**ŽILINSKÁ UNIVERZITA V ŽILINE**  
 FAKULTA RIADENIE A INFORMATIKY



**Semestrálna práca 2**Predmet: Diskrétna simuláciaŠtudijný program: Informačné systémy

Vypracoval: Zimen Dávid Školský rok: 2023/2024  
Cvičiaci: Ing. Andrea Galadíková Študijná skupina: 5ZIS12

**OBSAH**

[Zoznam obrázkov 3](#_Toc163543925)

[Zoznam tabuľiek 3](#_Toc163543926)

[1 Úloha 4](#_Toc163543927)

[2 Postup tvorby simulačného modelu 4](#_Toc163543928)

[3 Diagram udalostí 4](#_Toc163543929)

[4 Implementácia 5](#_Toc163543930)

[4.1 Generátory 5](#_Toc163543931)

[4.1.1 DiscreteUniformGenerator 6](#_Toc163543932)

[4.1.2 ContinuousUniformGenerator 6](#_Toc163543933)

[4.1.3 ContinuousEmpiricalGenerator 6](#_Toc163543934)

[4.1.4 ExponentionalGenerator 7](#_Toc163543935)

[4.1.5 TriangularGenerator 8](#_Toc163543936)

[4.2 Jadro 9](#_Toc163543937)

[4.2.1 SimulationCore 9](#_Toc163543938)

[4.2.2 EventSimulationCore 10](#_Toc163543939)

[4.2.3 CompanyEventSimulation 10](#_Toc163543940)

[4.3 Zákazník 11](#_Toc163543941)

[4.3.1 Enumy pre zákazníka 11](#_Toc163543942)

[4.4 Komponenty 13](#_Toc163543943)

[4.4.1 Queue<T> 13](#_Toc163543944)

[4.4.2 Service<T> 13](#_Toc163543945)

[4.4.3 ServingDeskQueue 13](#_Toc163543946)

[4.5 Udalosti 14](#_Toc163543947)

[4.5.1 AbstractEvent 14](#_Toc163543948)

[4.5.2 CompanyEvent 14](#_Toc163543949)

[4.5.3 CustomerArrivalEvent 14](#_Toc163543950)

[4.5.4 CustomerExitEvent 14](#_Toc163543951)

[4.5.5 PrintTicketStartEvent 15](#_Toc163543952)

[4.5.6 PrintTicketEndEvent 15](#_Toc163543953)

[4.5.7 CustomerServingStartEvent 15](#_Toc163543954)

[4.5.8 CustomerServingEndEvent 15](#_Toc163543955)

[4.5.9 DeskOcupationEvent 15](#_Toc163543956)

[4.5.10 PaymentStartEvent 15](#_Toc163543957)

[4.5.11 PaymentEndEvent 16](#_Toc163543958)

[4.6 Užívateľské rozhranie 16](#_Toc163543959)

[5 Experimenty 18](#_Toc163543960)

[5.1 Odporúčania 19](#_Toc163543961)

[Záver 20](#_Toc163543962)

# Zoznam obrázkov

[Obrázok 1 Diagram udalostí, časť 1 4](#_Toc163544518)

[Obrázok 2 Diagram udalostí, časť 2 5](#_Toc163544519)

[Obrázok 3 Diagram tried pre generátory 6](#_Toc163544520)

[Obrázok 4 Graf porovnania pravdepodobností 7](#_Toc163544521)

[Obrázok 5 Overenie exponenciálneho generátora 7](#_Toc163544522)

[Obrázok 6 Overenie trojuholníkového generátora 8](#_Toc163544523)

[Obrázok 7 Jadro simulačného modelu 9](#_Toc163544524)

[Obrázok 8 Diagram tried zákazníka a jemu pridružených enumov 11](#_Toc163544525)

[Obrázok 9 Diagram tried komponentov 13](#_Toc163544526)

[Obrázok 10 Diagram tried pre udalosti 14](#_Toc163544527)

[Obrázok 11 GUI v móde SINGLE 16](#_Toc163544528)

[Obrázok 12 GUI v móde REPLICATIONS 16](#_Toc163544529)

[Obrázok 13 Tab pre závislosť času v rade na lístky od pokladní 17](#_Toc163544530)

# Zoznam tabuľiek

[Tabuľka 1 Porovnanie pravdepodobností 6](#_Toc163543915)

[Tabuľka 2 Výsledky experimentov 18](#_Toc163543916)

# Úloha

Semestrálna práca 2 sa zaoberá simuláciou a optimalizáciou procesov v novom predajnom zariadení firmy Elektrokomponenty v Žiline. Úlohou je vytvoriť udalostne orientovaný simulačný model, ktorý zohľadňuje prúd zákazníkov, typy objednávok, platby a obslužné procesy.

Cieľom je navrhnúť optimálny počet pokladní a obslužných miest tak, aby priemerný čas strávený zákazníkom v predajni neprekročil 25 minút a priemerný čas čakania v rade pred automatom bol nižší ako 3 minúty. Je potrebné vykonať experimenty, vyhodnotia výsledky a odporučiť vhodný počet zariadení na základe štatisticky overených replikácií.

