Aplikace pro demonstraci Buffonovy úlohy o jehle

Peter Revaj

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Peter Revaj**
Osobní číslo:	**A19611**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Aplikace pro demonstraci Buffonovy úlohy o jehle**
Téma práce anglicky:	**Application to Demonstrate the Buffon's Needle Problem**

Zásady pro vypracování

1. Zpracujte literární rešerši na téma geometrické pravděpodobnosti. Zaměřte se na Buffonovu úlohu o jehle, analyzujte různá řešení a možná rozšíření (jiné struktury, na které jehla dopadá).
2. Proveďte rešerši vytvořených aplikací na dané téma.
3. Navrhněte a vytvořte uživatelsky přívětivou aplikaci demonstrující Buffonovu úlohu o jehle. Aplikaci doplňte o vybraná rozšíření této úlohy.
4. Podrobně popište jednotlivé funkce vytvořené aplikace, volitelné parametry a výstupy.
5. Využití aplikace demonstrujte na experimentu odhadu Ludolfova čísla.
6. Zhodnoťte výsledky provedených experimentů.
7. Srovnejte vytvořenou aplikaci s aplikacemi vytvořenými v minulosti, nastiňte výhody a přínosy navržené aplikace.

Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. PŁOCKI, Adam. *O náhodě a pravděpodobnosti: pro účastníky matematické olympiády*. Praha: Mladá fronta, 1982. Škola mladých matematiků.
2. HYKŠOVÁ, Magdalena: *Filosofická pojetí pravděpodobnosti v pracích českých myslitelů.* (Czech). Praha: Matfyzpress, vydavatelství Matematicko-fyzikální fakulty Univerzity Karlovy v Praze, 2011. ISBN 978-80-7378-192-7.
3. MATHAI, A. M. *An Introduction to Geometrical Probability: Distributional Aspects with Applications*. Amsterdam: Gordon and Breach Science Publishers, 1999. ISBN 978-90-5699-681-9.
4. HLEDÍK, Jakub. *Buffonova úloha o jehle a její zobecnění*. 2018. Bakalářská práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra pravděpodobnosti a matematické statistiky.
5. DANÍČEK, Ladislav. *Využití geometrické pravděpodobnosti při experimentálním ověření Ludolfova čísla*. Bakalářská práce. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005, 37 s., 7s. příloh.

Vedoucí bakalářské práce: **doc. Ing. Bc. Bronislav Chramcov, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **15. ledna 2022**
Termín odevzdání bakalářské práce: **20. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 18.5.2022

Peter Revaj, v.r.
podpis studenta

ABSTRAKT

Hlavným cieľom bakalárskej práce bolo vytvoriť užívateľsky prívetivú desktopovú aplikáciu pre aproximáciu Ludolfovho čísla pomocou Buffonovej úlohy o ihle. Daný cieľ sme splnili použitím frameworku Qt s použitím programovacieho jazyka QML a C++. Výsledkom je spustiteľná aplikácia, ktorá po zvolení parametrov užívateľom dokáže zobraziť priebeh experimentu animáciou, vytvoriť grafy závislosti Ludolfovho čísla na vstupných parametroch experimentu. Súčasťou aplikácie je tiež tutoriál obsahujúci teoretické informácie o danej problematike. Aplikácia bola navrhnutá pre potenciálne použitie študentami a pre lepšie pochopenie a vizualizáciu problému Buffonovej úlohy. Súčasťou práce je aj teoretická časť, ktorá popisuje geometrickú pravdepodobnosť a možné riešenia danej problematiky.

Kľúčové slová: Geometrická pravdepodobnosť, Ludolfovo číslo, Buffonova úloha o ihle, Monte Carlo, Qt, QML

ABSTRACT

The main goal of the bachelor's thesis was to create a user-friendly desktop application for approximation of Ludolf number via Buffon's needle problem. The given goal was achieved using Qt framework with the usage of programing languages QML and C++. The final product is an executable application which after setting parameters by the user can showcase the flow of experiment via animation, create graphs of Ludolf number dependencies on entry parameters of the experiment. This application contains a tutorial of theoretical information regarding the given problematics. The application was designed to be used by students for better understanding and visualization of Buffon's needle problem. The thesis also contains a theoretical part that explains Geometrical probability and possible solutions of given problematics.

Keywords: Geometrical probability, Ludolf number, Buffon's needle problem, Monte Carlo, Qt, QML

Chtěl bych se touto cestou poděkovat mému vedoucímu doc. Ing. Bc. Bronislavovi Chramcovovi, Ph.D. za bezproblémovou spolupráci, ochotu, odborné rady a pomoc při řešení všech problémů.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČASŤ **10**
1 ÚVOD DO TEORETICKEJ ČASŤI BAKALÁRSKEJ PRÁCE **11**
2 PRAVDEPODOBNOSŤ **12**
2.1 POJMY 12
2.1.1 Experiment 12
2.1.2 Náhodný jav 12
2.1.3 Nemožný a istý jav 13
2.1.4 Náhodné číslo 13
2.1.5 Iterácia 13
2.1.6 Graf závislosti jednej veličiny na druhej 13
2.2 GEOMETRICKÁ PRAVDEPODOBNOSŤ 14
2.2.1 Historický vývoj geometrickej pravdepodobnosti 14
2.3 LUDOLFOVO ČÍSLO 15
2.3.1 Ludolph van Ceulen 15
2.3.2 Vývoj presnosti Ludolfovho čísla 15
2.3.3 Metódy vyčíslenia Ludolfovho čísla 16
2.4 METÓDA MONTE CARLO 17
3 BUFFONOVA ÚLOHA O IHLE **18**
3.1 GEORGES-LOUIS LECLERC, COMTE DE BUFFON 18
3.2 ROVNOBEŽKY 18
3.3 ROZŠÍRENIA BUFFONOVEJ ÚLOHY 19
3.3.1 Obdĺžniková mriežka 19
3.3.2 Rovnostranný trojuholník 20
4 DOSTUPNÉ RIEŠENIA BUFFONOVEJ ÚLOHY O IHLE **22**
II PRAKTICKÁ ČASŤ **26**
5 ÚVOD DO PRAKTICKEJ ČASTI BAKALÁRSKEJ PRÁCE **27**
6 NÁVRH APLIKÁCIE **28**
6.1 QT FRAMEWORK 28
6.1.1 Qt Creator 28
6.1.2 QML 29
6.2 ŠTRUKTÚRA ZDROJOVÉHO KÓDU 30
6.2.1 Hlavičkové súbory 30
6.2.2 Zdrojové súbory 33
6.2.3 Časť zdrojov 33
6.2.4 Prepojenie c++ časti s qml časťou 34
6.3 NÁVRH APLIKÁCIE PODĽA ZADANIA 35
6.3.1 Flow diagram 35
6.3.2 Farby, písmo a motív 35
7 IMPLEMENTÁCIA **37**
7.1 GUI 37
7.1.1 Vlastné nastaviteľné tlačidlo 37

7.1.2 Main Window 38
7.1.3 Loader 38
7.1.4 StackView 39
7.1.5 Obrazovky 40
7.1.6 Vykreslenie ihiel a plochy 50
7.1.7 Vykreslenie grafov 54
7.2 VÝPOČTOVÁ ČASŤ 56
7.2.1 Trieda calculations 57
7.2.2 Nastavenie, uloženie a resetovanie parametrov 57
7.2.3 Štart experimentu 59
7.2.4 Generovanie náhodných čísel 60
7.2.5 Výpočet Pi – rovnobežky 60
7.2.6 Výpočet Pi – obdĺžniková mriežka 62
7.2.7 Výpočet Pi – rovnostranný trojuholník 63
7.2.8 Výpočet Pi pre viac iterácií 65
7.3 TEORETICKÝ ÚVOD 66
7.4 VIDEOUKÁŽKA 66
7.5 VYTVORENIE APLIKÁCIE A PRENOSITEĽNOSŤ NA INÉ ZARIADENIA 67
7.5.1 Qmake 67
7.5.2 Prenositenosť 67
8 EXPERIMENTY NA VÝPOČET LUDOLFOVHO ČISLA 68
8.1 EXPERIMENTY NA PLOCHE ROVNOBEŽKY 68
8.1.1 Experimenty podobnosti výsledku v prípade rovnosti pomerov parametrov 68
8.1.2 Experimenty s malou ihlou 70
8.1.3 Experimenty s polovičnou hodnotou veľkosti ihly 72
8.1.4 Experimenty 2/3 veľkosťou ihly 76
8.1.5 Experimenty s použitím viacerých iterácií 76
8.1.6 Experimenty pre získanie najlepšej hodnoty Pi 79
8.2 EXPERIMENTY NA PLOCHE OBDĹŽNIKOVÁ MRIEŽKA 80
8.2.1 Experimenty s malou ihlou 80
8.2.2 Experimenty s polovičnou ihlou 82
8.3 EXPERIMENTY NA PLOCHE ROVNOSTRANNÝ TROJUHOLNÍK 84
8.3.1 Experimenty s malou ihlou 84
8.3.2 Experimenty s polovičnou ihlou 85
8.3.3 Experimenty s najväčšou možnou ihlou 87
9 VÝSLEDOK EXPERIMENTOV A POROVNANIE S EXISTUJÚCIMI APLIKÁCIAMI 90
9.1 ZHODNOTENIE VÝSLEDKOV APLIKÁCIE 90
9.2 POROVNANIE S EXISTUJÚCIMI APLIKÁCIAMI 92
9.2.1 GUI 92
9.2.2 Počet iterácií 92
9.2.3 Rozšírenia 93
9.2.4 Presnosť experimentov 93
ZÁVER 94
SEZNAM POUŽITÉ LITERATURY 95

ZOZNAM OBRÁZKOV 97
ZOZNAM TABULIEK 99
ZOZNAM PRÍLOH 100

ÚVOD

Mať možnosť pochopiť problém pri štúdiu inou formou ako len výkladom profesora či čítaním skrípt je pre študenta nesmiernou výhodou a navodzuje lepší pocit z učenia. V oblasti pravdepodobnosti nie je vždy jednoduché pochopiť danú problematiku bez praktickej ukážky. Práve tieto dôvody ma motivovali vybrať si prácu s cieľom vytvoriť študijnú pomôcku.

Problematikou vyčíslenia hodnoty Ludolfovho čísla pomocou Buffonovej úlohy o ihle sa zaoberajú matematici už niekoľko storočí. Od fyzických experimentov až po jednoduché simulácie či výpočty, sa riešenie úlohy posúva stále vpred. Bohužiaľ, v súčasnej dobe neboli k dispozícii žiadne adekvátne aplikácie nápomocné výučbe danej problematiky. Existujúce experimenty s presnosťou na jedno až dve desatinné miesta nevrhajú na validitu úlohy príliš dobré svetlo, pričom je úloha pomerne triviálna. Buffonova úloha o ihle poskytuje viaceré rozšírenia, a to hlavne čo sa týka výberu plochy, na ktorú sú ihly vrhané. Avšak dostupné aplikácie sa zaoberajú len základným riešením na plochu rovnobežiek.

Ako hlavný cieľ práce sme si stanovili vytvorenie aplikácie, ktorá bude pohodlná na obsluhu a prinesie používateľovi čo najlepšiu vizuálnu skúsenosť pri procese pochopenia problematiky. Ďalší, nemenej dôležitý cieľ, sme určili zvýšenie presnosti. Výpočtová technika je v dnešnej dobe na vysokej úrovni, a preto sme očakávali, že zvýšenie presnosti bude pomerne výrazné. Tretím cieľom, ktorý sme si stanovili, bolo rozšírenie základnej Buffonovej úlohy o možnosť výberu z viacerých plôch. Predpokladali sme, že rozšírenie na inú plochu by malo pomôcť k zvýšeniu presnosti.

I. TEORETICKÁ ČASŤ

1 ÚVOD DO TEORETICKEJ ČASŤI BAKALÁRSKEJ PRÁCE

V teoretickej časti našej bakalárskej práce sme najskôr podrobne popísali problematiku pravdepodobnosti, Ludolfovo číslo a metódu Monte Carlo. Následne sme priblížili čo je to Buffonova úloha o ihle a jej princíp. Ďalej sme ozrejmili riešenie úlohy na základnej ploche rovnobežiek a rozšírenia na ďalšie plochy. Poslednou časťou, ktorú sme v teoretickej časti bakalárskej práce rozoberali boli existujúce aplikácie, riešiace problematiku Buffonovej úlohy.

2 PRAVDEPODOBNOSŤ

V každodennom živote počujeme slovo pravdepodobnosť. Pre niekoho to môže byť len fráza označujúca šancu, s akou bude nasledujúci deň pršať alebo aká bude možnosť výhry v lotérií. Avšak za slovom pravdepodobnosť sa skrýva oveľa viac. Z matematického hľadiska je pravdepodobnosť číslo, okolo ktorého osciluje pomerná početnosť. Je dokonca známe, ako pravdepodobnosť javov určovať s vysokou, a v niektorých prípadoch aj so sto percentnou presnosťou. Pravdepodobnosť javu A je práve jedno číslo, javu priradené P(A). Číslo P(A) je z množiny od 0 po 1 [1].

V danej kapitole rozoberieme, čo je to geometrická pravdepodobnosť a aké je jej využitie. Následne sa bližšie zameriame na Ludolfovo číslo, metódy jeho vyčíslenia a princíp simulačnej metódy Monte Carlo.

2.1 Pojmy

Pre uvedenie do problematiky pravdepodobnosti považujeme za dôležité vysvetliť nasledujúce pojmy: experiment, náhodný, nemožný a istý jav, náhodné číslo, iterácia a graf závislosti jednej veličiny na druhej.

2.1.1 Experiment

Pre dokázanie hypotézy je nutné danú hypotézu otestovať. Činnosť overovania hypotéz sa nazýva experiment. Experimenty môžu hypotézu potvrdiť, vyvrátiť, ale aj skončiť s nejasným výsledkom. V prípade nejasného výsledku je nutné navrhnúť nový, efektívnejší experiment.

2.1.2 Náhodný jav

Výsledkom jedného pokusu experimentu je náhodný jav. Ako klasický príklad náhodného javu spomenieme hod hracou kockou. Jeden hod kockou, teda náhodný pokus, skončí náhodným javom, napríklad padnutím čísla 4. Výsledkom náhodného pokusu nemusí vždy byť len jeden náhodný jav. Pri padnutí čísla 4 je náhodným javom aj to, že padlo párne číslo. Základnou vlastnosťou náhodného javu je neurčitosť výsledku pred vykonaním pokusu.

2.1.3 Nemožný a istý jav

Na rozdiel od náhodného javu sa nemožný a istý jav vyznačujú určitosťou. Nemožný jav je taký, že nie je možné aby pri danom pokuse nastal. Príkladom nemožného javu je hodenie čísla 57 pri hode hracou kockou. Pravdepodobnosť nemožného javu je teda 0

Istý jav je opakom javu nemožného. Pri náhodnom pokuse nastane vždy. Ako príklad uvedieme pri hode kocky tento jav: hodíme číslo menšie ako 7. Pravdepodobnosť istého javu je vyjadrená číslom 1 pripadne ako 100%.

2.1.4 Náhodné číslo

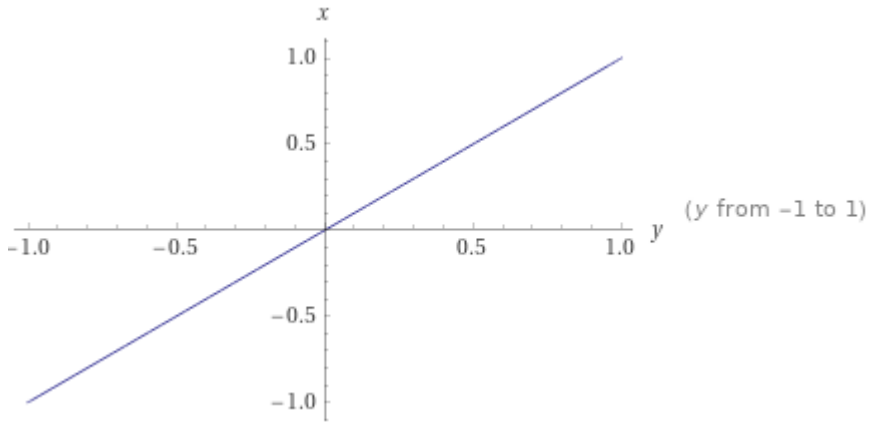
Náhodné číslo je podmnožinou náhodného javu. Pre overovanie hypotéz je často nutné pracovať s veľkým počtom dát. Generovanie náhodných čísel ako vstupov experimentov prispieva k presnejšiemu získaniu výsledku, ale môže prispieť aj k nájdeniu chyby.

2.1.5 Iterácia

Pri experimentálnom overovaní hypotéz sa pokusy experimentov môžu niekoľkokrát opakovať. Či už na samotné dokázanie hypotézy alebo na zlepšenie výsledku. Jedno opakovanie pokusu či sady pokusov sa nazýva iterácia. Cieľom experimentov je dosiahnuť čo najpresnejší výsledok pri najmenšom možnom počte iterácií.

