Mendelova univerzita v Brně
Provozně ekonomická fakulta

Implementace výukové aplikace pro VIDA! Science centrum

Diplomová práce

Vedoucí práce:
Mgr. Tomáš Foltýnek, Ph.D.

Bc. Jakub Drobný

Brno 2016

Děkuji vedoucímu diplomové práce Mgr. Tomášovi Foltýnkovi, Ph.D. za výstižné připomínky a cenné rady, které mi pomohly při psaní této práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Implementace výukové aplikace pro VIDA! Science centrum** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 25. května 2016 _____

Abstract

Drobný, J. Implementation of educational application for VIDA! Science centre. Diploma thesis. Brno: Mendel University, 2016.

This diploma thesis deals with an implementation of educational application for the specific exhibit in VIDA! science centre. Application is known as "Beer distribution game" and serves to demonstrate a number of key principles of supply chain management.

Keywords

Educational application, VIDA! SC, C#, Beer Distribution game, Supply chain management

Abstrakt

Drobný, J. Implementace výukové aplikace pro VIDA! Science centrum. Diplomová práce. Brno 2016.

Tato diplomová práce se zabývá implementací výukové aplikace pro specifický exponát ve VIDA! Science centru. Aplikace je známá pod pojmem „Distribuce limonád" a slouží k demonstraci základních principů v oblasti řízení dodavatelského řetězce.

Klíčová slova

Výuková aplikace, VIDA! SC, C#, Distribuce limonád, řízení dodavatelského řetězce

Obsah

Seznam obrázků

Seznam tabulek

Nebyla nalezena položka obsahu.

1 Úvod a cíl práce

1.1 Úvod

V dnešní době lze jen stěží nalézt obor lidské činnosti, ve kterém by nenašla uplatnění výpočetní technika. I v oblasti podnikání a vedení firem tomu není jinak. Drtivá většina podniků působících na trhu využívá ke své denní činnosti počítačový software, avšak je za potřebí si uvědomit, že klíčová rozhodnutí vždy dělá člověk. Každý manažer musí denně řešit několik problémů. Z jedné strany je tlačen konkurencí, která ho nutí ke stálému snižování ceny a z druhé strany působí zákazník a jeho neustále stoupající nároky na kvalitu výrobků. Jedinec, firma nebo i velké společnosti, vytváří s okolím mnoho vazeb. Pro člověka je potom velmi obtížné provádět rozhodnutí tak, aby byla správná a vedla k vytyčenému cíli. Často některá rozhodnutí negativně ovlivní chod firmy.

Díky velkému rozvoji informačních technologií jsme schopni vytvářet stále složitější modely reálných systémů, které nám pomohou k jejich poznání a porozumění všech jejich součástí a vazeb. Jednou z oblastí, která se zabývá systémy, vazbami, chováním a jeho vývojem v čase, je i systémová dynamika. Firmy, ale nejen ony, jsou schopny díky tomuto nástroji vidět důsledky svého rozhodnutí ještě před tím, než vůbec budou učiněny. Dle (STERMAN 2000) je tvorba modelů a následná simulace mohutným nástrojem pro všechny firmy zejména v oblasti plánování. Tento nástroj dokáže ušetřit mnoho času, finančních prostředků a energie.

Jednou z oblastí, ve které je obrovský potenciál na úsporu nákladů, je i problematika řízení dodavatelského řetězce (SCM - Supply Chain Management). Než se finální výrobek dostane do rukou koncového zákazníka, absolvuje dlouhou cestu, která začíná těžbou surovin, následně přichází hrubé opracování a končí až finální výrobou, montáží, balením a velkoobchodní distribucí do maloobchodních prodejen. Na téhle cestě je výrobek mnohokrát přepravován, přesouván a skladován, což stojí hodně peněz a času. SCM je jeden z oborů moderního managementu pro optimalizaci všech činností a systémů pro zabezpečení dodávky produktů a služeb od dodavatelů surovin až ke koncovému spotřebiteli. (LAMBERT, 2008)

VIDA! science centrum je zábavní vědecký park v Brně, který obsahuje přes 150 interaktivních exponátů rozdělených do čtyř tematických celků: Planeta, Civilizace, Člověk a Mikrosvět. Návštěvníci VIDA! SC mají nespočet možností, jak zde svůj volný čas strávit a rozšířit si své znalosti zábavnou formou. Mou snahou je využít potenciál informačních technologií a vytvořit pro návštěvníky novou výukovou aplikaci, jejíž cílem bude demonstrovat základní principy v oblasti SCM a zdokonalit tak mentální modely hráčů v této problematice.

1.2 Cíl práce

Cílem této diplomové práce je implementovat výukovou aplikaci „Distribuce limonád“ pro exponát „Kulatý stůl“ ve VIDA! science centru.

Výsledná aplikace by měla sloužit k návštěvníkům VIDA! SC k tvorbě popř. zdokonalení jejich mentálních modelů v oblasti řízení dodavatelského řetězce. Po skončení hry by si každý hráč skupiny měl uvědomit základní principy v SCM a na jejich základě se být schopen i efektivněji rozhodovat.

1.3 Metodika

Za použití odborné literatury bude zpracován přehled možných způsobů uvažování o systémech. Jejich výhody a nevýhody budou demonstrovány na praktických příkladech, které by měly čtenáři pomoci pochopit hlavní úskalí jednotlivých způsobů uvažování. Poté bude provedena samotná analýza hry Distribuce limonád, ve které bude čtenář seznámen s modelem hry, pravidly a ekonomickými jevy, které jsou pro tuto hru typické. Následně bude vytvořen přehled současných prací odborného charakteru, které se zabývají problematikou Beer game. Na základě poznatků bude navrženo řešení, které bude později implementováno. Výsledná aplikace poté bude otestována v ostrém provozu exponátu. V průběhu testovacího provozu bude se zaměstnanci VIDA! SC současné řešení komunikováno a jeho nedostatky budou obratem zapracovány tak, aby mohla být nová verze aplikace co nejrychleji nasezena.

Pro vytvoření přehledu současných prací budou prohledány tyto webové archivy:

- MENDELU1[1]
- ČVUT2[2]
- VŠB3[3]
- VUT4[4]

Další publikované práce odborného charakteru budou vyhledávány na webovém portálu *theses.cz* podle klíčových slov abstraktu. Vyhledány budou práce, které nejsou starší pěti let.

[1] http://is.mendelu.cz/zp/

[2] https://dip.felk.cvut.cz/

[3] http://dspace.vsb.cz

[4] https://www.vutbr.cz/studium/zaverecne-prace

K vytvoření přehledu současných odborných článků a knih budou prohledány následující databáze:

- Web of Science[5]
- ScienceDirect[6]
- Ovid[7]
- SpringerLink[8]

Výsledná aplikace bude vyvíjena podle metodiky extrémního programování (XP). Extrémní programování je jedna z metodik agilního vývoje SW, která je založena na iterativním a inkrementálním vývoji. Podstatou XP je rychlé a časté dodávání SW zákazníkovi a kvalitní komunikace všech zainteresovaných subjektů. XP umožňuje rychlý vývoj softwaru a zároveň dokáže reagovat na změnu požadavků v průběhu vývojového cyklu. (AGILE SOFTWARE DEVELOPMENT, 2016), (EXTREME PROGRAMMING, 2016)

[5] http://apps.webofknowledge.com/

[6] http://www.sciencedirect.com/

[7] http://ovidsp.tx.ovid.com/

[8] http://link.springer.com/

2 Uvažování o systémech

2.1 Intuitivní myšlení

Intuitivní myšlení, neboli selský rozum, je druh myšlení, který je postaven na empirických zkušenostech a umožňuje získávat vědomosti bez užití úsudku. Jeho aplikace nám umožňuje fungovat v běžném životě a dělat spoustu věcí automaticky. Intuitivní myšlení je nicméně uzpůsobeno pro přemýšlení o jednoduchých systémech. V minulosti to většinou stačilo – člověk se zabýval pouze jednoduchými skutečnostmi a případné komplexní systémy, které ho obklopovaly (např. počasí), stejně ovlivnit nedokázal. Dnes se ovšem komplexním systémům nevyhneme, dokonce se ani nevyhneme tomu, abychom je ovlivňovali. Najednou nám intuitivní myšlení přestává dostačovat. Proto je nutné rozvíjet některé alternativní druhy přemýšlení. Než se vrhneme na přehled alternativních stylů myšlení, pojďme se podívat na několik typických způsobů intuitivního myšlení. (MEADOWS, 2008)

2.1.1 Lineární uvažování

Systémy skutečného světa jsou z velké části nelineární. I přesto mají lidé tendenci extrapolovat systémové trendy a hledat tak lineární závislosti mezi sledovanými jevy. Tendence k lineárnímu uvažování byla prokázána i psychologickými výzkumy a lze ji pěkně ilustrovat příkladem s řasami v rybníku. Představme si velký rybník. V rybníku se objevily řasy a začaly se rychle množit. Množí se tak rychle, že se jejich počet každý den zdvojnásobí. První den je jich velmi málo, nicméně pokud nedojde k žádnému zásahu, tak během třiceti dní pokryjí celý rybník a vše živé v rybníku zahyne. Správce rybníku pozoruje, jak se mu postupně na rybníku množí řasy. Protože jich však je na začátku málo, tak si říká, že s tím není třeba nic dělat. Pro zásah se rozhodne, až když bude polovina rybníku zamořená řasami. Ovšem tento okamžik nastane 29. den a na záchranu rybníka zbyde pouhý jeden den. Stejně tak jako správce v modelovém příkladu, i většina lidí vykazuje podobné chování a v první fázi problém podcení a myslí si, že na vyřešení problému zbude více času. (STERMAN, 2000)

2.1.2 Krátkodobý výhled

V dávné minulosti, v době lovců a sběračů, nemělo dlouhodobé plánování velký význam. Lidé byli schopni vnímat a rozumět pouze jednoduchým systémům a tak povětšinou řešili otázku současnosti a blízké budoucnosti. V dnešní době má dlouhodobý výhled význam zásadní, nicméně naše myšlení zůstává často na úrovni tehdejších lovců a sběračů. Dlouhodobý výhled nám mnohdy nejde.

Pro komplexní systémy je charakteristické, že zásahy s pozitivním dlouhodobým efektem často vedou ke krátkodobému zhoršení. Ovšem vzhledem k našemu krátkodobému myšlení a krátkodobému způsobu fungování našich organizací, bývá velký problém dlouhodobě výhodná řešení prosadit. (STERMAN, 2001)

2.1.3 Zjednodušené uvažování o příčinách

Lidé mají rádi jednoduchá vysvětlení. Následky, zejména ty špatné, si žádají jasné příčiny. Nastalo X, protože Y. Druhá světová válka vypukla, protože Hitler byl schopný demagog. Teplota stoupá, protože stoupá koncentrace CO_2. Jednoduchá vysvětlení dělají svět pochopitelnějším a my přeci chceme světu rozumět. V uvažování o příčinných vztazích se dopouštíme hned několika chyb. První typickou chybou je uvažování ve stylu „poté, tedy proto“ neboli zaměnění konsekvence za kauzalitu. Šaman provedl rituál a poté začalo pršet. Dobrá, šamany už jsme prokoukli, takže příliš nevěříme, že by déšť byl důsledkem šamanova rituálu. Ale co když centrální banka sníží úrokovou míru a stav ekonomiky se následně zlepší, když fotbalový tým vymění trenéra a následné výsledky jsou mizerné, nebo když finanční krize v zemi A následuje po finanční krizi v zemi B? Dokážeme i v těchto situacích dobře rozlišit konsekvenci a kauzalitu a nepodlehnout zjednodušenému vysvětlení? (STERMAN, 2000)

2.1.4 Neintuitivnost komplexních systémů

Když porovnáme seznam charakteristik komplexních systémů a častých chyb lidského myšlení, není divu, že komplexní systémy se chovají často velmi neintuitivně a naše zásahy do nich mají nežádoucí důsledky. Jakmile máme co do činění s komplexními systémy, musíme být velmi ostražití a nepodlehnout nástrahám našeho příliš zjednodušujícího intuitivního myšlení.

2.2 Systémové myšlení

Jak bylo řečeno, k pochopení složitějších systémů nám intuitivní přístup nepostačí. Je nutno jej vylepšit. Jednou z alternativních možností je myšlení systémové. Zjednodušeně řečeno, systémové myšlení se cíleně snaží o pohled na systém jako celek. Zabývá se studií vztahů mezi jednotlivými částmi systému. Myslet systémově znamená vědět, na jakých principech systémy fungují, jaké jsou jejich zákonitosti vzniku, růstu a zániku a vidět tudíž veškeré intervence do systému ve všech jejich souvislostech.

Avšak jako svět není černobílý, tak ani naše myšlení ne – je tedy nutno pamatovat, že v praxi vždy používáme kombinaci různých typů myšlení. (STERMAN, 2000)

2.2.1 Holismus a redukcionismus

Holismus systémového myšlení znamená, že celý systém není pouhým součtem jeho částí, tudíž k tomu, abychom systém jako celek poznali, nám nestačí důkladně poznat všechny jeho části, ale musíme klást důraz i na vztahy mezi nimi.

Redukcionismus je opakem holismu a je založen na přesvědčení, že systému jako celku můžeme porozumět na základě detailního porozumění jednotlivým částem. (MEADOWS, 2006)

2.2.2 Induktivní a deduktivní myšlení

Dvěma základními metodami uvažování o systémech jsou dedukce a indukce. Dedukce je logicky korektní, avšak málokdy je reálně používaná. Deduktivní myšlení vede od obecného ke konkrétnímu. Z obecně platných principů vyvozujeme logickou úvahou platné závěry. Deduktivní uvažování je korektní a snadno formálně uchopitelné. Vědecké argumenty se (většinou) snaží postupovat podle zásad deduktivního myšlení. Každodenní realitě však deduktivní myšlení neodpovídá – lidé zvládají přemýšlet deduktivně pouze o jednoduchých a jasně formulovaných problémech.

Oproti tomu induktivní myšlení postupuje od konkrétního k obecnému, od pozorování příkladů k zobecňování a usuzování o budoucím vývoji. Induktivní uvažování sice není příliš korektní, nicméně lidé ho velmi často a úspěšně používají. Typickým příkladem ze života je ovládání nového přístroje (např. mobilního telefonu). Deduktivní přístup by odpovídal tomu, že si sedneme, přečteme si kompletní návod, a když potřebujeme s přístrojem něco provést, tak na základě informací z návodu odvodíme posloupnost kroků, které máme udělat. Induktivní přístup odpovídá tomu, co dělá většina lidí, prostě začneme přístroj používat a na základě zkušeností se nejdříve učíme posloupnosti konkrétních kroků a postupně i obecné principy fungování přístroje. (MEADOWS, 2006)

2.2.3 Decentralizované myšlení

Jednou z častých chyb při intuitivním myšlení je (jak bylo zmíněno v kapitole 2.1.3) tendence lidí hledat jednoduchá vysvětlení. Tato vysvětlení bývají centralizovaná, to znamená, že hledáme jednu příčinu, jeden zdroj problémů, jednoho původce. Pro porozumění komplexním systémům je však často potřeba naučit se uvažovat decentralizovaně. Velmi užitečným nástrojem k názorné demonstraci decentralizovaného myšlení jsou modely, které ukazují, jakým způsobem mohou systémy fungovat bez globálního vedení. (STERMAN, 2001)

2.3 Systémová dynamika

Je vědní disciplína, která zkoumá vývoj systému, jeho závislosti a chování v čase. Cílem systémové dynamiky je nalezení trendů, závislostí, vazeb či vzorců chování mezi jednotlivými veličinami v systému. Za zakladatele systémové dynamiky je považován profesor J. W. Forrester, který v polovině 20. století působil na MIT (Massachusetts Institute of Technology). První publikace o systémové dynamice vznikaly v 60. letech minulého století.

Systémová dynamika kombinuje teorii, metody a filosofii a vytváří modely, které nám napomáhají analyzovat chování celého systému.

Základem každého počítačového modelu je mentální model — naše představa o tom, jak určitý systém funguje. Mentální modely tedy odrážejí představu jednotlivce. Avšak i mentální modely mají své omezení, která jsou daná omezenými možnostmi vnímání lidského mozku. Je proto zřejmé, že používání pouze mentálních

modelů při zkoumání složitých, dynamicky se vyvíjejících systémů, může vést k naprosto špatným rozhodnutím. Proto je žádoucí k modelování využít i výhody a možnosti moderní techniky. Dynamické modelování za pomoci výpočetní techniky tak představuje velmi účinný nástroj pro rozšíření našich mentálních modelů. (STERMAN, 2000), (STERMAN, 2001)

2.4 Simulátory

Rozvoj výpočetní techniky měl pro systémovou dynamiku velmi pozitivní význam. Díky pokroku v tomto odvětví bylo vlastně teprve možné vytvořit rozsáhlé modely nejen v podnikové praxi a řešit problémy, které vysoce převyšují schopnosti lidského intelektu. Dle (STERMAN, 2000) je simulace považována za jednu z možností, jak testovat mentální modely. Bez simulací bychom byli schopni zlepšovat své mentální modely pouze na základě zpětné vazby z reálného světa.

3 Distribuce limonád

Jedná se o simulační hru, ve světě známou jako Beer game nebo Beer distribution game (BEERGAME, 2012).

Hlavním účelem této hry je demonstrovat základní principy z oblasti řízení dodavatelského řetězce. Beer game se nejčastěji hraje jako desková hra, ve které je dodavatelský řetězec reprezentován čtyřmi rolemi. Maloobchodník – velkoobchodní – distributor – továrna. Každá role je pak obsazena jedním, nebo skupinou hráčů, kteří každé hrací kolo činí rozhodnutí, kolik beden piva objednají od následujícího článku v řetězci. Bedny s pivem zde nepředstavují předmět studia, ale pouze libovolně zvolený artikl pro lepší představu.

Odehrání jednoho hracího kola reprezentuje dobu jednoho týdne. Maloobchodník obdrží objednávku od koncového zákazníka a odešle mu pivo ze svých zásob. Zná tedy koncovou poptávku zákazníka. Podle situace a potřeby poté objednává bedny s pivem u velkoobchodníka. Obdobně postupují všechny články řetězce, kdy objednávají od vyššího článku a nižší článek uspokojují ze svých zásob. Posledním článkem je továrna, která pivo vyrábí a má nekonečnou zásobu surovin. Každá pozice je zatížena informačním zpožděním (z pravidla 2 týdny) a zpožděním při doručení zásob (také zpravidla 2 týdny). Výjimkou je továrna, která má informační zpoždění pouze 1 týden. Doručení zásob ale trvá stejně jako u ostatních rolí a to 2 týdny.