# Postup tvorby simulačného modelu

Ako prvý krok bolo detailne zanalyzované zadania. Na základ zadania sa následne vytvoril diagram udalostí. Následne sa začalo s implementáciou.

V prvom kroku implementácia sa vytvorili požadované generátory a overila sa ich funkčnosť a správnosť, tak aby nedošlo ku potenciálnemu skresleniu výsledkov.

V ďalšom kroku bolo vytvorené jadro aplikácie so všetkými abstraktnými triedami. Nad vytvoreným jadrom bol následne vytvorený konkrétny simulačný model so všetkými generátormi, stanovišťami, radmi a zbieraním štatistík a ku nemu sa podľa diagramu udalostí naprogramovali udalosti obsahujúce všetku logiku simulačného modelu.

Ako posledné bolo vytvorené užívateľské rozhranie pre vizualizáciu stavu simulácie.

# Diagram udalostí

Na základe popisu plánovanej organizácie predajne elektrokomponentov bol vytvorený diagram udalostí, ktorý slúži ako nástroj na vizualizáciu a pochopenie procesov od vstupu zákazníka do predajne až po jeho odchod.

Obrázok, na ktorom je text, diagram, snímka obrazovky, rad

Automaticky generovaný popisCieľom tohto diagramu je vizualizovať následnosť jednotlivých krokov a tak uľahčiť implementačnú časť.

Obrázok 1 Diagram udalostí, časť 1

**Obrázok, na ktorom je text, snímka obrazovky, diagram, rad

Automaticky generovaný popis**

Obrázok 2 Diagram udalostí, časť 2

Simulačný model začína udalosťou PríchodZákazníka, a ak je niekto pri automate na lístky, tak sa postaví do Radu pred automatom na lístky. Ak je automat voľný tak rovno nasleduje udalosť VydajLístok. Tam zákazník čaká na vydanie lístka s časom.

Potom ak je v Čakacej zóne menej ako 9 ľudí nasleduje udalosť ObsluhaZákazníkaStart, kde nadiktuje objednávku a čaká kým mu ju zamestnanec donesie a dochádza k udalosti ObsluhaZákazníkaEnd.

Po danej udalosti, nasleduje podľa veľkosti objednávky buď udalosti UvoľniMiesto, alebo ObsaďMiesto. V tomto prípade UvolniMiesto povolí obsluhu ďalšieho zákazníka a obslúžený zákazník odchádza do Radu pred pokladňou alebo začína rovno udalosť PlatenieStart. Ak sa zákazník iba vrátil po objednávku, tak nasleduje udalosť OdchodZákazníka.

Udalosť ObsaďMiesto nechá objednávku na obslužnom mieste a zákazník sa po ňu musí vrátiť po skončení udalosti PlatenieEnd. Ak nie je potreba vrátiť sa po veľkú objednávku, tak zákazník z predajne odchádza v udalosti OdchodZákazníka.

# Implementácia

Pre implementáciu zadania bol zvolený programovací jazyk Kotlin, ktorý je postavený na platforme JVM a má schopnosť využívať knižnice naprogramované v jazyku Java, ale ponúka jednoduchšiu syntax a možnosť funkčného programovania.

## Generátory

V každej simulácii veľmi záleží na generovaní náhodných čísiel, takže tejto oblasti je potrebné dať potrebnú dôležitosť a precíznosť.

Bolo potrebné naprogramovať vlastný generátor pre spojité empirické, exponenciálne a trojuholníkové rozdelenie. Avšak pre uľahčenie následnej implementácie, sme vytvorili nasledovnú štruktúru tried.

Obrázok, na ktorom je text, snímka obrazovky, diagram, rad

Automaticky generovaný popisVšetky triedy implementujú spoločné rozhranie Generator, ktoré teda sprístupňuje iba metódu sample(), ktorá má vrátiť iba vygenerovanú hodnotu typu Double.

Obrázok 3 Diagram tried pre generátory

Taktiež máme statický Random objekt, ktorý ako seed berie systémový čas. Ten následne generuje násady do atribútov generator v jednotlivých generátoroch.

### DiscreteUniformGenerator

Pomocou atribútu generator, vygeneruje celé číslo z intrevalu <minValue; maxValue).

### ContinuousUniformGenerator

Vracia hodnotu z intervalu <minValue; maxValue) ako reálne číslo.

### ContinuousEmpiricalGenerator

Atribút mainGenerator vygeneruje hodnotu z intervalu <0; 1) a na základe tejto hodnoty sa vyberie z mapy generators príslušný objekt IntervalRandom, ktorý obsahuje príslušné hodnoty intervalu a inštanciu triedy Random a vráti vygenerovanú hodnotu z intervalu.

Pre overenie korektného naprogramovania bolo vygenerovaných 500 000 hodnôt, ktoré boli spracované a pre jednotlivé intervaly boli porovnané teoretické a skutočné pravdepodobnosti.