2.1.6 Graf závislosti jednej veličiny na druhej

Graf je grafické zobrazenie vývoja výsledku skúmanej problematiky. Rôzne interpretácie, ako napríklad čiarový, koláčový či stĺpcový graf slúžia pre širokú škálu úloh. Pre najlepšiu vizualizáciu závislosti dvoch veličín sa používa čiarový graf. Skladá sa z dvoch na seba kolmých osí. Každá z osí interpretuje hodnoty jednej veličiny. Graf čítame tak, že veličina nanesená na osi y je závislá na veličine nanesenej na osi x. Jednoduchý graf závislosti x od y je možné vidieť na obrázku 1.



Obrázok 1 Graf závislosti x na y

2.2 Geometrická pravdepodobnosť

Geometrická pravdepodobnosť rozširuje klasickú pravdepodobnosť o náhradu množiny všetkých prípadov množinou ohraničenou predom určenými mierami. Medzi miery geometrickej pravdepodobnosti patria dĺžka, obsah či objem. Miery môžeme označovať obecným názvom Lebesqueova či Hausdorffova miera [2].

2.2.1 Historický vývoj geometrickej pravdepodobnosti

Prvý problém geometrickej pravdepodobnosti nastolil Isaac Newton. Modifikoval Huygensovu teóriu. Teória pracuje s pomermi častí kruhu a odvodzuje hodnotu očakávanej výhry. Newton však teóriu rozšíril pre prípad iracionálneho pomeru šancí na výhru. Newton uvažoval kruh s pomerom častí $2 : \sqrt{5}$ a guličku, ktorá padá smerom k stredu kruhu. Výsledkom je tvrdenie, že pomer obsahov výsekov kruhu je rovný pomeru pravdepodobností zásahu výsekov guličkou [2].

Ďalším významným problémom geometrickej pravdepodobnosti sú Buffonove úlohy. O nich si bližšie povieme v kapitole 3.

Za najväčších českých prispievateľov problematiky geometrickej pravdepodobnosti sa považujú Emanuel Czuber (1851 - 1925), Joseph Bertrand (1822 - 1900) či Bohuslav Hostinský (1884 -1951) [2].

2.3 Ludolfovo číslo

Π nazývané aj Ludolfovo číslo je jedna z najznámejších matematických konštánt. Hodnota konštanty vyjadruje pomer medzi obvodom a priemerom kružnice. Presnosť konštanty sa rokmi zlepšovala. V súčasnosti dokážeme pomocou moderných technológií vyčísliť hodnotu Ludolfovho čísla na 62.8 triliónov desatinných miest [3]. Avšak Pi je iracionálne číslo, čo znamená, že jeho presnú hodnotu nie je možné vyčísliť.

2.3.1 Ludolph van Ceulen

O pôvode jedného z najslávnejších nemeckých matematikov Ludolpha van Ceulena (1540 - 1610) svedčí už preklad jeho mena: Ludolf z Kolína, napriek tomu že sa narodil v nemeckom univerzitnom meste Hildesheim. Napriek veľkému nadaniu mu kvôli chudobe jeho rodiny nebolo umožnené študovať na univerzite. Bez potrebného vzdelania nedokázal prekladať texty z latinčiny a gréčtiny, a preto sa pri štúdiu textov musel spoliehať na priateľov [4].

Vrcholom jeho práce bolo vyčíslenie hodnoty Pi s presnosťou 35 desatinných miest. Pre aproximáciu použil pravidelný útvar, pri ktorom zvýšil, oproti jeho predchodcom, počet vrcholov z niekoľko tisíc až na hodnotu 2^{62}, čo odpovedá hodnote viac ako 4,5 trilióna. Aproximácia Pi bola jeho celoživotnou prácou. Do smrti publikoval výsledok hodnoty Pi „len“ na 20 desatinných miest v roku 1596. Po Ludolphovej smrti manželka zverejnila jeho posmrtnú prácu s presnosťou na 33 desatinných miest a v roku 1621 bola zverejnená kompletná aproximácia s presnosťou na 35 desatinných miest. Na Ludolphovom náhrobku je vytesaná hodnota jeho výsledku: 3.14159265358979323846264338327950288. Číslo Pi sa v odbornej literatúre zvykne nazývať Ludolfovo číslo [4].

2.3.2 Vývoj presnosti Ludolfovho čísla

Prvou motiváciou zlepšovania presnosti Pi bol vynález kolesa približne 6000 rokov pred naším letopočtom v Mezopotámii. Dôvodom bola nutnosť získania obvodu kolesa. Pre výpočet opísali a vpísali kružnici štvorec. Obvody týchto dvoch štvorcov sčítali a polovicu súčtu považovali za obvod kolesa. Hodnota bola o niečo väčšia ako 3, čo odpovedá Pi. Pre spresnenie používali neskôr šesťuholníky či dvanásťuholníky. Použitím šesťuholníka dostali hodnotu 3,125 [5].

Významným menom pri odhadovaní hodnoty čísla Pi bol aj grécky matematik a fyzik Archimedes zo Syrakúz. Pri jeho výpočte Pi Archimedes nahradil kružnicu pravidelným mnohouholníkom. Po opísaní a vpísaní mnohouholníka, v prvom prípade pravidelného

šesťuholníka, získal dve hodnoty. Tieto hodnoty slúžili ako interval, v ktorom sa Pi musí nachádzať. Pre spresnenie (zmenšenie) intervalu logicky zvyšoval počet uhlov, a teda aj strán mnohouholníka. Postupne vyskúšal 12, 24, 48 až 96 strán. Výsledkom bol interval $3{,}1408 < \pi < 3{,}1429$. Archimedes teda nikdy presnú hodnotu Pi neodhadoval. [5].

Od doby Archimeda (približne 200 pred našim letopočtom) až po prácu Ludolfa van Ceulena odhad hodnoty Pi výrazne nepokročil. Zvýšenie presnosti na 35 desatinných miest zásluhou Ludolfa bolo jedným z veľkých pokrokov.

S novodobými technológiami a výpočtovou silou počítačov je skok presnosti oproti rokom minulým neporovnateľný. Napriek tomu sa hodnota stále spresňuje. Avšak, ako je známe, presnú hodnotu nie je možné zistiť.

2.3.3 Metódy vyčíslenia Ludolfovho čísla

Pre vyčíslenie Ludolfovho čísla matematici používali rôzne metódy. V danej podkapitole spomenieme niektoré z metód. V minulosti sa na vyčíslenie hodnoty Pi pozeralo ako na geometrický problém. Postupom času matematici začali na konštantu Pi pozerať analytickým pohľadom. Či už pomocou radov, algoritmov či integrálov.

Jedným radom, ktorý spomenieme je Fourierov rad. Fourier zistil, že funkciu je možné zapísať ako rad pomocou sínusov a kosínusov. Rad má tvar zobrazený na nasledujúcej rovnici [5]:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right) \quad (1)$$

Dôležité sú koeficienty a_n a b_n nazývané aj Fourierove koeficienty. Pomocou koeficientov dokážeme potom vyjadriť hodnotu Pi. Medzi ďalšie rady použité na vyčíslenie hodnoty Pi patria: Leibnizov rad či Ramanujanov rad. Ďalšími dostupnými riešeniami sú napríklad algoritmus Spigot či riešenie pomocou Gausovho integrálu [5].

Štatistické metódy používané v danej problematike sú Buffonova úloha a metóda Monte Carlo, ktoré popíšeme nižšie v texte.

2.4 Metóda Monte Carlo

Metóda Monte Carlo je simulačnou metódou skúmajúcou geometrickú pravdepodobnosť. Predchodcom metódy je Buffonova úloha o ihle, ktorú podrobne popíšeme v kapitole 3. Pomocou metódy je možné experimentálne vyčísliť hodnotu Pi. Princíp experimentu je pomerne jednoduchý. V súčasnej dobe sa pre spresnenie používajú rôzne algoritmy, a keďže je výpočtová technika na vysokej úrovni, je možné vykonať čoraz väčšie množstvo experimentov. Avšak v minulosti matematikom stačil len štvorcový papier. Do jeho stredu nakreslili kruh. Opísaním kružnice vznikol štvorec s dĺžkou strany rovnajúcou sa priemeru kruhu. Následne sa do štvorca náhodne vpisovali body. Bod buď ležal vo vnútri kruhu alebo mimo neho. Pravdepodobnosť, že bod sa nachádzal v kružnici potom vyjadrili ako pomer počtu bodov vnútri a počtu bodov mimo kruhu. Tento pomer je vyjadrený nasledujúcou rovnicou [1]:

$$P = \frac{počet\ úspešných}{počet\ všetkých} = \frac{\pi r^2}{(2r)^2} \tag{2}$$

Kde počet úspešných bodov je vyjadrený ako obsah kruhu a počet všetkých ako obsah štvorca s veľkosťou 2 krát polomer kruhu, teda priemer. Po úprave dostávame rovnicu:

$$P = \frac{\pi}{4} \tag{3}$$

Hodnota Pi je potom štvornásobkom pravdepodobnosti, že bod sa nachádza v kruhu [1].

3 BUFFONOVA ÚLOHA O IHLE

V tretej kapitole našej bakalárskej práce si podrobne popíšeme Buffonovu úlohu o ihle. Rozoberieme princíp úlohy, historické míľniky, základné riešenie na rovnobežkách, rozšírenia na obdĺžnikovú mriežku a rovnostranný trojuholník.

3.1 Georges-Louis Leclerc, Comte de Buffon

Vlastným menom Goerges Buffon (1707 - 1788) bol najstarším z piatich súrodencov. Goerges pochádzal z bohatej rodiny. Jeho matka, príbuzná bohatého bankára, zdedila veľké množstvo peňazí a to umožnilo Goergesovmu otcovi nadobudnúť titul lord Buffonu a Montbard. Po presťahovaní do mesta Dijon začal George študovať na univerzite. Snom Georgovho otca bolo, aby sa Goerge stal právnikom. Avšak George prejavoval viac nadanie na matematiku. Vo veku 20 rokov objavil teorém nazývaný binomická veta. Binomická veta je dôležitá v oblasti kombinatoriky, no využíva sa aj v iných odvetviach ako napríklad vo fyzike. Najväčším príspevkom do rozvoja matematiky bol experiment skúmajúci pravdepodobnosť s akou po hodení paličiek na vydláždenú podlahu dopadne ihla medzi dlaždice. Experiment pomenovaný podľa Georgesa, Buffonova úloha o ihle, pomohol rozvoju chápaniu problematiky pravdepodobnosti. Pomocou experimentu je možné vyčísliť hodnotu Ludolfovho čísla [6].

3.2 Rovnobežky

Problém Buffonovej ihly nadväzuje na experiment hádzania paličiek na vydláždenú plochu. Paličky sú nahradené ihlami o konštantnej veľkosti l a dlaždica je nahradená rovnobežkami s nekonečnou dĺžkou vzdialenými od seba konštantnú vzdialenosť h. Buffon experimentálne zistil, že pre ihly splňujúce podmienku $l < h$ je pravdepodobnosť pretnutia niektorej z rovnobežiek vyjadrená nasledujúcou rovnicou [7] :

$$p = \frac{2l}{\pi h} \tag{4}$$

Z pravdepodobnosti (vyjadrenej pomerom počtu pretnutí n a počtom všetkých ihiel N) je potom možné vyjadriť hodnotu π rovnicou:

$$\pi = \frac{Nh}{n2l}$$

(5)

Pozícia ihly je jasne určená vzdialenosťou jej stredu od najbližšej rovnobežky a uhlom α medzi ihlou a rovnobežkou [7].

3.3 Rozšírenia Buffonovej úlohy

V nasledujúcej podkapitole priblížime rozšírenia Buffonovej úlohy na obdĺžnikovú mriežku a na rovnostranný trojuholník

3.3.1 Obdĺžniková mriežka

Samotný Buffon sa snažil rozširovať svoju úlohu na viaceré plochy. Ako uvádza Hledík [8], v Buffonovom diele sa nachádza chyba pri pokuse rozšíriť úlohu z rovnobežiek na obdĺžnikovú mriežku. Toto rozšírenie sa v správnej podobe nachádza v diele Laplacea o takmer 50 rokov neskôr. Preto sa toto rozšírenie zvykne označovať aj ako Laplaceovo rozšírenie na obdĺžnikovú mriežku.

Princípom rozšírenia je hádzanie ihly na plochu tvorenú obdĺžnikovou mriežkou. Obdĺžniky s konštantnou veľkosťou strán (*a , b*). Ihla je jasne určená pozíciou stredu v závislosti od zvoleného vrcholu obdĺžnika a uhlom α, ktorý ihla zviera s kratšou stranou obdĺžnika [8].

Pre jednoduchosť riešenia predpokladáme veľkosť ihly *l:* $0 < l \leq b$, kde b je veľkosť väčšej strany obdĺžnika. Pravdepodobnosť pretnutia ihly rozdelíme na viac častí. Obdĺžniková mriežka sa skladá z vertikálnych a horizontálnych rovnobežiek. Pravdepodobnosť pretnutia rovnobežky sme určili v podkapitole 3.2. Pravdepodobnosť pretnutia vertikálnej P_A a pravdepodobnosť pretnutia horizontálnej P_B rovnobežky vyjadríme nasledovne [8]:

$$P_A = \frac{2l}{\pi a}$$

(6)

$$P_B = \frac{2l}{\pi b}$$

(7)

Avšak treba brať do úvahy aj prípad, keď ihla pretne vertikálnu aj horizontálnu rovnobežku súčasne (obe strany obdĺžnika) P_{AB}. Výsledný vzťah pre výpočet pravdepodobnosti vyjadríme rovnicou [8]:

$$P = P_A + P_B - P_{AB} \tag{8}$$

Body pretínajúce obe strany obdĺžnika sa musia nachádzať v blízkosti jeho vrcholov. Konkrétne v obdĺžniku so stranami veľkosti *1/2sin(α)* a *1/2cos(α)*. Pravdepodobnosť P_{AB} vyjadríme rovnicou [8]:

$$P_{AB} = \frac{l^2}{\pi ab} \tag{9}$$

Po dosadení do rovnice 7 dostávame výslednú pravdepodobnosť pretnutia ihly pri obmedzení $0 < l \leq a$ [8]:

$$P = \frac{1}{\pi ab}(2al + 2bl - l^2) \tag{10}$$

Vyjadrenie pravdepodobnosti pre ihly v intervale (a, b> je zložitejšie, a preto uvedieme len výsledok [8]:

$$P = \frac{1}{\pi ab}\left(a^2 + 2bl - 2\sqrt{l^2 - a^2} + 2ab \arccos\left(\frac{a}{l}\right)\right) \tag{11}$$

Z rovnice pravdepodobností rovnako ako aj pri úlohe o rovnobežkách dokážeme vyjadriť hodnotu Ludolfovho čísla.

3.3.2 Rovnostranný trojuholník

Pre rozšírenie rovnostranného trojuholníka najprv definujeme veľkosť ihly *l* voči strane trojuholníka *a*. Ihla s dĺžkou *l* rovnakou alebo väčšou ako strana *a* s určitosťou pretne aspoň jednu stranu trojuholníka.