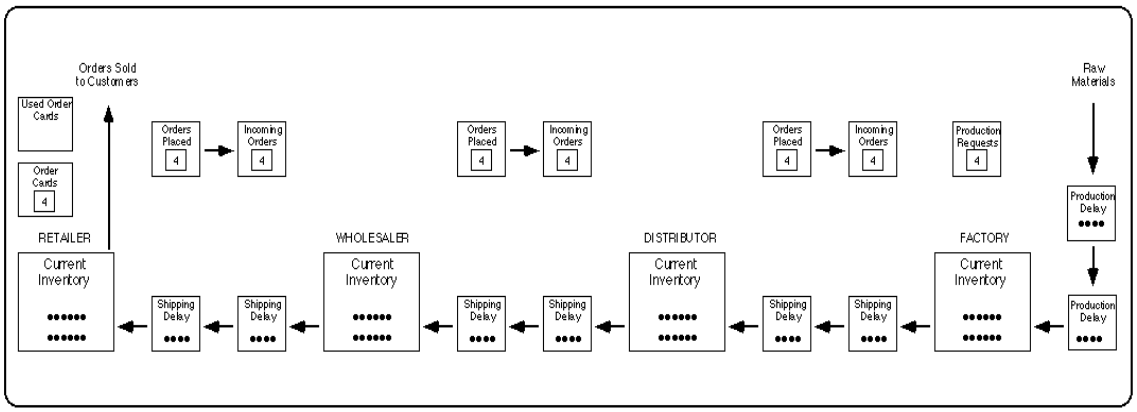
Každý hráč má přehled pouze o svém stavu zásob a objednávek. Jediný, kdo zná koncovou poptávku, je maloobchodník. Hráči si v průběhu hry nesmí sdělovat žádné informace. Zpoždění a omezený okruh informací je tedy důvodem, proč nejsou subjekty schopné koordinovat svá rozhodnutí a plánovat společnou strategii, přestože je jejich úkolem minimalizovat celkové náklady řetězce. (STERMAN, 2000)

Jak už bylo řečeno, cílem hry je tedy minimalizovat celkové náklady řetězce (pozor, ne jednotlivce). Řetězce poté soupeří mezi sebou o dosažení co nejmenší sumy nákladů. Náklady jednotlivých rolí jsou počítány za každé hrací kolo (1 týden) a jsou vypočteny takto:

náklady = 50 x bedny na skladě + 100 x neuspokojené poptávky

Pravidla hry jsou nastavena tak, že je sice možné dosáhnout dobrého výkonu, nicméně téměř vždy dojde k neuspokojivému vývoji, který si hráči zaviní sami svým chováním. Tento rozpor dává dostatek zajímavých podnětů pro rozbor konečného výsledku a pro poučení.

Počáteční konfigurace hry je znázorněna na následujícím obrázku. Každý článek řetězce má na skladě 12 beden piva a všechny objednávky a vyrobené bedny na cestě jsou nastaveny na hodnotu 4.



Obr. 1 Základní konfigurace hry Beer game (STERMAN, 1992)

Pro dosažení co nejnižších nákladů v rámci celého týmu by hráči měli udržovat co nejnižší skladové zásoby a přitom být schopni plnit příchozí požadavky. Pokud hráč nedokáže vyplnit některou objednávku, dostává postih a počet beden, které dluží, se přenáší do dalšího hracího kola, kde může znovu dojít k postihu a k přenesení dlužného počtu beden do dalšího kola. Tento kumulativní charakter nesplněných objednávek si je potřeba dobře uvědomit.

Jediná informace, kterou hráči neznají, je hodnota poptávaného množství beden koncového zákazníka. Zde přichází důležitá pointa hry. Poptávané množství projde za celou dobu jen jednou změnou. I přesto však dojde u většiny týmů k vysokým oscilacím v objednávkách a tudíž i ke špatnému výsledku. První 4 kola je poptávka zákazníku ve výši 4 beden za týden. Po uplynutí 4 hracích kol (týdnů) se poptávka koncových zákazníků skokově zvýší na 8 beden za hrací kolo. Tato změna je neohlášená. Hráčům se sdělí, že hra bude trvat 50 týdnů, z důvodu vyhnutí se efektu horizontu[9] se však přerušuje již po 36 kolech.

3.1 Typické výsledky her

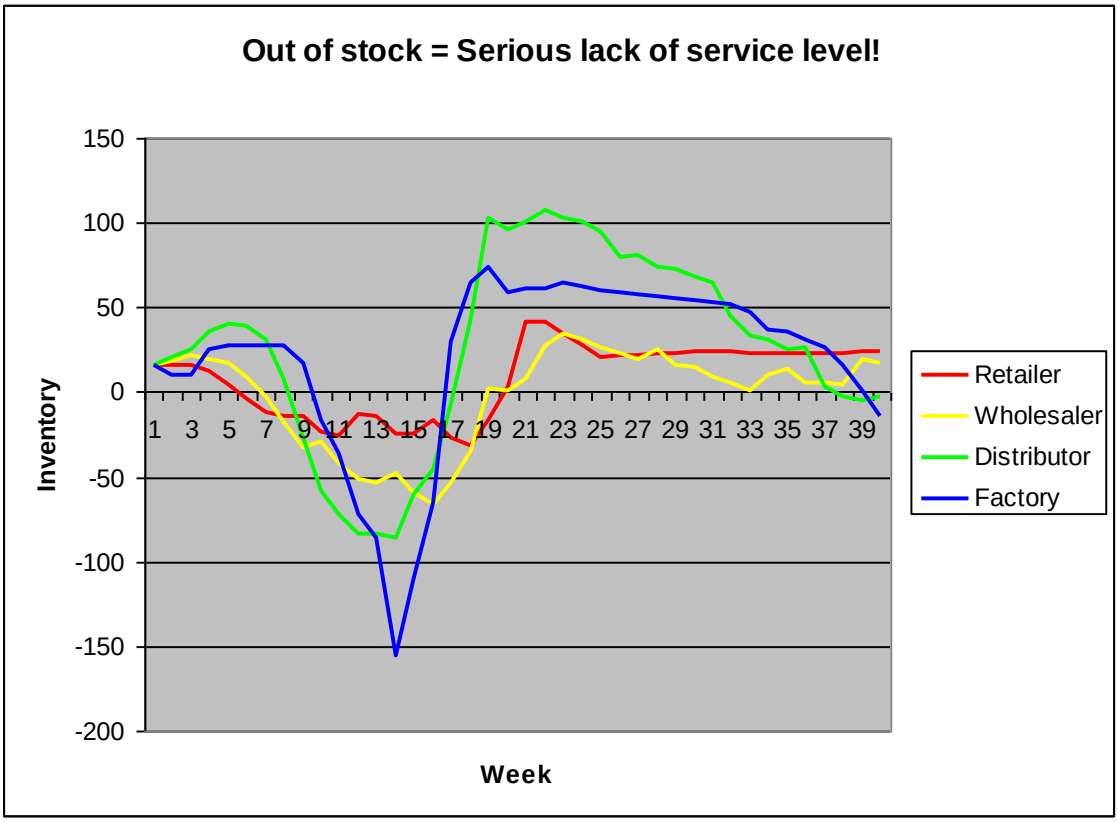
I přesto, že koncová poptávka projde za celou hru jen jedinou změnou, objednávky mezi hráči a stav zásob v průběhu hry divoce oscilují. Tyto oscilace se zvětšují směrem od maloobchodníka k továrně.

Po zvýšení koncové poptávky u zákazníků ze 4 beden na 8 zvýší i maloobchodník své objednávky. Nicméně než se zvýšené objednávky projeví, dojdou maloobchodníkovi jeho zásoby na skladě a vzniknou mu nesplněné zakázky. To vede k dalšímu zvyšování objednávaného množství. K velkoobchodníkovi dorazí se zpožděním zvyšující se objednávky od maloobchodníka a i jemu postupně začnou docházet zásoby na skladě. Stejně jako maloobchodník reaguje a zvyšuje objednávané množství. Takto postupují zvyšující se objednávky až k továrně, kde mohou dosáhnout až několik desítek beden. K maximálním objednávkám a objemům nesplněných zakázek dochází typicky ve 20. až 25. týdnu. Továrna začne vyrábět na

[9] Efekt horizontu má za následek, že hráč začne pod tíhou blížícího se konce měnit svá rozhodnutí.

plné obrátky a zasílat bedny s nápoji směrem k maloobchodníkovi a koncovému zákazníkovi. Hráči si však většinou nejsou schopni spočítat bedny „na cestě“ a v panice způsobené nesplněnými zakázkami objednali příliš. Jakmile k nim tedy začnou bedny se zpožděním dorážet, sklady se jim naopak přeplní a začnou posílat velice nízké objednávky, blížící se k nule. K maximálnímu naplnění skladu dochází většinou mezi 25. a 30. týdnem. (STERMAN, 2000)

Tento průběh se samozřejmě neopakuje vždy a s úplnou přesností, nicméně v základních rysech ho lze pozorovat téměř vždy a platí pro široké spektrum hráčů od středoškolských studentů až po zkušené manažery ve firmách.



Obr. 2 Typický průběh hry Beer game (BEERGAME, 1999)

3.2 Oscilace

V modelu Beer game dominuje objednávkám a zásobám poměrně velká fluktuační amplituda s periodou asi 20 týdnů. V průběhu této amplitudy dojde k poklesu zásob u jednoho článku v řetězci. Následuje pokles zásob u všech článků a subjekty začnou zvyšovat své objednávky. To má za následek vznik záporné čisté zásoby. Maximum nesplněných zakázek nastává typicky v rozmezí 20. až 25. kola. Továrna začne vyrábět na plné obrátky s cílem uspokojit vysoké příchozí požadavky a vyrobené bedny putují směrem k zákazníkovi. Poptávka je naplněna a přebytečné bedny se začnou kumulovat na skaldech. K maximálnímu stavu skladu obvykle do-

chází v 25. až 30. kole. V důsledku přebytečného množství beden na skladě začnou hráči požadovat velmi malé množství beden (blízké k nule), dojde k vyprázdnění skladů a vzniku nesplněných zakázek. Celý průběh se poté opakuje. K demonstraci nám však stačí pouze jedna takováto amplituda. (STERMAN, 1992)

3.3 Zesílení

Amplituda a rozptyl objednávek zesilují od zákazníka směrem k továrně. Maximální objednávky u továrny jsou průměrně 2x větší než objednávky, které dostává maloobchodník. Tento efekt je známý pod pojmem „Efekt biče“ (Bullwhip effect) a bude mu později věnována samostatná kapitola. Právě Efekt biče, který byl zdokumentován v celé řadě studií (např. STERMAN, 2000), je považován za zdroj největších neefektivností v každém produkčně-distribučním řetězci.

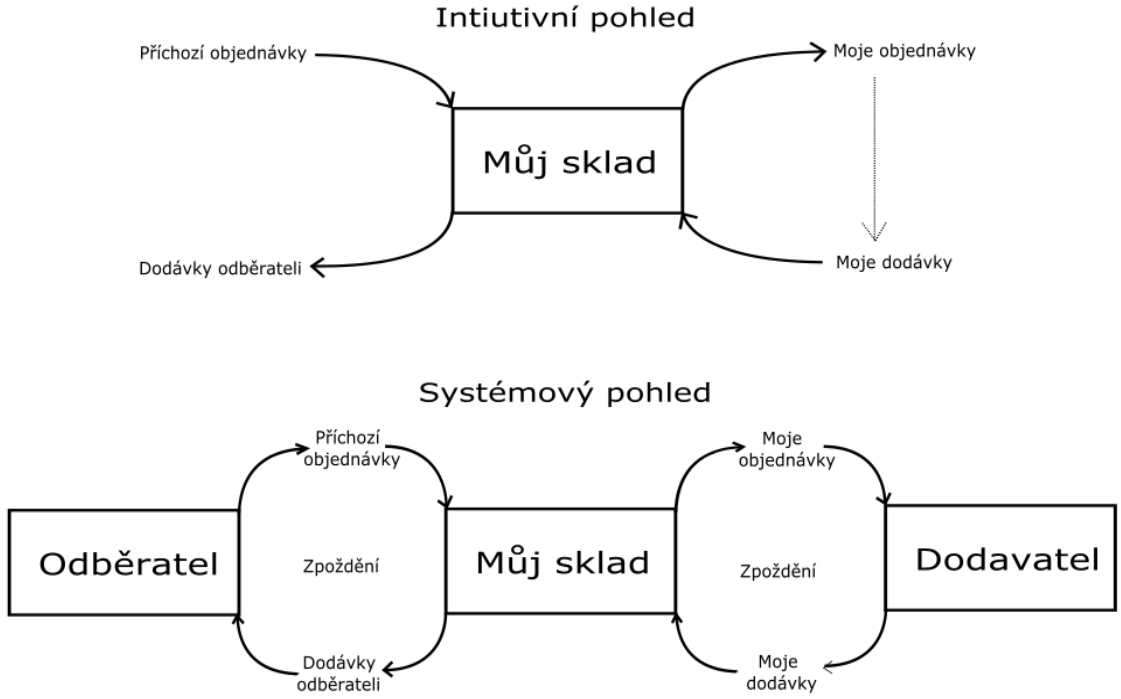
Zesílení ve stavu zásob má ovšem opačný směr. Továrna má typicky menší výkyvy v zásobách a menší minimální úroveň čistých zásob než ostatní články dodavatelského řetězce. To je dáno tím, že celkové zpoždění dodávky pro továrnu činí 3 týdny a tím pádem je schopna doplňovat své zásoby rychleji než ostatní články. Další výhodou pro továrnu je spolehlivost dodávek. Jelikož fiktivní entita, od které továrna objednává, má nekonečnou výrobní kapacitu. Továrně se tedy nemůže stát, že svoji objednávku nedostane. To ostatní role říci nemohou. (STERMAN, 1992)

3.4 Fázové zpoždění

Výše objednávek dosahují svého maxima tím později, čím více se posunujeme vzhůru v řetězci (tj. od maloobchodníka k továrníkovi). V 5. týdnu dosahují maxima objednávky zákazníků, kolem 15. týdne poté objednávky maloobchodníků a velkoobchodníků a u továrny registrujeme maximální objednávky kolem 20. kola. Toto zpoždění je jednak způsobeno nastaveným zpožděním objednávek v systému (ve hře) a také rozhodovacím procesem samotných hráčů. (STERMAN, 1992)

3.5 Debrífing

Velice důležitou částí hry je závěrečné zhodnocení výsledků a rozbor. Přestože je hráč několikrát upozorněn na to, že je důležité dbát na výsledek celého týmu, zajímá ho často pouze vlastní stav zásob a ostatní články řetězce opomíjí. Na následujícím obrázku je zobrazen rozdíl mezi „systémovým“ a intuitivním pohledem hráče.



Obr. 3 Rozdíl mezi systémovým a intuitivním pohledem hráče (STERMAN, 2000)

Zvládnou-li si všichni hráči uvědomit, že jeho objednávky dorážejí se zpožděním a také, že velkými nárazovými objednávkami nejen nevyřeší své problémy, ale naopak je způsobí svému spoluhráči, od kterého objednává, má tato skupina velkou šanci dosáhnout dobrého konečného výsledku.

Další typickou chybou, které se hráči dopouštějí, je tzv. lineární extrapolace trendu doručených objednávek, která vede k neopodstatněnému navýšení odeslaných objednávek. Například, když maloobchodník tři kola po sobě zvyšuje objednávané zboží o 10. Velkoobchodník tento trend extrapoluje a předpokládá, že maloobchodník v tom nejspíše bude pokračovat a rovnou si objedná o 10 beden více. Což sice zní na první pohled rozumně, ale realita je jiná. Velkoobchodník přeci nemůže vědět lépe než maloobchodník, jaká bude konečná poptávka zákazníka. Tento problém navíc zvýrazňuje struktura systému. I když část hráčů uvažuje správně a neudělá uvedené chyby, stačí, aby jeden z týmu začal posílat přehnaně velké požadavky, a ostatní nemají žádnou možnost tuto chybu korigovat a nezbývá jim nic jiného, než dál šířit velké objednávky napříč celým dodavatelským řetězcem.

3.6 Bullwhip efekt

Bullwhip effect, neboli efekt biče, je jeden ze základních fenoménů dodavatelských řetězců. Obecně i malé výkyvy v poptávce dokáží způsobit velké výkyvy v objemech objednávek. Dochází pak k vytváření nadměrných bezpečnostních zásob a tedy i ke zvyšování nákladů. Efekt biče říká, že se výkyvy v zásobách v závislosti na

změně poptávky zvětšují tím více, čím dále se článek pohybuje v dodavatelském řetězci, tedy ve směru od maloobchodníka k továrně.

Efekt biče má za následek neefektivní alokaci zdrojů. Dochází ke snižování konkurenceschopnosti jednotlivých článků dodavatelského řetězce. Velké výkyvy v poptávce tedy znamenají vetší zásoby a tedy i větší vázanost kapitálu a vyšší náklady. Efekt biče popisuje plýtvání kapacitami z důvodu nekomunikace, které se promítá do vyšších konečných cen pro zákazníka. (HERLYN, 2014)

3.6.1 Důvody vzniku Bullwhip efektu a jeho řešení

Hlavním důvodem vzniku Bullwhip efektu je nedostatek informací pro přesnější odhad poptávky. Ve hře Beer game představuje nedostatek informací zpoždění informací o počtu objednávek a také objednaného materiálu. Platí, že čím větší zpoždění je, tím větší je i síla efektu.

Dalším důvodem vzniku bullwhip efektu je objednávání zboží po dávkách. V důsledku to ale znamená, že následující článek v dodavatelském řetězci musí mít na skladě více zásob, aby se vyhnul jeho vyprázdnění. Bullwhip efekt může vzniknout také fluktuací ceny na trhu. Cena se však ve hře Beer game neuvažuje. Významnou roli při vzniku efektu biče představuje stav, kdy poptávka překročí stav na skladě, nebo pokud se jednotlivý článek řetězce domnívá, že by k tomu mohlo dojít. V této chvíli se začne předzásobovat. Cílem je vykrýt (budoucí) zvýšenou poptávku. Pokud je objednávka nevykryta, nebo vykryta jen z části, navýší se tím příští objednávka o nedodané množství, čímž se vytvoří umělá poptávka, na kterou daný článek reaguje objednáním většího množství u svého dodavatele. (XU, 2008)

4 Současný stav

V současnosti je ve VIDA! SC na exponátu Kulatý stůl implementována pouze jedna aplikace a to hra Rybolov. V této hře návštěvníci vysílají na virtuální lov různé velikosti lodí s cílem maximalizovat jejich výlov, avšak udržet stav populace ryb na takové úrovni, aby byla zajištěna jejich reprodukce a tudíž i možnost opětovného výlovu.

4.1 Přehled současných prací

Následující seznam uvádí přehled současných bakalářských a diplomových prací.

- PEŠALOVÁ, L. 2013. Metody a principy v řízení dodavatelských řetězců
- KVASNICA, J. 2012, Síťová asynchronní aplikace Piškvorky
- HÁJEK, J. 2010. Zkoumání chování distribučního řetězce na příkladu Beer Game

Bakalářská práce slečny Pešalové (PEŠALOVÁ, 2013) rozebírá pouze problematiku SCM a jeho základní principy v řízení dodavatelských řetězců. Autorka na začátku práce podrobně vysvětlí pojem dodavatelský řetězec a uvádí také toky, které v dodavatelském řetězci proudí. Následně přehledně rozebírá jednotlivé faktory vedoucí k vývoji SCM. Práce je dobře strukturovaná a srozumitelně popisuje podstatu SCM, jeho cíle a nástroje. Práce je zakončená kritickým zhodnocením daného tématu.

Další prací, která se dotýká tématu této diplomové práce, je bakalářská práce pana Kvasnici (KVASNICA, 2012). Autor zde popisuje a implementuje známou hru piškvorky v programovacím jazyce C#. Stejně jako v této prací využívá asynchronní komunikaci. Práce obsahuje vysvětlení základních i rozšiřujících principů sockotového mechanismu. Autorova výsledná aplikace je též postavena na bázi klient-server a demonstruje základní síťovou komunikaci. Desktopová síťová aplikace/hra je přehledně graficky zpracovaná.