Tabuľka 1 Porovnanie pravdepodobností

|  |  |  |  |
| --- | --- | --- | --- |
| **Interval** | **Počet vygenerovaných hodnôt** | **Skutočná pravdepodobnosť** | **Teoretická pravdepodobnosť** |
| <0,1 - 0,3) | 100103 | 0,100103 | 0,1 |
| <0,3 - 0,8) | 349288 | 0,349288 | 0,35 |
| <0,8 - 1,2) | 200670 | 0,20067 | 0,2 |
| <1,2 - 2,5) | 150362 | 0,150362 | 0,15 |
| <2,5 - 3,8) | 149541 | 0,149541 | 0,15 |
| <3,8 - 4,8) | 50036 | 0,050036 | 0,05 |

Obrázok 4 Graf porovnania pravdepodobností

Ako vyplýva z uvedenej tabuľky a grafu, tak rozdiel medzi teoretickou a skutočnou pravdepodobnosťou je na úrovní tisícin až desaťtisícin, takže generátor sa môže považovať za korektný.

### ExponentionalGenerator

Generátor exponenciálneho rozdelenia, ktorý berie ako vstupný parameter lambdu. Podľa neho na základe vzorca pre exponenciálne rozdelenie vráti vygenerovanú hodnotu.

Obrázok, na ktorom je text, snímka obrazovky, diagram, vývoj

Automaticky generovaný popisPre overenie správnosti, bolo vygenerovaných náhodných hodnôt s parametrom lambda 1 / 2. Následne sme tieto dáta vložili do programu Input Analyzer a získali nasledovné výsledky.

Obrázok 5 Overenie exponenciálneho generátora

Ako vidno z obrázka po vykonaní Chi kvadrát testu vyšla p-hodnota 0,09. Táto hodnota je väčšia ako 0,05. To znamená, že pôvodná hypotéza o správnosti generátora sa nedá vyvrátiť, takže generátor sa môže považovať za korektný.

### TriangularGenerator

Generátor trojuholníkového rozdelenia, ktorý berie ako vstupné parametere minimum, modus a maximum. Podľa nich na základe vzorca pre trojuholníkové rozdelenie vráti vygenrovanú hodnotu.

Obrázok, na ktorom je text, snímka obrazovky, vývoj, diagram

Automaticky generovaný popisoverenie správnosti, bolo vygenerovaných náhodných hodnôt s parametrami minimum 60, modus 120 a maximum 480. Následne sme tieto dáta vložili do programu Input Analyzer a získali nasledovné výsledky.

Obrázok 6 Overenie trojuholníkového generátora

Ako vidno z obrázka po vykonaní Chi kvadrát testu vyšla p-hodnota väčšia ako 0,75. Táto hodnota je väčšia ako 0,05. To znamená, že pôvodná hypotéza o správnosti generátora sa nedá vyvrátiť, takže generátor sa môže považovať za korektný.

## Jadro

Na nasledujúcom obrázku je zobrazený diagram tried pre jadro simulačného modelu.Obrázok, na ktorom je text, snímka obrazovky, písmo, dokument

Automaticky generovaný popis

Obrázok 7 Jadro simulačného modelu

### SimulationCore

Všeobecné simulačné jadro, ktoré má metódy runSimulation, stopSimulation, pauseSimulation a resumeSimulation, ktoré sú finálne a nedajú sa prekryť, lebo v nich sa ovláda zmena stavu simulácie. Poskytuje však metódy onSimulationRun, onSimulationStop, onSimulationPause a onSimulationResume pre vlastnú implementáciu logiky behu vyššie spomenutých metód.

### EventSimulationCore

Jadro pre udalostnú simuláciu. Je potomkom triedy SimulationCore a prekrýva jeho metódy onSimulationRun, onSimulationStop, onSimulationPause a onSimulationResume. Taktiež poskytuje vlastné metódy na prekrytie beforeSimulation, beforeReplication, afterSimulation a afterReplication pre vlastnú programovú logiky v jednotlivých fázach simulácie.

Obsahuje prioritný front eventsQueue, do ktorej sa vkladajú udalosti, pričom priorita je čas vykonania udalosti. Čím menší čas, tým vyššia priorita.

Privátna metóda executeEvents, v cykle while berie udalosti z frontu a vykonáva ich pomocou ich spoločnej metódy execute. Udalosti sa vykonávajú až dokým nie je front prázdny. Táto metóda sa volá v tele metódy simulationRun, kde sa vykonávajú jednotlivé replikácie, tak že jedna replikácia je jedno volanie executeEvents.

Taktiež je k dispozícii metóda updateSimulationState, ktorá nastaví atribúty v triede EventSimulationState na aktuálne a tie môžu byť poskytnuté užívateľovi na grafickom rozhraní.

Jadro pracuje v dvoch módoch podľa enumu EventSimulationMode. V móde SINGLE sa do kalendára udalostí plánujú aj udalosti na dočasné pozastavenie simulácie, čím sa simuluje tok reálneho času. V móde REPLICATIONS sa tieto udalosti neplánujú, čo znamená, že simulácia bude bežať čo najrýchlejšie.