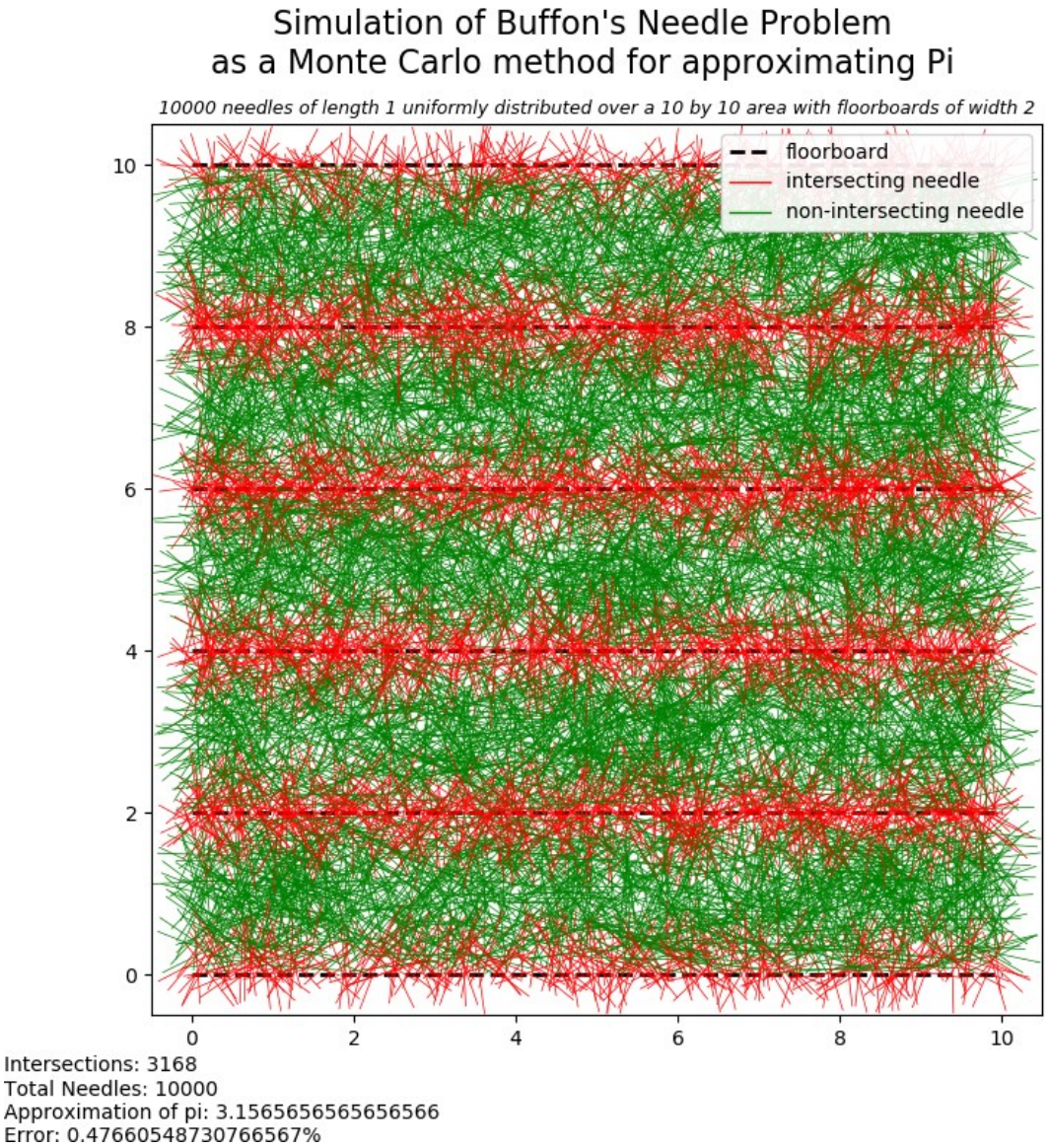
Na rozdiel od prístupu v podkapitole 3.3.1 nájdeme vo vnútri trojuholníka plochu takú, že stredy ihiel umiestnené v tejto ploche s určitosťou nepretnú žiadnu zo strán rovnostranného

trojuholníka. Z obsahu plochy dostávame pravdepodobnosť nepretnutia, z ktorej dokážeme vyjadriť hodnotu Ludolfovho čísla. Pravdepodobnosť vyjadríme pre ihly menšie ako ťažnica trojuholníka rovnicou [8]:

$$P_{nepretnutia} = \frac{\pi(3a^2 + 2l^2) + 3\sqrt{3}l(l - 4a)}{3\pi a^2} \quad (12)$$

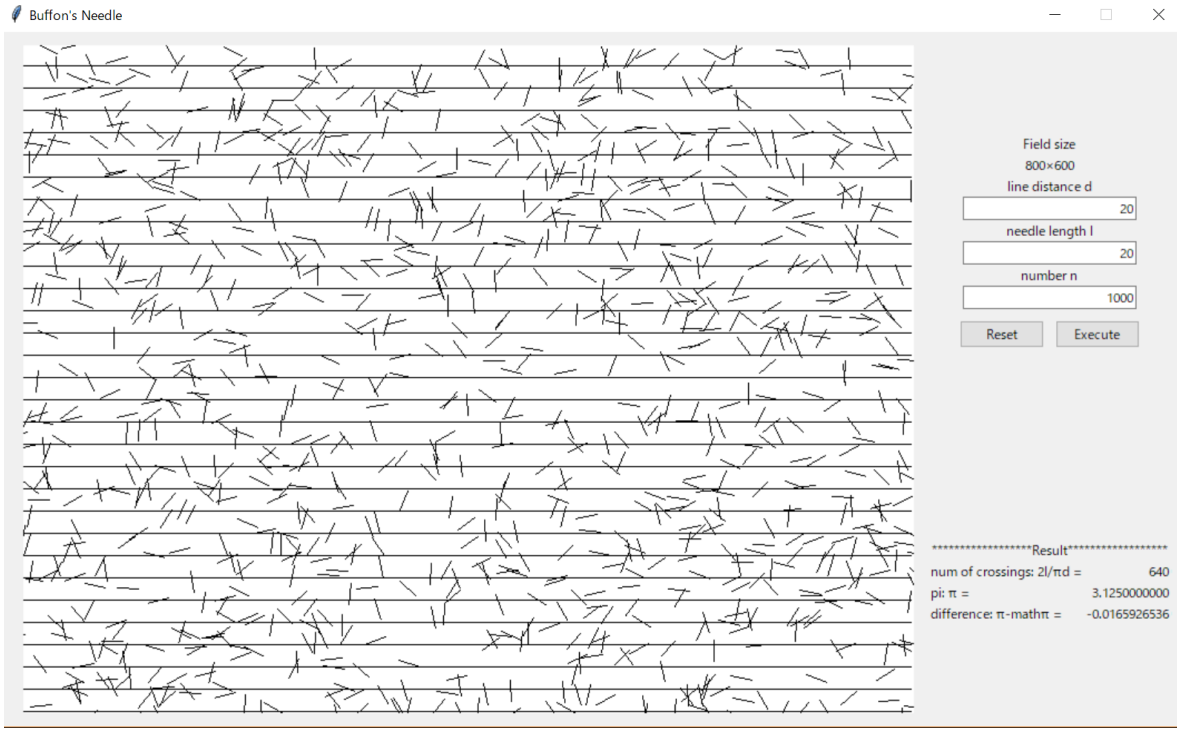
4 DOSTUPNÉ RIEŠENIA BUFFONOVEJ ÚLOHY O IHLE

Simulačné aplikácie sa vo veľkej miere implementujú v programovacom jazyku **python**. Dôvodom je knižnica **numpy,** poskytujúca rôzne matematické funkcie a objekty. V jazyku python bola implementovaná aj aplikácia [9] verejne prístupná na úložiskovom serveri **GitHub**. Aplikácia je založená na simulačnej metóde **Monte Carlo**. Aplikácia najskôr „hodí“ všetky ihly na plochu, a potom vykreslí výsledok s odhadnutou hodnotou Pi. Parametre ako veľkosť ihly či počet ihiel nie sú nastaviteľné bez úpravy zdrojového kódu. Na obrázku 2 je možné vidieť výsledok riešenia. Presnosť dosahuje takmer 2 desatinné miesta. Podobnou je aj aplikácia od Dahma [10], rovnako založená na modeli Monte Carlo.



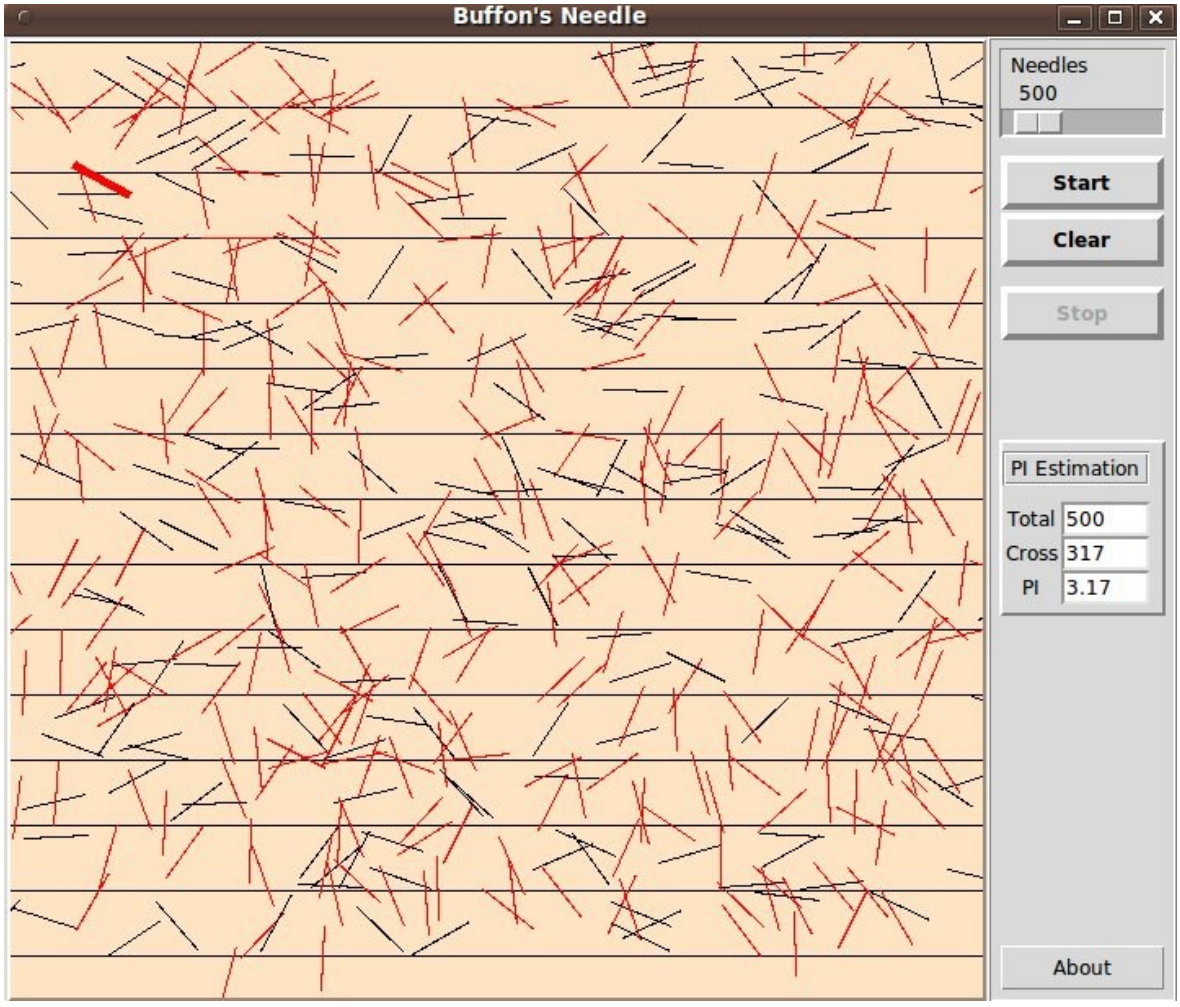
Obrázok 2 Výsledok experimentu pomocou Monte Carlo simulácie [9]

Nevýhodou implementácie v pythone je limitovaná grafická stránka. Druhá aplikácia, rovnako implementovaná v jazyku python [11], na rozdiel od prvej obsahuje možnosť zvolenia parametrov priamo na obrazovke. Avšak grafická stránka aplikácie je veľmi prostá. Snímku obrazovky danej aplikácie je možné vidieť na obrázku 3. Po nastavení parametrov vzdialenosti rovnobežiek, veľkosti ihly a počtu ihiel sa po odštartovaní vykreslia všetky ihly a výsledkom experimentu je hodnota Pi spolu s rozdielom od skutočnej hodnoty.

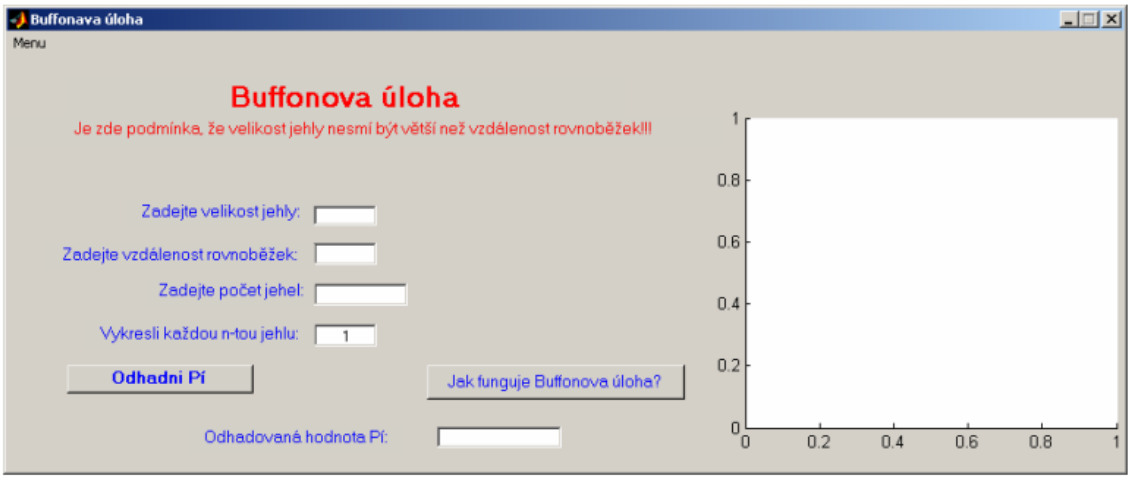


Obrázok 3 Aplikácia pre výpočet Pi v jazyku python [11]

Užívateľsky prívetivejšie, no aj tak pomerne jednoduché na dizajn, sú aplikácia od Vettera [12], implementovaná v jazyku **Tool Command Language** a aplikácia od Daníčka [13], implementovaná v **Matlabe**. Rovnako ako predchádzajúca aplikácia, ponúkajú aj tieto možnosť nastavenia parametrov veľkosti ihly, počtu ihiel a vzdialenosti rovnobežiek. Na obrázkoch 4 a 5 môžeme porovnať grafickú podobu aplikácií. Aplikácie boli vytvorené v rokoch 2003 a 2005, a tomu aj odpovedajú použité technológie. Oproti predchádzajúcim aplikáciám je pridaná možnosť zobrazenia informácií o fungovaní Buffonovej úlohy. Presnosť výsledku je opäť maximálne 2 desatinné miesta.



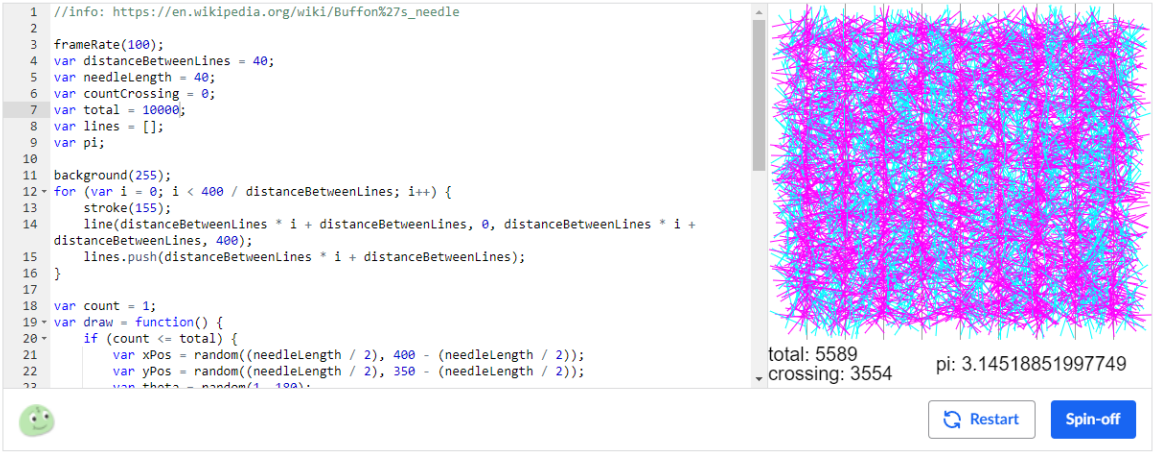
Obrázok 4 Aplikácia v jazyku Tool Command Language [12]



Obrázok 5 Aplikácia v Matlabe [13]

Poslednou popisovanou aplikáciou je aplikácia od Martineza [14]. Ukážka výstupu aplikácie sa nachádza na obrázku 6. Na rozdiel od predchádzajúcich štyroch aplikácií, dokáže vykresľovať ihly postupne s navodením dojmu animácie dopadu. Výsledkom je hodnota Pi

meniaca sa po každej vrhnutej ihle. Táto aplikácia je zo všetkých popisovaných najpresnejšia. Presnosť výsledku voči skutočnej hodnote Ludolfovho čísla je pri zvolených parametroch veľkosti ihly = 40 a vzdialenosti rovnobežiek = 40 v rozmedzí 2-3 desatinných miest.



Obrázok 6 Aplikácia s animáciou [14]

II. PRAKTICKÁ ČASŤ

5 ÚVOD DO PRAKTICKEJ ČASTI BAKALÁRSKEJ PRÁCE

V praktickej časti našej bakalárskej práce sme najskôr podrobne popísali návrh aplikácie spolu s použitými nástrojmi pre jej realizáciu a základnú štruktúru zdrojového kódu. Následne sme sa zamerali na popis samotnej aplikácie, ktorú sme rozdelili do dvoch častí: grafická časť a časť výpočtov. V grafickej časti sme sa venovali popisu použitých komponentov na vykreslenie obrazoviek, animácií a grafov. V časti výpočtov sme detailne priblížili spôsob implementácie výpočtu Ludolfovho čísla pomocou Buffonovej úlohy o ihle. Po nastolení problematiky implementácie aplikácie sme pristúpili s vykonaniu experimentov. Experimenty sme realizovali s rôznymi parametrami a na rôznych plochách pre dosiahnutie čo najlepšieho výsledku. V poslednej časti praktickej časti bakalárskej práce sme zhodnotili výsledky vykonaných experimentov a porovnali sme aplikáciu s existujúcimi riešeniami danej problematiky na základe grafickej, výpočtovej a funkcionálnej stránky.

6 NÁVRH APLIKÁCIE

V šiestej kapitole podrobne popíšeme použitý framework, štruktúru zdrojového kódu, jej jednotlivé časti a samotný návrh aplikácie podľa zadania.