Diplomová práce pana Hájka se zabývá zkoumáním chování distribučního řetězce pomocí systémově dynamického modelu. V teoretické části je představena systémová dynamika, Beer Game a simulační prostředí, ve kterém byl vytvořen model řetězce. V praktické části je analyzována reakce pěti zkoumaných pravidel rozhodování o výši objednávky na 4 vzory poptávky. Práce zcela naplňuje vytyčený cíl a řadí se mezi velmi povedené. Práce vyúsťuje shrnutím výsledků experimentování, doporučením nejvhodnějšího pravidla a návrhem dalšího možného rozšíření a využití modelu v teorii a praxi (HÁJEK, 2010).

4.2 Přehled odborných článků a knih

Následující seznam uvádí přehled odborných článků a knih, jejichž téma souvisí s problematikou řešenou v této diplomové práci.

- EDALI, M. A Mathematical Model of the Beer Game
- MACDONALD, J. FROMMER, D. Decision making in the beer game and supply chain performance
- COPPINI, M. ROSSIGNOLI, Ch. ROSSI, T. Bullwhip effect and inventory oscillations analysis using the beer game model
- STROZZI, F. BOSCH, J. Beer game order policy optimization under changing custo-mer demand. Decision Support Systems
- NIENHAUS, J. ZIEGENBEIN, A. How human behaviour amplifies the bullwhip effect. A study based on the beer distribution game online.
- SARKAR, S. KUMAR, S. Demonstrating the Effect of Supply Chain Disruptions through an Online Beer Distribution Game

V práci (EDALI, 2014) je podobně rozebrána problematika Beer game. Autor sestavil matematický model, který přesně odpovídá originální deskové verzi hry Distribuce limonád. I přesto, že důkladně rozebírá úskalí tvorby matematických modelů, je jeho model sestaven velice precizně se všemi náležitostmi. Podrobně popisuje všechny proměnné modelu, jednotky a parametry. Autor dále uvádí R kód modelu a diskutuje o prospěšnosti R kódu při sestavování a experimentování s modely. Cílem práce bylo pomocí matematického modelu usnadnit případné budoucí studie problematiky Beer game.

J. Macdonald a D. Frommer ve své práci (MACDONALD, 2008) porovnávali výkonost modelu v krátkodobém a dlouhodobém časovém úseku. Analyzovali model uváděný v (STERMAN, 1989), který simuluje rozhodování. V tomto modelu může systém vykazovat chaotické chování v závislosti na heuristice rozhodovacích entit (hráčů). Autoři zkoumali, jak rychle je systém schopen dosáhnout stabilního stavu, ve kterém je efekt biče minimální. Ukázalo se, že krátkodobě výhodná rozhodnutí nemusí vždy znamenat úspěch v dlouhém období. Na konci práce jsou diskutovány výsledky simulací a jejich praktické důsledky.

V práci (COPPINI, 2010) je podrobně zkoumán efekt biče generovaný na každé ze čtyř úrovní dodavatelského řetězce. Autor v svém výzkumu demonstruje paradox efektu biče. Uvádí, že ti, kdo vytváří efekt biče v malém rozsahu, jsou nakonec nejvíce zasaženi jeho důsledky. Autor také zavádí novou jednotku měřitelnosti oscilací skladových zásob. Na konci práce ověřuje, že nově zavedená jednotka má větší vypovídající schopnost o výkonnosti řetězce než měření efektu biče.

Optimální politikou objednávek se zabývá (STROZZI, 2007). Optimální politika je nalezena pomocí genetického algoritmu (GAs). GAs je speciálně navržený algoritmus pro řešení právě těchto problémů. Vyznačuje se vysokou dimenzí prohledávacího prostoru, jelikož funkce celkového výkonu řetězce má mnoho lokálních mi-

nim. Autor na základě svého výzkumu uvádí, že nejlepšího výkonu lze dosáhnout pouze rozdílnou strategií objednávek u každého článku v řetězci.

Důkladný rozbor problematiky Beer game je obsažen v práci (NIENHAUS, 2006). Autor se svými teoretickými znalostmi opírá o celou řadu odborné literatury a podrobně popisuje efekt biče se všemi jeho náležitostmi. Poté vytvořil webovou simulaci hry Distribuce limonád, která dává možnost porovnat lidské rozhodování s jednoduchými strategiemi. Poukazuje na to, že aspekty lidského chování musí být rozpoznány dříve, než dojde k zesilování efektu biče. Autor prokazuje, že výměna informací nad rámec pouhého předávání objednávek snižuje efekt biče, nicméně lidský faktor působí jako překážka pro tok informací v dodavatelském řetězci.

Tvorbou výukového nástroje se zabýval článek (SARKAR, 2016). Tento nástroj je adaptací Beer Game a demonstruje snížení efektu biče při sdílení informací a spolupráci. Hra obsahuje několik scénářů podle toho, jakým směrem mohou být informace sdíleny a jaká z rolí je poskytuje. Hra dává mnoho možností k diskuzi o pozitivním dopadu sdílení informací v dodavatelském řetězci.

4.3 Problematika supply chain management

Dnešní, dynamicky se rozvíjející, trh představuje pro mnohé organizace zásadní změny v jejich struktuře a chování. Konkurenční prostředí nutí firmy chovat se efektivně při správě svých interních a externích zdrojů. Být perspektivní a úspěšný na trhu znamená pro mnohé organizace vytvářet nové obchodní modely, odstraňovat překážky komunikace, zjednodušit procesy výroby a dodávky. Cílem je pružně reagovat a uspokojovat stále se měnící potřeby zákazníků. Moderní model spolupráce vyžaduje schopnost propojení heterogenních procesů a možnost rychlé realizace nových myšlenek a nápadů bez rozdílů, kde vznikly, nebo na které konkrétní platformě pracují. Právě tuto problematiku se snaží řešit nástroje pro SCM. (LAMBERT, 2008)

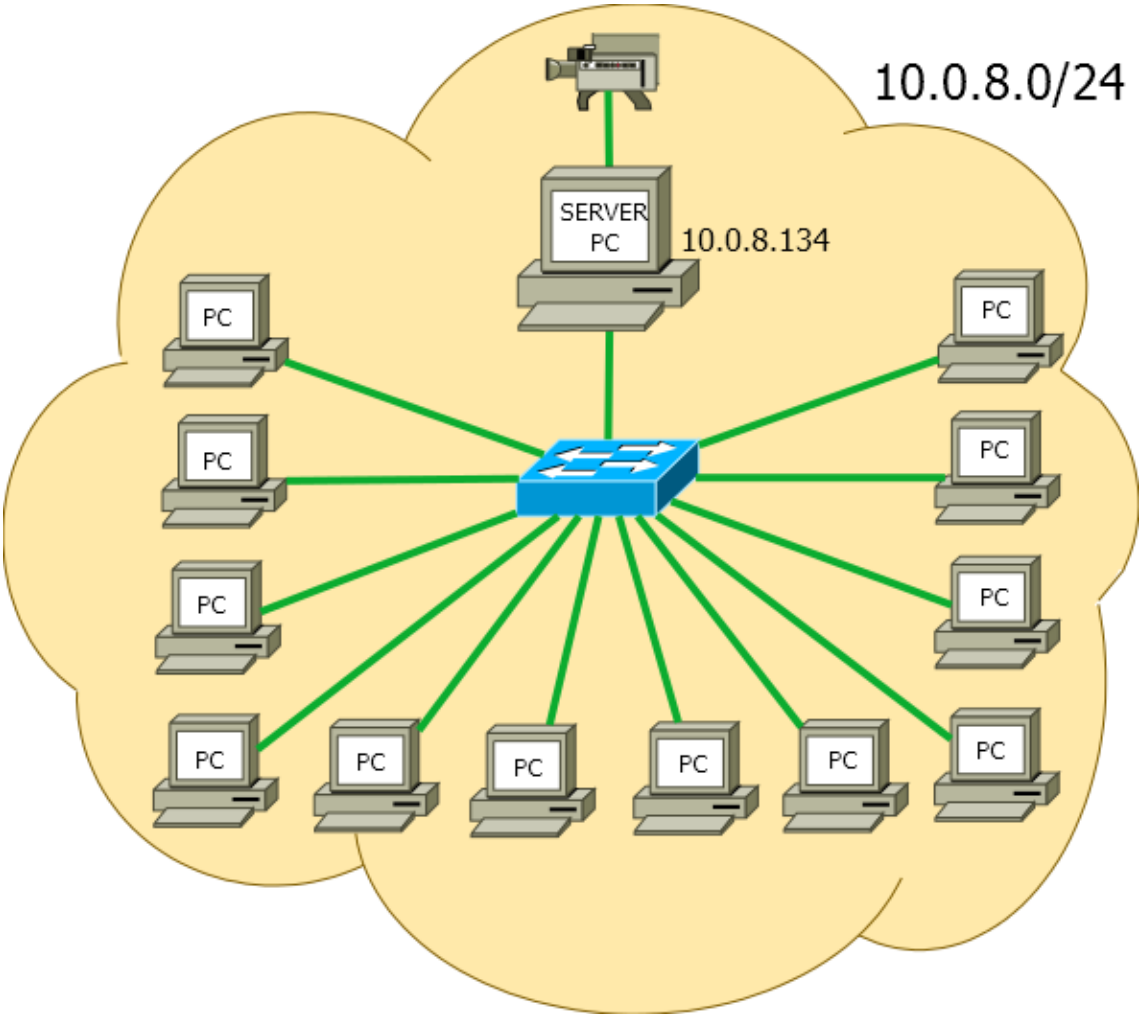
SCM v poslední době velice rychle nabylo na významu a řadí se mezi moderní přístupy dnešní doby. Cílem této práce však není detailní popis všech částí SCM, nýbrž poukázat na jeho důležitost a zábavnou formou naučit návštěvníky VIDA! SC základním principům v této oblasti.

5 Požadavky a analýza

V této kapitole je zpracována analýza požadavků, které vznikly na základě vzájemné komunikace s pracovníky VIDA! SC o konečné podobě softwaru.

5.1 Analýza exponátu Kulatý stůl

Exponát Kulatý stůl je jeden z mnoha interaktivních exponátů ve VIDA! SC. Skládá se z dvanácti navzájem propojených počítačů a jedním centrálním, ke kterému je připojen projektor k zobrazování aktuálních informací. Uživatel poté pomocí jednoho z dvanácti dotykových monitorů okolo stolu interaguje s centrální stanicí, která dle potřeby zobrazuje informace za pomocí projektoru. Všechny počítače spolu komunikují prostřednictvím LAN s využitím technologie Ethernet.



Obr. 4 Schéma exponátu kulatý stůl

5.1.1 Technická specifikace

Koncovou stanici představuje patnáctipalcový dotykový LCD displej ELO TOUCHSYSTEMS 1537L s rozlišením 1024 x 768 pixelů, který je připojen pomocí konektoru DVI-D k počítači s operačním systémem Windows 7. Páteř sítě LAN je tvořena switchem TP-LINK TL-SG1016D.

5.1.2 Watch dog

Jedná se o aplikaci třetí strany, která je schopna periodicky kontrolovat přítomnost procesu v OS. V případě, že daný proces neběží, tak jej spustí. Tato aplikace je součástí každého klientského počítače exponátu a je využívána i pro hru distribuci limonád. Přijde-li skupina hráčů k exponátu s úmyslem si zahrát distribuci limonád, bude už tato hra spuštěna na koncových stanicích. Jakmile dohrají, serverová aplikace hry provede svůj restart do iniciálního nastavení a Watch dog znovu spustí klientské aplikace, které se na server připojí a hra je znovu připravena na příchod další hrací skupiny.

5.2 Stanovení požadavků na komunikaci

Cílovým exponátem pro výukovou aplikaci Distribuce limonád je tedy výše zmíněný exponát Kulatý stůl. Klíčovým požadavkem pro správné fungování aplikace bude zajistit nezávislou komunikaci mezi jednotlivými hráči s centrálním počítačem, který bude obsluhovat přicházející požadavky a zasílat zpět odpovědi příslušným koncovým stanicím, resp. hráčům. Tato komunikace by měla probíhat za pomocí LAN a měla by být velice spolehlivá. Nesmí se stát, aby cílová stanice obdržela neúplnou zprávu či vůbec žádnou. Proto je nutné, aby komunikace probíhala přes protokol TCP, který garantuje spolehlivé doručování a správné pořadí paketů.

5.3 Stanovení požadavků na vzhled aplikace

Dalším, neméně důležitým požadavkem, je grafická přehlednost a snadné ovládání aplikace. Aplikace by měla svým rozložením ovládacích prvků dát najevo, v jaké části dodavatelského řetězce se konkrétní hráč nachází. Samotné ovládání hry by mělo být intuitivní. Cílem je minimalizovat čas, kdy se hráč bude rozkoukávat a zjišťovat, co je potřebné udělat k tomu, aby odehrál a ukončil hrací kolo.

Rozšiřujícím postulátem z pohledu bezpečnosti exponátu je nedat uživateli možnost nikterak ze hry vyskočit. V případě, že by se tak stalo, mohl by uživatel pomocí dotykové obrazovky přímo zasáhnout do operačního systému Windows a ohrozit tím chod celého exponátu.

5.4 Stanovení požadavků na funkcionalitu

Aby nemusela obsluha exponátu neustále kontrolovat jeho stav a dokola spouštět všechny aplikace na klientských stanicích, je nutné zajistit, aby se hra po jejím

skončení sama inicializovala. Do jisté míry se o toto stará aplikace Watch dog. Serverovou aplikaci však Watch dog nespravuje a proto je nezbytné zajistit, aby restart učinila aplikace sama.

Na základě dohody s pracovníky VIDA! SC bylo ujednáno, že aplikace by měla zobrazovat tyto informace: aktuální kolo, aktuální náklady hráče, příchozí a odchozí počet beden, příchozí a odchozí požadavky (objednávky), skladové zásoby a nesplněné požadavky. Aplikace začne tím, že zobrazí hráči pravidla hry, které budou srozumitelné a snadno pochopitelné. Poté, co si je hráč přečte, bude přesměrován na hlavní obrazovku. Jakmile hráč odehraje své kolo, budou mu všechny ovládací prvky skryty a skutečnost, že čeká na ostatní, až také dohrají, mu bude dána jasně najevo.

Důležitým požadavkem na funkcionalitu je automatická volba role. Tento požadavek úzce souvisí s požadavkem na rozdílnost rozložení ovládacích prvků v závislosti na roli. Serverová aplikace by měla podle potřeby role klientům přidělovat a měla by si hlídat počet připojených hráčů. Jakmile se hra naplní (počet hráčů je roven 4), serverová aplikace odmítne každé další příchozí spojení a nedá tím možnost se 5. hráči připojit.

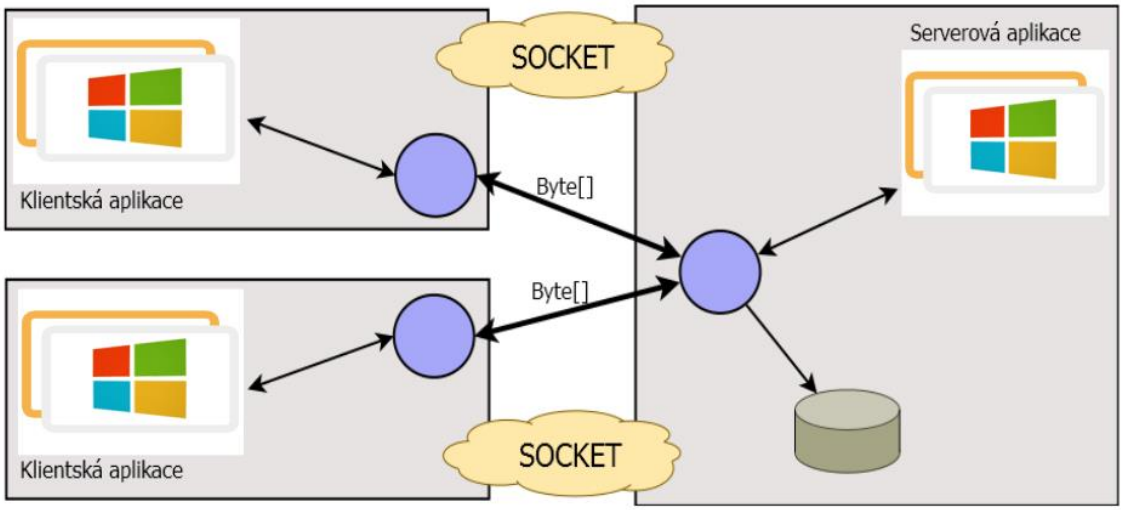
V případě nějaké anomálie např. pádu aplikace či chyby sítě by měla serverová aplikace na všech zbylých klientech korektně hru ukončit a sama provést svůj restart.

6 Návrh řešení

6.1 Návrh socketové komunikace

Výsledná aplikace se tedy bude skládat ze dvou částí. Serverová aplikace a klientská aplikace, která se bude na server připojovat. Serverová aplikace dostane od řídicího systému přidělený port, na kterém bude naslouchat a pod kterým bude spuštěna jedna instance hry. Číslo portu bude přebírat přes parametr aplikace.

Serverová aplikace bude dále udržovat spojení s klientem a bude si hlídat počet připojených klientů do hry. Komunikaci bude vždy iniciovat klient (požadavek) a server odpoví. Bude zaveden systém komunikačních značek, které budou od sebe odlišovat jednotlivé druhy komunikace a tím i odlišné reakce serveru. Jako zpráva bude zasílána unifikovaná struktura v bytové podobě, která bude podle její struktury příjemcem rozparsována. Schéma socketové komunikace je zobrazeno na následujícím obrázku.

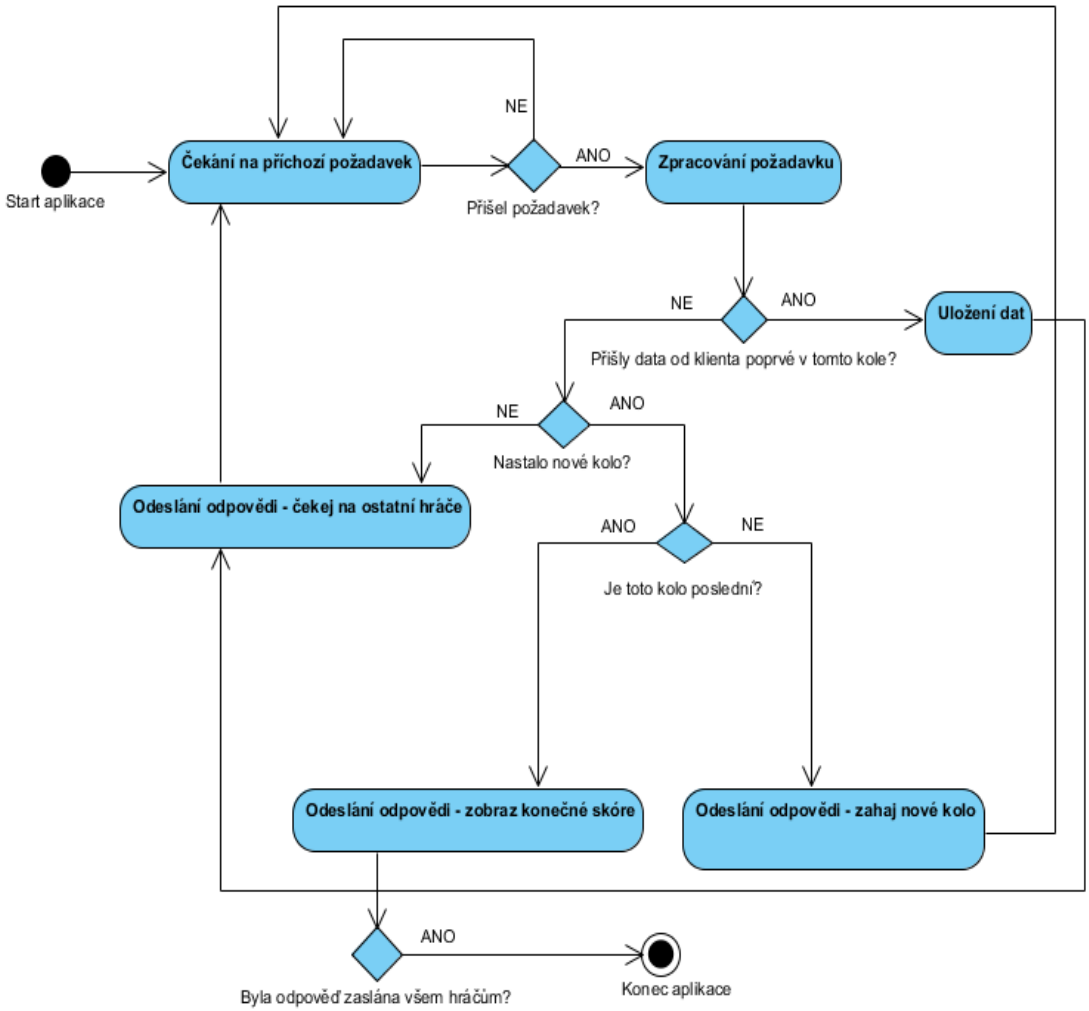


Obr. 5 Schéma navržené socketové komunikace

Jakmile všichni hráči odešlou své požadavky, serverová aplikace rozpozná tento stav a zahájí nové kolo.