### CompanyEventSimulation

Potom triedy EventSimulationCore. Je konkrétnou implementáciou zadania, ktorá združuje všetky potrebné generátory. Taktiež je možné nastaviť čas otvorenia predajne, čas ukončenia automatu a čas zavretia prevádzky. Podľa zadania je možné nastaviť aj počet pokladní a počet obslužných miest.

Taktiež obsahuje triedy pre zbieranie štatistík potrebných pre neskoršie vyhodnotenie simulačných experimentov.

Prekrýva dostupné metódy z predka a vykonáva s nimi tieto akcie:

* beforeSimulation – inicializácia stavu, celkových štatistík a generátorov
* beforeReplication – inicializácia pokladní a obslužných miest podľa parametrov, naplánovanie príchodu prvého zákazníka
* afterSimulation – upovedomenie užívateľa o výsledku
* afterReplication – pridanie štatistík replikácie do celkových štatistík

## Zákazník

Na nasledujúcom diagrame tried vidíme triedu Customer, ktorý reprezentuje zákazníka v predajni.

Obrázok, na ktorom je text, snímka obrazovky, písmo, diagram

Automaticky generovaný popisZákazník obsahuje svoj identifikátor, ticketTime pre čas aký mu vytlačil automat na lístky. Potom obsahuje štatistické atribúty systemStartTime, ticketMachineQueueStartTime, servingDeskQueueStartTime, cashDeskQueueStartTime, ktoré zaznamenávajú časy príchodov na jednotlivé stanovištia a potom v konkrétnych udalostiach sú používané do štatistík v jadre.

Obrázok 8 Diagram tried zákazníka a jemu pridružených enumov

Atribút cashDeskQueue predstavuje pokladňu, kde čakal v rade. CashDeskPaid pokladňu, kde reálne zaplatil a servingDesk miesto kde diktoval a prevzal objednávku.

### Enumy pre zákazníka

Všetky enumy obsahujú metódu retrieve(double probability). Na základe parametra probability, ktorý je vygenerovaný ako hodnota z intervalu <0; 1) generátorom z jadra pre príslušný enum sa vráti hodnota z enumu. Všetky enum atribúty pre triedu Customer sú generované už pri inicializácii objektu pri udalosti PríchodZákazníka.

#### OrderSize

Má dve hodnoty:

* NORMAL – objednávka ma klasickú veľkosť a po prevzatí môže ísť aj s ňou na pokladňu
* BIG – po prevzatí musí ísť najprv zaplatiť a potom sa pre ňu vrátiť, obslužné miesto je zatiaľ obsadené

#### OrderType

Má 3 hodnoty, na základe ktorých sa potom generuje čas trvania prípravy objednávky:

* NORMAL
* SLIGHTLY\_COMPLEX
* COMPLEX

#### CustomerType

Má 3 hodnoty:

* ONLINE – má špeciálne obslužné miesto
* COMMON – má miesta spolu so zmluvným
* CONTRACTED – má miesta spolu s obyčajným, pričom má pred ním prednosť

#### PaymentType

Má 2 hodnoty, na základe ktorých sa generuje doba platenia na pokladni:

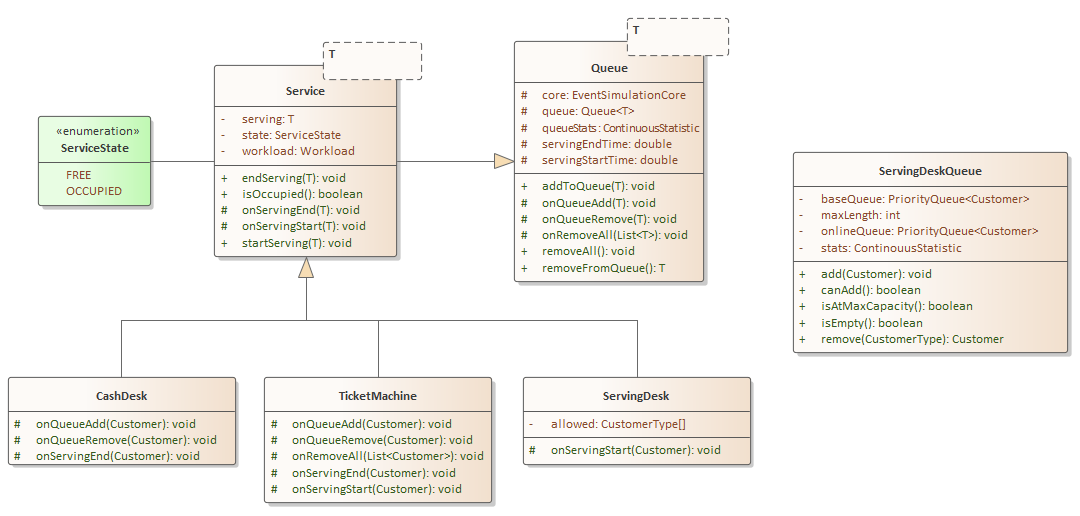
* CASH
* CARD

#### CustomerState

Tieto hodnoty predstavujú momentálne miesto zákazníka v predajni a menia sa mu pri príslušných udalostiach:

* ARRIVED - Príchod zákazníka.
* IN\_TICKET\_QUEUE - Zákazník čaká v rade pri automate na lístky.
* PRINTING\_TICKET - Zákazník je pri automate na lístk.
* WAITING\_FOR\_SERVING - Lístok bol vytlačený, teraz čaká v rade na objednávku.
* BEING\_SERVED - Zákazník požaduje svoju objednávku a čaká na obsluhu pri servisnom pulte.
* WAITING\_FOR\_PAY - Zákazník bol obslúžený a teraz čaká v rade na pokladni.
* PAYING - Momentálne je u pokladne a platí za svoju objednávku.
* PICKING\_UP - Ak mal zákazník veľkú objednávku a musel sa vrátiť k servisnému pultu, aby si ju vyzdvihol.
* LEFT - Zákazník opustil predajňu po obsluhe.
* LEFT\_TICKET\_MACHINE - Zákazníci, ktorí opustili predajňu kvôli uzavretiu automatu na lístky.

## Komponenty

Na nasledujúcom obrázku sa nachádza diagram tried pre komponenty, ktoré vznikli na odseparovanie repetitívnej logiky z udalostí, čím sa dosiahne jednoduchší, čitateľnejší a udržateľnejší kód.

Obrázok 9 Diagram tried komponentov

### Queue<T>

Tried pre rad v simulácii. Obsahuje odkaz na jadro, ktorému patrí a queue, kde sa ukladajú agenti. Jej výhodou je, že automaticky počíta spojitú štatistiku pre dĺžku radu v čase.

Pre vlastnú nadstavbovú logiku poskytuje metódy onQueueAdd, onQueueRemove a onRemoveAll na prekrytie.

### Service<T>

Služba alebo miesto, ktoré môže obsluhovať agentov. Je potomkom triedy Queue<T>, čo znamená, že má v sebe implementovaný rad pre čakanie na obsluhu.

Automaticky ráta pracovnú záťaž podľa poskytnutých informácii o začatí a skončení zmeny v predkovi.

Uchováva si v sebe stav FREE alebo OCCUPIED, podľa toho či momentálne obsluhuje nejakého agenta.

Pre implementáciu vlastnej logiky pri začatí alebo skončení obsluhovania poskytuje na prekrytie metódy onServingStart a onServingEnd.

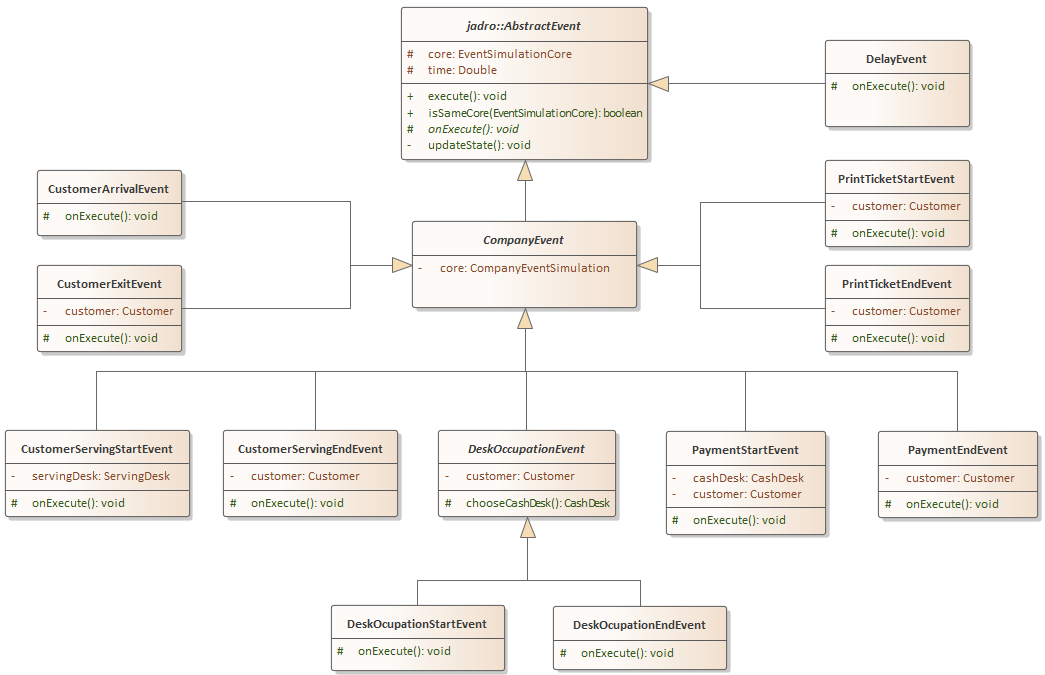
Jej potomkami sú triedy:

* CashDesk – predstavuje pokladňu predajne
* TicketMachine – automat na vydávanie lístkov po vstupe do predajne
* ServingDesk – obslužné miesto pre diktovanie a chystanie objednávok

### ServingDeskQueue

Predstavuje zónu, kde zákazníci čakajú kým budú zavolaný ku obslužnému miestu. Vznikla pre potrebu implementovať pomocou 2 prioritných frontov, keďže online zákazníci majú vlastné obslužné miesta.