6.1 Qt framework

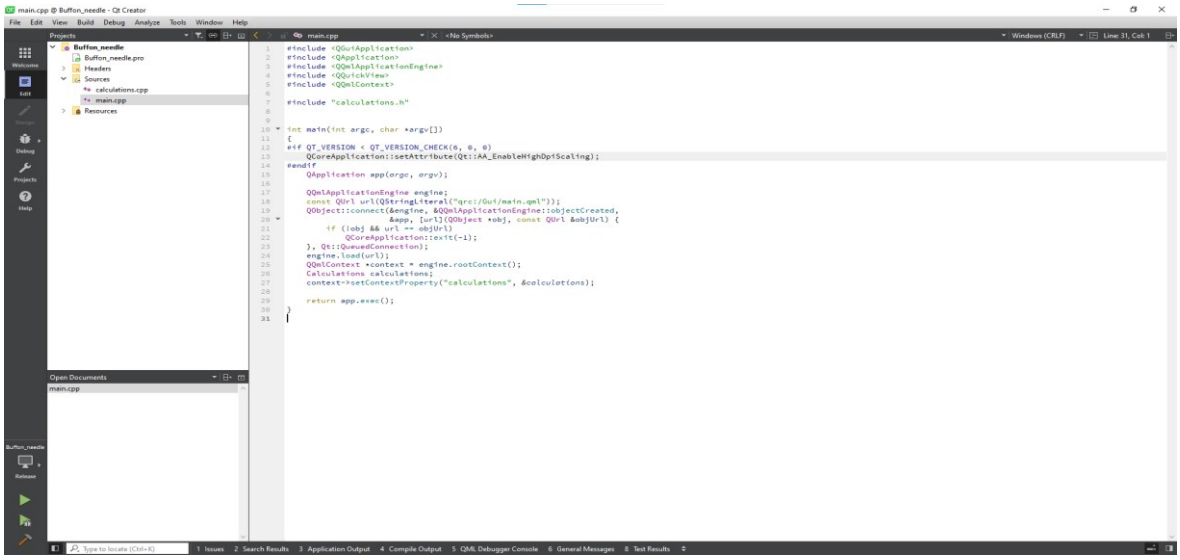
„*Qt je multiplatformový framework pre vývoj aplikácií na stolných počítačoch, vstavaných a mobilných zariadeniach*“ [15]. Hlavnou črtou Qt je jeho preprocesor. Vďaka nemu je možné projekty implementované v tomto frameworku premeniť na natívny c++ kód, kompilovateľný na všetkých podporovaných systémoch. [15]

Z historického hľadiska je Qt pomerne moderné. Vývoj začal v roku 1990 vo firme Trolltech. V súčasnosti sa firma nazýva **The Qt Company** [15]. Základnou politikou organizácie je filozofia „každý, kto chce, môže prispieť“. Ako prínosná kontribúcia sa považuje písanie samotného kódu, dokumentácie či reportovanie chýb (bugov) [16].

Súčasťou inštalačného balíka Qt sú aj tutoriály. Problematika rozoberaná v tutoriáloch má náučný charakter a pomáha užívateľom lepšie pochopiť zdrojový kód a jeho použitie.

6.1.1 Qt Creator

Qt framework disponuje vlastným editorom zdrojového kódu. Qt Creator je použiteľný, rovnako ako aj samotný framework, multiplatformovo. Editor umožňuje zdrojový kód kompilovať, vykonať a ladiť. Ukážka grafického rozhrania editoru je zobrazená na obrázku 7.



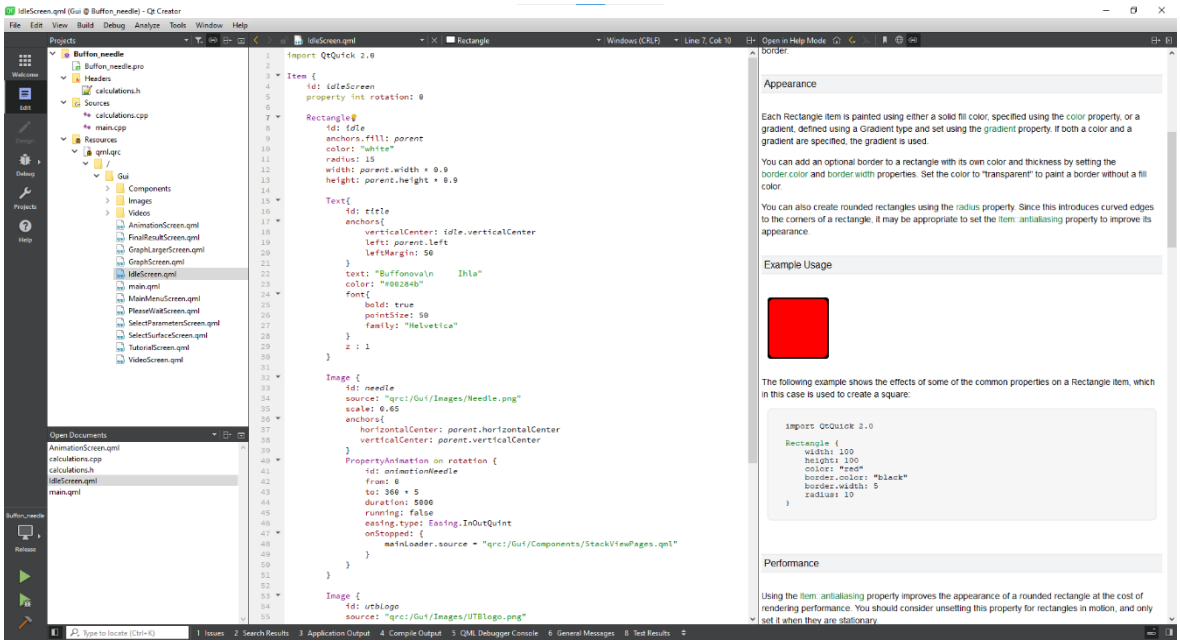
Obrázok 7 Grafické rozhranie Qt Creatoru

6.1.2 QML

Qt framework ponúka dva prístupy implementácie grafického užívateľského rozhrania:

- GUI pomocou QWidgetov – modelovanie objektov na obrazovku v grafickom editore
- GUI pomocou jazyka QML – modelovanie objektov v jazyku, založenom na jazyku javaScript so syntaxou podobnou JSONu

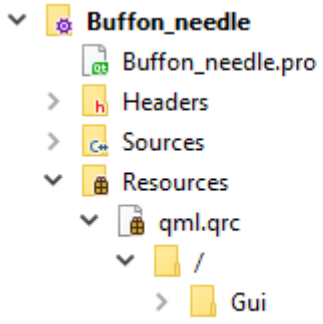
V aplikácii sme vybrali druhý z vyššie uvedených prístupov. Výhodou QML je jeho vysoká čitateľnosť a prehľadnosť. Umožňuje vytvárať graficky prívetivé aplikácie s mnohými možnosťami prispôsobenia. Veľkou výhodou jazyka je aj jeho detailná dokumentácia, od detailného popisu funkcionality jednotlivých komponentov, až po konkrétne ukážky použitia v kóde. Pre prístup k dokumentácii komponentu slúžia webové stránky [17]. Po stlačení klávesy F1 v zdrojovom kóde editora Qt Creator, je možné dokumentáciu daného komponentu zobraziť priamo v editore. Na obrázku 8 je na pravej strane editora možné vidieť dokumentáciu objektu **Recangle**, konkrétne časť popisujúcu vzhľad a príklad použitia v zdrojovom kóde.



Obrázok 8 Dokumentácia objektu Rectangle v Qt Creatore

6.2 Štruktúra zdrojového kódu

Zdrojový kód v Qt aplikáciách je založený na princípe stromovej štruktúry definovanej v projektovom **.pro** súbore, ktorý sa nachádza v každej Qt aplikácii. Príklad takéhoto stromu je zobrazený na obrázku 9. Tri základné pod-stromy sú **Headers** (časť hlavičkových súborov), **Sources** (časť zdrojových súborov) a **Resources** (časť zdrojov).



Obrázok 9 Štruktúra zdrojového kódu

6.2.1 Hlavičkové súbory

Hlavičkové súbory slúžia na deklaráciu premenných, funkcií, štruktúr či tried. Deklarácia sa pri programovaní používa z dôvodu možnosti použitia (volania) deklarovaného objektu

v ľubovoľnej časti zdrojového kódu. Deklarovať je možné aj priamo v zdrojovom kóde, ale kvôli prehľadnosti boli zavedené hlavičkové súbory. Pre sprístupnenie obsahu hlavičkového súboru na ktoromkoľvek mieste kódu je nutné hlavičkový súbor zahrnúť v danom súbore *#include „názov.h“*.

Samostatnou časťou sú pri prístupnosti **triedy**. Trieda je údajový typ využívaný v objektovo orientovanom programovaní, ktorý si zadefinuje užívateľ podľa svojich potrieb. Môže obsahovať ľubovoľný počet premenných, nazývaných inštancie triedy a funkcií, nazývaných metódy. Metódy, ktoré nijak nemodifikujú inštancie triedy sa zvyknú deklarovať ako konštantné pomocou kľúčového slova **const**.

Triedy môžu byť medzi sebou prepojené. Spôsob prepojenia medzi triedami sa nazýva **dedičnosť**. Dcérska trieda „dedí“ všetky prístupné atribúty a metódy od nadradenej rodičovskej triedy a je schopná ich používať ako vlastné. V prípade, že by dcérska trieda potrebovala metódu z rodičovskej triedy modifikovať, musí byť táto metóda virtuálna. Dcérska trieda ju potom dokáže modifikovať pomocou kľúčového slova **override,** použitého v deklarácii metódy.

Nie vždy je žiadané aby boli všetky inštancie a metódy triedy použiteľné v ktorejkoľvek časti zdrojového kódu. Pre každú triedu existujú preto tri druhy prístupnosti k jej obsahu:

1. **Public** – neobmedzená prístupnosť
2. **Protected** – inštancie a metódy sú prístupné len pre samotnú triedu, pre jej dcérske triedy a všetky triedy, ktoré majú na vrchole stromu dedičnosti práve túto triedu
3. **Private** – s privátnymi inštanciami a metódami dokáže pracovať len samotná trieda

Je zvykom všetky inštancie deklarovať ako privátne. Takto sa zamedzí nechceným prepisom. Pre prístup k inštanciám tried sa používajú **gettery** a **settery.** Getter je konštantná, verejná (public) metóda triedy, ktorej jedinou funkciou je vrátiť hodnotu inštancie v mieste kódu kde je volaná. Setter je verejná (public) metóda triedy, avšak už nie konštantná, keďže modifikuje inštanciu triedy. Parametrom settera je nová hodnota, ktorá sa v setteri uloží do inštancie.

Špeciálnymi metódami triedy sú v Qt **signal** a **slot**. Slúžia na komunikáciu medzi dvomi alebo viacerými objektami. Objekt 1 po vykonaní časti svojho kódu dáva signál ostatným pripojeným objektom, že sa tento kód vykonal. Následne pripojené objekty vykonajú svoj kód, ktorý je v slote. Pre odoslanie signálu je nutné zavolať signál s kľúčovým slovom **emit**.

Na obrázkoch 10 a 11 sa nachádza kód komunikácie medzi objektom 1 z c++ a objektom 2 z QML. Zaužívaným formátovaním je pomenovať slot spôsobom **on+Signal**.

```
void Object1::someFunction()
{
    //code

    emit object1Signal();
}
```

Obrázok 10 Emit signálu

```
Connections{
    target: object1
    function onObject1Signal(){
        //code
    }
}
```

Obrázok 11 Prepojenie slotu so signálom

Metódy c++ triedy je možné sprístupniť aj pre QML časť aplikácie s použitím **Q_INVOKABLE** pred deklaráciou metódy. Na obrázku 12 je ukážka deklarácie triedy, ktorá dedí z rodičovskej triedy QObject.

```
#ifndef TRIEDA_H
#define TRIEDA_H

#include <QObject>

class Trieda : public QObject
{
    Q_OBJECT
public:
    explicit Trieda(QObject *parent = nullptr);

    int premenna() const;
    void setPremenna(int newPremenna);

    Q_INVOKABLE bool jePremennaKladna() const;

public slots:
    void onInySignal();
signals:
    void signal1();

private:
    void funkcia();
    int premenna_ = 0;
};

#endif // TRIEDA_H
```

Obrázok 12 Ukážková trieda

Pre použitie triedy a jej metód je nutné triedu vytvoriť. V Qt je možnosť použiť explicitný konštruktor alebo si vytvoriť svoj vlastný. Triedu potom možno vytvoriť ako premennú *Trieda trieda;* a k jej metódam pristupovať pomocou bodkovej notácie *trieda.premenna();*

6.2.2 Zdrojové súbory

Zdrojové súbory slúžia na definíciu zadeklarovaných funkcií. Definícia funkcie obsahuje samotný kód, ktorý funkcia vykonáva. Spracúva vstupné parametre a vracia návratovú hodnotu vzhľadom na typ funkcie. Špeciálnym typom zdrojového súboru je hlavný súbor pre beh programu **Main.**

Main.cpp je základným súborom, v ktorom sa štartuje aplikácia. Spracúva vstupné parametre, zadané pri spúšťaní. Následne zavolá prvú funkciu, štartujúcu aplikáciu. Pre QGui aplikácie je v main funkcii volaná funkcia *exec();*. Kód funkcie main QGui aplikácie, vygenerovanej v QtCreatore po založení novej aplikácie sa nachádza na obrázku 13. Vytvorí sa prepojenie na QML časť a spustí sa celá aplikácia.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>


int main(int argc, char *argv[])
{
#if QT_VERSION < QT_VERSION_CHECK(6, 0, 0)
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    const QUrl url(QStringLiteral("qrc:/main.qml"));
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                     &app, [url](QObject *obj, const QUrl &objUrl) {
        if (!obj && url == objUrl)
            QCoreApplication::exit(-1);
    }, Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

Obrázok 13 Funkcia main pre QGui aplikácie

6.2.3 Časť zdrojov

V časti zdrojov sa nachádzajú všetky GUI časti kódu. Dôležitým súborom je súbor **res.qrc**. V tomto súbore sú uložené všetky použité zdroje. V adresárovej štruktúre operačného

systému sú zdroje v zložke /GUI. Pre prehľadnosť sa zvyknú jednotlivé typy zdrojov rozdeľovať do ďalších podadresárov, napríklad adresár pre obrázky či videá. Východiskovým súborom je **main.qml**. Každej obrazovke prislúcha vlastný .qml súbor. Komponenty, ako napríklad vlastné tlačidlá, majú rovnako svoj unikátny .qml súbor. V jednotlivých .qml súboroch sa pridávajú podľa potreby objekty, funkcie a premenné v jazyku založenom na JavaScripte.

Main.qml definuje základné okno aplikácie. Obrázok 14 popisuje obsah súboru main.qml. Okno je definované objektom **Window.** Window má viacero nastaviteľných vlastností. V stave po vygenerovaní zdrojového kódu, sú to len vlastnosti šírka, výška, viditeľnosť a nadpis, ale veľmi jednoducho sa môžu modifikovať a pridávať aj ďalšie.

```
import QtQuick 2.15
import QtQuick.Window 2.15

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello World")
}
```

Obrázok 14 Main.qml

6.2.4 Prepojenie c++ časti s qml časťou

Pre optimálnu funkčnosť Qt aplikácie je žiadané, aby v QML časti bolo možné pristupovať k inštanciám z časti c++ zdrojového kódu. Ak je potrebné sprístupniť celú triedu pre QML, je to možné poslaním adresy triedy do QML ako **contextProperty**. Poslanie adresy a nie samotnej triedy zaručí, že po zmene v triede sa táto zmena prejaví aj v QML súboroch. Ukážka zdrojového kódu nastavenia contextProperty je zobrazená na obrázku 15.

```
QQmlContext *context = engine.rootContext();
Trieda trieda;
context->setContextProperty("trieda", &trieda);
```

Obrázok 15 Sprístupnenie triedy pre qml súbory

V c++ kóde je zvykom používať rôzne štruktúry, vektory a podobne. Avšak jazyk QML nedisponuje danými dátovými typmi. Vhodnou náhradou je dátový typ **QVariant**. Je ho možné použiť v c++ časti a zároveň aj v QML časti kódu.

6.3 Návrh aplikácie podľa zadania

Prvotným kritériom návrhu bola jednoduchosť. Aplikáciu musí zvládnuť a ovládať aj človek, ktorý ju vidí a skúša po prvýkrát. Na každej obrazovke by malo byť len to, čo je relevantné.