Po skončení hry serverová aplikace postupně odpojí všechny klientské a následně se znovu spustí v iniciálním stavu. To zaručí, že jakmile aplikace Watch dog spustí novou klientskou aplikaci na některém z počítačů, bude se mít tato aplikace kam připojit a hra bude moci začít.

Podrobnější návrh socketové komunikace popisuje až následující diagram aktivit, který ukazuje, jakým způsobem by serverová aplikace měla reagovat na příchozí požadavky klientů. Každý příchozí požadavek bude zpracován zvlášť. Bude-li se jednat o data, která jsou v probíhajícím kole nová, budou data uložena a klient uveden do stavu čekání, až kolo dohrají i ostatní hráči. Jakmile tento stav nastane, server odešle všem klientům zprávu, která vyvolá nové kolo, a hráči budou moci ve hře pokračovat. Hra bude ukončena v momentě, kdy bude odehráno poslední kolo a všem klientům bude odeslána odpověď, která na všech počítačích zobrazí konečné skóre skupiny.

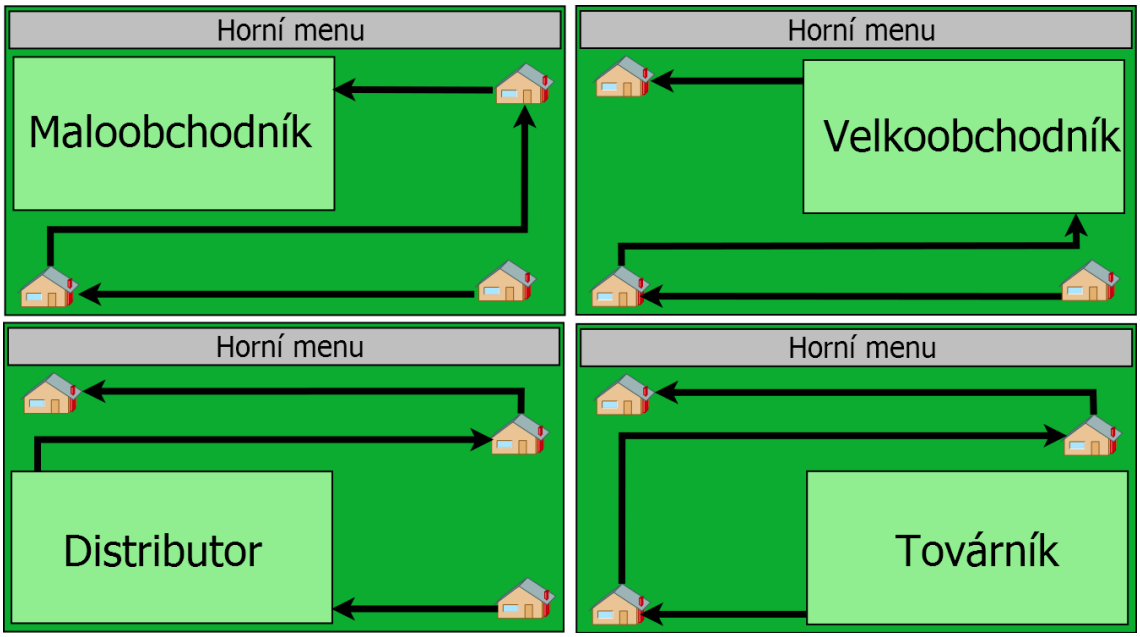


Obr. 6 Diagram aktivit komunikace z pohledu serveru

6.2 Návrh vzhledu aplikace

Proto, aby si každý hráč dokázal představit, jakou roli a pozici v dodavatelském řetězci zastupuje, je nutné každému z nich zobrazit lehce odlišné rozložení aplikace. Klientská aplikace dopředu neví, pod jakou rolí a s jakým rozložením se bude

spouštět. Hned poté, co se klient připojí na serverovou aplikace, proběhne konfigurace klientské aplikace. Tato komunikace je oddělena od ostatní a probíhá jen a pouze na startu klientské aplikace. Server obdrží požadavek na konfiguraci, vyhodnotí, jaká role v řetězci ještě není zastoupena žádným hráčem a tuto roli klientovi vzápětí přidělí. Klient poté jen zareaguje na příchozí odpověď a nastaví příslušné rozložení prvků. Následující obrázek zobrazuje rozdílné možnosti rozložení v rámci všech rolí v řetězci.

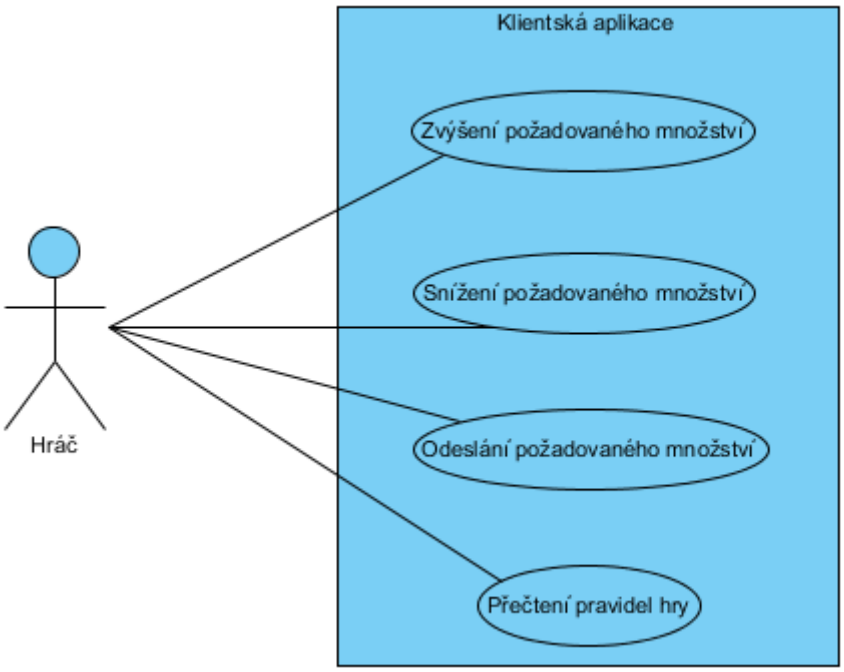


Obr. 7 Rozložení hrací obrazovky v závislosti na roli hráče

Aby hráč neměl žádnou možnost ovlivnit svým chováním bezpečný chod celého exponátu, poběží aplikace v režimu fullscreen. Navíc bude skryt horní panel, který by teoreticky umožňoval hru minimalizovat či vypnout. K tomu, aby hráč nemohl použít žádnou z klávesových zkratek, musí být zaručeno, že nebude moci v prostředí hry vyvolat softwarovou klávesnici, na které by mohl klávesovou zkratku stlačit. Proto je nutné v celém prostředí u všech ovládacích prvků tuto možnost potlačit.

6.3 Návrh požadované funkcionality

Pro přehledné znázornění navrhované funkcionality jsem zvolil diagram užití (usecase diagram). Usecase diagram znázorňuje interakci uživatele (aktéra) se systémem. V mém případě v roli aktéra vystupuje hráč a systém představuje klientská aplikace, kterou hráč ovládá. Jednotlivé případy užití jsou úkony, které hráč může v klientské aplikaci provádět. Jak je vidět na následujícím obrázku, uživatel příliš mnoho možností mít nebude. Hráčova pozornost se bude soustředit pouze na to, aby zvolil množství, které si bude přát nechat dodat. Tím se hra patřičně zjednoduší a zpřehlední.



Obr. 8 Usecase diagram hráče a klientské aplikace

7 Použité technologie

7.1 Hardwarové prostředky

7.1.1 LAN

Všechny počítače (i centrální stanice) exponátu kulatý stůl jsou připojeny k jedinému přepínači značky TP-LINK model TL-SG1016D kroucenou dvojlinkou se standardní koncovkou typu RJ-45.

7.2 Programovací jazyk C#

Jedná se o vysokoúrovňový, objektově orientovaný programovací jazyk vyvíjený firmou Microsoft. Jazyk C# je součástí platformy .NET, která představuje jak rozsáhlou knihovnu pro tvorbu aplikací tak i operační prostředí (.NET runtime), v němž jsou programy spouštěny. C# je navržen pro psaní aplikací jak pro zařízení se sofistikovanými operačními systémy, tak pro zařízení s omezenými možnostmi. Přestože by programy psané v C# neměly plýtvat s přiděleným procesorovým časem a pamětí, nemohou se měřit s aplikacemi psanými v C nebo jazyce symbolických adres.

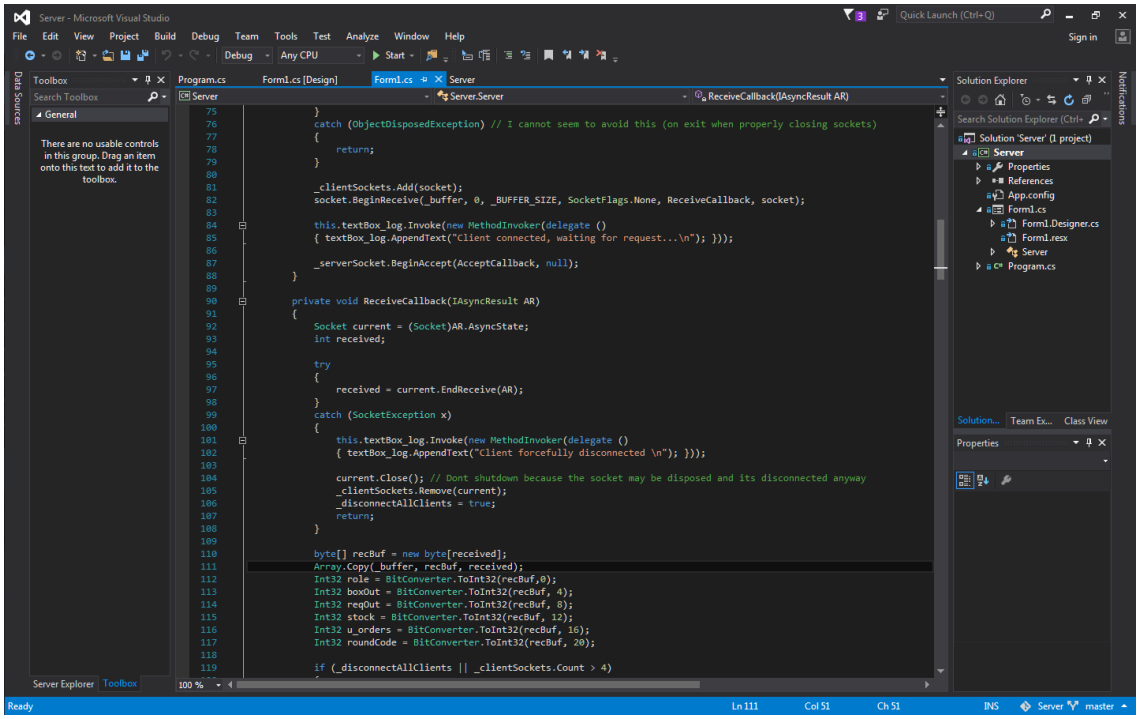
V C# neexistuje vícenásobná dědičnost. V praxi to znamená, že každá třída může být potomkem nejvýše jedné třídy. Dále neexistují žádné globální proměnné a metody, všechny musí být deklarovány uvnitř tříd. Náhradou za globální proměnné a metody jsou statické metody a proměnné veřejných tříd.

V objektově orientovaném programování se z důvodu dodržení principu zapouzdření často používá vzor, kdy k datovým atributům třídy lze zvenčí přistupovat pouze nepřímo, a to pomocí dvou metod: metody get (accessor) a metody set (mutator). V C# lze místo toho definovat tzv. property, která zvenčí stále funguje jako datový atribut, ale uvnitř obsahuje prostor pro definici obou těchto metod. Výhodou je jednodušší práce s datovým atributem při zachování principu zapouzdření. Jazyk C# rozlišuje mezi velkými a malými písmeny, je tedy case sensitive. (C Sharp, 2016)

7.3 Visual studio 2015

Jedná se o vývojové prostředí od firmy Microsoft, které může být použito k vývoji celé řady aplikací jak s grafickým rozhraním (Windows Forms) tak například pro webové aplikace, mobilní zařízení apod. Visual Studio 2015 obsahuje editor kódu podporující IntelliSense a refaktorování. Během psaní kódu jej Visual Studio 2015 na pozadí kompiluje, aby poskytlo informace o syntaktických a kompilačních chybách, které jsou podtrženy červenou vlnovkou. Kompilace na pozadí negeneruje spustitelný kód, protože používá jiný kompilátor než ten, který generuje spustitelný kód. Integrovaný debugger pracuje jak na úrovni kódu, tak na úrovni stroje. Další vestavěné nástroje zahrnují designer formulářů pro tvorbu aplikací s GUI,

designer webu, tříd a databázových schémat. Vyvíjené řešení je snadno možné připojit na některý z verzovacích systémů.



Obr. 9 Ukázka vývojového prostředí Visaul Studio 2015

8 Řešení

V této kapitole bude detailně popsán vývoj a konečná podoba výukové aplikace. Výsledná aplikace se skládá ze dvou navzájem spolupracujících programů. Klientská aplikace běžící na koncových stanicích a serverová aplikace, která je spuštěna na centrálním počítači exponátu, naslouchá na zvoleném portu a čeká, až se na ni klientské aplikace připojí.

8.1 Serverová aplikace

Hlavním úkolem této aplikace je řídit průběh celé hry. Jedná se prakticky o program, který v sobě obsahuje veškou logiku hry Distribuce limonád a podle jejich pravidel ovlivňuje chování koncových (klientských) aplikací. Jelikož exponát kulatý stůl umožňuje hrát hru Distribuce limonád až třem skupinám zároveň, je nezbytné, aby každá instance serverové aplikace naslouchala na odlišném portu. O jeho přidělení by se měl v budoucnu postarat řídicí systém exponátu, který je vyvíjen v rámci diplomové práce pana Saviče. V současnosti je port zvolen staticky, avšak aplikace je na toto rozšíření připravena a potřebná funkcionalita je již implementována a otestována. Klíčovým konceptem, který je pro správné fungování obou aplikací nezbytný, je asynchronní síťová komunikace. Této problematice je vyčleněna samostatná kapitola.

Serverová aplikace každému hráči dynamicky přiděluje roli v dodavatelském řetězci (továrník, distributor…), tudíž hra není zatěžkána žádnými počátečními úvahami o volbě role. Jelikož je počet hráčů této hry stanoven na číslo čtyři, serverová aplikace si v sobě uchovává počet připojených instancí a v případě, že by se do hry chtěl připojit pátý, toto spojení odmítne.

Jakmile dojde k odpojení posledního hráče od serveru, server tento stav rozpozná, nastartuje novou instanci serveru, který poslouchá na stejném portu jako původní a sebe ukončí. Díky tomu bude na centrálním počítači existovat vždy (minimálně) jedna instance serverové aplikace a tím pádem bude zaručeno, že klientské aplikace se budou mít kam připojit a hra začne, jakmile k tomu dá hráč pokyn.