## Udalosti

Na nasledujúcom obrázku je zobrazený diagram tried udalostí, ktorý bol navrhnutý podľa diagramu udalostí z kapitoly 2.

### AbstractEvent

Obrázok 10 Diagram tried pre udalosti

Abstraktná udalosť, ktorej metóda execute sa volá v jadre simulácie. V tejto metóde sa vykoná metóda onExecute, ktorú musia potomkovia prekryť a taktiež na konci vykonania aktualizuje stav simulácie.

Má v sebe čas, kedy má byť udalosť vykonaná a odkaz na jadro, ktorému patrí. Jedným jej priamym potomkov je DelayEvent, ktorý iba pozastaví vlákno jadra na čas vyrátaný podľa nastavenej rýchlosti simulácie.

### CompanyEvent

Je potomkom triedy AbstractEvent a taktiež je abstraktnou triedou. Vznikla iba preto aby mohla prekryť atribút core z EventSimulationCore na CompanyEventSimulation. Takto nie je potom v potomkoch zakaždým pretypovávať, aby sa mohlo dostať ku metódam CompanyEventSimulation.

### CustomerArrivalEvent

Vytvorí nového zákazníka do predajne a podľa obsadenosti automatu ho buď pridá do radu pred automat, alebo rovno naplánuje udalosť PrintTicketStartEvent. Ak nie je automat na lístky zatvorený, tak naplánuje podľa exponenciálneho generátora nový CustomerArrivalEvent.

### CustomerExitEvent

Pridá zákazníka do listu obslúžených zákazníkov a aktualizuje štatistiky o čase v predajni a poslednom odchode zákazníka a obslúžených zákazníkoch. Už neplánuje žiadnu udalosť.

### PrintTicketStartEvent

Ak tejto udalosti bol poskytnutý zákazník, tak ho začne obsluhovať. Ak nebol, tak vyberie jedného z radu pre automat. Naplánuje udalosť PrintTicketEndEvent, kde čas bude o vygenerovanú reálnu hodnotu z intervalu <30; 120).

### PrintTicketEndEvent

Ukončí pobyt zákazníka pri automate. Ak v priestore pre čakanie na obsluhu je menej ako 9 ľudí, zároveň rad pred automatom nie je prázdny a zároveň automatu neskončila obslužná doba, tak naplánuje udalosť PrintTicketStartEvent.

Ak je voľné obslužné miesto, ktoré môže obslúžiť zákazníka, tak je pre to miesto rovno naplánované udalosť CustomerServingStartEvent. Ak nie je žiadne voľné, tak pridá zákazníka do čakacej zóny.

### CustomerServingStartEvent

Vyberie si zákazníka na obslúženie z čakacej zóny. Ak tá nie je na maximálnej kapacite, tak naplánuje udalosť PrintTicketStartEvent.

Naplánuje udalosť CustomerServingEndEvent o čas podľa typu zákazníka. Ak je online, tak sa mi vygeneruje čas vyzdvihovania podľa trojuholníkového rozdelenia minimum 60, modus 120 a maximum 480.

Ak je bežný alebo zmluvný, tak sa vygeneruje čas ukončenia ako súčet vygenerovanej hodnoty pre diktovania a následné nachystanie objednávky zamestnancom.

### CustomerServingEndEvent

Podľa veľkosti objednávky zákazníka naplánuje DeskOcupationStartEvent ak je veľkosť normálna alebo DeskOcupationEndEvent ak je objednávka príliš veľká. Čas týchto udalostí bude rovnaký ako čas tejto udalosti.

### DeskOcupationEvent

Abstraktný predok pre DeskOcupationStartEvent a DeskOcupationEndEvent. Poskytuje im metódu, pomocou ktorej dokážu zvoliť správnu pokladňu podľa zadaných podmienok. Ak nevie zvoliť pokladňu priamo, tak pridá zákazníka do nejakého radu.

#### DeskOcupationStartEvent

Táto udalosť nastane iba keď je objednávka príliš veľká, takže pre vybratú pokladňu, ak nejaké je, naplánuje udalosť PaymentStartEvent.

#### DeskOcupationEndEvent

Uvoľní obslužné miesto od zákazníka. Ak tento zákazník mal veľkú objednávku, tak je preň naplánované udalosť CustomerExitEvent v opačnom prípade PaymentStartEvent alebo zaradenie do radu nejakej pokladne.

Ak toto miesto dokáže obslúžiť niekoho z čakacej zóny, tak naplánuje aj CustomerServingStartEvent.

### PaymentStartEvent

Ak má priradeného zákazníka, tak začne s jeho obsluhou. Ak nemá, tak si vyberie prvého z radu. Vygeneruje hodnotu koľko má platenie trvať a podľa toho naplánuje udalosť PaymentEndEvent.

### PaymentEndEvent

Ukončí platenie zákazníka. Ak bola jeho objednávka veľká, tak naplánuje udalosť DeskOcupationEndEvent o čas vygenerovaný na vyzdvihnutie objednávky od pokladne. V opačnom prípade naplánuje CustomerExitEvent.

