários sem experiência e com compartilhamento de informações na cadeia 2



Fonte: o autor.

5.2 Usuários com experiência em jogo da cerveja

Para o experimento com usuários com experiência, estiveram presentes 8 voluntários na qual foram divididos em 2 cadeias que utilizaram o simulador simultaneamente. Houve prêmio para o jogador e para a cadeia com melhor desempenho.

Os usuários utilizaram o jogo em dois momentos, no primeiro sem compartilhamento de informações e no segundo com compartilhamento de informações.

5.2.1 Sem compartilhamento de informações

A primeira aplicação sem uso da *blockchain* para compartilhamento de informações durou 12 semanas para a cadeia 1 e 9 semanas para cadeia 2.

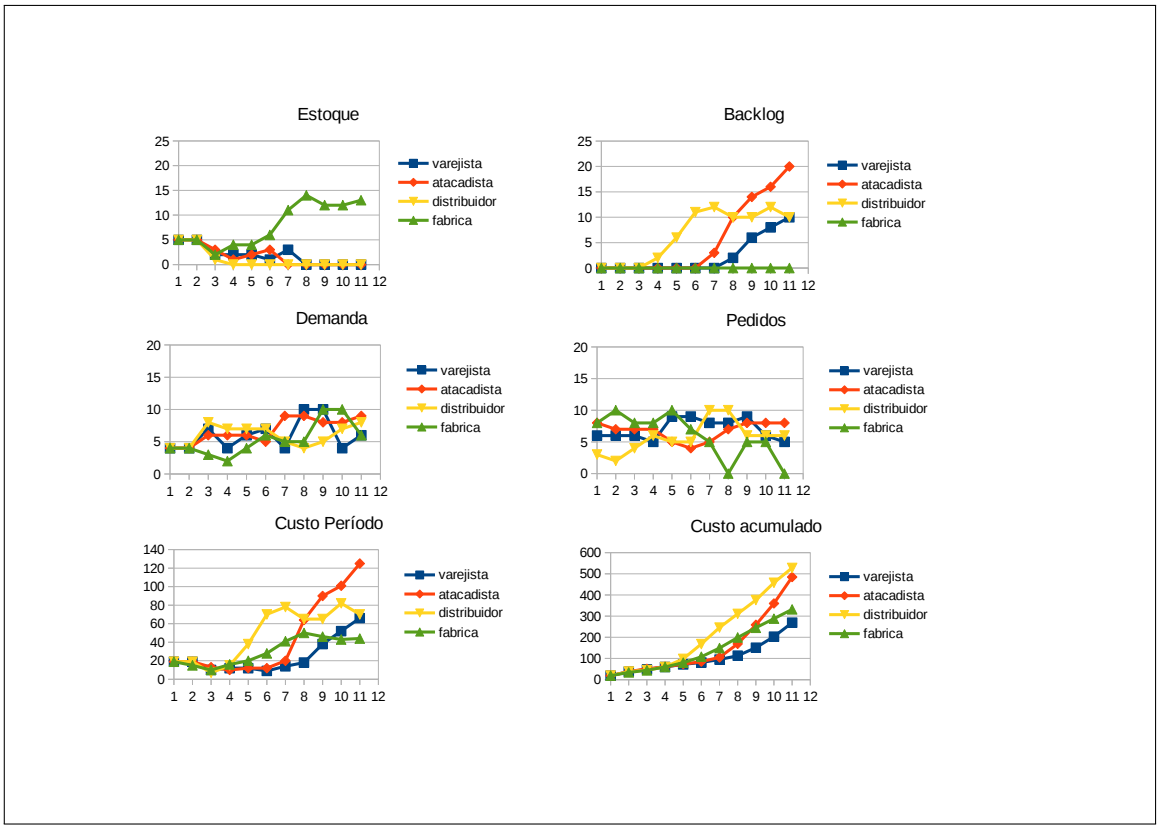
Para análise, foram consideradas semanalmente quantidade de estoque, pedidos realizados, pedidos recebidos, demanda não atendida, quantidade de produtos em transporte, custos semanais e custo total acumulado.

Os custos eram de $1 por unidade em transporte, $3 por unidade em estoque e $6 por unidade não atendida. Os resultados obtidos nesta aplicação são mostrados nas figuras 9 e 10.

Na cadeia 1 é possível notar o efeito chicote no estoque da fábrica e que os demais membros não conseguiram entregar os produtos e terminaram com altos níveis de atraso. O estoque do varejista e do distribuidor foi caindo até chegar a -10 e o estoque do atacadista chegou até -20. Já o estoque da fábrica chegou a ter 2 produtos e terminou com 13.

Com relação aos pedidos, não houve uma grande variação. Os pedidos do varejista variaram entre 5 e 9, do atacadista entre 4 e 8, do distribuidor entre 2 e 10 e da fábrica entre 0 e 10.

Figura 9 – Resultados obtidos na aplicação com usuários com experiência e sem compartilhamento de informações na cadeia 1



Fonte: o autor.

Quem obteve o menor custo acumulado foi o varejista seguido da fábrica, o distribuidor e o atacadista tiveram um desempenho pior.

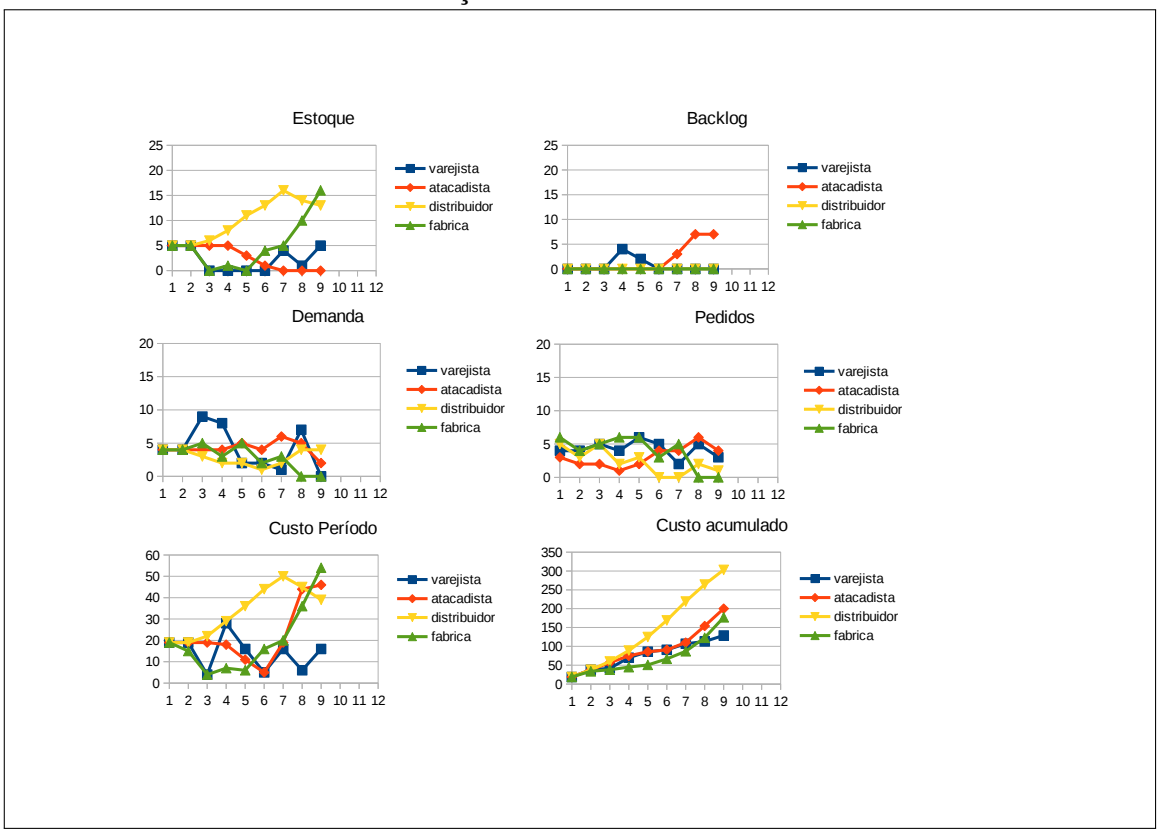
Na cadeia 2 os estoques de cada membro tiveram comportamentos diferentes. O