8.1.1 Rozbor důležitých částí programového kódu

Následující kód popisuje unifikovanou strukturu zprávy serverové aplikace, která je zasílána všem klientům (hráčům). Jak je možné vidět, zpráva obsahuje identifikátor role, pod kterou byla zpráva odeslána, dále počet beden, které odesílá nadřazenému článku v řetězci, počet beden, které požaduje od podřízeného článku a číslo kola, ve kterém byly tyto hodnoty odeslány. Konstruktor struktury převezme čtyři parametry typu Integer a uloží je. Následně je volaná metoda *getMessageByteArray,* která převede tyto hodnoty do pole bytů a vrátí jej. Toto pole je poté posíláno sítí ke všem klientům, kde je díky známé struktuře aplikován opačný postup pro získání hodnot z bytového pole.

```
public struct Message
{
      Int32 role;       // identifikátor role
      Int32 boxOut;     // počet odchozích beden
      Int32 boxReqOut;  // počet požadovaných beden
      Int32 roundCode;  // identifikátor kola (hracího týdne)

// konstruktor struktury
public Message(Int32 p_role, Int32 p_boxOut, Int32 p_boxReqOut, Int32
p_roudCode)
{
      role = p_role;
      boxOut = p_boxOut;
      boxReqOut = p_boxReqOut;
      roundCode = p_roudCode;
}
// převedení všech čísel do bytového pole a vrácení tohoto pole
public byte[] getMessageByteArray()
{
      byte[] data1 = BitConverter.GetBytes(role);
      byte[] data2 = BitConverter.GetBytes(boxOut);
      byte[] data3 = BitConverter.GetBytes(boxReqOut);
      byte[] data4 = BitConverter.GetBytes(roundCode);
      byte[] data = new byte[data1.Length + data2.Length + da-
      ta3.Length + data3.Length];

      Buffer.BlockCopy(data1, 0, data, 0, data1.Length);
      Buffer.BlockCopy(data2, 0, data, data1.Length, data2.Length);
      Buffer.BlockCopy(data3, 0, data, data1.Length + data2.Length,
      data3.Length);
      Buffer.BlockCopy(data4, 0, data, data1.Length + data2.Length +
      data3.Length, data4.Length);
      return data;
}}
```

Serverová aplikace každé hrací kolo ukládá průběhy her do souborů. Pro jednoduchost následného vyhodnocení byl zvolen formát CSV. Je nutné zajistit, aby v jednom okamžiku nezapisovalo do souboru více vláken aplikace, o to se stará zámek, který pouští do kritické sekce vždy pouze jedno vlákno. Zkrácená podoba metody pro ukládání do souboru vypadá následovně:

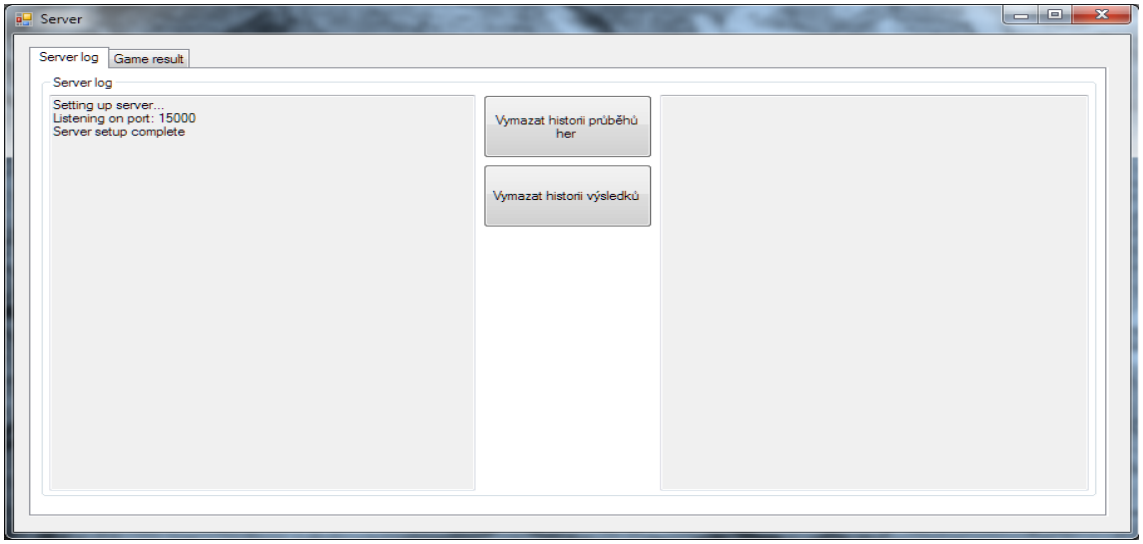
```
public void writeToFile(int role, int stock, int u_orders)
{
...
lock (_locker) // zámek kritické sekce zápisu do souboru
{
      if(!File.Exists(path))
      {
            string header = "ID;KOLO;ID_HRAC;HODNOTA;";
            // využití objektu StreamWriter k zápisu do souboru
```

```
            using (StreamWriter sw = File.CreateText(path))
            {
                  sw.WriteLine(header);
            }
      }
      // zapsání hodnot do souboru
      using (StreamWriter sw = File.AppendText(path))
      {
            sw.WriteLine(barrels);
      }}}
```

Ve hře proti sobě směřují dva toky informací. Jeden tok představují fyzické bedny, které putují od továrníka k maloobchodníkovi a zákazníkovi a druhý tok představují objednávky, které putují od zákazníka resp. maloobchodníka k továrníkovi. Oba tyto toky jsou v serverové aplikaci reprezentovány samostatnými poli hodnot. Každá role má určenou neměnnou kombinaci indexů do pole s bednami a do pole s objednávkami. Server podle toho, jakou roli chce obsloužit, načte data z polí pod těmito indexy a zašle je klientovi. Posun všech hodnot v poli s bednami nebo v poli s objednávkami pak zajistí pouhé vložení nové hodnoty na začátek pole. Výsledná podoba metody je uvedena v příloze A v sekci *sendDataByRole*.

8.1.2 Grafické rozhraní

Cílem serverové aplikace není nikterak oslnit svým vzhledem, ale spíše zaručit plynulý a bezproblémový průběh hry. Serverová aplikace je spuštěna na pozadí a nikterak nevyužívá centrální projektor exponátu. Z tohoto důvodu obsahuje pouze dva pomocné logy pro možnost zpětného dohledání chyb a dvě tlačítka pro mazání historie všech průběhů her a historie výsledků pouze dohraných her.



Obr. 10 Ukázka grafického rozhraní serverové aplikace

8.2 Klientská aplikace

Cílem klientské aplikace je poskytnout hráči přehledné prostředí, ve kterém se snadno zorientuje a jeho pozornost nebude rušena žádnými přebytečnými elementy. Chování klientské aplikace je řízeno serverovou aplikací a uživatel má pouze omezené pole působnosti. Jedinou možností, jak hru ovlivnit, je odeslat objednávku na počet beden podřízenému článku řetězce. Ač to na první pohled nemusí být zjevné, hráč nemůže ovlivnit počet beden, které odesílá nadřazenému článku v řetězci. Aplikace tento krok dělá za něj a to podle toho, jak vysoký je příchozí požadavek a kolik beden má v dané chvíli hráč na skladě. V případě, že příchozí požadavek (objednávka) je větší jak počet beden na skladě, je nucen hráč dát vše, co na skladě má a rozdíl se mu započítá v podobě nesplněných zakázek, které se přenášejí do dalších hracích kol. V případě, že příchozí požadavek (objednávka) je menší jako počet beden na skladě, hráč odešle přesně tolik beden, kolik je požadováno a zbytek mu na skladě zůstává. Je tedy zřejmé, že po každém hracím kole musí být počet nesplněných zakázek nebo počet beden na skladě rovný nule.

Klientská aplikace potřebuje ke svému spuštění dva parametry a to číslo portu, na kterém naslouchá serverová aplikace a IP adresu počítače, kde serverová aplikace běží. Současný stav exponátu (bez řídicího systému) však nedovoluje spuštění aplikace se dvěma parametry, jelikož nynější řídicí systém exponátu byl vyvíjen externí společností, která nenabídla možnost vkládání nových her. Implementace soudobého řídicího systému je zcela odstíněna od možnosti customizace a pracovníci VIDA! SC nemohou do chodu systému nijak zasáhnout. Číslo portu a IP adresa jsou proto přímo součástí programového kódu, nicméně vše je pro budoucí stav exponátu připraveno a potřebná funkcionalita je již implementována.

8.2.1 Rozbor programového kódu

Unifikovaná struktura zprávy, která putuje od klientů k serveru, je mírně odlišná od té, která putuje opačným směrem. Každý klient posílá navíc ještě stav svého skladu a výši nesplněných zakázek. Poslední hodnotou zprávy je její kód. Ten slouží k oddělení různých druhů komunikace mezi klientem a serverem. Například pro načtení počáteční konfigurace klientské aplikace je jiný kód jako pro uvedení klientské aplikace do stavu čekání na dokončení kola ostatních hráčů.

Zpracování předdefinované struktury zprávy však probíhá podle stejného principu, jako tomu bylo u serverové aplikace. Výsledná podoba zdrojového kódu je uvedena v příloze B, sekce *Struktura zprávy*.

Stěžejní metodou klientské aplikace je metoda *SendRequest()* pro odesílání požadavků na server. Tato metoda je volána stisknutím tlačítka odeslat, kterým každý hráč ukončuje své hrací kolo. Jejím úkolem je posbírat potřebná data, sestavit zprávu a zaslat ji serverové aplikaci. Mimo to jsou ještě volány další pomocné metody, které provádí například aktualizaci nákladů či skrytí ovládacích prvků.

```
private void SendRequest()
{
      try
      {
      // načtení aktuálních hodnot z aplikace
      int un_order = (int)stringToInt(lbl_dluh.Text);
      int stock = (int)stringToInt(lbl_sklad.Text);
      int customerRequest = (int)stringToInt(lbl_reqIn.Text);
      updateCLientStockAndUnfulfilledOrders(un_order, stock);

      add2CostSum(); // přičtení nákladů do celkové sumy
      updateScore(); // zobrazení aktuálních nákladů
      add2Chart(customerRequest); // vynesení bodů do grafů
      _roundNumber++; // navýšení čísla kola

      // sestavení zprávy k odeslání
      Message m = new Message(_ROLE,
      (Int32)stringToInt(lbl_boxOut.Text),
      (Int32)stringToInt(lbl_reqOut.Text), _stock, _unfulfilledOrders,
      -500);

      // převedení sestavené zprávy do pole bytů
      byte[] buffer = m.getMessageByteArray();

      resetAllCells(); // reset prvků do iniciálního stavu
      hideControls(); // skrytí všech viditelných prvků

      // odeslání dat na server
      _clientSocket.Send(buffer, 0, buffer.Length, SocketFlags.None);
      }
      catch(Exception e) // nastala chyba
      {
            ...
      }}
```

V aplikaci existují ještě dvě obdobné metody, které umožňují odeslat specifický požadavek na server. Jsou to metody *SendWaitingRequest()* a *loadConfigurationRequest()*. V případě, že hráč odehraje svoje kolo, dostává se do stavu, kdy čeká na ostatní, až také tak učiní a server zahájí kolo nové. Během této doby se klientská aplikace periodicky ptá serveru, jestli už tak nenastalo. K tomuto právě slouží metoda *SendWaitingRequest()* resp. kód zprávy -300.

```
private void SendWaitingRequest()
{
      Message m = new Message(_ROLE, 300, 300,300,300,-300);
      byte[] buffer = m.getMessageByteArray();
      _clientSocket.Send(buffer, 0, buffer.Length, SocketFlags.None);
}
```

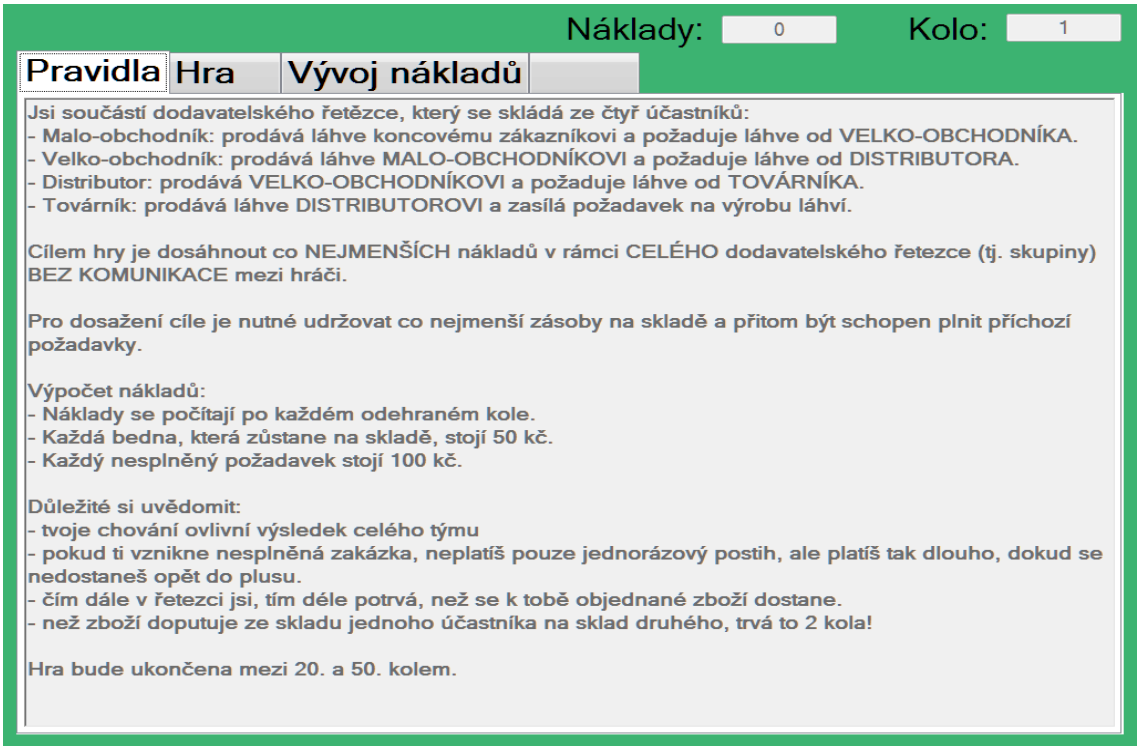
Po počátečním připojení klienta na server si musí mezi sebou tyto dvě aplikace domluvit roli, pod kterou se má klientská aplikace nastavit. K tomu slouží metoda *loadConfigurationRequest()* resp. kód zprávy -600.

```
private void loadConfigurationRequest()
{
      Message m = new Message(_ROLE, 600, 600, 600, 600, -600);
      byte[] buffer = m.getMessageByteArray();
      _clientSocket.Send(buffer, 0, buffer.Length, SocketFlags.None);
}
```

Naopak, aby byla klientská aplikace schopna data od serveru přijmout, byla pro tento účel implementována metoda *ReceiveResponse()*. Ta nejdříve rozparsuje příchozí pole bytů podle předem známé struktury zprávy, převede jednotlivé sekvence bytů na číselné hodnoty a poté reaguje na příchozí kód zprávy, který klientské aplikaci jasně říká, jak se má zachovat. Jelikož se jedná o poměrně obsáhlou metodu, je uvedena v příloze B v sekci *RecieveResponse*.

8.2.2 Grafické rozhraní

Po připojení aplikace na server a dohodnutí role jsou hráči zobrazena pravidla hry. V celém GUI je potlačeno vyvolání softwarové klávesnice z důvodu bezpečnosti exponátu.

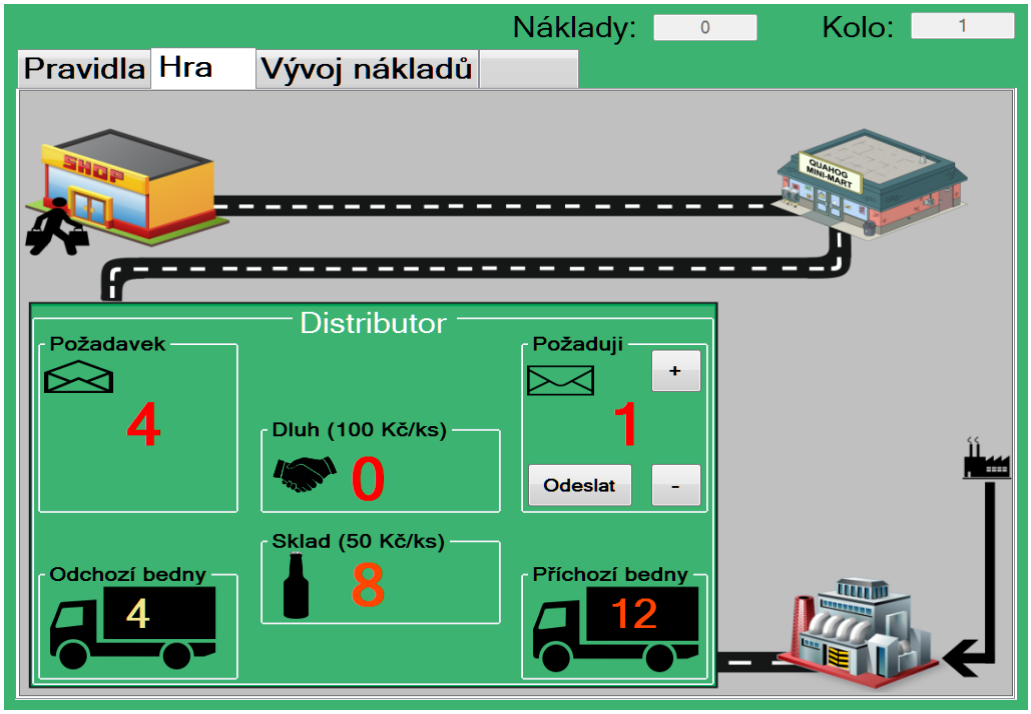


Obr. 11 Ukázka úvodní obrazovky s pravidly klientské aplikace

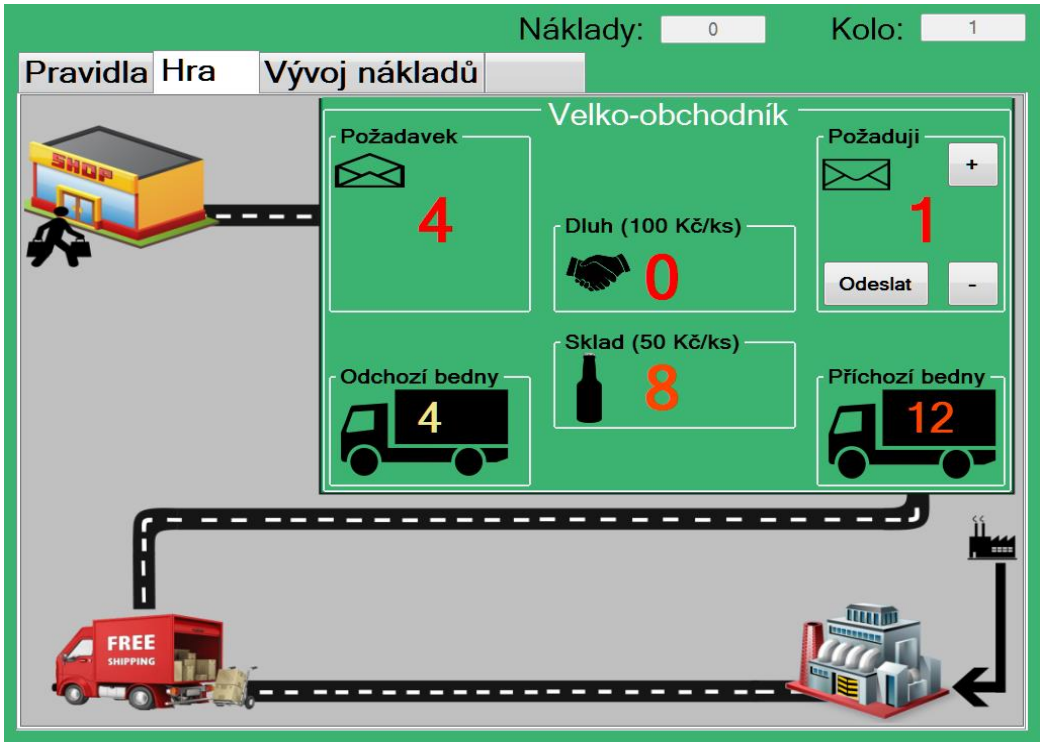
V pravém horním rohu jsou pak zobrazeny informace o aktuální výši nákladů daného hráče (nikoliv součtu za celou skupinu) a o kole, v němž se hráč nachází.

Po přečtení pravidel může hráč pokračovat kliknutím na záložku „Hra“, kde se mu zobrazí hlavní obrazovka hry. Její rozložení je dáno rolí v řetězci, pod kterou hráč vystupuje. Na následujících obrázcích jsou postupně zobrazeny všechny rozložení hrací obrazovky, která hra nabízí. Cílem designu všech obrazovek bylo dát hráči najevo, v jaké části dodavatelského řetězce se jeho role vyskytuje a také naznačit tok informací. Bedny s lahvemi se pohybují směrem od továrníka k maloobchodníkovi a objednávky (požadavky na bedny) se pohybují opačně, tedy od maloobchodníka k továrníkovi.