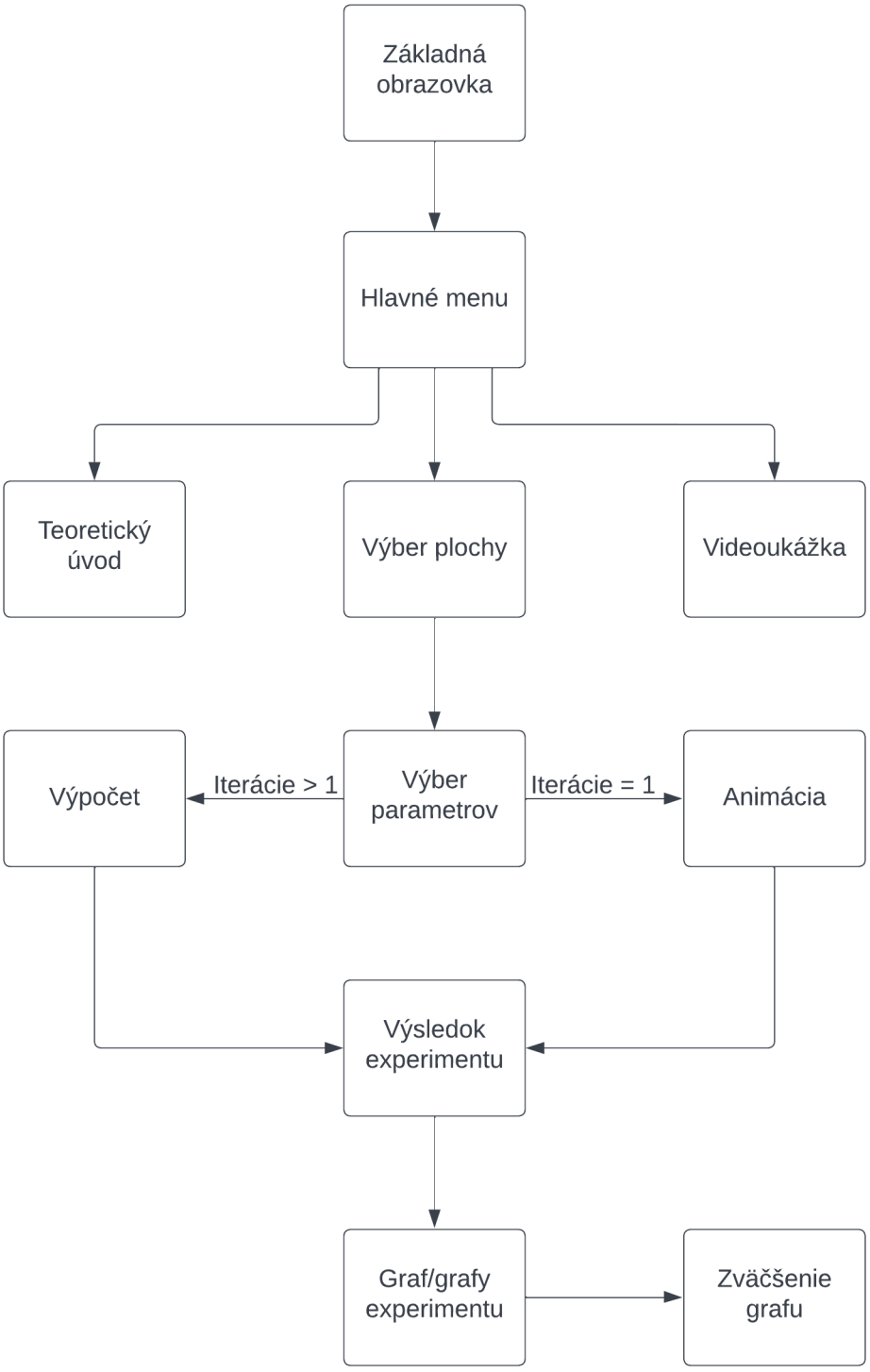
6.3.1 Flow diagram

Pred samotnou implementáciou obrazoviek sme vypracovali flow diagram ako je znázornené na obrázku 16. Každá obrazovka plní jedinečnú funkciu. Pri výbere parametrov sa podľa zvoleného počtu iterácií aplikácia rozhodne, ktorou vetvou diagramu bude pokračovať. Prechody medzi jednotlivými obrazovkami sme navrhli tak, aby boli intuitívne a ľahko pochopiteľné. Z každej obrazovky je možné zrušiť experiment a pri tých, kde je to logicky vhodné, je možné sa vrátiť o krok späť. Základná obrazovka slúži ako variant šetriča obrazovky. Späť do nej sa užívateľ dostane len z hlavného menu po stlačení tlačidla „Zzz“.

6.3.2 Farby, písmo a motív

Aplikácia Buffonovej úlohy o ihle má mať hlavne edukačný účel. Z daného dôvodu sme vybrali len bielu a modrú farbu pre vzhľad aplikácie, a pre ihly farbu zelenú, označujúcu ihlu pretínajúcu plochu a farbu červenú, ktorá označuje ihlu nepretínajúcu plochu.

Ako rodinu písma sme zvolili Helveticu. Všetky slová okrem teoretického úvodu začínajú v aplikácii veľkým začiatočným písmenom, a je pre nich nastavený parameter **bold** - tučný. Pre prechody medzi obrazovkami sme použili tlačidlá. Tlačidlá sme navrhli tak, aby bolo užívateľom jasne viditeľné, že tlačidlo bolo stlačené. Tlačidlo, ktoré nie je v aplikácii povolené je odlíšené od aktívnych sivou farbou. Pre nastavenie hodnoty číselných parametrov sme zvolili jednoduchý posúvač.



Obrázok 16 Flow diagram aplikácie

7 IMPLEMENTÁCIA

V siedmej kapitole sme podrobne popísali implementáciu aplikácie, ktorá experimentálnym spôsobom overuje Buffonovu úlohu o ihle slúžiacu na aproximáciu Ludolfovho čísla. Dvomi hlavnými časťami sú implementácia grafického užívateľského rozhrania **GUI** a samotná **výpočtová** časť. V krátkosti spomenieme aj vytvorený teoretický úvod a videoukážku experimentu, ktorá je súčasťou aplikácie.

Na záver tejto kapitoly priblížime spôsob, akým sa aplikácia prekladá, spúšťa a jej prenositeľnosť na iné zariadenia.

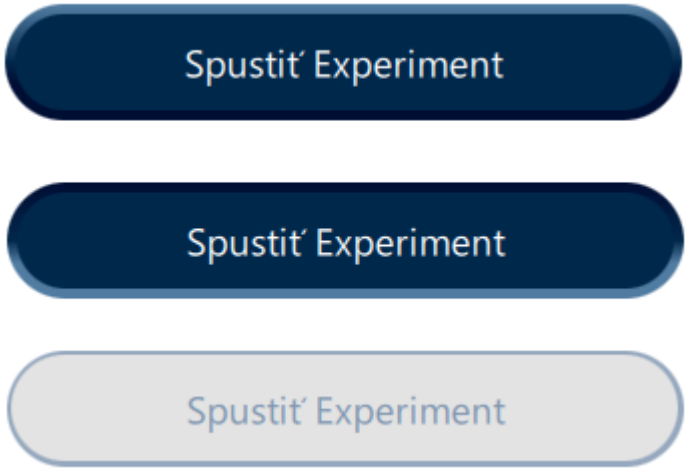
7.1 GUI

Gui časť aplikácie sme implementovali v jazyku **QML**. Jazyk je založený na JavaScripte. Jednotlivé komponenty je možné samostatne implementovať a následne ich vkladať na obrazovku. Umiestnenie je tiež konfigurovateľné a môže sa odvíjať od samotnej obrazovky alebo od ostatných komponentov na obrazovke.

7.1.1 Vlastné nastaviteľné tlačidlo

Prvým komponentom, ktorý sme vytvorili, bolo vlastné nastaviteľné tlačidlo. Tlačidlá sú použité na takmer každej obrazovke, a preto konfigurovateľnosť práve tohto komponentu bola jednou z priorít na začiatku implementácie. Základom väčšiny komponentov v QML je obdĺžnik **Rectangle**. Inak tomu nie je ani pri tlačidle. Výhodou obdĺžnika je jeho všestrannosť. Pomocou neho je možné vytvoriť čiary alebo aj kruhy. Na obrázku 17 sa nachádza tlačidlo v aktivovanom, stlačenom a zakázanom stave. Pre účely tlačidla sme použili dva obdĺžniky. Prvý, tvoriaci okraj a druhý, o niečo menší, pre vytvorenie 3D vzhľadu. Oblé hrany sú nastavené vlastnosťou obdĺžnika **radius**, ktorú sme zvolili pre tlačidlá na hodnotu 40. Pre dotvorenie vzhľadu je tlačidlo vo viacerých odtieňoch modrej farby. Vlastnosť **gradient** umožňuje nastavovať rozličné farby na zvolených pozíciách objektu. V tomto prípade je farba odlišná na začiatku, v polovici a od 7/10 obdĺžnika. V strede tlačidla sa nachádza text, objekt **Text**. Veľkosť textu závisí od veľkosti tlačidla. Užitočná vlastnosť textu **fontSizeMode** nastavená na **Text:HorizontalFit,** zabezpečí automatické zmenšenie či zväčšenie textu vzhľadom na veľkosť tlačidla.

Základnou vlastnosťou tlačidla je kliknutie. Táto vlastnosť je splnená aj pre vlastné tlačidlo. V prvom rade je nutné určiť, na akú plochu je kliknutie možné. Na určenie plochy slúži objekt **MouseArea**. Objektu MouseArea je následne pridelené ukotvenie na vyplnenie takzvaného rodiča, ktorým je v tomto prípade okrajový obdĺžnik tlačidla. Pre ukotvenie sa používa vlastnosť **anchors**, s parametrom vyplnenia rodiča **fill: parent**. Po určení plochy je nutné oboznámiť aj ostatné objekty, ktoré kliknutie ovplyvní. Ako sme spomenuli vyššie v texte, komunikáciu medzi objektami obstarávajú signály. Signál pre kliknutie je potom volaný ako **myButton.clicked()**.



Obrázok 17 Aktivované, stlačené a zakázané tlačidlo Spustiť Experiment

7.1.2 Main Window

V súbore main.qml sa nachádza objekt **Window**. Je to základný objekt Qt Gui aplikácie. Pre aplikáciu Buffonovej úlohy o ihle sme veľkosť hlavného okna stanovili nasledovne: výška 864 a šírka 1536. Oknu aplikácie nie je možné meniť veľkosť. QML sa dokáže síce veľmi dobre prispôsobovať zmene veľkosti za behu aplikácie, avšak pri vykresľovaní padania ihiel na plochu dochádzalo k odchýlkam, a preto bola možnosť zmeny veľkosti vypnutá. Aplikácia je v dostatočne veľkom rozlíšení pre dobrú viditeľnosť všetkých jej dôležitých súčastí. Hlavné okno tvorí pozadie aplikácie. Jeho farba je nastavená vlastnosťou *color: "#00284b",* čo odpovedá tmavo modrej farbe. Názov okna je Buffon's needle.

7.1.3 Loader

Pre prepínanie medzi jednotlivými obrazovkami sme použili objekt **Loader**. Z dokumentácie tohto objektu [18] ide o objekt, umožňujúci načítanie iného objektu, ktorý je zadaný vo

vlastnosti **source** ako URL cesta alebo komponent. V aplikácii je Loader umiestnený v strede hlavného okna s posunutými okrajmi dovnútra. Umiestnenie je zabezpečené opäť vlastnosťou ukotvenie, teda **anchors**.

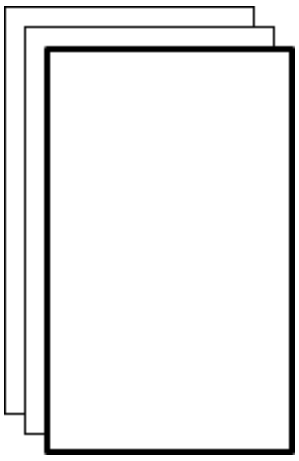
```
anchors{
    fill:parent
    margins: 45
    }
```

Margins značí vzdialenosť od okrajov rodiča. Ako východiskový zdroj Loadera sme nastavili základnú obrazovku, ktorá je uložená v súbore IdleScreen.qml.

source: "qrc:/Gui/IdleScreen.qml"

7.1.4 StackView

Loader ako prepínač medzi obrazovkami je dostačujúci v prípade, že nie je nutné uchovávať históriu. Pre pohyb medzi obrazovkami dopredu či dozadu sme použili v aplikácii **StackView**. Tento model pracuje na princípe zásobníkovej logiky, čo vyplýva už z názvu. Na obrázku 28 je naznačená zásobníková logika, využitá v tomto modeli. V aplikácii je opäť ukotvený v rodičovi *anchors.fill : parent*. Rodičom je Loader definovaný v main.qml. Východiskový objekt modelu je obrazovka hlavného menu: *initialItem: "qrc:/Gui/MainMenuScreen.qml".*



Obrázok 18 StackView [19]

StackView sa po ukončení animácie na základnej obrazovke nastaví ako zdroj pre Loader a všetky ďalšie obrazovky sa vykresľujú zásobníkovým princípom. Pre pridanie obrazovky sa volá funkcia *push()* a pre vrátenie sa o obrazovku späť, teda pre výber zo zásobníka

funkcia *pop()*. Funkcia pre výber zo zásobníku sa využíva aj pre odstránenie všetkých obrazoviek okrem tej východiskovej, použitím funkcie s parametrom null: *pop(null)*.

StackView riadi prechody medzi obrazovkami. Prechody môžu byť sprevádzané animáciami. V aplikácii sme zvolili animáciu zmeny priehľadnosti. Na obrázku 19 sa nachádza časť kódu, v ktorej sa nastavuje animácia pri pridaní obrazovky do zásobníka. Aktívna obrazovka po dobu 1 sekundy(1000 milisekúnd) mení svoju priehľadnosť z 0 na 1, a naopak obrazovka vkladaná do zásobníka z 1 na 0.

```
pushEnter: Transition {
        PropertyAnimation {
            property: "opacity"
            from: 0
            to:1
            duration: 1000
        }
    }
pushExit: Transition {
    PropertyAnimation {
        property: "opacity"
        from: 1
        to:0
        duration: 1000
    }
}
```

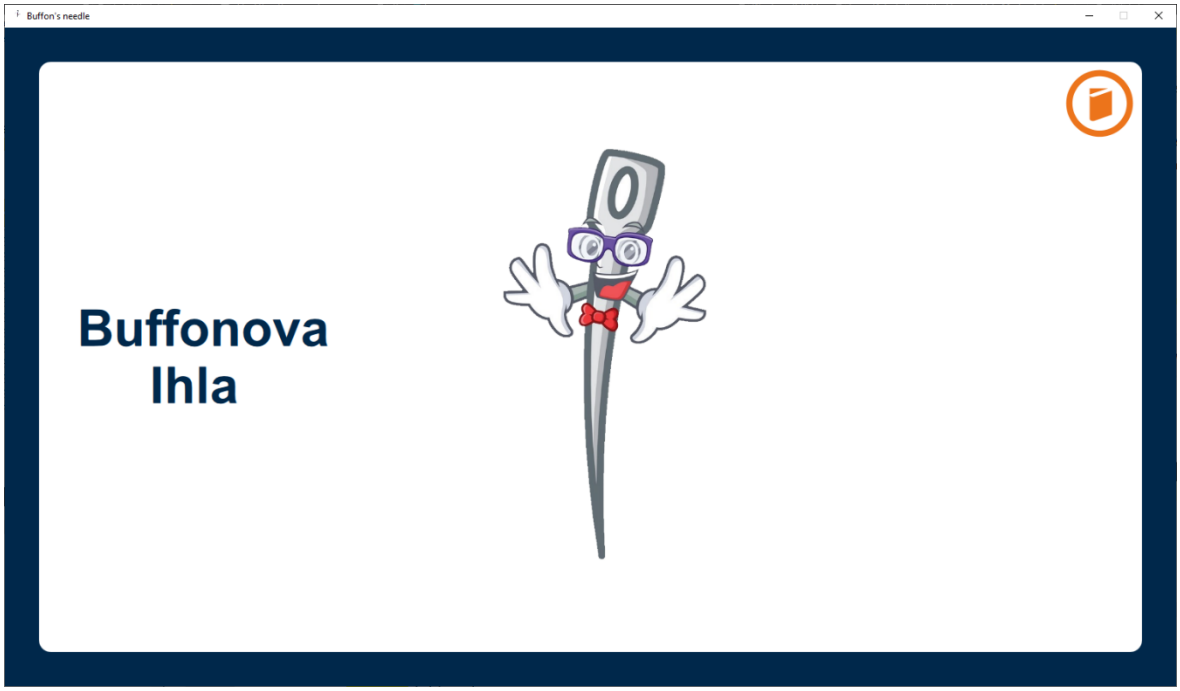
Obrázok 19 Animácia prechodu obrazoviek

7.1.5 Obrazovky

Ako sme spomenuli vyššie v texte, každá obrazovka má svoj samostatný .qml súbor. Každá obrazovka, okrem tej úvodnej, má navrchu názov pre lepšiu prehľadnosť. Tento text je ukotvený v horizontálnom strede a hornej hrane obrazovky s hornou maržou **topMargin,** nastavenou na 20 a líši sa len v názve, preto sa mu už nebudeme pri detailoch ďalších obrazoviek venovať. Úvodná obrazovka má tento názov na ľavom boku. Všetky obrazovky sa skladajú zo základného obdĺžnika so zakrivením okrajov **radius,** nastaveným na 15 a farbou nastavenou na bielu *color: white.*

Na obrázku 20 je možné vidieť úvodnú obrazovku aplikácie. Obrazovka sa skladá z obdĺžnika, ktorého identifikátor je *id: idle.* Nachádza sa na nej text Buffonova Ihla. Text sme umiestnili na ľavú stranu vo vertikálnom strede rodiča (obdĺžnik idle). Do pravého horného rohu sme pridali logo Univerzity Tomáša Bati. Pre obrázky sme použili objekt **Image**.