varejista variou um pouco entre estoque negativos e positivos, chegou a ter -4, mas terminou com 5. O atacadista sofreu com atraso, manteve o estoque em 5 até a quarta semana mas daí foi caindo até terminar devendo 7. O distribuidor sofreu com excesso de estoque, ele foi crescendo, chegou a ter 16 produtos e terminou com 14. Na fábrica deu pra notar o efeito chicote, ela chegou a ter nenhum produto em estoque e terminou com 16.

Em relação aos pedidos houve pouca variação, com todos variando entre 0 e 8.

O desempenho de todos exceto o distribuidor foi bem parecido em relação aos custos, o varejista teve o menor custo acumulado seguido de perto da fábrica e do atacadista.

Figura 10 – Resultados obtidos na aplicação com usuários com experiência e sem compartilhamento de informações na cadeia 2



Fonte: o autor.

5.2.2 Com compartilhamento de informações

A segunda aplicação com uso da *blockchain* para compartilhamento de informações durou 8 semanas para a cadeia 1 e 9 semanas para cadeia 2.

Para análise, foram consideradas semanalmente quantidade de estoque, pedidos realizados, pedidos recebidos, demanda não atendida, quantidade de produtos em transporte,

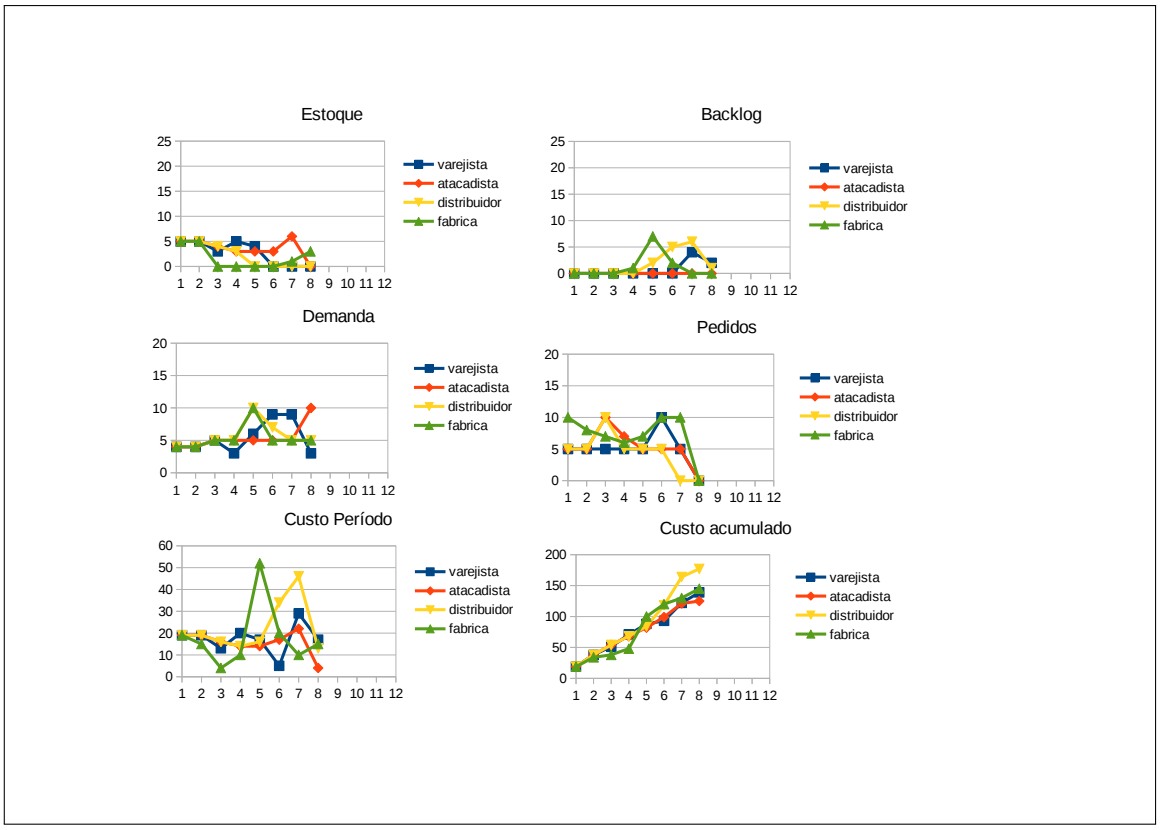
custos semanais e custo total acumulado.

Os custos eram de $1 por unidade em transporte, $3 por unidade em estoque e $6 por unidade não atendida. Os gráficos obtidos nesta aplicação são mostrados nas figuras 11 e 12.

Na cadeia 1 os estoques variaram um pouco entre positivos e negativos, mas com poucos produtos em atraso. O atacadista manteve seu estoque sempre positivo chegando até 6 e terminando em 0. Os demais, o varejista variou entre 5 e -4, o distribuidor começou com 5 chegou a ter -6 e terminou com -1 e a fábrica começou caindo chegando até -7 e terminou com 0. Esses números mostram um efeito chicote bem mais fraco do que foi visto em outras aplicações.

Em relação aos pedidos houve pouca variação, todos variaram entre 0 e 10.

Figura 11 – Resultados obtidos na aplicação com usuários com experiência e com compartilhamento de informações na cadeia 1



Fonte: o autor.