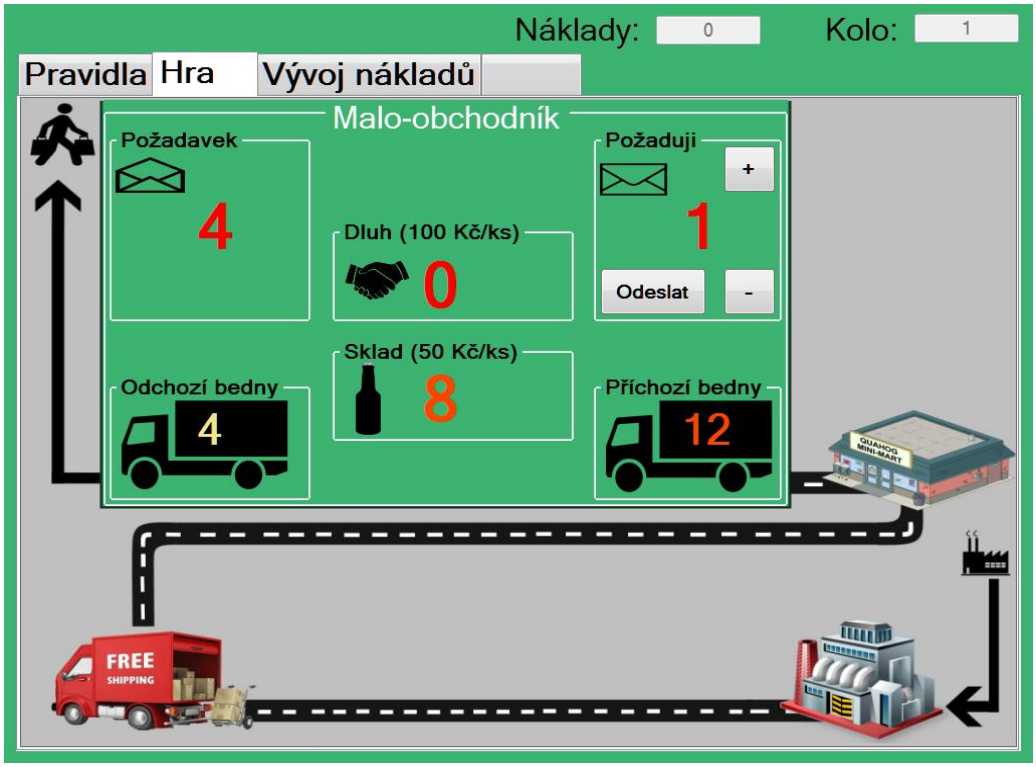
Obr. 12 Ukázka rozložení hrací obrazovky pro roli „Továrník“



Obr. 13 Ukázka rozložení hrací obrazovky pro roli „Distributor“



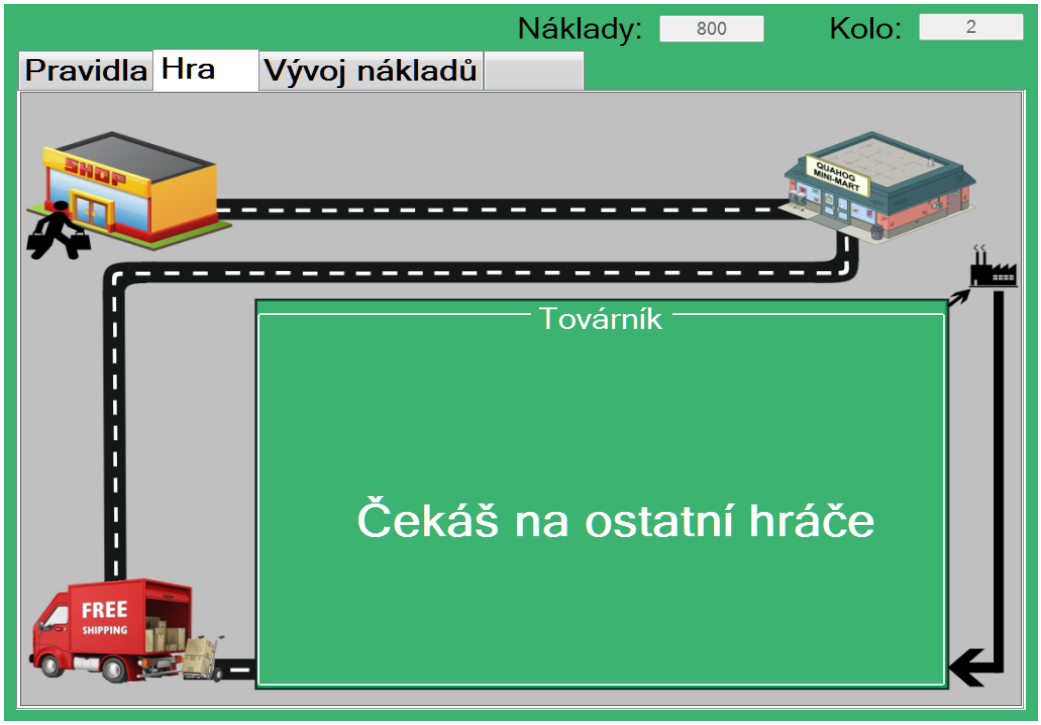
Obr. 14 Ukázka rozložení hrací obrazovky pro roli „Velkoobchodník“



Obr. 15 Ukázka rozložení hrací obrazovky pro roli „Maloobchodník“

8.2.3 Stav čekání klientské aplikace

V momentě, kdy hráč odešle svůj požadavek a ukončí tím své hrací kolo, ocitne se ve stavu, kdy čeká, až tak učiní všichni. Veškeré jeho ovládací prky hry zmizí. Na následujícím obrázku je tento stav zobrazen.



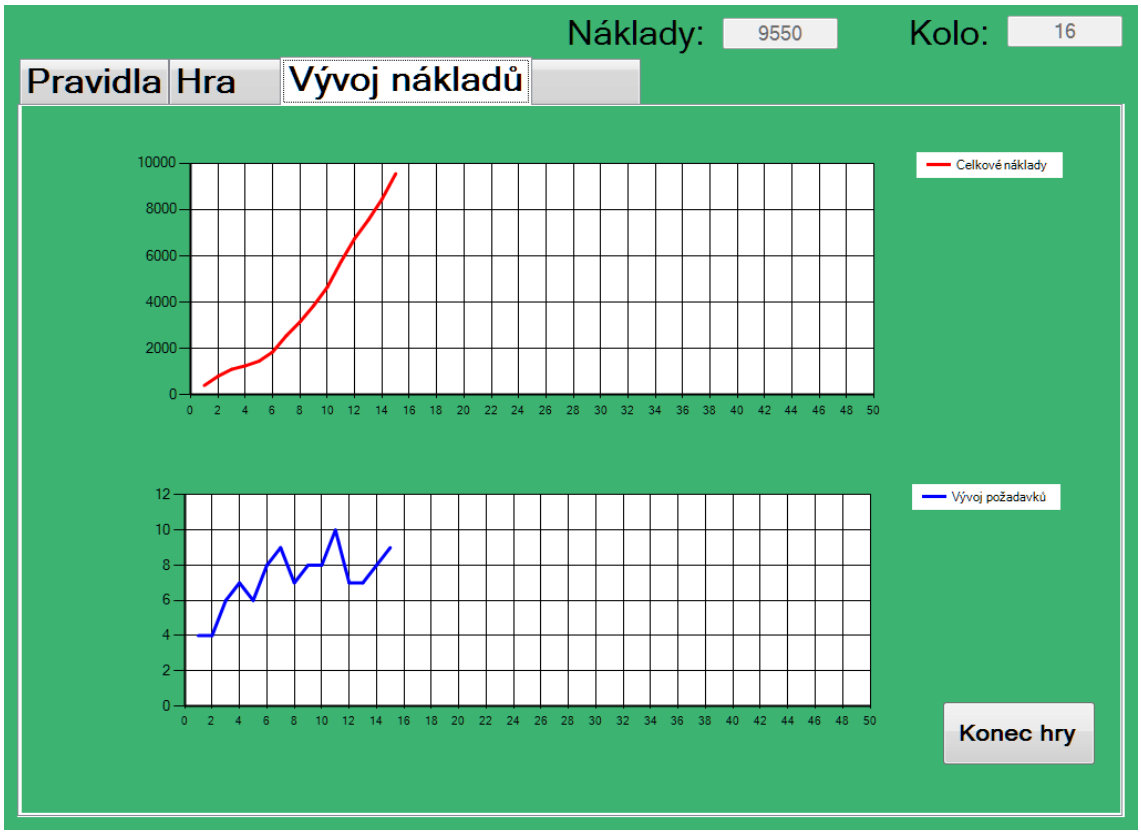
Obr. 16 Ukázka stavu čekání na ostatní hráče

Mezi tím aplikace na pozadí nastartuje timer (metoda *waitingLoop()*), který se periodicky ptá serveru, zda už všichni odehráli a může se začít nové kolo. V momentě, kdy tak nastane, server všem hráčům, kteří se ho na nové kolo ptají, odpoví kladně a tím pádem je vysvobodí z čekací smyčky. Každému hráči se pak znovu zobrazí veškeré ovládací prvky a hra může pokračovat dalším kolem. Metody, které řeší přechod do stavu „čekání“ a zněj, jsou rozebrány níže.

```
public Form1(int portNumber, string ipAddress)
{
      _PORT = portNumber;                  // uložení čísla portu
      _IP_ADDRESS = ipAddress;             // uložení IP adresy
      InitializeComponent();               // inicializace komponent GUI
      waitingTimer.Interval = 1000;        // nastavení intervalu na 1s
      waitingTimer.Enabled = false;        // deaktivace timeru
      waitingTimer.Elapsed += OnTimedEvent; // navázání události
}

// Metoda pro zahájení čekací smyčky
private void waitingLoop()
{
      waitingTimer.Start();   // start timeru
}
```

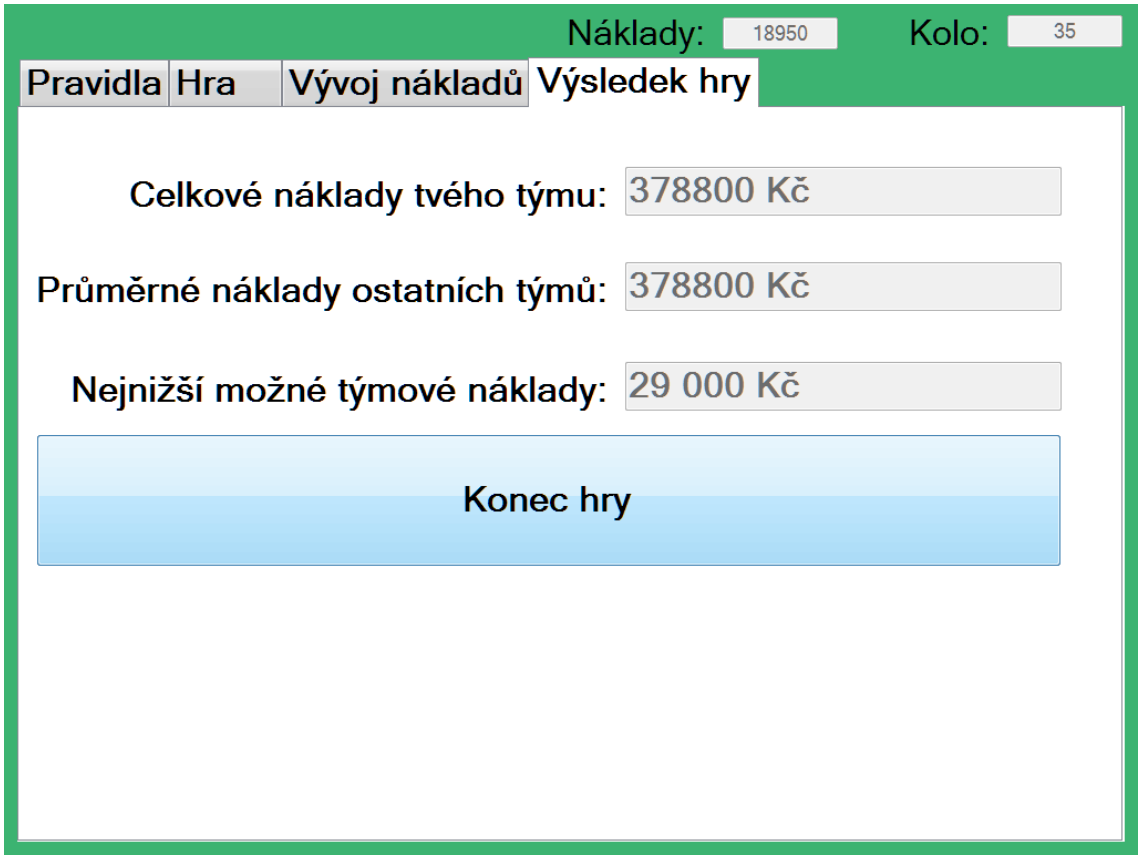
```
// Metoda pro vyklouznutí z čekací smyčky
private void stopWaiting()
{
      waitingTimer.Stop();    // zastavení timeru
}

// Metoda, která se volá vždy po vypršení intervalu nastaveného výše
private void OnTimedEvent(Object source, System.Timers.ElapsedEventArgs e)
{
      SendWaitingRequest();   // poslání dotazu na server
      ReceiveResponse();      // obdržení odpovědi
}
```

Během čekání na odehrání ostatních si může hráč prohlédnout dva grafy, které se postupem hry mění v závislosti na velikosti sledovaných veličin. Grafy mají v první řadě hráče informovat o velikosti poptávaného množství. Prozíravější uživatelé se poté mohou pokusit vysledovat určité trendy v poptávce a přizpůsobit tak svoje chování v rámci celého dodavatelského řetězce. Ukázka takových grafů je na obrázku číslo 17. V dolní části obrazovky se nachází tlačítko, které po stisku odpojí daného klienta od serveru. Hra pro celou skupinu v této chvíli končí. Server postupně ukončí spojení se zbylými klienty a restartuje se do iniciálního stavu, kdy čeká na nově připojivší se hráče.



Obr. 17 Ukázka grafu vývoje celkových nákladů a vývoje příchozích požadavků

Pokud všichni hráči odehrají poslední kolo, server rozpozná konec hry, spočítá celkový součet nákladů v rámci celé skupiny (dodavatelského řetězce), průměrné náklady, které dosáhli ostatní skupiny a tato data zašle každému z klientů. Klientovi se bezprostředně poté zaktivuje poslední hrací záložka s výsledkem hry a tyto hodnoty se vloží do příslušných polí. Ostatní záložky (až na záložku s pravidly) se deaktivují. Poté, co jakýkoli hráč klikne na tlačítko konec hry, server postupně odpojí i zbylé hráče a nastartuje se znovu na stejném portu. Na každém klientském PC poté aplikace Watch dog (kapitola 5.1.2) znovu spustí novou instanci hry, která se připojí na nově vzniklý server a exponát je připraven na příchod nové skupiny hráčů. Finální obrazovka s výsledky hry je vidět na následujícím obrázku.



Obr. 18 Všichni odehráli své poslední kolo a nastal konec hry

8.3 Asynchronní síťová komunikace Klient-Server

8.3.1 Server

Klíčová funkcionalita, na které hra distribuce limonád stojí, je asynchronní síťová komunikace. V následující kapitole se pokusím vysvětlit základní principy použité v mé aplikaci. V jazyce C# je asynchronní komunikace řešena poměrně intuitivně. Poté, co proběhne inicializace všech komponent GUI serveru, zavolá se metoda

SetupServer(), která má za úkol vytvořit serverový soket, který bude akceptovat veškerou komunikaci probíhající na daném portu prostřednictvím protokolu TCP.

```
public void SetupServer()
{
...
      // vytvoření serverového soketu
      _serverSocket = new Socket(AddressFamily.InterNetwork, SocketTy-
pe.Stream, ProtocolType.Tcp);
      // přiřazení koncového bodu vytvořenému soketu
      _serverSocket.Bind(new IPEndPoint(IPAddress.Any, _PORT));
      _serverSocket.Listen(5); // uvedení soketu do stavu naslouchání

      // zahájení asynchronní operace pro přijetí všech
      // příchozích požadavků na připojení se k serveru
      _serverSocket.BeginAccept(AcceptCallback, null);
      ...
}
```

Jakmile je spojení mezi klientem a serverem navázáno, server si uchová do své paměti toto spojení a zahájí samotný přenos dat mezi těmito aplikacemi. Nesmíme zapomenout dát šanci se připojit dalším klientům. Na závěr metody je tedy nutné zavolat znovu metodu pro příjem příchozího spojení.

```
private void AcceptCallback(IAsyncResult AR)
{
      Socket socket;
      try
      {
            // soket spojení mezi klientem a serverem
            socket = _serverSocket.EndAccept(AR);
      }
      catch (ObjectDisposedException)
      {
            return;
      }
      // uložení spojení do seznamu všech spojení
      _clientSockets.Add(socket);
      // zahájení přenosu dat nad aktuálním spojením
      socket.BeginReceive(_buffer, 0, _BUFFER_SIZE, SocketFlags.None,
ReceiveCallback, socket);

      ...
      // zahájení čekání na další příchozí spojení
      _serverSocket.BeginAccept(AcceptCallback, null);
}
```

Veškerá logika, která určuje, jak se s danou zprávou a daty naloží je v metodě *ReceiveCallback().* Tato metoda je uvedena v příloze C v sekci *ReceiveCallback.* Nedojde-li během přenosu informací k žádnému problému, metoda z přenesených dat

sestaví zprávu, kterou zaslala klientská aplikace. Můžeme vidět, že to, co určuje, jakým způsobem bude s daty nakládáno a jaký typ odpovědi server klientovi zašle, je kód zprávy, pod kterým klient zprávu zaslal. Server má hned několik možností, jak zareagovat. Pokud zpráva obsahuje data, která jsou v rámci právě probíhajícího kola nová, server si data uloží a odpoví klientovi zprávou s kódem -300, což znamená uvedení klienta do stavu čekání na ostatní hráče. Klienti, kteří čekají, až odehrají všichni, zasílají periodicky dotaz na server, zda už se tak náhodou nestalo. Pokud ne, server odpoví zprávou s kódem -400, která pro klienta nemá žádný význam (nereaguje na ni). Jakmile však všichni hráči odehrají, server odpoví zprávou s kódem -200, což pro klienta znamená nové kolo. Periodické dotazy jsou ukončeny, všechny ovládací prvky jsou znovu zobrazeny a hra může pokračovat.

8.3.2 Klient

Z klientského pohledu komunikace není natolik složitá. Při inicializaci klientské aplikace dochází k volání metody *ConnectToServer()*, která se ve smyčce snaží připojit k serveru. Jakmile je připojení serverem akceptováno, smyčka se ukončí a spojení je navázáno. V tomto okamžiku je vše připraveno na samotný přenos dat, který je vyvolán stisknutím tlačítka odeslat a realizován již dříve popsanými metodami *SendRequest()* a *RecieveResponse()*.

```
private void Form1_Load(object sender, EventArgs e)
{
      ConnectToServer(); // připojení k serveru
      // načtení aktuálního stavu nákladů
      add2CostSum();
      // načtení počáteční konfigurace ze serveru
      loadConfigurationRequest();
      // obdržení konfigurace
      ReceiveResponse();
      // nastavení rozložení obrazovky pro přidělenou roli
      setScreenLayout(_ROLE);
      ...
}
private void ConnectToServer()
{
      // dokud se nepodaří připojit se k serveru, tak se snaž připojit
      while (!_clientSocket.Connected)
      {
            try
            {
                  // připojení klienta k serveru
                  _clientSocket.Connect(_IP_ADDRESS, _PORT);
            }
            catch (SocketException e) // nastala chyba
            {
                  ...
            }}}
```

9 Testovací provoz

Testovací provoz aplikace probíhal měsíc. Exponát Kulatý stůl byl uzpůsoben pro tyto účely. Zaměstnanci VIDA! SC připravili 4 koncové stanice výhradně pro hru Distribuce limonád. Během této doby hra zaznamenala 52 her, z nichž 14 her dosáhlo svého konce. Zbylých 38 her se nedohrálo. Zaměstnanci VIDA! SC reportovali pád aplikace ve 3 případech. Ostatní hry skončili kvůli tomu, že jeden hráč nebo celá skupina předčasně opustili hru. Možné důvody tohoto chování rozebírám později v závěru této práce.