## Užívateľské rozhranie

Na tvorbu užívateľského rozhrania bola použitá technológia JavaFX. Pre účely vykresľovania grafov sa využila knižnica JFreeChart.

Aplikácia je rozdelená na 2 taby. Prvý je pre sledovanie simulačného behu, na druhom sa nachádza graf pre závislosť času v rade pred automatom od počtu pokladní.

Prvý tab poskytuje možnosť nastaviť si počet replikácii, pokladní, obslužných miest, priestor pre čakanie na obsluhu. Taktiež je možné si zvoliť jednotlivé časy a upraviť rýchlosť. Pracuje v 2 módoch, kde prvý je SINGLE a druhý je REPLICATIONS.

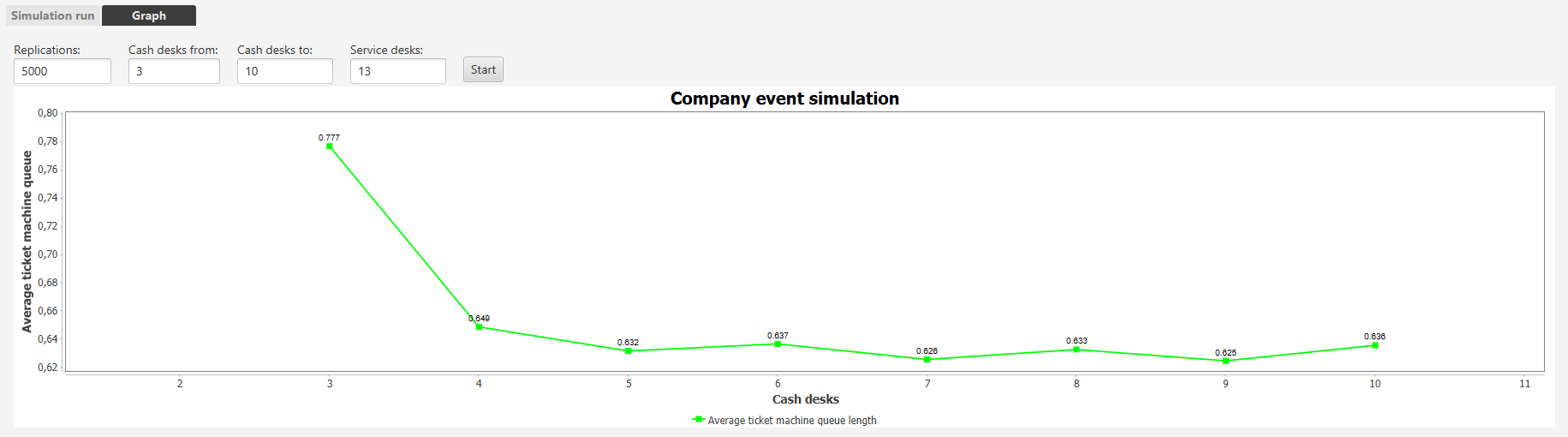
Obrázok 11 GUI v móde SINGLE

Obrázok, na ktorom je text, snímka obrazovky, číslo, rad

Automaticky generovaný popisObrázok, na ktorom je text, snímka obrazovky, číslo, dokument

Automaticky generovaný popisV móde SINGLE sa sleduje stav jednotlivých zákazníkov a zamestnancov v rámci jednej replikácie. Zobrazuje sa simulačný čas pre aktuálnu replikáciu a vypisujú sa štatistiky v rámci jednej replikácie.

Obrázok 12 GUI v móde REPLICATIONS

 V móde REPLICATIONS sa následne sledujú celkové štatistiky pre simuláciu a jednotlivých zamestnancov. Nezobrazuje sa simulačný čas, keďže replikácie idú rýchlo za sebou.

Obrázok 13 Tab pre závislosť času v rade na lístky od pokladní

Na tomto tabe si užívateľ môže zvoliť interval pre koľko pokladní sa má experiment vykonávať, počet obslužných miest a počet replikácii pre každú pokladňu.

Výsledkom je potom graf ako na obrázku vyššie.

# Experimenty

Vyhodnotenie experimentov a ich správne interpretácia sú kľúčovými aspektmi v procese získavania poznatkov a vytvárania informovaných rozhodnutí.

V rámci experimentov s modelom vytvoreným podľa zadania, je cieľom skúmať a optimalizovať fungovanie predajne, konkrétne procesu obsluhy zákazníko. V našom prípade sa zameriavame na určenie vhodného počtu pokladní a obslužných miest s ohľadom na dva kľúčové parametre: priemerný čas strávený zákazníkom v predajni a priemerný čas čakania v rade pred automatom na poradové číslo.

S modelom sme vykonali nasledujúce experimenty. Pre každý experiment bolo vykonaných 100 000 replikácii.