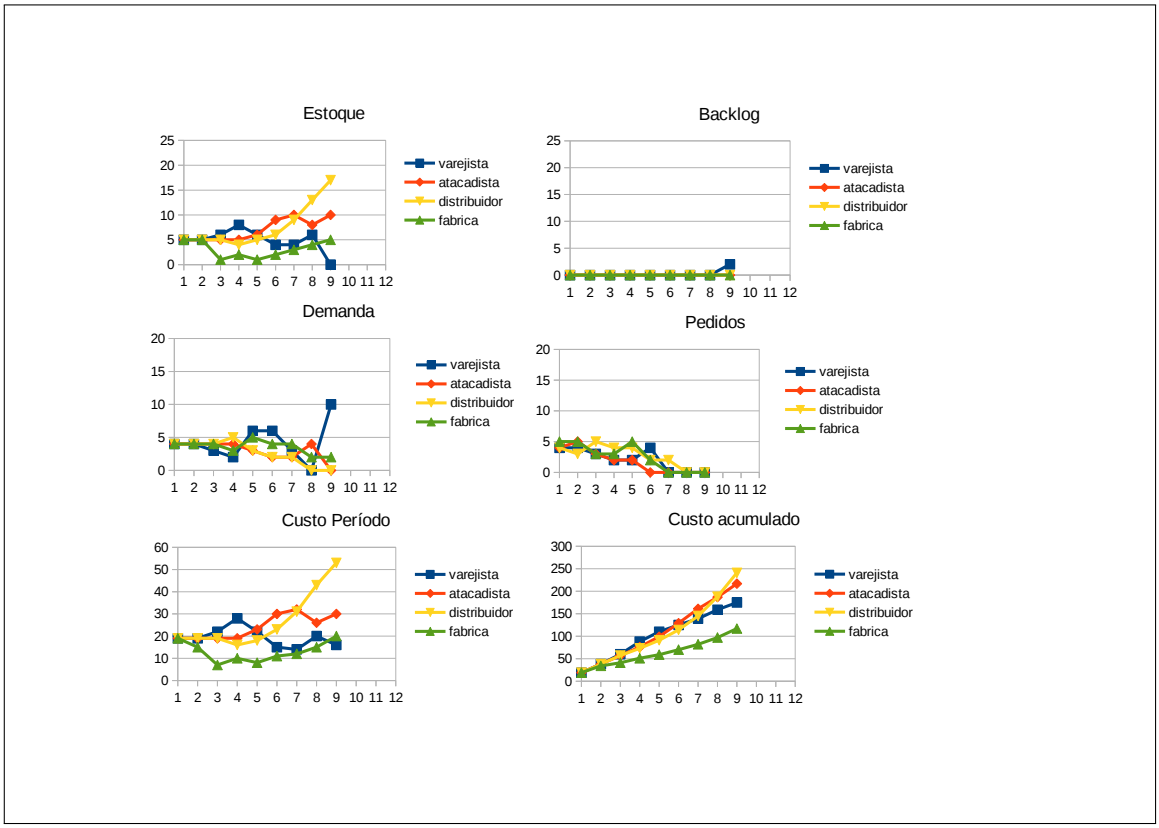
Os custos foram baixos e bem próximos, com o atacadista tendo o melhor rendimento.

Na cadeia 2, com exceção do distribuidor que teve problemas com excesso de estoque, os demais conseguiram se manter em equilíbrio. O varejista teve no máximo 8 produtos em estoque e terminou devendo apenas 2, o estoque do atacadista variou entre 5 e 10, o do distribuidor a partir da sexta semana cresceu e atingiu 17 e a fábrica variou entre 5 e 0.

Os pedidos de todos os membros variaram entre 5 e 0.

Os custos foram bem baixos com destaque para a fábrica, seguido pelo varejista, atacadista e por fim o distribuidor.

Figura 12 – Resultados obtidos na aplicação com usuários com experiência e com compartilhamento de informações na cadeia 2



Fonte: o autor.

5.3 Discussões

As tabelas abaixo mostram a variação total de estoque e pedidos para cada usuário em cada cadeia. Essa variação é calculada pela diferença entre o maior e menor valor que o indicador possuiu durante o experimento. Uma variação alta de estoque ou pedidos é o que caracteriza o efeito chicote. Como as cadeias jogaram por diferentes semanas em cada partida, não é possível comparar os custos.

Os experimentos com usuários sem experiência mostraram que o compartilhamento de informações ajuda bastante, mas a falta de experiência pode complicar no gerenciamento da cadeia. Na cadeia 1 mesmo com o compartilhamento de informações foi possível ver o efeito chicote nos estoques e nos pedidos. Também foi possível ver como o efeito chicote se propaga ao longo da cadeia de suprimentos. Já na cadeia 2 o compartilhamento de informações mitigou

	Estoque		pedido	
	sem info	com info	sem info	com info
Varejista	5	8	16	10
Atacadista	2	19	2	10
Distribuidor	1	18	2	10
Fábrica	6	30	6	10

Tabela 1 – Variação de estoque e pedidos na cadeia 1 do experimento com usuários sem experiência em jogo da cerveja

	Estoque		pedido	
	sem info	com info	sem info	com info
Varejista	13	11	12	6
Atacadista	18	9	6	8
Distribuidor	17	8	8	10
Fábrica	31	25	15	16

Tabela 2 – Variação de estoque e pedidos na cadeia 2 do experimento com usuários sem experiência em jogo da cerveja

	Estoque		pedido	
	sem info	com info	sem info	com info
Varejista	15	9	4	10
Atacadista	25	6	4	10
Distribuidor	17	11	8	10
Fábrica	14	12	10	10

Tabela 3 – Variação de estoque e pedidos na cadeia 1 do experimento com usuários com experiência em jogo da cerveja

	Estoque		pedido	
	sem info	com info	sem info	com info
Varejista	9	10	4	4
Atacadista	12	5	5	5
Distribuidor	11	12	5	5
Fábrica	16	5	6	5

Tabela 4 – Variação de estoque e pedidos na cadeia 2 do experimento com usuários com experiência em jogo da cerveja

os problemas causados pelo efeito chicote, enquanto na aplicação sem o compartilhamento de informações todos os membros sofreram com o efeito chicote, na aplicação com compartilhamento de informações apenas a fábrica teve problemas.

Nos experimentos com usuários com experiência foi possível ver uma diminuição dos problemas do efeito chicote nas duas cadeias. Na cadeia 1 houve problemas com atraso de pedido nas duas aplicações, mas percebeu-se que na aplicação com compartilhamento de informações os atrasos foram menores. Na cadeia 2 na aplicação sem compartilhamento de informações todos os membros tiveram problemas de diferentes formas, já na aplicação com

compartilhamento de informações apenas o distribuidor teve problemas.

Os resultados no geral mostram que o compartilhamento de informações através da *blockchain* ajudou a mitigar o efeito chicote, pois das 4 cadeias que usaram o simulador, em 3 delas houve uma melhora da aplicação sem o compartilhamento de informações para a aplicação com compartilhamento de informações.

6 TRABALHOS RELACIONADOS

Esta seção fará uma breve análise de trabalhos relacionados a este. Primeiramente abordará o trabalho de (KSHETRI, 2018) que fala sobre *blockchain* em cadeia de suprimentos e em seguida falará de três simuladores de jogo da cerveja, o UFVBeerGame (AMBRÓSIO *et al.*, 2006), o Near Beer Game (BEAN, 2006) e o Beer Distribution Game Simulator (VRANJES, 2017).

6.1 Kshetri (2018)

Kshetri (2018) fez uma análise baseada em estudos de casos do papel da blockchain em cadeia de suprimentos para conseguir objetivos, como reduções de custo, melhorar a eficiência e a velocidade da distribuição, entre outros. Como o nosso trabalho, Kshetri mostra como utilizar a blockchain em cadeia de suprimentos, porém não aborda o efeito chicote e o jogo da cerveja, e também não implementa uma aplicação.

Em outros trabalhos foram desenvolvidos softwares que representam virtualmente o jogo da cerveja. Alguns deles são: UFVBeerGame (AMBRÓSIO *et al.*, 2006), o Near Beer Game (BEAN, 2006) e o Beer Distribution Game Simulator (VRANJES, 2017).