Samotný obrázok je potom načítaný ako vlastnosť obrázka **source** a v adresárovej štruktúre je uložený v zložke \Gui\Images.



Obrázok 20 Základná obrazovka

Posledným a veľmi dôležitým prvkom úvodnej obrazovky je obrázok ihly. Ten totiž slúži ako prechod na ďalšiu obrazovku. Po kliknutí na ihlu sa začne točiť a po skončení animácie rotovania je do zásobníka vložená obrazovka Hlavné Menu. Zdrojový kód obrázku ihly spolu s animáciou a prechodom na ďalšiu obrazovku po zastavení animácie sa nachádza na obrázku 21.

```
Image {
    id: needle
    source: "qrc:/Gui/Images/Needle.png"
    scale: 0.65
    anchors{
       horizontalCenter: parent.horizontalCenter
       verticalCenter: parent.verticalCenter
    }
    PropertyAnimation on rotation {
        id: animationNeedle
        from: 0
        to: 360 * 5
        duration: 5000
        running: false
        easing.type: Easing.InOutQuint
        onStopped: {
            mainLoader.source = "qrc:/Gui/Components/StackViewPages.qml"
        }
    }
}
```

Obrázok 21 Zdrojový kód obrázku ihly s animáciou rotácie

Po skončení animácie je na vrchol zásobníka vložená obrazovka hlavného menu. Táto obrazovka sa skladá z troch základných tlačidiel:

- Začať Experiment – po jeho stlačení sa prejde na výber plochy
- Teoretický Úvod
- Videoukážka

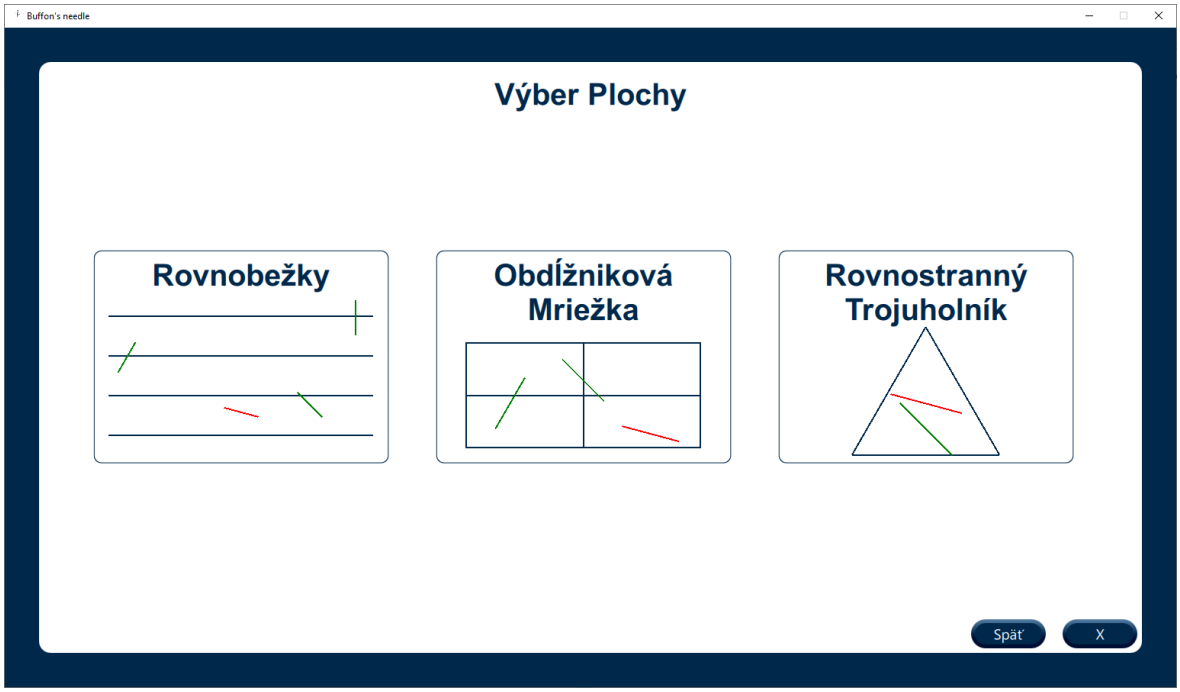
V pravom dolnom rohu sa nachádza tlačidlo pre „uspanie“ aplikácie. Nie je to nič iné ako návrat na základnú obrazovku, ktorá slúži ako variant šetriča obrazovky. Tri základné tlačidlá sme vytvorili v stĺpcovom usporiadaní **ColumnLayout**. Toto usporiadanie umožňuje pridávať viacero objektov pod seba so zachovaním konštantných rozostupov medzi jednotlivými objektami, nastavených vlastnosťou **spacing** s hodnotou 40 . Usporiadanie je ukotvené v strede rodičovského obdĺžnika *anchors.centerIn : parent*. Alternatívou k stĺpcovému usporiadaniu je usporiadanie riadkové **RowLayout,** fungujúce na rovnakom princípe. Po stlačení tlačidiel Teoretický Úvod alebo Videoukážka sa dostaneme do ďalších obrazoviek. Tieto budú detailnejšie popísané v podkapitolách 5.3 a 5.4.

Po stlačení tlačidla Začať Experiment sa aplikácia prepne do obrazovky Výber Plochy zobrazenej na obrázku 22. V aplikácii sú podporované tri plochy:

- Rovnobežky
- Obdĺžniková mriežka
- Rovnostranný trojuholník

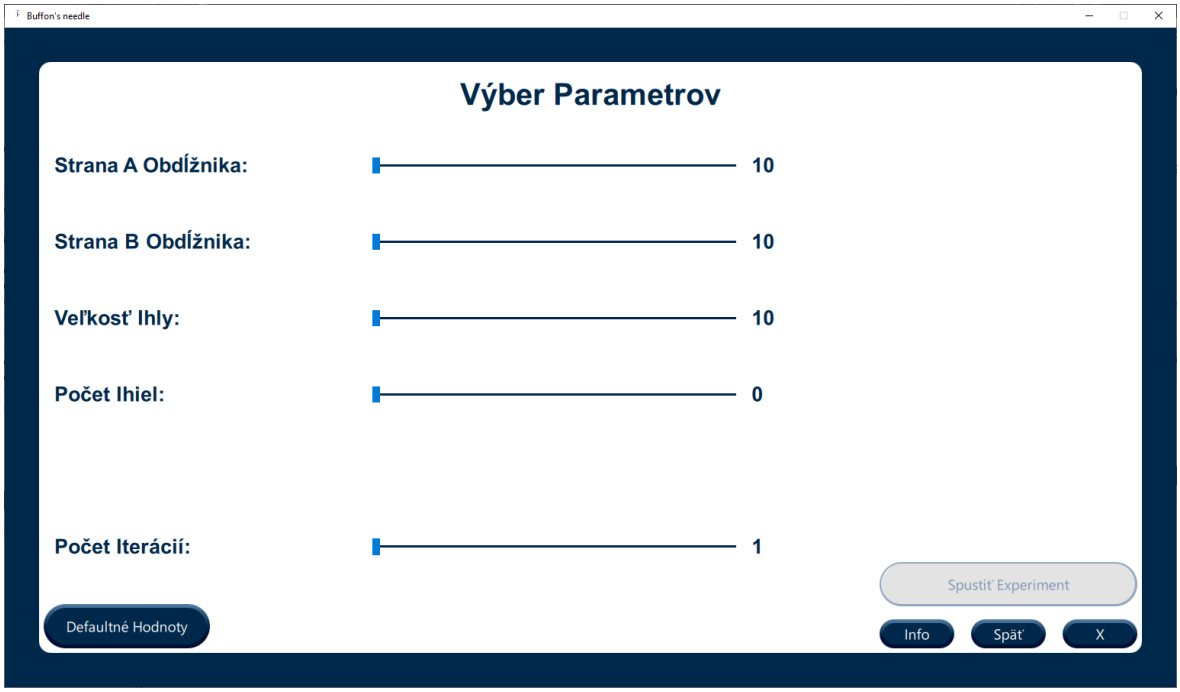
Pre výber plochy sme vytvorili tri obdĺžniky, umiestnené do riadkového usporiadania **RowLayout** s rozostupmi *spacing: 45*.

V pravom dolnom rohu sa nachádzajú tlačidlo „Späť“, ktoré po kliknutí vyberie aktuálnu obrazovku z vrcholu zásobníku stackView.*pop()* a tlačidlo „X“, ktoré vráti aplikáciu do hlavného menu *stackView.pop(null)*. Tlačidlo X vyresetuje všetky premenné aplikácie *calculations.resetAllVariables()* . Triedu **calculations** podrobnejšie popíšeme v podkapitole 5.2. Tlačidlá „Späť“ a „X“ sa objavujú aj na ďalších obrazovkách. Ich funkcionalita je všade rovnaká, a preto sa im nebudeme v ďalšom texte venovať znova.



Obrázok 22 Obrazovka Výber Plochy

Po kliknutí na ľubovoľnú plochu prejde aplikácia do obrazovky výberu parametrov, ukázanej na obrázku 23. Pre každý potrebný parameter je nutné vybrať hodnotu.



Obrázok 23 Obrazovka Výber Parametrov

Parametre sa vyberajú dvomi spôsobmi. Prvým je využitie posúvačov. Posúvač **Slider** je objekt, ktorý po kliknutí a posunutí takzvanej rukoväte **Handle** je schopný nastaviť hodnotu

premennej. Na obrázku 24 je možné vidieť zdrojový kód posúvača na výber veľkosti ihly. Medzi nastaviteľné vlastnosti patrí orientácia, v tomto prípade horizontálna, ďalej veľkosť kroku, ktorá je pri veľkosti ihly nastavená na hodnotu 1. Dôležité sú hodnoty od a do, ktoré určujú medze posúvača. **Slot** *onValueChanged()* je napojený na vlastnosť **value** posúvača a po jej zmene uloží do premennej obrazovky novú hodnotu veľkosti ihly. Ďalšími vlastnosťami sú ukotvenie, zabezpečujúce pozíciu objektu voči ostatným, výška, šírka a pozadie. Pozadie tvorí obdĺžnik modrej farby s výškou 3, vytvárajúci dojem čiary. Čísla napravo od posúvača na obrázku 23 sa v reálnom čase aktualizujú vzhľadom na hodnotu posúvača.

Druhým spôsobom je použitie tlačidla „Defaultné Hodnoty“. Toto tlačidlo nastaví všetky potrebné parametre na hodnoty, s ktorými experiment dosahuje najlepšie výsledky.

```
Slider {
    id: needleLengthSlider
    from: 10
    value: needleLengthValue
    to: calculations.maximumNeedleLength()
    stepSize: 1
    width: parent.width *0.33
    height: needleLength.height *2
    onValueChanged: {
        needleLengthValue = value
    }
    orientation: Qt.Horizontal

    anchors{
        verticalCenter: needleLength.verticalCenter
        left: parallelRangeSlider.left
    }

    background: Rectangle{
        width: parent.width
        height: 3
        anchors.centerIn: parent
        color: "#00284b"
    }
}
```

Obrázok 24 Zdrojový kód posúvača

Každý experiment má určité obmedzenia, aby malo zmysel ho vykonať. O týchto obmedzeniach sa používateľ dozvie keď klikne na tlačidlo „Info“. Po jeho stlačení sa zobrazí vyskakovacie okno s informáciami o parametroch daného experimentu. Použili sme objekt

Popup. V tomto Objekte sa nachádza obdĺžnik s textom a chybový text, ktorý je viditeľný v prípade, že boli parametre vyplnené nekorektne. Pre vypnutie okna je nutné kliknúť na tlačidlo „Ok“, ktoré okno zavrie. Počas doby, čo je okno otvorené, nie je možné klikať mimo neho, okolie okna je neaktívne. Na obrázku 25 sa nachádza ukážka vyskakovacieho okna pre plochu obdĺžniková mriežka s chybne vyplneným parametrom veľkosti ihly, ktorá nesmie byť väčšia ako strana A obdĺžnika. O korektné vyplnenie textu okna sa stará funkcia *resolveInfoText()*, ktorá na základe zvolenej plochy vyplní text.

Obrázok 25 Vyskakovacie okno

Posledným objektom na obrazovke Výber Parametrov je tlačidlo „Spustiť Experiment“. Tlačidlo sa môže nachádzať v aktivovanom alebo zakázanom stave, vzhľadom na správnosť vyplnenia parametrov. O kontrolu správnosti sa stará funkcia *areAllParametersFilledCorrectly()*, ktorá nastavuje funkčnosť tlačidla a v prípade nekorektného vyplnenia parametrov nastaví chybový text vyskakovacieho okna. Na obrázku 26 je zobrazená časť kódu funkcie *areAllParametersFilledCorrectly()* pre kontrolu parametrov plochy 1 (rovnobežky).

```
if(surfaceSelected === 1){
    if( needleCountValue ===0 || numberOfIterationsValue ===0){
        popUpErrorText = "Niektorá Hodnota Je 0!!"
        return false;
    }
    if(parallelRangeValue < 10 || needleLengthValue < 10){
        popUpErrorText = "Veľkosť Ihiel/Vzdialenosť Rovnobežiek Musí Byť Aspoň 10!!"
        return false;
    }
    if(parallelRangeValue < needleLengthValue){
        popUpErrorText = "Vzdialenosť Rovnobežiek Je Menšia \n Ako Veľkosť Ihly!!"
        return false;
    }
    return true;
}
```

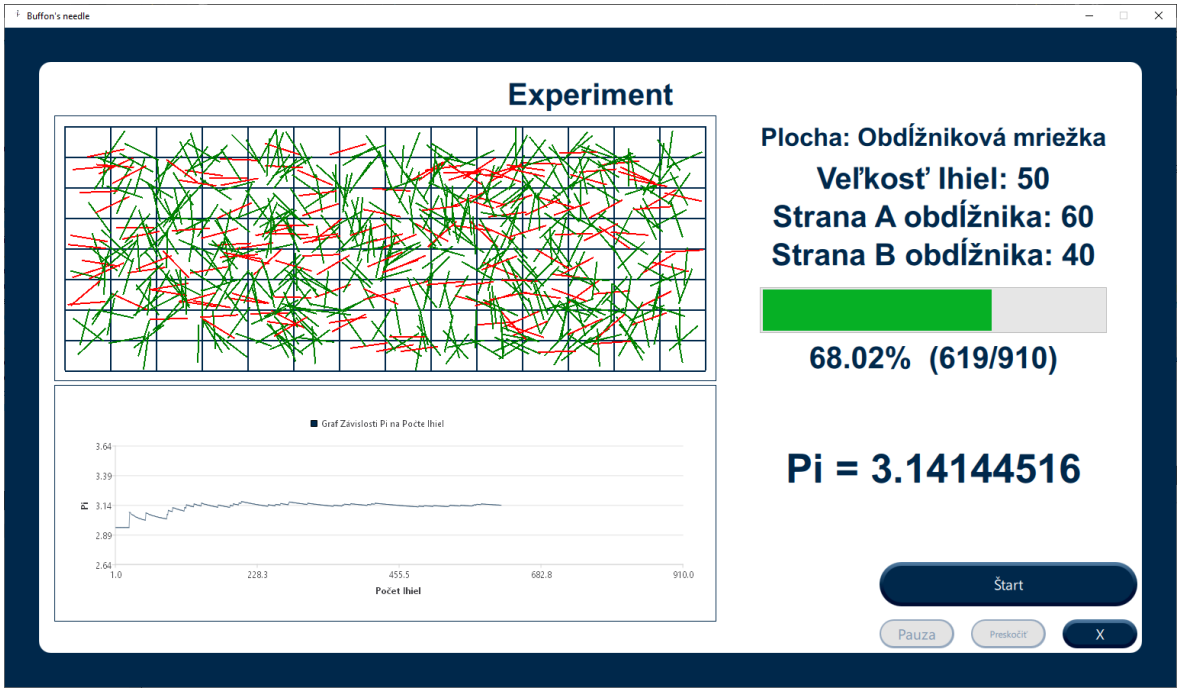
Obrázok 26 Časť zdrojového kódu funkcie *areAllParametersFilledCorrectly()*

V prípade korektného vyplnenia parametrov je tlačidlo „Spustiť Experiment“ aktívne. Po jeho kliknutí sa v závislosti od zvoleného počtu iterácií zmení obrazovka na obrazovku Animácie pre jednu iteráciu, alebo na obrazovku Výpočtu pre viacero iterácií.

Obrazovka Výpočtu je na vzhľad veľmi jednoduchá. Skladá sa okrem názvu už len z jedného textu informujúceho užívateľa, že prebieha výpočet. Po zobrazení obrazovky je spustený časovač na 1 sekundu a po jeho uplynutí sa odštartujú výpočty funkciou *calculations.start()*. Časovač sme použili z dôvodu, že pri nižšom počte iterácií výpočet prebehne veľmi rýchlo a obrazovka by sa zobrazila len na zlomok sekundy. Po skončení všetkých iterácií výpočtu je totiž z c++ emitovaný príslušný signál, na ktorý je napojená obrazovka Výpočtu. Po prijatí signálu je na zásobník vložená obrazovka Výsledok Experimentu, ktorá podrobnejšie popíšeme nižšie v texte.