Následující statistiky berou v úvahu pouze data z dohraných her. Průměrná výše nákladů za celou hrací skupinu činila 299 500 Kč. Tato hodnota je poměrně dost vysoká a výrazně ji zkreslují extrémní průběhy her, kde se konečné náklady skupiny pohybovaly až okolo 650 000 Kč. Medián ze skupinových nákladů dohraných her činil 223 675 Kč.

Maximální hodnota skupinových nákladů činila 689 800 Kč. Průběh této hry je podrobněji rozebrány v kapitole 10.3. Minimální hodnota skupinových nákladů byla 101 850 Kč. Tento výsledek lze považovat za velice slušný a můžeme říci, že hráči prováděli svá rozhodnutí velmi pečlivě a citlivě s ohledem na ostatní články v řetězci.

Během testovacího provozu žádnou skupinu hráčů nenapadlo aplikovat tzv. triviální strategii. Dle (STERMAN, 2001) spočívá v tom, že všichni hráči si vždy objednají přesně tolik, jak vysoký byl jejich poslední příchozí požadavek. Toto chování eliminuje divoké oscilace zásob a nesplněných požadavků, avšak není zdaleka optimální. Například u maloobchodníka vzniknou nesplněné zakázky, které už nikdy nedorovná. Touto strategií lze dosáhnout týmových nákladů cca 76 000 Kč. Dle (STERMAN, 1992) je však optimální výše skupinových nákladů stanovená okolo 20 000 Kč. Nelze předpokládat, že by se podařilo nějaké hráčské skupině této hodnotě bez předchozích zkušeností s Beer game přiblížit.

9.1 Zpětná vazba zaměstnanců VIDA! SC

V průběhu vývoje hry bylo se zaměstnanci VIDA! SC řešení několikrát konzultováno a jejich připomínky byly do příští schůzky zapracovány. S výslednou podobou aplikace jsou zaměstnanci VIDA! SC spokojeni. Jejich požadavek byl vytvořit fungující řešení, které bude schopné samostatného provozu a na které se bude moci v budoucnu navázat případnými rozšířeními.

9.2 Zpětná vazba hráčů

Jelikož jsem se několika her jako přihlížející účastnil, mohu říci, že největší problém hráčům činilo pochopit návaznost jednotlivých rolí v řetězci. Toto tvrzení mi mohou potvrdit i samotní zaměstnanci VIDA! SC. Někteří hráči měli problémy s tvorbou mentálního modelu hry a se správným zařazením své role. V těchto chví-

lích bylo nejlepší skupině názornou ukázkou souvislosti v řetězci vysvětlit. Někteří spíše potřebovali odehrát pár vzorových kol, aby si každý hráč mohl vyzkoušet, jak se hra chová a jakým způsobem v ní tečou informace. Každá skupina svým způsobem byla individuální a dotazy hráčů se často velmi lišily. Podstatnou roli zde hráli zkušenosti jednotlivých hráčů s problematikou SCM. S ovládáním hry problém nebyl a tak, když už hráči pochopili všechny potřebné náležitosti, často se stávalo, že hra pro ně nabyla nového rozměru a vtáhla je.

Do budoucna by se dalo lepší pochopení souvislostí podpořit ukázkovou animací, která by vše potřebné vysvětlila a doplnila textový výklad pravidel.

10 Vyhodnocení průběhů her

V následující kapitole postupně rozeberu nejzajímavější průběhy her, které aplikace během svého provozu na exponátu kulatý stůl nasbírala. Snahou bylo vybrat ty průběhy, které v sobě nesou cenné informace, ať už se jedná o typický průběh hry, či netradiční taktiku. V těchto praktických příkladech se budu snažit ověřit teoretický základ, který jsem během psaní této diplomové práce načerpal z odborné literatury a který jsou popsán v předchozích kapitolách. Pozornost bude soustředěna zejména na efekt biče. Dále se budu snažit identifikovat kritická rozhodnutí některých hráčů, která negativně ovlivnila výsledek celé hrací skupiny.

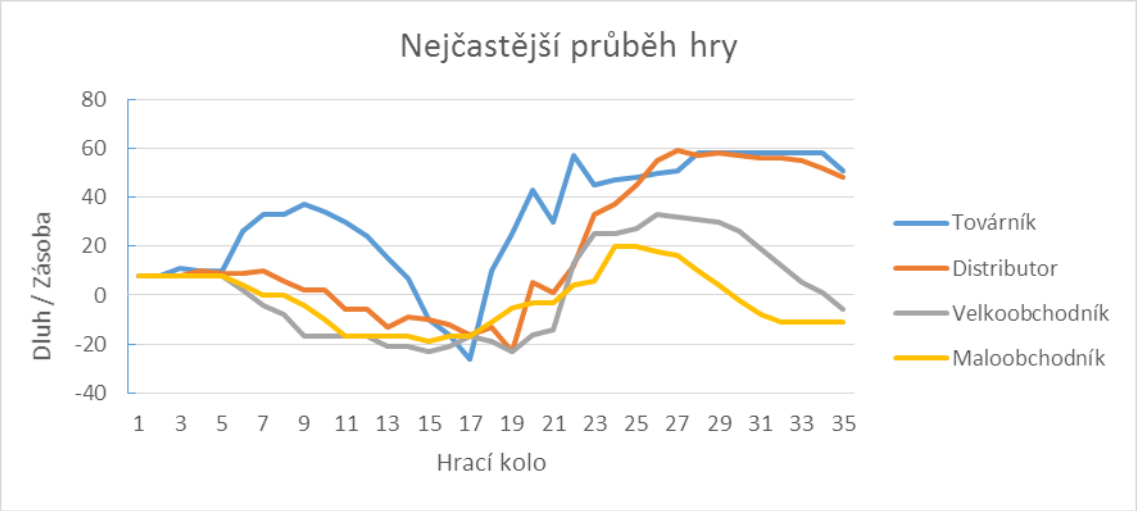
V každé následující podkapitole je uveden průběh jedné hry, jakožto zástupce daného chování hráčů. Jednotlivé grafy ukazují vývoj skladových zásob respektive nesplněných požadavků během hry. Vodorovná osa grafu představuje číslo hracího kola a svislá osa grafu potom stav skladových zásob resp. nesplněných požadavků podle toho, zda je číslo kladné či záporné.

10.1 Nejčastější průběh hry

Jako první hru k rozboru jsem si zvolil tu, která se svými rysy nejvíce podobá typickému průběhu hry, který je popsán v literatuře. Např. (STERMAN, 2000).

V úvodu hry můžeme vidět snahu továrníka o vytvoření mírné zásoby, která by mu měla pomoci v případě příchodu větší objednávky od distributora. I přesto však továrníkovi zásoby kolem 15. kola došly a dostal se do mínusu. Vyhodnotil to tak, že trend stoupajících požadavků bude pokračovat a začal vyrábět nepřiměřeně mnoho. Jakmile byly bedny vyrobeny, uspokojil poptávku, avšak přebytečné bedny mu zůstaly na skladě. Od té doby (kolem 28. kola) můžeme vidět, že již nevyrábí a čerpá pouze ze svých zásob. Chování továrníka se ze začátku hry od ostatních rolí lišilo, avšak postupem času všichni hráči činili svá rozhodnutí velmi podobně. Můžeme vidět, že úroveň skladových zásob a nesplněných požadavků vykazuje podobný trend. Všechny tři role (distributor, velkoobchodník a maloobchodník) se dostaly se svými skladovými zásobami do mínusu (rozmezí 8. až 11. kola) a poté s cílem naplnit znovu své sklady nepřiměřenými objednávkami způsobili, že továrník vyrobil mnohem více, než byla v té době poptávka a bedny se všem začaly kumulovat na skladech.

Tento průběh, ovšem s menšími rozdíly, se objevoval bezmála v polovině her, které byly za dobu 1 měsíce zaznamenány.



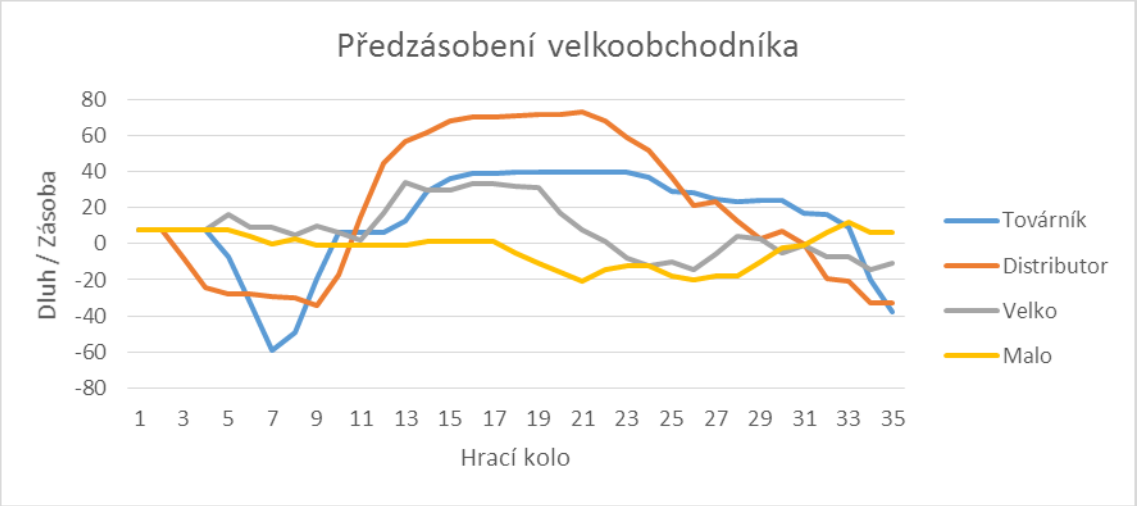
Obr. 19 Nejčastější průběh hry

Jelikož model Beer game slouží především k demonstraci efektu biče, lze očekávat, že v typickém průběhu hry bude možnost vidět i typický průběh efektu biče. A opravdu tomu tak je. Z obrázku je zřejmé, že výkyvy ve skladových zásobách se zvyšují, čím dále postupujeme od maloobchodníka k továrně.

10.2 Předzásobení

Někteří hráči měli tendenci si vytvořit rezervu na začátku hry, ze které by mohli v budoucnu čerpat. Výši této rezervy poté svými rozhodnutími (požadavky) korigovali. Na první pohled se může zdát tato taktika jako správná a logická, avšak nikdo z hráčů si neuvědomil, že toto chování vygeneruje mnoho problémů pro ostatní hráče v týmu.

V následujícím grafu můžeme vidět snahu velkoobchodníka o vytvoření pojistné rezervy z počátku hry. Je zřejmé, že maloobchodník tímto rozhodnutím zasažen není, ale to nelze říci o distributorovi a továrníkovi. Skladové zásoby těchto dvou postižených rolí jsou ve 2. respektive ve 4. kole zcela vyčerpány a vznikají nesplněné zakázky. Distributor zareagoval logicky a zvýšil své požadavky tak, aby se znovu dostal do kladných čísel skladové zásoby. Továrník požadované množství vyrobil a to doputovalo až k velkoobchodníkovi, který toto vše způsobil. Avšak jak je možné vidět, ani distributor ani továrník neodhadli tento záměr velkoobchodníka a paradoxně jim vznikla ještě větší skladová zásoba. Zhruba od 19. kola můžeme vidět, že velkoobchodník reguluje svoji skladovou zásobu. Jeho objednávané množství se blíží k nule a příchozí požadavky uspokojuje právě ze skladu. Později takto jednají i ostatní dvě role, které mají přebytek na skladě (distributor a továrník). Na konci hry můžeme vidět snahu maloobchodníka naplnit všechny své nesplněné zakázky, které vznikly už kolem 17. kola. Velkoobchodník, distributor ani továrník nejsou schopni včas zareagovat na zvýšené poptávané množství maloobchodníka a vzniká jim dluh.



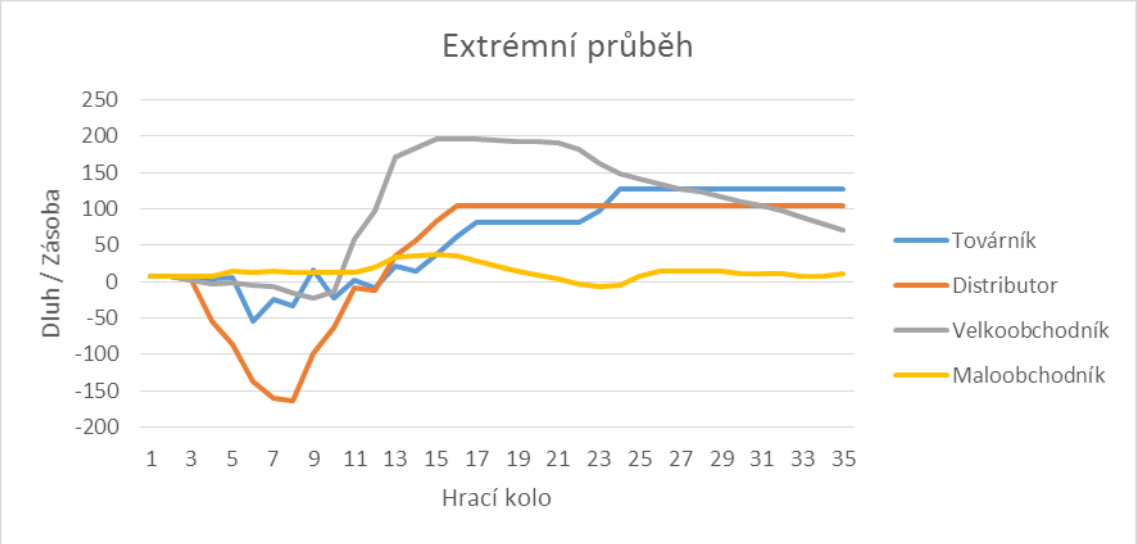
Obr. 20 Ukázka předzásobení velkoobchodníka

I přesto, že velkoobchodník zvolil z počátku hry taktiku tvorby pojistné zásoby, je efekt biče zřejmý a patrný. Dalo by se říci, že distributor a továrník jsou postiženi dopadem efektu biče zhruba stejně, resp. výkyvy v jejich skladových zásobách jsou srovnatelné.

10.3 Extrémní průběhy

Ze zkoumaného vzorku her se našly i takové průběhy her, ve kterých objednávky několikanásobně překračovaly hodnotu skutečné poptávky. Hráčům v těchto hrách typicky bylo jedno, zda bude mít jejich požadavek hodnotu 10 beden nebo 60. V kontextu skutečné poptávky, která se skokově změnila pouze jednou a to ze 4 beden na 8, je však tato benevolentnost hráčů fatální. Každá objednaná bedna by měla mít svoje opodstatnění a hráč by si měl dobře rozmyslet, kolik beden objedná. Poměrně velká část výsledků (asi 30 %) byla tímto chováním ovlivněna. I přesto, že taková hra nemá příliš velkou vypovídající hodnotu, rozhodl jsem se ji zde kvůli velkému procentnímu zastoupení uvést.

Jak je z grafu patrné, jediný, kdo hrál v rámci mezí je maloobchodník. Ten dokázal poměrně úspěšně kopírovat cílovou poptávku a nečinil přehnaná rozhodnutí. To ovšem nelze říci o ostatních hráčích. Velkoobchodník svým přemrštěným požadavkem z druhého kola způsobil obrovský dluh distributorovi. Jakmile továrník požadované bedny vyrobil a poslal, nastal stav, kdy sklady byly plné (100 a více beden) a cílová poptávka byla stále na úrovni 8 beden. Hráči poté odesílali velmi nízké požadavky a regulovali svoje skladové zásoby. Ovšem jedinému hráči, komu se to podařilo, byl velkoobchodník. Ostatním se skladové zásoby nezměnily, a proto není divu, že tato skupina dosáhla největších týmových nákladů po čas sběru dat, které činily 689 800 Kč.



Obr. 21 Ukázka průběhu hry s extrémními hodnotami

V tomto případě nemá příliš cenu zabývat se efektem biče a jinými ekonomickými souvislostmi. Tato hrací skupina se chovala velice neefektivně a zřejmě úplně neporozuměla principu hry. V ostatních hrách však takové extrémní hodnoty naměřeny nebyly a i když skupina dosáhla velmi vysokých nákladů, byl efekt biče patrný.

11 Shrnutí nasbíraných výsledků

Z analýz vybraných (i nevybraných) průběhů her vyplývá, že žádná skupina se nedokázala vyhnout efektu biče. Ten ve všech sehrávkách hrál větší či menší roli dle toho, jak se dané skupině dařilo jej eliminovat. V modelu Beer game je efekt biče způsobem třemi faktory: 1) nedostatek informací, 2) struktura dodavatelského řetězce, 3) nemožnost spolupracovat a absence globálního řízení.

Model Beer game simuluje dodavatelský řetězec s nízkou úrovní důvěry, kde pouze malá část informací je sdílena napříč řetězcem. Bez znalosti cílové poptávky se veškeré její předpovědi zakládají na výši došlých objednávek v každém článku dodavatelského řetězce. Prokázalo se, že neočekávané skokové zvýšení poptávky má za následek zvyšování počtu objednávek každého článku. Navýšení počtu objednávek bylo ještě umocněno v momentě, kdy některý z hráčů zpozoroval trend ve stoupajícím počtu objednávek a snažil se předejít vyprázdnění jeho skladů zvyšováním své pojistné zásoby. Později, kdy se ukázalo, že zvýšení bylo pouze dočasné, hráči měli typicky tendenci snižovat svoji pojistnou zásobu tím, že chvíli posílali objednávky, blížící se svojí výší k nule. Toto chování značně přispívá k zesílení efektu biče.

V některých skupinách se objevili hráči, kteří se svým chováním a rozhodováním o tom, kolik beden si objednají, přibližovali k fenoménu známému jako „dávkové objednávání“ (batch ordering). Podstatou tohoto jednání je cílené zadržování objednávek a následné vypuštění jedné souhrnné. V praxi se často setkáváme se situací, kdy objednání menšího množství je dražší než objednáme-li celý kamión. Kromě toho i mnozí dodavatelé nabízí množstevní slevy. Proto někteří manažeři jednají podle výše popsaného vzoru s cílem ušetřit náklady. Toto chování však v modelu Beer game značně ztěžuje odhad a předpověď cílové poptávky. Objednávání po dávkách teoreticky může vést ke snížení nákladů v krátkém období, nicméně přímo přispívá k efektu biče a tím i k růstu celkových nákladů v rámci celého dodavatelského řetězce.

Struktura dodavatelského řetězce svým designem ve formě samostatných článků a existencí zpoždění mezi nimi též přispívá k síle efektu biče. Čím je toto zpoždění větší, tím roste i jeho síla.

12 Závěr

Tato diplomová práce se zabývá analýzou a implementací výukové aplikace Distribuce limonád pro exponát Kulatý stůl ve VIDA! SC. Nejprve byl čtenář seznámen se základním konceptem hry a pravidly. Následně byla detailně rozebrána typická podoba stavu hry v jednotlivých fázích hry. Byly vysvětleny tři klíčové faktory (kolísání, zesílení a fázové zpoždění), které vznikají z důvodu informačního zpoždění modelu a nedostatku informací. Poměrně velký prostor jsem dále věnoval Efektu biče, který je ve větší či menší míře součástí každého produkčně-distribučního řetězce. Dále jsem rozebral jeho příčiny vzniku a kroky, které je potřeba učinit k tomu, aby byl dopad tohoto nežádoucího efektu co nejvíce omezen.

V praktické části jsem implementoval výukovou aplikaci v programovacím jazyce C#. Tato aplikace byla poté nasazena do testovacího provozu, kde byly odladěny všechny chyby a zapracovány připomínky ze strany VIDA! SC. Poté byla aplikace spuštěna v ostrém provozu, kde zaznamenávala průběhy her návštěvníků. Následně byly tyto průběhy vyhodnoceny a porovnány s typickými průběhy uváděnými v literatuře.