6.2 UFVBeerGame

No UFVBeerGame (AMBRÓSIO *et al.*, 2006) é possível ter mais de um jogador em cada nível da cadeia de suprimentos, os atrasos de pedido e entrega bem como os custos de estoque e de não entrega de um produto são personalizáveis, e também é possível personalizar o tamanho dos lotes de entrega, por exemplo, se o lote é de 100 caixas, então os pedidos devem ser feitos em múltiplos de 100, para simular as cargas totais dos caminhões.

Porém, o UFVBeerGame não implementa um modo com compartilhamento de informação e também não possui blockchain.

6.3 Near Beer Game

O Near Beer Game (BEAN, 2006) é bem mais simples comparado ao UFVBeerGame, o jogo possui apenas um jogador que tenta ajustar a produção do fabricante para atender a demanda dos consumidores. A demanda inicial começa em 10 e depois sobe para um valor

constante. O que o torna diferente deste na maioria dos aspectos.

6.4 Beer Distribution Game Simulator

O Beer Distribution Game Simulator (VRANJES, 2017), é o simulador que serve de base para o desenvolvido neste trabalho, ele possui uma área pra um administrador que pode iniciar, terminar e acompanhar dados sobre a partida. Os jogadores se conectam no jogo e se tiver um número múltiplo de 4 jogadores o administrador pode iniciar a partida, se houver mais de 4 jogadores o simulador cria vários grupos. O jogo não possui partidas simultâneas, se uma partida for iniciada, ninguém pode se conectar ao jogo ou iniciar outra partida até que a partida seja encerrada.

6.5 Comparativo

Nenhum dos simuladores implementa o jogo utilizando blockchain com o objetivo de melhorar os resultados dos jogadores e gerar discussões entre os comportamentos das cadeias de suprimentos com e sem blockchain.

O quadro 1 trás um comparativo entre os três simuladores citados anteriormente e o que foi desenvolvido neste trabalho.

Quadro 1 – Comparativo entre os simuladores UFVBeerGame (1), Near Beer Game (2), Beer Distribution Game Simulator (3) e Ledger Beer Game (4).

	simuladores			
	1	2	3	4
Compra e venda	X	X	X	X
Cadeia de suprimentos completa	X		X	X
Multijogadores	X		X	X
Multijogadores em um mesmo nível da cadeia de suprimentos	X			
Troca de informações diretamente entre os jogadores (chat)				
Acesso a informações para previsão de demanda				X
Blockchain				X

Fonte: elaborado pelo autor.

7 CONCLUSÕES E TRABALHOS FUTUROS

Durante os experimentos percebeu-se que o efeito chicote é um grande problema e que pode ser visto mesmo com poucas semanas de duração em um jogo. Também foi visto que quanto mais se percorre a cadeia de suprimentos, mais os problemas se intensificam, já que na maioria das vezes o efeito chicote causava uma variabilidade menor nos estoques e pedidos de quem tava mais perto do consumidor final.

Pelos resultados pode-se concluir que o compartilhamento de informações ajuda bastante a mitigar o efeito chicote, das 4 cadeias que participaram de experimentos em 3 houve uma melhora quando os usuários puderam compartilhar informações entre si. Também foi percebido que a falta de experiência também pode prejudicar, e fazer com que uma cadeia sofra com o efeito chicote mesmo tendo um compartilhamento de informações.

A *blockchain* se mostrou uma boa ferramenta para auxiliar no combate ao efeito chicote, durante os experimentos ela mostrou que pode entregar os dados de forma rápida e consistente, além de todos os seus outros benefícios como resistência a fraude.

Para um melhor aproveitamento da *blockchain*, ela pode ser remodelada para que algumas informações desnecessárias não sejam armazenadas, como as informações sobre partidas, e assim, a *blockchain* fique totalmente focada em armazenar e distribuir informações sobre transações entre os membros da cadeia. Outra implementação importante a se fazer, é o controle de acesso a *blockchain*, atualmente qualquer entidade pode acessar os dados através da API.

Para o melhor aproveitamento do jogo, algumas alterações deveram ser feitas. A demanda do consumidor final poderá ser mostrada ao varejista para uma melhor previsão de demanda no modo com compartilhamento de informações. Outra melhoria seria para que os dados de uma partida pudessem ser exportados para análise posterior. Uma aspecto crucial que atrapalha na aplicação do jogo é a demora na sua instalação devido a quantidade de pré-requisitos que o Hyperledger Fabric possui, assim para uma execução mais rápida do jogo, uma possível melhoria seria colocar o jogo executando em um servidor e disponível para que qualquer usuário pudesse acessar via web.

O jogo foi aplicado em partidas com duração entre 7 e 13 semanas, devido a problemas com o jogo e também em decorrência do tempo disponível, esse número de semanas é considerado pequeno, e seria interessante em trabalhos futuros experimentos com um número de semanas maior.

REFERÊNCIAS

AMBRÓSIO, B. G.; BRAGA, J. L.; PEREIRA, M. de O. Ufvbeergame: Intermediando o aprendizado em cadeias de fornecimento com simulação e jogos empresariais. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2006. v. 1, n. 1, p. 121–130.

BEAN, M. **Bullwhips and beer**: Why supply chain management is so difficult. 2006.

BEZERRA, E. **Princípios de Análise e Projeto de Sistema com UML**. [S.l.]: Elsevier Brasil, 2015. v. 3.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

CRIPTOGRAFIA. **Dicionário online Dicio**. 2018. Disponível em: <https://www.dicio.com.br/criptografia/>. Acesso em: 05 nov. 2018.

KSHETRI, N. 1 blockchain's roles in meeting key supply chain management objectives. **International Journal of Information Management**, Elsevier, v. 39, p. 80–89, 2018.

LEE, H. L.; PADMANABHAN, V.; WHANG, S. The bullwhip effect in supply chains. **Sloan management review**, v. 38, p. 93–102, 1997.

LINDELL, Y.; KATZ, J. **Introduction to modern cryptography**. [S.l.]: Chapman and Hall/CRC, 2014.

MAHESHWARI, S. **Blockchain basics**: Hyperledger fabric and hyperledger composer. 2018. Disponível em: <https://developer.ibm.com/articles/cl-blockchain-hyperledger-fabric-hyperledger-composer-compared/>. Acesso em: 05 nov. 2018.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. Working Paper, 2008.

NARAYANAN, A.; BONNEAU, J.; FELTEN, E.; MILLER, A.; GOLDFEDER, S. **Bitcoin and cryptocurrency technologies**: A comprehensive introduction. [S.l.]: Princeton University Press, 2016.

PUB, F. Secure hash standard (shs). **FIPS PUB 180**, Citeseer, v. 4, 2012.

STERMAN, J. Booms, busts, and beer: Understanding the dynamics of supply chains. **The Handbook of Behavioral Operations Management: Social and Psychological Dynamics in Production and Service Settings**, 2015.

STERMAN, J. D. Flight simulators for management education: The beergame. **Sloan School of Management, Massachusetts Institute of Technology**, 1992.

SVENSSON, G. The multiple facets of the bullwhip effect: Refined and re-defined. **International Journal of Physical Distribution & Logistics Management**, Emerald Group Publishing Limited, v. 35, n. 10, p. 762–777, 2005.