Obrazovka Animácie sa zobrazí pre jednu zvolenú iteráciu. Obrázok 27 znázorňuje prebiehajúci experiment. V pravom dolnom rohu sa nachádzajú tlačidlá ovládajúce obrazovku Animácie. Z logického hľadiska nemá zmysel, aby bolo sprístupnené tlačidlo „Späť“. Na jeho mieste pribudli dve tlačidlá. Prvým z nich je tlačidlo „Preskočiť“, slúžiace na preskočenie animácie a zobrazenie výsledku experimentu na obrazovke Výsledok Experimentu. Druhým tlačidlom je „Pauza“, ktoré slúži na pozastavenie bežiaceho experimentu. Nad týmito tlačidlami sa nachádza tlačidlo „Štart“, slúžiace na odštartovanie výpočtov, prípadne na obnovenie animácie po stlačení tlačidla „Pauza“. V pravom hornom rohu sa nachádzajú základné informácie o zvolených parametroch experimentu. Parameter zvolenej plochy sme získali z funkcie *resolveSurface()*. Pod nimi sa nachádza ukazovateľ postupu. Pre jeho implementáciu sme využili objekt **ProgressBar**. Ukazovateľ v reálnom čase zobrazuje progres

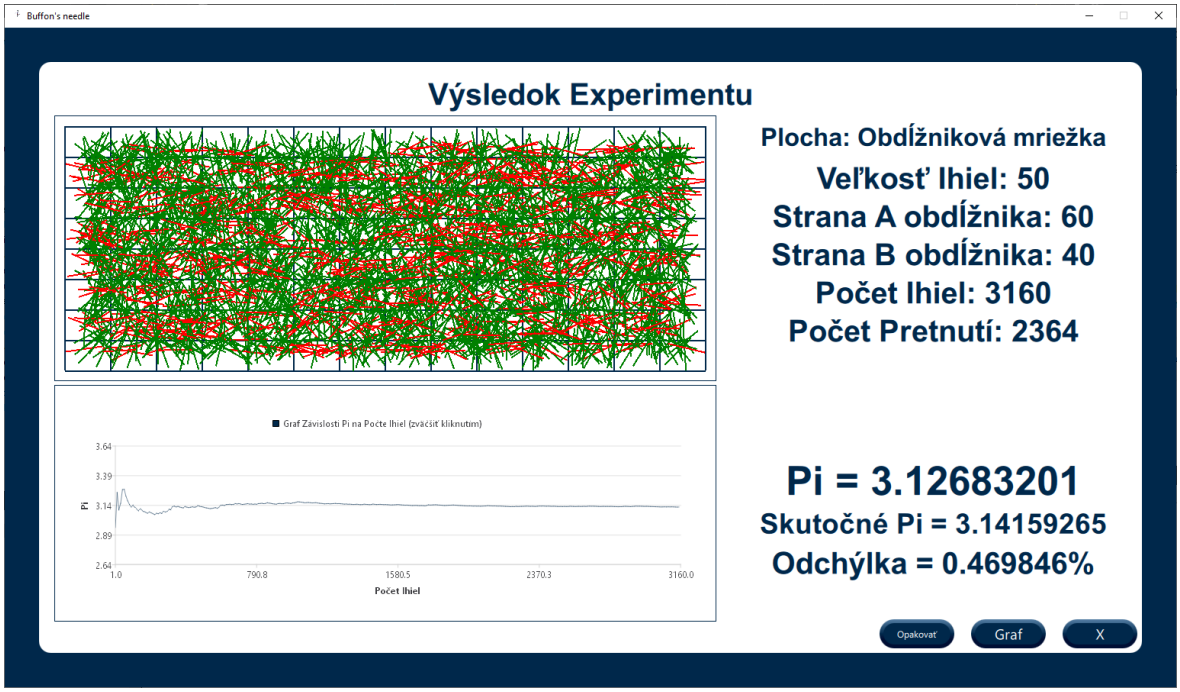
experimentu (počet hodených ihiel lomeno počet všetkých ihiel) vyjadrený v percentách. Posledným Objektom na pravej strane je hodnota Ludolfovho čísla Pi, rovnako meniaca sa v závislosti od počtu hodených ihiel. Pi je zobrazené ako desatinné číslo s ôsmymi desatinnými ciframi. Pre zmenu hodnoty sme použili **QVariant** naplnený v c++ časti zdrojového kódu po dokončení výpočtov.



Obrázok 27 Prebiehajúci experiment

Ľavá polovica obrazovky je zastúpená časťou ihiel padajúcich v reálnom čase a grafom závislosti Pi na počte hodených ihiel, rovnako vykreslenom v reálnom čase. Spôsob vykresľovania obsahu týchto objektov bude podrobne popíšeme v podkapitolách 5.1.6 a 5.1.7.

Po dokončení animácie je zobrazená obrazovka Výsledok experimentu. Na obrázkoch 28 a 29 sú zobrazené dva varianty vzhľadu obrazovky. Obrázok 28 obrazovku vykresľuje pre jednu iteráciu a obrázok 29 pre viacero zvolených iterácií.



Obrázok 28 Obrazovka Výsledok Experimentu 1 iterácia

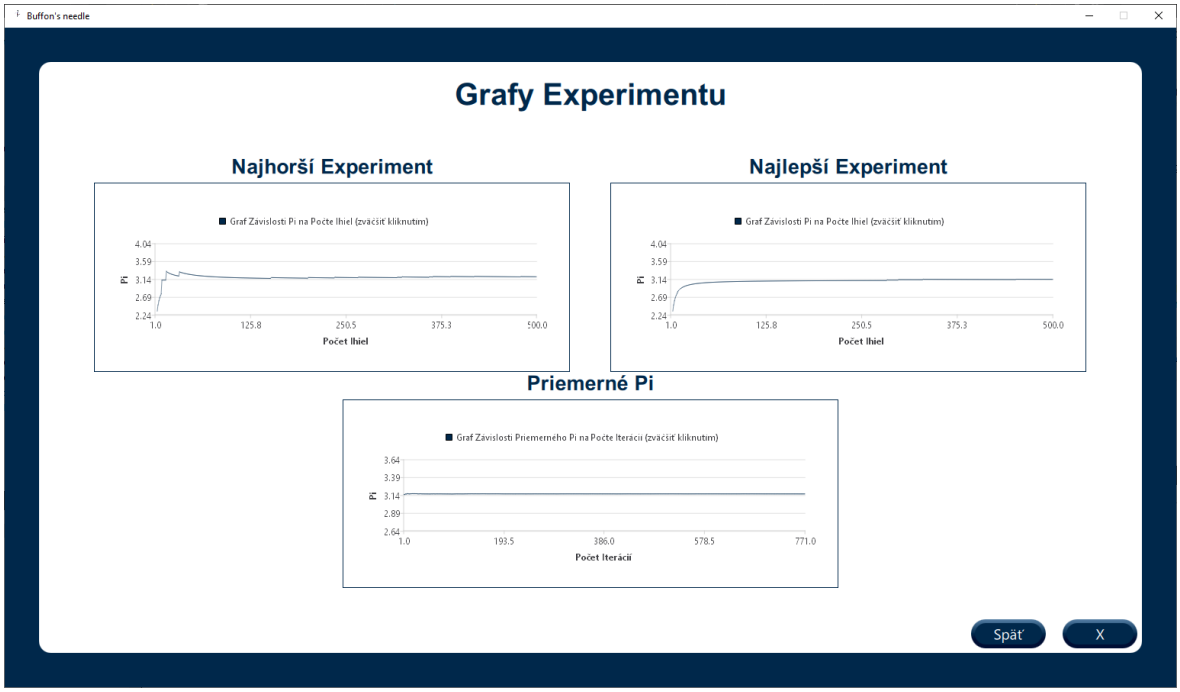
Obrázok 29 Obrazovka Výsledok Experimentu 771 iterácií

Na oboch variantoch obrazovky sa v pravom hornom rohu nachádzajú informácie o zvolených parametroch experimentu. Pri jednom experimente sme oproti obrazovke Animácie pridali informácia o počte pretnutí.

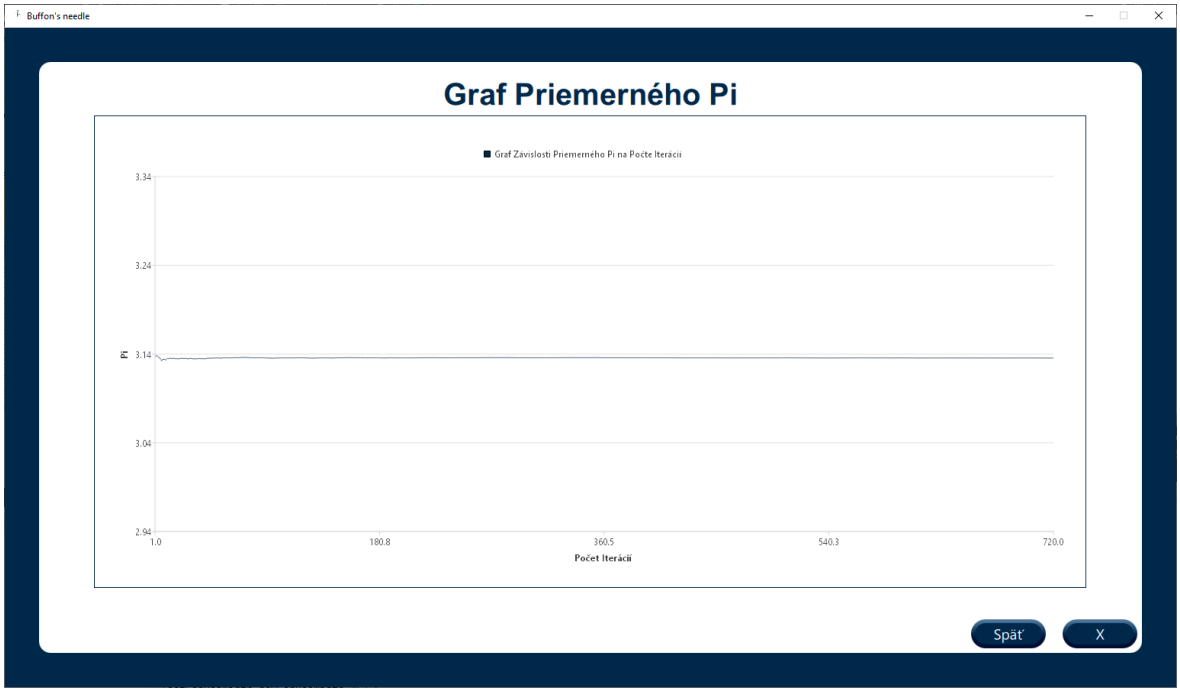
Na obrazovke sa nachádza porovnanie hodnoty Ludolfovho čísla ako výsledku experimentu s jeho skutočnou hodnotou. Zobrazená je taktiež odchýlka vypočítaná podľa vzorca

Math.abs(((*piValue* - *realPiValue*)/*realPiValue**100)), kde *piValue* je hodnota Pi z experimentu a *realPiValue* je hodnota skutočného Pi. Pre prípad, že by *piValue* bolo menšie ako skutočná hodnota Pi, bola použitá funkcia *abs()* z knižnice **Math**. Pre experiment s viacerými iteráciami nie je možné jasne vyjadriť hodnotu Pi. Preto sú na obrazovke zobrazené hodnoty Pi najlepšej a najhoršej iterácie a tiež priemerné Pi všetkých iterácií. Získanie hodnôt najlepšieho, najhoršieho a priemerného Pi podrobnejšie popíšeme v podkapitole 5.2.5. Pri experimente s jednou iteráciou je zobrazená plocha s ihlami a finálny graf závislosti Pi na počte hodených ihiel. Na graf je možné kliknúť a zväčšiť ho. Rovnako je možné zobraziť/zväčšiť grafy po kliknutí na tlačidlo „Graf/y“, tým sa na zásobník obrazoviek vloží obrazovka Graf/Grafy Experimentu. Rovnako ako na obrazovke Animácie je nie je dostupné tlačidlo „Späť“. Nahradili sme ho tlačidlom „Opakovať“. Po jeho stlačení sa experiment môže zopakovať s rovnakými parametrami.

Poslednými dvomi obrazovkami sú obrazovka Grafov a obrazovka Zväčšenia grafu. Ako je možné vidieť na obrázku 30 pri viacerých iteráciách sú výsledkom experimentu tri grafy. Pre lepšiu čitateľnosť sa po kliknutí na ľubovoľný graf zobrazí obrazovka Zväčšenia grafu. Na obrázku 31 sa nachádza zväčšený graf závislosti vývoja priemerného Pi na počte iterácií. Z grafu je možné vidieť, že pri zadaných parametroch sa priemerné Pi ustáli približne po 150 iteráciách.



Obrázok 30 Grafy experimentu s viacerými iteráciami



Obrázok 31 Graf závislosti priemerného Pi od počtu iterácií

7.1.6 Vykreslenie ihiel a plochy

Pri experimente Buffonovej úlohy s jednou iteráciou sa na obrazovke Animácie postupne pridávajú ihly. Všetky ihly spoločne s plochou sú vykreslené v jednom obdĺžniku. Toto je veľmi dôležité pre umiestňovanie ihiel, ale aj pre prípadné opakovanie experimentu, kde je nutné všetky padnuté ihly pred novým experimentom odstrániť. Všetky doteraz popísané objekty sme na obrazovky pridali staticky pri ich vytvorení. Avšak tento prístup by pre postupné pridávanie ihiel na obrazovku nefungoval, z dôvodu neurčitého počtu objektov pri písaní kódu. Preto sme zvolili dynamické pridávanie objektov na obrazovku.

Ihla alebo aj jednotlivé zložky plochy (rovnobežky, strany trojuholníka) sú v základnom ponímaní priamky. QML priamo neponúka možnosť vytvorenia objektu priamky či čiary. Ako jednoduchú alternatívu je však možné použiť obdĺžnik. Konkrétne obdĺžnik s výškou **height** 2. Po pridaní takéhoto objektu sa na obrazovke zobrazí úsečka s nastaviteľnou veľkosťou, ktorá korešponduje šírke **width** obdĺžnika. Rotáciu ihiel umožňuje vlastnosť obdĺžnika **rotation**, udávaná v stupňoch. Pre potreby dynamického pridávania objektov na obrazovku sme vytvorili komponent **MyRectangle**. Jeho vlastnosti sú zobrazené na obrázku 32.

```
Rectangle{
    id: rectangle
    width: 2
    height: 2
    border.color: "#00284b"
    border.width: 1
    z:1
}
```

Obrázok 32 MyRectangle

Samotné pridanie objektu dynamicky je vykonané v dvoch krokoch. Vytvorenie komponentu: *component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")* a vytvorenie objektu z tohto komponentu: *object = component.createObject(animationRectangle, {"x": objectX, "y": objectY, "rotation": 60, "width": objectWidth})*. Vo funkcii *createObject()* sa definuje rodič a všetky vlastnosti, ktoré užívateľ chce tomuto objektu nastaviť.

Vzhľadom na to, že každá priamka plochy a každá ihla má iné parametre, je potrebné mať tieto parametre uložené. Generácia pozície, rotácie či farby ihly, ako aj tvar a pozícia plochy sú generované v c++ časti zdrojového kódu. Ako sme spomenuli v podkapitole 4.2.4, **QVariant** sa používa na náhradu dátových typov pre uloženie väčšieho množstva dát. Pre vykreslenie plochy sme použili tieto QVarianty:

- linesVariant – pre uloženie vlastnosti y každej čiary plochy rovnobežky
- linesVariantHorizontal – pre uloženie vlastnosti x zvislých čiar plochy obdĺžniková mriežka
- linesVariantVertical – rovnaká funkcia ako linesVariant pre plochu obdĺžniková mriežka

Pre vykreslenie ihly sme použili tieto QVarianty:

- crossingsVariantVar – pre nastavenie farby danej ihly
- xVariantVar – pre uloženie súradnice x stredu ihly
- yVariantVar – pre uloženie súradnice y stredu ihly
- rotationVariantVar – pre nastavenie rotácie danej ihly

Každý QVariant obsahuje určitý počet hodnôt v závislosti na počte čiar na ploche a počte ihiel v experimente.

Pridanie čiar vybranej plochy na obrazovke Animácie sa vykoná po prijatí signálu o ukončení výpočtu experimentu z c++ časti zdrojového kódu. Po prijatí tohto signálu je volaná

funkcia *addLines()*. Na obrázku 33 môžeme vidieť časť zdrojového kódu funkcie *addLines()* pre plochu rovnobežky. Pre všetky rovnobežky sme vytvorili objekt ukotvený na ľavej strane rodičovského obdĺžnika **animationRectangle** a na pozícii y získanej z i-teho prvku QVariantu linesVariantVar. Rovnobežky majú šírku rovnakú ako rodič, preto vlastnosť **width** je nastavená na šírku rodiča. Vlastnosť **height** je nastavená na východiskovú hodnotu 2.

```
if(surfaceSelected === 1){
    if(animationRectangle.children.length === 0){
        for(var i = 0; i < linesVariantVar.length; i++){
            component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
            line = component.createObject(animationRectangle, {"anchors.left":animationRectangle.left , "y":linesVariantVar[i],
                                          "width": animationRectangle.width})
        }
    }
}
```

Obrázok 33 Časť kódu funkcie *addLines()* pre plochu rovnobežky.