Z počátku jsem se potýkal s malým množstvím dohraných her, který byl zapříčiněný zřejmě větší obtížností hry, nutnosti uvažování a také časovou náročností. Drtivou většinu návštěvníků VIDA! SC tvoří děti ze základních škol, pro které je aplikace příliš složitá a časově náročná. Tyto děti častěji preferují více interaktivní exponáty, jejichž návštěva nezabere tolik času a výsledek se dostaví několikanásobně rychleji, než je tomu u hry distribuce limonád.

Rozbor vybraných her ukázal, že většina hráčů činí svá rozhodnutí, aniž by si uvědomovala jejich dopad v rámci celého řetězce a to i přesto, že vědí, že cílem je minimalizace nákladů v rámci celé skupiny. Cenným poznatkem pro takové hráče je zjištění, že pokud se budou snažit o dosažení nejlepších výsledků (nejnižších nákladů) jen v rámci své role (syndrom zajištění vlastní pozice), bude mít toto jednání často negativní dopad na řetězec jako celek. Jsou-li všichni hráči schopni vidět souvislosti daného systému v komplexním pohledu, směřují i jejich rozhodnutí správným směrem a výsledek celé hrací skupiny může dosáhnout nadprůměrného výsledku.

Hraní hry Distribuce limonád výrazným způsobem přispívá k rychlé tvorbě mentálních modelů, které využíváme před učiněním každého rozhodnutí v rámci hry. Tvorba a následné vylepšování mentálního modelu napomáhá k vidění souvislostí v komplexním pohledu na systém a tím i ke kvalitnějším rozhodnutím.

Hra Distribuce limonád představuje pouze zjednodušený model reality oproštěný od mnoha skutečností, které se však v praxi vyskytují. I přesto tento model zřetelně poukazuje na faktory, které stojí za neúspěchem naprosté většiny týmů. Prvním faktorem jsou nedostatečné informace (zejména o koncové poptávce) a druhým faktorem je zpoždění systému. Především pak zpoždění systému je tím, co velmi často přehlížíme nejen v tomto modelu, ale i v nejrůznějších životních situacích. Nejsme-li schopní zpoždění vnímat nebo jej přehlížíme, vede to k pocitu, že

svět resp. systém, ve kterém se pohybujeme, je nepředvídatelný a preferujeme raději krátkodobé řešení na úkor dlouhodobých.

Hra Distribuce limonád je skvělou příležitostí pro návštěvníky VIDA! SC naučit se základním principům a zákonitostem v produkčně-distribučním řetězci. Zábavnou formou si hráči mohou vyzkoušet, jaké je to být součástí dodavatelského řetězce a činit zdánlivě jednoduchá rozhodnutí o tom, kolik beden s nápoji si objednají. Jak vyplývá z teoretické části práce, běžný hráč a dokonce i někteří manažeři si nejsou vědomi zpoždění v řetězci. Hráči nedokáží vnímat rozdíl mezi skutečným a požadovaným stavem a nemají představu o zpětném působení vlastních rozhodnutí na svou pozici.

Do budoucna by se hra dala rozšířit o fázi debrífingu, která by se mohla stát součástí hry po jejím skončení. Hráči by zde mohli odpovědět na několik otázek, které jsou typické pro tuto fázi a využít potenciál dotykové obrazovky s centrálním projektorem pro sdílení svých myšlenek. Dále vidím velký prostor pro zlepšení v oblasti vzhledu klientské aplikace. VIDA! SC má vlastní motiv, který integruje do všech svých aplikací a který by se mohl stát součástí i této aplikace. Další možné vylepšení by mohlo spočívat v různých jazykových mutacích hry. Řídicí systém, který bude implementován v rámci diplomové práce pana Saviče, by předal klientské aplikaci parametr, který by určoval, v jaké jazykové mutaci se má daná aplikace spustit. Hodnota parametru by korespondovala s jazykovou mutací řídicího systému. Jakmile by si hráč nastavil jazyk v řídicím systému, všechny aplikace, které by poté spustil, by byly inicializovány ve zvolené jazykové mutaci. Pro mezinárodní skupinu hráčů by to znamenalo, že každý hráč by měl jazyk aplikace přizpůsobený dle svých preferencí, aniž by byl ovlivněna celková funkčnost aplikace.

13 Literatura

AGILE SOFTWARE DEVELOPMENT. *Agile software development - Wikipedia, the free encyclopedia* [online]. 2016 [cit. 2016-05-06]. Dostupné z: <https://en.wikipedia.org/wiki/Agile_software_development>

BEERGAME, Typical results. W3.org [online]. 1999 [cit. 2016-04-19]. Dostupné z: <http://www.beergame.org/the-game/typical-results>

BEERGAME, Welcome. W3.org [online]. 2012 [cit. 2016-05-05]. Dostupné z: <http://www.beergame.org/>

COPPINI, M. ROSSIGNOLI, Ch. ROSSI, T. *Bullwhip effect and inventory oscillations analysis using the beer game model*. International Journal of Production Research. 2010, 48(13), 3943-3956. DOI: 10.1080/00207540902896204. ISSN 0020-7543. Dostupné také z: <http://www.tandfonline.com/doi/abs/10.1080/00207540902896204>

DOGAN, G., STERMAN, J., When Less Leads to More: Phantom Ordering in the Beer Game? International Conference of the System Dynamics Society, Boston, Massachusetts, USA. 2005.

C Sharp. C Sharp – Wikipedie [online]. 2016 [cit. 2016-05-01]. Dostupné z: <https://cs.wikipedia.org/wiki/C_Sharp>

EDALI, M. A *Mathematical Model of the Beer Game*. England: JASSS, 2014, 17(4). ISSN 1460-7425.

EXTREME PROGRAMMING. *Extreme programming - Wikipedia, the free encyclopedia* [online]. 2016 [cit. 2016-05-06]. Dostupné z: <https://en.wikipedia.org/wiki/Extreme_programming>

HÁJEK, J. *Zkoumání chování distribučního řetězce na příkladu Beer Game*. Praha, 2012. Dostupné z: <https://theses.cz/id/a45d4t>. Diplomová práce, Vysoká škola ekonomická v Praze. Vedoucí práce Inka Neumaierová.

HERLYN W.: *The Bullwhip Effect in expanded Supply Chains and the Concept of Cumulative Quantities* , epubli Verlag, Berlin, 2014, ISBN 978-3-8442-9878-9

KVASNICA, J. *Síťová asynchronní aplikace Piškvorky*. Jihlava, 2012. Dostupné z: <https://theses.cz/id/lh108z>. Bakalářská práce. Vysoká škola polytechnická v Jihlavě. Vedoucí práce František Smrčka.

LAMBERT, D M. *Supply chain management: processes, partnerships, performance*. Sarasota: Supply Chain Management Institute, 2008. ISBN 978-0-9759-9493-1.

MACDONALD, J. FROMMER, D. *Decision making in the beer game and supply chain performance*. Operations Management Research. 2013, 6(3-4), 119-126. DOI: 10.1007/s12063-013-0083-4. ISSN 1936-9735. Dostupné také z: <http://link.springer.com/10.1007/s12063-013-0083-4>

MEADOWS, Donella H a Diana WRIGHT. *Thinking in systems: a primer*. White River Junction, Vt.: Chelsea Green Pub., 2008. ISBN 1603580557.

MILDEOVÁ, S., VOJTKO, V. a kol. (2006). *Manažerské simulace dynamických procesů*. 1. vyd. Praha: Oeconomica, 2006. 105 s. ISBN 80-245-1055-3.

NIENHAUS, J. ZIEGENBEIN, A. *How human behaviour amplifies the bullwhip effect. A study based on the beer distribution game online*. Production Planning. 2006, 17(6), 547-557. DOI: 10.1080/09537280600866587. ISSN 0953-7287. Dostupné také z: <http://www.tandfonline.com/doi/abs/10.1080/09537280600866587>

PEŠALOVÁ, L. *Zkoumání chování distribučního řetězce na příkladu Beer Game*. Ostrava, 2012. Dostupné z: <http://dspace.vsb.cz/handle/10084/99342>. Bakalářská práce. Vysoká škola báňská v Ostravě. Vedoucí práce Pavel Wicher.

SARKAR, S. KUMAR, S. *Demonstrating the Effect of Supply Chain Disruptions through an Online Beer Distribution Game*. Decision Sciences Journal of Innovative Education. 2016, 14(1), 25-35. DOI: 10.1111/dsji.12091. ISSN 15404595. Dostupné také z: <http://doi.wiley.com/10.1111/dsji.12091>

STERMAN, John D. *The Beer Game*. Flight Simulators for Management Education [online]. Cambridge: Sloan School of Management, 1992 [cit. 2016-03-

15]. Dostupné z:
<http://web.mit.edu/jsterman/www/SDG/beergame.html>

STERMAN, John D. *System Dynamics Modeling: Tools for Learning in a Complex World*. California Management Review. 2001, 43(4), 8-25. DOI: 10.2307/41166098. ISSN 00081256. Dostupné také z:
<http://cmr.ucpress.edu/cgi/doi/10.2307/41166098>

STERMAN, J. *Business dynamics: systems thinking and modelling for a complex world*. Boston: McGraw-Hill, 2000. ISBN 0-07-231135-5.

STERMAN, J. *Modeling Managerial Behavior: Misperceptions of Feedback in a Dynamic Decision Making Experiment*. Management Science. 1989, 35(3), 321-339. DOI: 10.1287/mnsc.35.3.321. ISSN 0025-1909. Dostupné také z:
<http://pubsonline.informs.org/doi/abs/10.1287/mnsc.35.3.321>

STROZZI, F. BOSCH, J. *Beer game order policy optimization under changing customer demand. Decision Support Systems*. 2007, 42(4), 2153-2163. DOI: 10.1016/j.dss.2006.06.001. ISSN 01679236. Dostupné také z:
<http://linkinghub.elsevier.com/retrieve/pii/S0167923606000790>

XU, R., *The Analysis of Bullwhip Effect in Supply Chain Based on Strategic Alliance*. Integration and Innovation Orient to E-Society Volume 1. Boston, MA: Springer US, 2008, 452 s. DOI: 10.1007/978-0-387-75466-6_51. ISBN 978-0-387-75465-9. Dostupné také z:
<http://link.springer.com/10.1007/978-0-387-75466-6_51>

Přílohy

A Ukázka zdrojového kódu serverové aplikace

sendDataByRole

```
private void sendDataByRole(Int32 role, Socket current)
{
      Int32 boxIn;
      Int32 boxReqIn;

      switch (role)
      {
            case 0:
                  boxIn = _materialQue[1];
                  boxReqIn = _infoOrderQue[6];
                  break;
            case 1:
                  boxIn = _materialQue[3];
                  boxReqIn = _infoOrderQue[4];
                  break;
            case 2:
                  boxIn = _materialQue[5];
                  boxReqIn = _infoOrderQue[2];
                  break;
            case 3:
                  boxIn = _materialQue[7];
                  boxReqIn = _infoOrderQue[0];
                  break;
            default:
                  boxIn = 0;
                  boxReqIn = 0;
                  break;
      }

      if(_roundNumber > 35) // nastal konec hry?
      {
            if(!_endOfGame) // kontrolní metoda pro konec hry
            {
                  // spočítání celkových nákladů
                  _finalCosts = getFinalCosts();
                  // spočítání průměrných nákladů
                  _averageCosts = Convert.ToInt32(getAverageCosts());
                  _endOfGame = true;
                  // zápis skóre do souboru
                  writeToFinalScore(_finalCosts);
            }
```

```
            // sestavení zprávy pro všechny klienty
            Message m = new Message(role, _averageCosts, _finalCosts, -
      1000);
            byte[] data = m.getMessageByteArray();
            // odeslání zprávy
            current.Send(data);
      }
      else // nenastal konec hry – hraj dál
      {
            // sestavení zprávy nové kolo a zaslání klientovi
            Message m = new Message(role, boxIn, boxReqIn, -200);
            byte[] data = m.getMessageByteArray();
            current.Send(data);
            ...
      }
}
```

B Ukázka zdrojového kódu klientské aplikace

Struktura zprávy

```
public struct Message
{
      Int32 role;
      Int32 boxOut;
      Int32 boxReqOut;
      Int32 stock;      // stav skladu
      Int32 u_orders;   // stav nesplněných zakázek
      Int32 mess_Code;  // kód zprávy

// konstruktor struktury
public Message(Int32 p_role, Int32 p_boxOut, Int32 p_boxReqOut, Int32
p_stock, Int32 p_orders, Int32 p_messCode)
{
      role = p_role;
      boxOut = p_boxOut;
      boxReqOut = p_boxReqOut;
      stock = p_stock;
      u_orders = p_orders;
      roundCode = p_roudCode;
}
// převedení všech čísel do bytového pole a vrácení tohoto pole
public byte[] getMessageByteArray()
{
      byte[] data1 = BitConverter.GetBytes(role);
      byte[] data2 = BitConverter.GetBytes(boxOut);
      byte[] data3 = BitConverter.GetBytes(boxReqOut);
      byte[] data4 = BitConverter.GetBytes(stock);
      byte[] data5 = BitConverter.GetBytes(u_orders);
      byte[] data6 = BitConverter.GetBytes(mess_Code);

      byte[] data = new byte[data1.Length + data2.Length + da-
      ta3.Length + data3.Length + data4.Length + data5.Length + da-
      ta6.Length];
      Buffer.BlockCopy(data1, 0, data, 0, data1.Length);
      Buffer.BlockCopy(data2, 0, data, data1.Length, data2.Length);
      Buffer.BlockCopy(data3, 0, data, data1.Length + data2.Length,
      data3.Length);
      ...
      return data;
}
```

RecieveResponse

```
private void ReceiveResponse()
{
      var buffer = new byte[2048];
      // načtení dat ze soketu
      int received = _clientSocket.Receive(buffer, SocketFlags.None);
      if(received == 0) return;
            // sestavení pole z přijatých dat
            var data = new byte[received];
            // provedení kopie pole
            array.Copy(buffer, data, received);
            // rozparsování pole dle struktury zprávy
            Int32 role = BitConverter.ToInt32(data, 0);
            Int32 boxInOut = BitConverter.ToInt32(data, 4);
            Int32 reqInOut = BitConverter.ToInt32(data, 8);
            Int32 roundCode = BitConverter.ToInt32(data, 12);
            // server odpověděl zprávou – zahaj čekání na ostatní
            if(roundCode == -300)
            {
            // čekací smyčka – do té doby, než server neřekne jinak
                  waitingLoop();
            }
            // server odpověděl zprávou – nastalo nové kolo
            else if(roundCode == -200)
            {
                  stopWaiting(); // konec čekací smyčky
                  EnableControls(); // zaktivování ovládacích prvků
                  updateRound();    // aktualizace čísla kola
                  updateScore();    // aktualizace skóre
                  showControls();   // zobrazení ovládacích prvků
...
            // server odpověděl zprávou – ukonči aplikaci
            // vynucené ukončení hry
            } else if(roundCode == -900) // close game
            {
                  Exit();
            // server odpověděl zprávou – nastal korektní konec hry
            }else if(roundCode == -1000)
            {
                  endGameOccur();
            // zobraz konečné skóre týmu a porovnání s ostatními
            this.tb_teamTotal.Invoke(new MethodInvoker(delegate (){
            tb_teamTotal.Text = reqInOut.ToString() + " Kč"; }));
            this.tb_comparison.Invoke(new MethodInvoker(delegate ()
            { tb_comparison.Text = boxInOut.ToString() + " Kč"; }));
      }
}
```

C Ukázka zdrojového asynchronní komunikace klient – server

ReceiveCallback

```
private void ReceiveCallback(IAsyncResult AR)
{
      Socket current = (Socket)AR.AsyncState;
      int received;
      try
      {
            // ukončení přenosu dat a jejich uložení
            received = current.EndReceive(AR);
      }
      catch (SocketException x) // nastala chyba při přenosu?
      {
            ...
            // uzavření spojení mezi klientem a serverem
            current.Close();
            // vymazání spojení ze seznamu udržovaných spojení
            _clientSockets.Remove(current);
            // nastavení příznaku pro ukončení všech ostatních spojení
            _disconnectAllClients = true;
            return;
      }

      // sestavení zprávy z přenesených dat podle její struktury
      byte[] recBuf = new byte[received];
      Array.Copy(_buffer, recBuf, received);
      Int32 role = BitConverter.ToInt32(recBuf,0);
      Int32 boxOut = BitConverter.ToInt32(recBuf, 4);
      Int32 reqOut = BitConverter.ToInt32(recBuf, 8);
      Int32 stock = BitConverter.ToInt32(recBuf, 12);
      Int32 u_orders = BitConverter.ToInt32(recBuf, 16);
      Int32 roundCode = BitConverter.ToInt32(recBuf, 20);

      if(_disconnectAllClients || _clientSockets.Count > 4)
      {
            // odpoj všechny
            Message m = new Message(0, 400, 400, -900);
            byte[] data = m.getMessageByteArray();
            current.Send(data);
            current.Close();
            _clientSockets.Remove(current);
```

```
        if(_clientSockets.Count == 0)
        {
              System.Diagnostics.Process.Start("Server.exe",
        _PORT.ToString());
              Environment.Exit(0);
        }
  }
  else
  {
        // zaslání počáteční konfigurace klientovi
        if(roundCode == -600) // load configuration + role
        {
              initClientByRole(current);
        // klient odeslal svá data
        else if(reqOut != 0 && boxOut != 0 && roundCode == -500)
        {
              // uložení dat a zapsání do souboru
              updeteRoundCounter(role);
              updateDataQues(role, boxOut, reqOut);
              writeToFile(role, stock, u_orders);

              ...
        // Uvedení klienta do stavu čekání
        Message m = new Message(role, 300, 300, -300);
        byte[] data = m.getMessageByteArray();
        // odeslání zprávy klientovi
        current.Send(data);
        }
        // klient čeká, až server pošle signál na nové kolo
        else if(roundCode == -300)
        {
                if(isNextRound()) // je nové kolo?
                {
                    lock (_locker)// omezení pouze na jedno vlákno
                    {
                        if(_allPlayersReady[role] == false)
                        {
                            _allPlayersReady[role] = true;
                        }
                    }
              // posunutí obou front
              if(!_dataShifted)
              {
                    shiftQueByNewValue();
                    _dataShifted = true; // data posunuta
                    _roundNumber++;
              }
              if(arePlayerReady()) // jsou obslouzeni všichni?
              {
```

```
                        resetRoundCounter(); // vymazání všech čítačů
                    }
                    // místo, kde se všechny vlákna zastaví a počkají, až
                    // server dokončí svoji práci
                    Thread.Sleep(2000);
                    // odeslání dat klientovi podle jeho role
                    sendDataByRole(role, current); // new round
                }
                else
                {
                    // nové kolo nenastalo, čekej dál
                    Message m = new Message(role, 400, 400, -400);
                    byte[] data = m.getMessageByteArray();
                    current.Send(data);
                }
            }
            else
            {
                // nové kolo nenastalo, čekej dál
                Message m = new Message(role, 400, 400, -400);
                byte[] data = m.getMessageByteArray();
                current.Send(data);
            }
            // zahaj znovu přenos dat mezi klientem a serverem
            current.BeginReceive(_buffer, 0, _BUFFER_SIZE, SocketFlags.None,
ReceiveCallback, current);
        }}
```