VRANJES, M. **Beer Distribution Game Simulator**. 2017. Disponível em: <https://github.com/MironV/beerdistribution>. Acesso em: 05 nov. 2018.

APÊNDICE A – DIAGRAMAS USADOS NO DESENVOLVIMENTO DO SIMULADOR

Para projetar o simulador foram criados diagrama de atividade, classes e caso de uso.

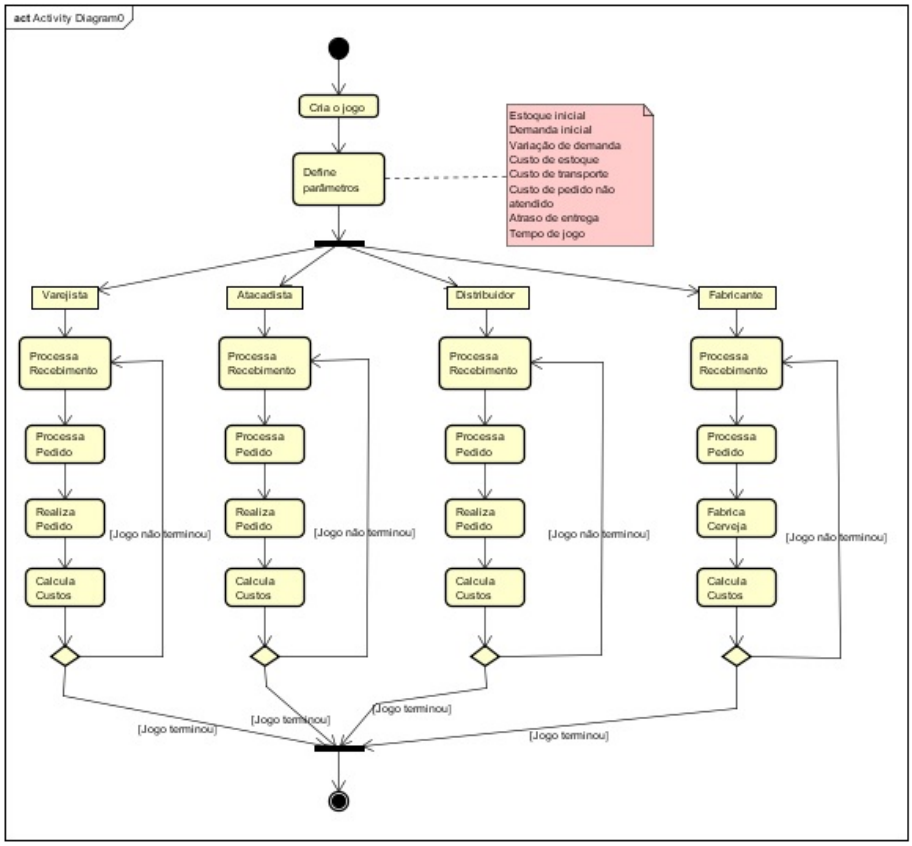


Figura 13 – Diagrama de atividades

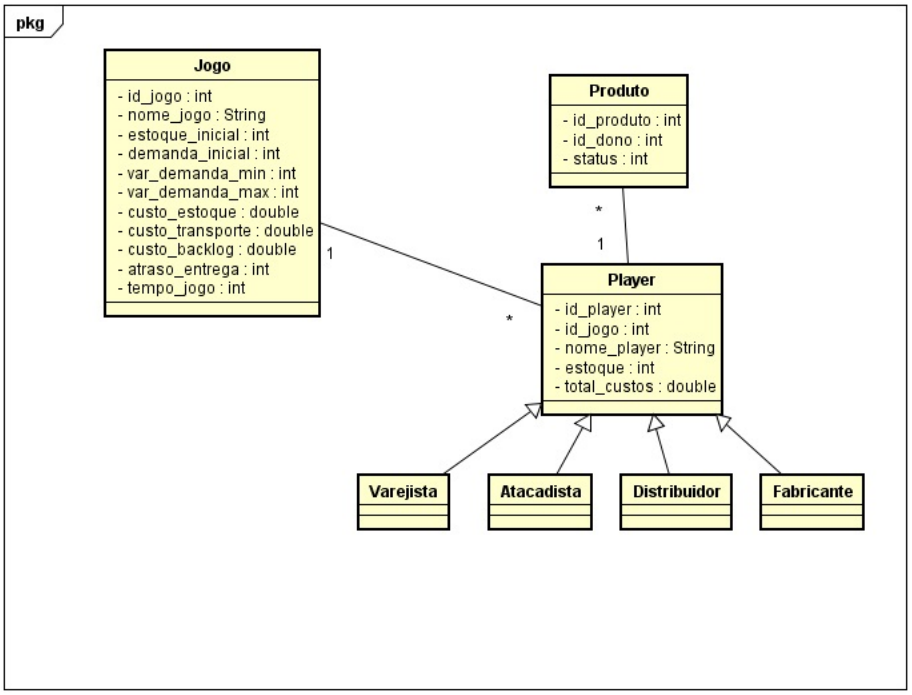


Figura 14 – Diagrama de classes

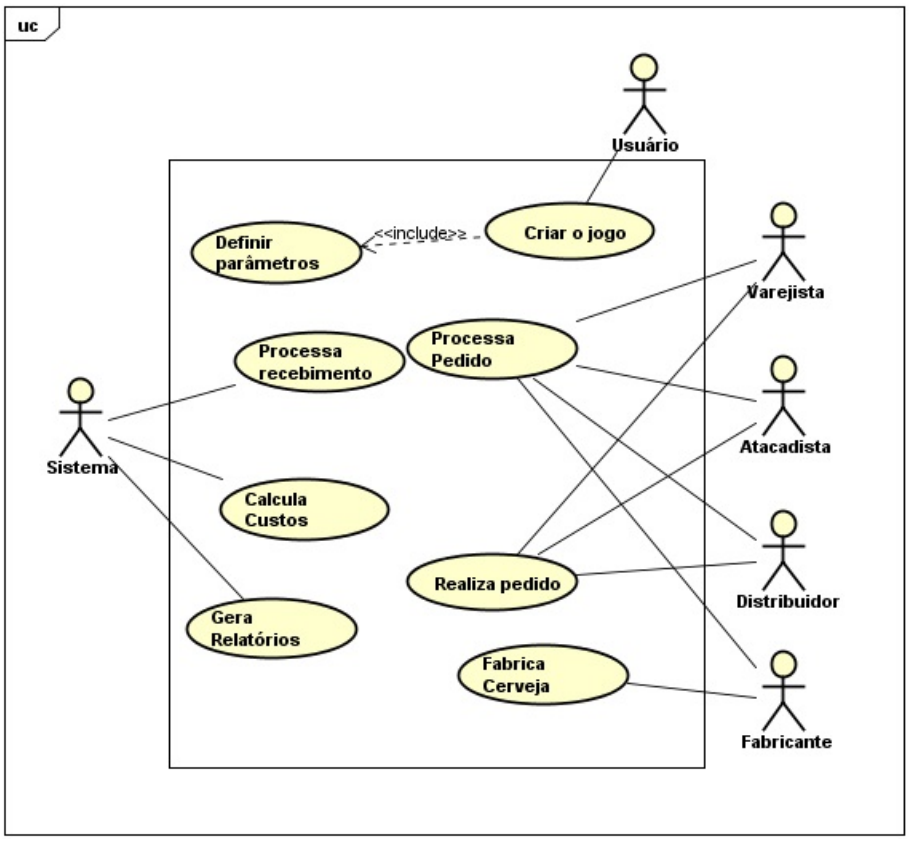


Figura 15 – Diagrama de caso de uso

APÊNDICE B – CÓDIGOS-FONTE UTILIZADOS PARA MODELAR O SIMULADOR

Código-fonte 1 – Modelagem da partida

```
/*
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/**
 * Write your model definitions here
 */

namespace br.com.ledgerbeergame.match

asset Match identified by match_id {
  o String match_id
  o String match_name
```

```
  o String match_pass
  o Integer start_inventory
  o Integer max_demand
  o Integer min_demand
  o Double inventory_cost
  o Double transport_cost
  o Double backlog_cost
  o Lead_time lead_time
  o Integer match_time
  o Double match_cost
  o Integer week
  o Status status
  o Boolean info
}

enum Lead_time {
  o TWO_WEEKS
  o FOUR_WEEKS
}

enum Status {
  o WAIT
  o INITIATED
  o FINISHED
}


//create match

transaction CreateMatch {
  o String match_name
  o String match_pass
```

```
  o Integer start_inventory
  o Integer max_demand
  o Integer min_demand
  o Double inventory_cost
  o Double transport_cost
  o Double backlog_cost
  o Lead_time lead_time
  o Integer match_time
  o Boolean info
}

//change status

transaction ChangeStatus {
  o String match_id
  o Status status
}
```

Código-fonte 2 – Modelagem do pedido

```
namespace br.com.ledgerbeergame.order

import br.com.ledgerbeergame.player.Player


asset Order identified by order_id {
  o String order_id //match_name - week - player_position
  o Integer order
  o Status status
  --> Player client optional
  --> Player provider optional
}
```

```
13
14 enum Status {
15   o OPEN
16   o CLOSE
17 }
18
19 //submit order
20
21 transaction SubmitOrder {
22   o Integer order
23   o String match_id
24   o String week
25   o String player_position
26   o String client_id optional
27   o String provider_id optional
28 }
29
30 //submit order
31
32 transaction CloseOrder {
33   o String order_id
34 }
35
36 event deliverOrder {
37   --> Player distributor
38 }
```

Código-fonte 3 – Modelagem do jogador

```
1 namespace br.com.ledgerbeergame.player
2
3 import br.com.ledgerbeergame.match.Match
```

```
4
5  abstract participant Player {
6    o Integer inventory
7    o Integer backlog
8    o Double cost
9    o Integer week
10   o Integer down_ship
11   --> Match match optional
12 }
13
14 participant Retailer identified by player_name extends
      Player {
15   o String player_name
16   --> Wholesaler provider optional
17 }
18
19 participant Wholesaler identified by player_name extends
      Player {
20   o String player_name
21   --> Retailer client optional
22   --> Regional provider optional
23 }
24