Tabuľka 2 Výsledky experimentov

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Počet obslužných miest** | **Počet pokladní** | **Priemerný čas v predajni**  [95 % interval spoľahlivosti] | **Priemerný čas v rade pred automatom na lístky** | **Priemerný počet obslúžených zákazníkov** |
| 3 | 1 | 204.456 min [204.37; 204.543] | 124.877 min | 78.213 |
| 6 | 2 | 126.016 min  [125.921; 126.111] | 76.781 min | 143.55 |
| 12 | 4 | 27.319 min  [27.276; 27.362] | 2.211 min | 238.237 |
| 12 | 5 | 26.365 min  [26.326; 26.405] | 2.021 min | 238.576 |
| 13 | 4 | 23.784 min  [23.765; 23.803] | 1.262 min | 239.287 |
| 14 | 3 | 25.131 min  [25.112; 25.15] | 1.208 min | 239.388 |
| 14 | 4 | 22.662 min  [22.652; 22.673] | 1.15 min | 239.373 |
| 15 | 3 | 25.135 min  [25.115; 25.154] | 1.206 min | 239.486 |
| 15 | 4 | 22.644 min [22.634; 22.655] | 1.151 min | 239.331 |
| 25 | 8 | 21.163 min [21.158; 21.168] | 1.133 min | 239.443 |

Pre prvý experiment sme si zvolili počet pokladní 1 a počet obslužných miest 3. Ako môžeme vidieť v Tabuľke 2, tak výsledné časy sú ďaleko za cieľovými hodnotami. Preto pre ďalší experiment sme v oboch prípadoch zdvojnásobili počet. Aj v tomto prípade sú výsledky ďaleko od cieľového času, takže znova zdvojnásobujeme na počet pokladní 4 a počet obslužných miest na 12.

Pri tomto experimente sme sa dostali na čas v rade pod 3 minúty, avšak čas v predajni neklesol pod 25 minút. Avšak tieto čísla už sú blízko, takže pristúpime iba ku jemnej úprave a zvýšime počet pokladní na 5.

V tomto prípade nedošlo ku takmer žiadnej zmene a čas v predajni je stále nad úrovňou 25 minút.

Pre ďalší experiment teda skúsime nastaviť počet obslužných miest na 13 a počet pokladní na 4. V tomto prípade sme sa pre oba meraná ukazovatele dostali pod požadované hodnoty. Konkrétne priemerný čas v predajni je 23,784 minút a priemerný čas v rade pred automatom je 1262 minút.

Následne sme ešte skúsili vykonať ďalšie experimenty postupným pridávaním obslužných miest a pokladní. V prípadoch pre počet obslužných miest 14 a 15 pri počte pokladní 3 je vidieť, že celkový čas v predajni je nad hranicou 25 minút. Takže tieto prípady nespĺňajú požiadavky.

V prípadoch pre počet obslužných miest 14 a 15 pri počte pokladní nastaveného na 4 sú v oboch prípadoch takmer rovnaké výsledky, a to priemerný čas v predajni na úrovni 22,6 minút a priemerný čas v rade pred automatom na úrovni 1,15 minúty. Pri daných výsledkoch, môžeme aj tieto 2 konfigurácie považovať za korektné.

Posledný experiment bol vykonaný s nastaveniami počet pokladní na 8 a počet obslužných miest na 25. Bol zvolený pre zistenie, či sa s výrazným navýšením počtom aj viditeľnejšie zlepšia priemerné časy. V tomto prípade nie. Celkový čas sa zlepšil oproti predošlým iba zhruba o 1,5 minúty a čas pred automatom iba o zhruba 0,02 minúty.

Ďalšie experimenty už neboli vykonávané, keďže priemerné časy sa stabilizovali a pridávaním ďalších obslužných miest by nebol moc priestor na zlepšenie.

## Odporúčania

Na základe vyhodnotenia jednotlivých experimentov sme usúdili, že najlepšia kombinácia pre predajňu bude 13 obslužných miest a 4 pokladne.

Aj keď niektoré ďalšie kombinácie vykazovali jemne lepšie výsledky, tak si môžeme všimnúť, že navýšenie počtu zamestnancov by aj tak neviedlo ku viacerým obslúženým zákazníkom, a tým pádom ku potencionálne väčšiemu zisku. Pri všetkých prístupných kombináciách je počet obslúžených zákazníkov na úrovni 239. To je vďaka tomu, že zóna pre čakanie na obsluhu má kapacitu iba 9 zákazníkov a s tým nespraví nič ani navýšenie počtu zamestnancov.

V našom prípade by nárast počtu zamestnancov znamenal iba dodatočné náklady, takže odporúčaním je 13 obslužných miest a 4 pokladne.

# Záver

V tejto semestrálnej práci sme vypracovali udalostne orientovaný simulačný model predajne spoločnosti Elektrokomponenty. Tento model bol návrhovo a implementačne komplexný, zahŕňajúci všetky hlavné aspekty predajného procesu, ako sú príchod zákazníkov, vydávanie poradových čísiel, obsluha zákazníkov na rôznych typoch miest a platenie na pokladniach.

S cieľom optimalizovať čas strávený zákazníkom v predajni a čas čakania v rade pred automatom sme vykonali sériu experimentov. Na základe týchto experimentov sme identifikovali optimálny počet pokladní a obslužných miest, ktoré zabezpečujú poskytovanie služieb na požadovanej úrovni s čo najmenšími nákladmi na pracovnú silu