25 participant Regional identified by player_name extends
      Player {
26   o String player_name
27   --> Wholesaler client optional
28   --> Factory provider optional
29 }
30
31 participant Factory identified by player_name extends
      Player {
```

```
  o String player_name
  --> Regional client optional
}


//Create player

transaction CreatePlayer {
  o String player_name
  o Integer player_position
  o String match_id
}

//Update player

transaction UpdatePlayer {
  o String player_name
  o String match_id
  o String player_position
  o Integer inventory
  o Integer backlog
  o Double cost
  o Integer week
  o Integer down_ship
}

//assign match

transaction AssignMatch {
  o String match_id
}

```

```
64 //leave match
65
66 transaction LeaveMatch {
67   o String match_id
68 }
```

Código-fonte 4 – Modelagem do produto

```
1  namespace br.com.ledgerbeergame.product
2
3  import br.com.ledgerbeergame.player.Player
4
5  asset Product identified by product_id {
6    o String product_id //product - match - week
7    o String product
8    o Local local
9    --> Player owner optional
10 }
11
12 enum Local {
13   o INVENTORY
14   o TRANSPORT1
15   o TRANSPORT2
16   o FABRIC
17 }
18
19 //create product
20
21 transaction CreateProduct {
22   o String id
23   o String product
24   o String match
```

```
25    o String week
26    o Local local
27    o String player_id
28    o Integer player_pos
29  }
30
31  //change local
32
33  transaction ChangeLocal {
34    o String product_id
35    o Local local
36  }
37
38  //deliver product
39  transaction DeliverProduct {
40    o String product_id
41    o String customer_id
42    o Integer customer_pos
43  }
44
45  //deliver Final
46  transaction DeliverFinal {
47    o String product_id
48  }
```

Código-fonte 5 – Modelagem das permissões

```
1  /*
2   * Licensed under the Apache License, Version 2.0 (the "
       License");
3   * you may not use this file except in compliance with the
       License.
```

```
4  * You may obtain a copy of the License at
5  *
6  * http://www.apache.org/licenses/LICENSE-2.0
7  *
8  * Unless required by applicable law or agreed to in
       writing, software
9  * distributed under the License is distributed on an "AS
       IS" BASIS,
10 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
       express or implied.
11 * See the License for the specific language governing
       permissions and
12 * limitations under the License.
13 */
14
15 rule NetworkAdminUser {
16     description: "Grant business network administrators
          full access to user resources"
17     participant: "org.hyperledger.composer.system.
          NetworkAdmin"
18     operation: ALL
19     resource: "**"
20     action: ALLOW
21 }
22
23 rule NetworkAdminSystem {
24     description: "Grant business network administrators
          full access to system resources"
25     participant: "org.hyperledger.composer.system.
          NetworkAdmin"
26     operation: ALL
27     resource: "org.hyperledger.composer.system.**"
```

```
28      action: ALLOW
29  }
```

Código-fonte 6 – Modelagem das querys

```
1
2   ////////////////////Match Queries/////////////////
3   query getMatchs {
4     description: "Return all matchs where status = wait or
          initiated"
5     statement:
6       SELECT  br.com.ledgerbeergame.match.Match
7       WHERE (status == _$statusInitiated OR status ==
            _$statusWait)
8   }
9
10  query getMatchById {
11    description: "Return match by id"
12    statement:
13      SELECT br.com.ledgerbeergame.match.Match
14      WHERE (match_id == _$id)
15  }
16
17  query lastMatchId {
18    description: "return the id of the last match inserted"
19    statement:
20      SELECT br.com.ledgerbeergame.match.Match
21      ORDER BY [match_id DESC]
22      LIMIT 1
23  }
24
25
```

```
26 ////////////////////Players Queries////////////////////
27 query getAllPlayers {
28   description: "Return all Players"
29   statement:
30     SELECT br.com.ledgerbeergame.player.Player
31 }
32 query getAllRetailers {
33   description: "Return all Retailers"
34   statement:
35     SELECT br.com.ledgerbeergame.player.Retailer
36 }
37 query getAllWholesalers {
38   description: "Return all Wholesalers"
39   statement:
40     SELECT br.com.ledgerbeergame.player.Wholesaler
41 }
42 query getAllRegionals {
43   description: "Return all Regionals"
44   statement:
45     SELECT br.com.ledgerbeergame.player.Regional
46 }
47 query getAllFactorys {
48   description: "Return all Factorys"
49   statement:
50     SELECT br.com.ledgerbeergame.player.Factory
51 }
52
53 query participantTransactions {
54   description: "Return transactions invoked by a
       participant in historian"
55   statement:
56     SELECT org.hyperledger.composer.system.HistorianRecord
```

```
57      WHERE (participantInvoking == _$participant)
58  }
59
60  query getPlayersUpdate {
61    description: "Return update players transactions order by
          week and player_position"
62    statement:
63      SELECT br.com.ledgerbeergame.player.UpdatePlayer
64      WHERE (match_id == _$match_id)
65      ORDER BY [week ASC, player_position ASC]
66  }
67
68  ////////////////Products Queries////////////////
69  query getInventory {
70    description: "Return all products in player s inventory"
71    statement:
72      SELECT br.com.ledgerbeergame.product.Product
73      WHERE (owner == _$owner_id AND local ==  INVENTORY )
74  }
75
76  query getTransport1 {
77    description: "Return all products with status TRANSPORT1
        of owner"
78    statement:
79      SELECT br.com.ledgerbeergame.product.Product
80      WHERE (owner == _$owner_id AND local ==  TRANSPORT1 )
81  }
82
83  query getTransport2 {
84    description: "Return all products with status TRANSPORT2
        of owner"
85    statement:
```

```
86      SELECT br.com.ledgerbeergame.product.Product
87      WHERE (owner == _$owner_id AND local ==  TRANSPORT2 )
88  }
89
90  query getProductFabric {
91    description: "Return all products with status FABRIC of
          owner"
92    statement:
93      SELECT br.com.ledgerbeergame.product.Product
94      WHERE (owner == _$owner_id AND local ==  FABRIC )
95  }
96
97  //////////////////Order Queries////////////////
98  query getNextOrder {
99    description: "Return the first submited order for a
          specific provider"
100   statement:
101     SELECT br.com.ledgerbeergame.order.Order
102     WHERE (provider == _$provider_id AND status ==  OPEN )
103     ORDER BY [order_id ASC]
104     LIMIT 1
105 }
106
107 query getNextFabricOrder {
108   description: "Return the first submited order for Fabric"
109   statement:
110     SELECT br.com.ledgerbeergame.order.Order
111     WHERE (client == _$client_id AND status ==  OPEN )
112     ORDER BY [order_id ASC]
113     LIMIT 1
114 }
115
```

```
query getAllSubmitOrder {
  description: "Return submit order transactions order by
     week and player_position"
  statement:
    SELECT br.com.ledgerbeergame.order.SubmitOrder
    WHERE (match_id == _$match_id AND week !=  00  AND week
        !=  01  AND player_position !=  -1 )
    ORDER BY [week ASC, player_position ASC]
}
```

Código-fonte 7 – Modelagem das transações

```
////////////Transaction functions////////////


////////////Match Transactions////////////

/**
 * Create Match Transaction
 * @param {br.com.ledgerbeergame.match.CreateMatch}
    matchData
 * @transaction
 */

function createMatch(matchData) {

  return getAssetRegistry("br.com.ledgerbeergame.match.
     Match").then(function (matchRegistry) {

    return newMatchID().then(function (id) {
      var factory = getFactory();
      var NS = "br.com.ledgerbeergame.match";
```

```
19      var matchId =  1 ;
20
21      if (id) {
22        matchId = id.toString();
23      }
24      var matchResource = factory.newResource(NS, "Match",
           matchId);
25
26      matchResource.match_name =  matchData.match_name;
27      matchResource.match_pass =  matchData.match_pass;
28      matchResource.start_inventory = matchData.
           start_inventory;
29      matchResource.max_demand =  matchData.max_demand;
30      matchResource.min_demand =  matchData.min_demand;
31      matchResource.inventory_cost =  matchData.
           inventory_cost;
32      matchResource.transport_cost =  matchData.
           transport_cost;
33      matchResource.backlog_cost =  matchData.backlog_cost;
34      matchResource.lead_time =  matchData.lead_time;
35      matchResource.match_time =  matchData.match_time;
36      matchResource.match_cost =  0.0;
37      matchResource.week =  0;
38      matchResource.status =   WAIT ;
39      matchResource.info = matchData.info;
40
41      return matchRegistry.add(matchResource);
42    }).catch(function (error) {
43      throw new Error(error);
44    });
45  }).catch(function (error) {
46    throw new Error(error);
```

```
47    });
48  }
49
50  function newMatchID() {
51    return query("lastMatchId").then(function (result) {
52      if (result.length > 0) {
53        return parseInt(result[0].match_id) + 1;
54      } else {
55        return null;
56      }
57    }).catch(function (error) {
58      return (null, error);
59    });
60  }
61
62  /**
63   * Chage Match Status Transaction
64   * @param {br.com.ledgerbeergame.match.ChangeStatus}
       matchData
65   * @transaction
66   */
67
68  function changeStatus(matchData) {
69    var regs;
70    return getAssetRegistry("br.com.ledgerbeergame.match.
       Match").then(function (matchRegistry) {
71      regs = matchRegistry;
72      return matchRegistry.get(matchData.match_id);
73    }).then(function (matchResource) {
74
75      matchResource.status =  matchData.status;
76
```

```
77      return regs.update(matchResource);
78    }).catch(function (error) {
79      throw new Error(error);
80    });
81  }
82
83  ////////////Player Transactions////////////
84  /**
85   * Create Player Transaction
86   * @param {br.com.ledgerbeergame.player.CreatePlayer}
       playerData
87   * @transaction
88   */
89
90  function createPlayer(playerData) {
91
92    var pns;
93    switch (playerData.player_position) {
94      case 0:
95        pns = "Retailer";
96        break;
97      case 1:
98        pns = "Wholesaler";
99        break;
100     case 2:
101       pns = "Regional";
102       break;
103     case 3:
104       pns = "Factory";
105       break;
106
107     default:
```

```
108         break;
109     }
110
111     return getParticipantRegistry("br.com.ledgerbeergame.
            player."+pns).then(function (playerRegistry) {
112
113       var factory = getFactory();
114       var NS = "br.com.ledgerbeergame.player";
115
116       var playerResource = factory.newResource(NS, pns,
              playerData.player_name);
117
118       playerResource.inventory = 0;
119       playerResource.backlog = 0;
120       playerResource.cost = 0.0;
121       playerResource.week = 0;
122       playerResource.down_ship = 0;
123
124       var rel_match = factory.newRelationship("br.com.
              ledgerbeergame.match", "Match", playerData.match_id)
              ;
125       playerResource.match = rel_match;
126
127       return playerRegistry.add(playerResource);
128     }).catch(function (error) {
129       throw new Error(error);
130     });
131 }
132
133 /**
134  * Update Player Transaction
135  * @param {br.com.ledgerbeergame.player.UpdatePlayer}
```

```
    playerData
 * @transaction
 */

function updatePlayer(playerData) {
  var pns;
  switch (parseInt(playerData.player_position)) {
    case 0:
      pns = "Retailer";
      break;
    case 1:
      pns = "Wholesaler";
      break;
    case 2:
      pns = "Regional";
      break;
    case 3:
      pns = "Factory";
      break;

    default:
      break;
  }
  var regs;
  return getParticipantRegistry("br.com.ledgerbeergame.
     player."+pns).then(function (playerRegistry) {
    regs = playerRegistry;
    return playerRegistry.get(playerData.player_name);
  }).then(function (playerResource) {

    playerResource.inventory = playerData.inventory;
    playerResource.backlog = playerData.backlog;
```

```
166     playerResource.cost = playerData.cost;
167     playerResource.week = playerData.week;
168     playerResource.down_ship = playerData.down_ship;
169 
170     return regs.update(playerResource);
171   }).catch(function (error) {
172     throw new Error(error);
173   });
174 }
175 
176 ////////////Product Transactions////////////
177 
178 /**
179  * Create Product Transaction
180  * @param {br.com.ledgerbeergame.product.CreateProduct}
       productData
181  * @transaction
182  */
183 
184 function createProduct(productData) {
185   var pns;
186   switch (productData.player_pos) {
187     case 0:
188       pns = "Retailer";
189       break;
190     case 1:
191       pns = "Wholesaler";
192       break;
193     case 2:
194       pns = "Regional";
195       break;
196     case 3:
```

```
197         pns = "Factory";
198         break;
199
200       default:
201         break;
202     }
203     return getAssetRegistry("br.com.ledgerbeergame.product.
          Product").then(function (productRegistry) {
204
205
206       var factory = getFactory();
207       var NS = "br.com.ledgerbeergame.product";
208       var productId = productData.product+"-"+productData.
            match+"-"+productData.week+"-"+productData.
            player_pos+"-"+productData.id;
209
210       var productResource = factory.newResource(NS, "Product
            ", productId);
211
212       productResource.product =  productData.product;
213       productResource.local =  productData.local;
214
215       var rel_player = factory.newRelationship("br.com.
            ledgerbeergame.player", pns, productData.player_id);
216       productResource.owner = rel_player;
217
218       return productRegistry.add(productResource);
219
220     }).catch(function (error) {
221       throw new Error(error);
222     });
223 }
```

```
224
225 /**
226  * Chage Product Local Transaction
227  * @param {br.com.ledgerbeergame.product.ChangeLocal}
        productData
228  * @transaction
229  */
230
231 function changeLocal(productData) {
232   var regs;
233   return getAssetRegistry("br.com.ledgerbeergame.product.
        Product").then(function (productRegistry) {
234     regs = productRegistry;
235     return productRegistry.get(productData.product_id);
236   }).then(function (productResource) {
237
238     productResource.local =  productData.local;
239
240     return regs.update(productResource);
241   }).catch(function (error) {
242     throw new Error(error);
243   });
244 }
245
246 /**
247  * Chage Product owner Transaction
248  * @param {br.com.ledgerbeergame.product.DeliverProduct}
        productData
249  * @transaction
250  */
251
252 function deliverProduct(productData) {
```

```
253  var pns;
254  switch (productData.customer_pos) {
255    case 0:
256      pns = "Retailer";
257      break;
258    case 1:
259      pns = "Wholesaler";
260      break;
261    case 2:
262      pns = "Regional";
263      break;
264    case 3:
265      pns = "Factory";
266      break;
267
268    default:
269      break;
270  }
271  var regs;
272  return getAssetRegistry("br.com.ledgerbeergame.product.
       Product").then(function (productRegistry) {
273
274    regs = productRegistry;
275    return productRegistry.get(productData.product_id);
276  }).then(function (productResource) {
277    var factory = getFactory();
278
279    var rel_player = factory.newRelationship("br.com.
         ledgerbeergame.player", pns, productData.customer_id
         );
280    productResource.owner =  rel_player;
281    productResource.local =  TRANSPORT1 ;
```

```
282
283     return regs.update(productResource);
284   }).catch(function (error) {
285     throw new Error(error);
286   });
287 }
288
289 /**
290  * Chage Product owner to null
291  * @param {br.com.ledgerbeergame.product.DeliverFinal}
        productData
292  * @transaction
293  */
294
295 function deliverFinal(productData) {
296
297   var regs;
298   return getAssetRegistry("br.com.ledgerbeergame.product.
        Product").then(function (productRegistry) {
299     regs = productRegistry;
300     return productRegistry.get(productData.product_id);
301   }).then(function (productResource) {
302
303     productResource.owner =  null;
304
305     return regs.update(productResource);
306   }).catch(function (error) {
307     throw new Error(error);
308   });
309 }
310
311 ////////////Order Transactions////////////
```

```
312
313 /**
314  * Submit Order Transaction
315  * @param {br.com.ledgerbeergame.order.SubmitOrder}
        orderData
316  * @transaction
317  */
318
319 function submitOrder(orderData) {
320
321   return getAssetRegistry("br.com.ledgerbeergame.order.
        Order").then(function (orderRegistry) {
322
323
324     var factory = getFactory();
325     var NS = "br.com.ledgerbeergame.order";
326
327     //var w = orderData.week;
328     if (parseInt(orderData.week) < 10) orderData.week =
          0 +orderData.week;
329     var orderId = orderData.match_id+"-"+orderData.week
         +"-0"+orderData.player_position;
330
331     var orderResource = factory.newResource(NS, "Order",
         orderId);
332
333     orderResource.order =  orderData.order;
334     orderResource.status =  OPEN ;
335     if (orderData.client_id) {
336       var pns;
337       switch (parseInt(orderData.player_position)) {
338         case 0:
```

```
339           pns = "Retailer";
340           break;
341         case 1:
342           pns = "Wholesaler";
343           break;
344         case 2:
345           pns = "Regional";
346           break;
347         case 3:
348           pns = "Factory";
349           break;
350
351         default:
352           break;
353       }
354       var rel_client = factory.newRelationship("br.com.
             ledgerbeergame.player", pns, orderData.client_id);
355       orderResource.client = rel_client;
356     }
357     if (orderData.provider_id) {
358       var pns;
359       switch (parseInt(orderData.player_position)) {
360         case -1:
361           pns = "Retailer";
362           break;
363         case 0:
364           pns = "Wholesaler";
365           break;
366         case 1:
367           pns = "Regional";
368           break;
369         case 2:
```

```
370             pns = "Factory";
371             break;
372
373           default:
374             break;
375         }
376         var rel_provider = factory.newRelationship("br.com.
              ledgerbeergame.player", pns, orderData.provider_id
              );
377         orderResource.provider = rel_provider;
378       }
379       return orderRegistry.add(orderResource);
380
381     }).catch(function (error) {
382       throw new Error(error);
383     });
384 }
385
386 /**
387  * Close Order Transaction
388  * @param {br.com.ledgerbeergame.order.CloseOrder}
       orderData
389  * @transaction
390  */
391
392 function closeOrder(orderData) {
393
394   var regs;
395   return getAssetRegistry("br.com.ledgerbeergame.order.
        Order").then(function (orderRegistry) {
396     regs = orderRegistry;
397     return orderRegistry.get(orderData.order_id);
```

```
398   }).then(function (orderResource) {
399
400     orderResource.status =   CLOSE ;
401
402     return regs.update(orderResource);
403   }).catch(function (error) {
404     throw new Error(error);
405   });
406 }
```