Na obrázku 34 je zobrazená časť zdrojového kódu funkcie *addLines()* pre plochu obdĺžniková mriežka. Pre túto plochu sme boli nútení pridať čiary horizontálne a vertikálne. Pri výbere tejto plochy je obdĺžniková mriežka vycentrovaná na stred rodičovského obdĺžnika. Z tohto dôvodu sme vytvorili pomocné premenné **xStart** a **yStart**, určujúce konštantnú súradnicu x pri horizontálnych čiarach, respektíve konštantnú súradnicu y pri čiarach vertikálnych. Pre pridanie horizontálnych čiar sme použili prvý for cyklus. Nastavované vlastnosti objektov sú pozícia x z premennej **xStart** pozícia y získaná z i-teho prvku QVariantu linesHorizontalVariantVar a šírka čiary **width** definovaná rovnako ako pozícia x v c++ časti zdrojového kódu. Pridanie vertikálnych čiar sme realizovali rovnakým princípom. V tomto prípade je pre všetky čiary konštantná súradnica y z premennej **yStart** a pozícia x získaná z i-teho prvku QVariantu linesVerticalVariantVar. Pre dosiahnutie vertikálnej čiary je šírka **width** obdĺžnika nastavená na 2 a dĺžka čiary sa nastavuje vlastnosťou výška **height** (naopak ako pri horizontálnej čiare)**.**

```
else if(surfaceSelected === 2){
    var xStart = calculations.getRectangleXStart()
    var yStart = calculations.getRectangleYStart()
    var lineWidth = calculations.lineXLength()
    var lineHeight = calculations.lineYLength()

    if(animationRectangle.children.length === 0){
        for(var j = 0; j < linesHorizontalVariantVar.length; j++){
            component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
            line = component.createObject(animationRectangle, {"x": xStart , "y":linesHorizontalVariantVar[j],
                                                               "width": lineWidth})
        }

        for(var k = 0; k < linesVerticalVariantVar.length; k++){
            component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
            line = component.createObject(animationRectangle, {"y": yStart , "x":linesVerticalVariantVar[k],
                                                               "height": lineHeight})
        }
    }
}
```

Obrázok 34 Časť kódu funkcie *addLines()* pre plochu obdĺžniková mriežka.

Na obrázku 35 zobrazujeme časť zdrojového kódu funkcie *addLines()* pre poslednú plochu rovnostranný trojuholník. Na obrazovku sme pridali len tri strany trojuholníka. Prvá strana je vložená do horizontálneho stredu rodiča, na pozíciu y definovanú pri výpočtoch v c++ časti zdrojového kódu. Pozície zvyšných strán trojuholníka sú odvodené z Pytagorovej vety. Pre rovnostranný trojuholník platí, že všetky strany majú rovnakú veľkosť, preto vlastnosť **width** je pre všetky tri objekty rovnaká. Vlastnosť **rotation** dotvára vzhľad trojuholníka.

```
else{
    component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
    line = component.createObject(animationRectangle, {"anchors.horizontalCenter": animationRectangle.horizontalCenter ,
                                  "y": calculations.bottomTriangleLineY() , "width": triangleAVar})

    var xRight = line.x + line.width/4
    var xLeft = line.x - line.width/4
    var y = line.y  - Math.sqrt((line.width*line.width) - ((line.width/2)*(line.width/2)))/2
    component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
    line = component.createObject(animationRectangle, {"x": xLeft, "y": y, "rotation": 120, "width": triangleAVar})

    component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
    line = component.createObject(animationRectangle, {"x": xRight, "y": y, "rotation": 60, "width": triangleAVar})
}
```

Obrázok 35 Časť kódu funkcie *addLines()* pre plochu rovnostranný trojuholník.

Vykresľovanie ihiel na obrazovke sme realizovali postupne po jednej ihle. Na obrázku 36 možno vidieť zdrojový kód funkcie *addNeedle()* použitej na vykreslenie práve jednej ihly na obrazovku. Parametrom funkcie je index. Tento index je pomocná premenná pre vykreslenie správnej ihly. Vykreslenie väčšieho množstva objektov pomocou cyklu je príliš rýchle pre navodenie dojmu animácie. Z tohto dôvodu sme pre obrazovku Animácie vytvorili časovač. Pre časovač sme použili objekt **Timer** s vlastnosťami **running** a **repeat** nastavenými na false, čo znamená že je nutné tento časovač ručne spúšťať a po uplynutí intervalu nastaveného podľa podmienky *interval: piVariantVar.length > 500 ? 10 : 100*, opäť znova spustiť.

Princíp vykresľovania ihiel je nasledovný:

1. Po prijatí signálu o ukončení výpočtov sa vykreslí prvá ihla s indexom 0 *addNeedle(0)*
2. Zvýši sa hodnota premennej **variantIndex** *variantIndex++*
3. Spustí sa časovač s daným intervalom *calcTimer.start()*
4. Po uplynutí časovača sa vykreslí ihla s indexom variantIndex *addNeedle(variantIndex)*
5. Zvýši sa hodnota premennej **variantIndex** *variantIndex++*
6. Reštartuje sa časovač *calcTimer.start()*
7. Po uplynutí časovača sa vykreslí ďalšia ihla a proces sa opakuje. V prípade, že boli vykreslené všetky ihly, prechádza sa na obrazovku Výsledok Experimentu.

Samotná funkcia *addNeedle(index)* pridáva obdĺžniky s výškou 2 rovnakým princípom ako pri pridávaní čiar plôch. Nastavených je však viacero hodnôt. Je nutné dopočítať súradnicu x, triviálnym výpočtom (od stredu ihly je odčítaná polovica jej dĺžky). Pridanými vlastnosťami sú **color** nastavená na červeno alebo zeleno v závislosti na pretnutí plochy a vlastnosť rotácie ihly.

```
function addNeedle(index){
    var color = crossingsVariantVar[index] ? "green" : "red"
    var component = Qt.createComponent("qrc:/Gui/Components/MyRectangle.qml")
    var needle = component.createObject(animationRectangle, {"x":xVariantVar[index] - needleLengthVar/2, "y":yVariantVar[index] ,
                                        "width": needleLengthVar, "border.color": color, "rotation": rotationVariantVar[index] })
}
```

Obrázok 36 Kód funkcie *addNeedle()* pre vykreslenie ihly.

Ihly a čiary plochy sa vykresľujú aj na obrazovke Výsledok experimentu. V tejto obrazovke sú všetky čiary a ihly vykreslené pri zobrazení obrazovky. Nie je nutné použiť časovač. Funkciu *addNeedle(index)* nahrádza funkcia *addNeedles()*, ktorá pridáva všetky ihly ako komponenty v cykle for.

Na obrazovku Animácie je možné sa vrátiť v prípade opakovania experimentu. Je nutné odstrániť všetky ihly s predchádzajúceho experimentu. Na obrázku 37 ukazujeme zdrojový kód funkcie *removeRectangles()*, ktorej úlohou je odstrániť všetky ihly a čiary plochy z predchádzajúceho experimentu. Keďže všetky dynamicky vytvorené objekty majú identického rodiča, obdĺžnik **animationRectangle**, je možné odstrániť všetkých jeho potomkov. Potomkov je nutné odstraňovať od posledného po prvého, a preto bol použitý spätný cyklus for.

```
function removeRectangles(){
    for(var i = animationRectangle.children.length; i > 0 ; i--) {
        animationRectangle.children[i-1].destroy()
    }
}
```

Obrázok 37 Kód funkcie *removeRectangles()*

Na obrazovku Výsledok experimentu je možné sa vrátiť z obrazovky Graf Experimentu. V tomto prípade je ale nežiadúce opätovné vykreslenie všetkých objektov. Preto je po prvom vykreslení všetkých dynamických objektov nastavená premenná **printed** na hodnotu true. V prípade vrátenia sa na obrazovku sa nebudú objekty opätovne vykresľovať.

7.1.7 Vykreslenie grafov

V aplikácií je možné vykresliť dva druhy grafov. Graf závislosti Pi na počte ihiel a graf závislosti Pi na počte iterácií. Zdrojový kód kostry grafu možno vidieť na obrázku 38. Pre

vykreslenie rôznych grafických reprezentácií grafov, legiend a podobne sa v Qt používa objekt **ChartView** [20]. Pre potreby grafu závislosti dvoch premenných je vhodným grafom čiarový graf **LineSeries**. Pred samotným vytvorením objektu LineSeries je nutné definovať osi grafu, pomocou objektu **ValueAxis**. V aplikácii sme vytvorili pre každý graf samostatné osi. Pomenovanie osí zabezpečuje vlastnosť **titleText** a ohraničenie osí vlastnosti **min** a **max**. Minimum a maximum na osi y je pre experimenty na ploche rovnostranný trojuholník vo väčšom rozpätí z dôvodu väčších výkyvov hodnôt na začiatku experimentu.

```
ChartView {
    anchors.fill: parent
    antialiasing: true

    ValueAxis {
        id: xAxis
        min: 1
        max: calculations.needleCount()
        gridVisible: false
        titleText: "Počet Ihiel"
        titleVisible: true
    }
    ValueAxis {
        id: yAxis
        max: surfaceSelected === 3 ? 4.04: 3.64
        min: surfaceSelected === 3 ? 2.24: 2.64
        titleText: "Pi"
        titleVisible: true
    }

    LineSeries {
        id: series
        name: "Graf Závislosti Pi na Počte Ihiel"
        axisX: xAxis
        axisY: yAxis
        color: "#00284b"
        width: 0.5
    }
}
```

Obrázok 38 Zdrojový kód kostry grafu

Po vytvorení kostry grafu je nutné graf naplniť dátami. Obrázok 39 obsahuje zdrojový kód funkcie *addToGraph(index)* slúžiacej na pridanie jednej hodnoty do grafu. Táto funkcia sa vykonáva na obrazovke Animácie. Rovnako ako funkcia *addNeedle(index)* pridáva len jednu hodnotu, pre aktualizovanie grafu v reálnom čase po padnutí ihly. Nanášanie hodnôt do grafu má dve podmienky:

- Ak je počet ihiel väčší ako 1000, vykreslí sa len každá desiata ihla.
- Ak je hodnota hodnoty Pi na danom indexe mimo ohraničenie minima a maxima na osi y, ihla sa nevykreslí.

Samotné pridanie bodov do grafu je realizované funkciou *append(x ,y)*, kde x je hodnota počtu ihiel/iterácií vzhľadom na typ grafu a y je hodnota Pi pri danom indexe. Pri zvolení opakovania experimentu sú body z grafu vymazané pomocou funkcie *removePoints()*.

```
function addToGraph(index){
    var iterationNumber = piVariantVar.length < 1000 ? 1 : 10
    if(index % iterationNumber === 0){
        if(piVariantVar[index] <= yAxis.max && piVariantVar[index] >= yAxis.min)
            series.append(index+1, piVariantVar[index])
    }
}
```

Obrázok 39 Zdrojový kód funkcie *addToGraph(index)*

V obrazovke Výsledok Experimentu nie je nutné hodnoty grafu vykresľovať v reálnom čase. Preto je funkcia *addToGraph(index)* nahradená funkciou *addToGraph()* zobrazenej na obrázku 40. Pre vykreslenie je použitý cyklus for. Opätovnému vykresleniu zabraňuje premenná **printed** rovnako ako pri funkcii *addNeedle()*.

```
function addToGraph(){
    piVariantVar = calculations.piVariant()
    var iterationNumber = piVariantVar.length < 1000 ? 1 : 10
    for(var index = 0; index < piVariantVar.length; index+=iterationNumber){
        if(piVariantVar[index] <= yAxis.max && piVariantVar[index] >= yAxis.min)
            series.append(index+1, piVariantVar[index])
    }
}
```

Obrázok 40 Zdrojový kód funkcie *addToGraph()*

Na obrazovkách Graf/Grafy Experimentu a Zväčšený Graf sa body na graf vykresľujú po zobrazení obrazovky. Princíp zobrazovania je rovnaký ako pri funkcii *addToGraph()*. Opätovnému vykresľovaniu grafov pri vrátení sa na obrazovku zabraňuje premenná **graphsPrinted**, nastavená na true po prvom vykreslení grafu.

7.2 Výpočtová časť

V tejto podkapitole rozoberieme c++ časť aplikácie a jednotlivé výpočty potrebné k uskutočneniu experimentu. Celú logiku výpočtu Ludolfovho čísla sme implementovali v triede calculations.

7.2.1 Trieda calculations

Calculations je trieda dediaca od public triedy QObject. Má jeden explicitný konštruktor. V triede sa nachádzajú všetky potrebné inštancie s notáciou **názovPremennej_**, spoločne s ich gettermi a settermi. Pre prístup ku getterom a setterom sme tieto metódy deklarovali ako Q_INVOKABLE. Ďalšie public Q_INVOKABLE metódy sú napríklad *resetAllVariables(), start() či redoExperiment()*. Signály používané pre oznámenie dokončenia výpočtu sú jedinečné pre každý typ výpočtu. V neposlednom rade sa v triede nachádzajú metódy pre samotný výpočet ako napríklad *calculatePiParallelsSingleIteration()*. Na obrázku 41 sa nachádza kód deklarácie enum classy **Sufrace** slúžiacej na prehľadnejšie rozpoznanie o akú plochu sa jedná.

```
enum class Surface{
  Parallels = 1,
  Rectangle,
  EquilTriangle
};
```

Obrázok 41 Kód enum classy Surface

7.2.2 Nastavenie, uloženie a resetovanie parametrov

Nastavovanie parametrov je dôležité pre správne vyhodnotenie experimentu. Prvý parameter nastavovaný na obrazovke Výber Plochy je samotná plocha. Po kliknutí na zvolenú plochu sa pomocou settera *setSurfaceSelected(int newSurface)* uloží vybraná plocha do premennej triedy calculations **surfaceSelected_**. Následne na obrazovke Výber Parametrov sme použili getter premennej **surfaceSelected_** *surfaceSelected()* a na jeho základe je od užívateľa požadované vyplnenie ostatných parametrov. Pre každý parameter sme nastavili hornú hranicu. Horná hranica je konfigurovateľná v triede calculations pre každú premennú. Na obrázku 42 je zobrazený zdrojový kód metódy *maximumNeedleLength()* určujúcej hornú hranicu veľkosti ihly vzhľadom na vybranú plochu. Maximálna dĺžka ihly je100 pre rovnobežky a obdĺžnikovú mriežku a 300 pre rovnostranný trojuholník.

```
double Calculations::maximumNeedleLength() const
{
    switch (static_cast<Surface>(surfaceSelected_)) {
    case Surface::Parallels:
        return maximumNeedleLengthParallels_;
    case Surface::Rectangle:
        return maximumNeedleLengthRectangle_;
    case Surface::EquilTriangle:
        return maximumNeedleLengthTriangle_;
    }
    return maximumNeedleLengthParallels_;
}
```

Obrázok 42 Zdrojový kód metódy *maximumNeedleLength()*

Implementáciu metódy *onDefaultButtonClicked()* je možné vidieť na obrázku 43. Nastavuje parametre na hodnoty, s ktorými experimenty dávajú čo najpresnejšie výsledky.

```
void Calculations::onDefaultButtonClicked()
{
    switch (static_cast<Surface>(surfaceSelected_)) {
    case Surface::Parallels:
        parallelRange_ = 100.0;
        needleLength_ = 95.0 ;
        needleCount_ = 5000;
        numberOfIterations_ = 1;
        return;
    case Surface::Rectangle:
        rectangleA_ = 100.0;
        rectangleB_ = 40.0;
        needleLength_ = 50.0 ;
        needleCount_ = 10000;
        numberOfIterations_ = 1;
        return;
    case Surface::EquilTriangle:
        triangleA_ = 300.0;
        needleLength_ = 254.0 ;
        needleCount_ = 250;
        numberOfIterations_ = 1;
        return;
    }
}
```

Obrázok 43 Zdrojový kód metódy *onDefaultButtonClicked()*

Po stlačení tlačidla „Spustiť Experiment“ na obrazovke Výber parametrov sú volané settery všetkých potrebných premenných pre daný experiment. Na obrázku 44 sa nachádza časť zdrojového kódu funkcie *onClicked* po stlačení tlačidla. Pre všetky tri plochy sa vždy nastavujú premenné **needleLength_, needleCount_** a **numberOfIterations_**.

```
onClicked: {
    calculations.setNeedleLength(needleLengthValue)
    calculations.setNeedleCount(needleCountValue)
    calculations.setNumberOfIterations(numberOfIterationsValue)
    if(surfaceSelected === 1){
        calculations.setParallelRange(parallelRangeValue)
    }
    else if(surfaceSelected === 2){
        calculations.setRectangleA(rectangleAValue)
        calculations.setRectangleB(rectangleBValue)
    }
    else if(surfaceSelected === 3){
        calculations.setTriangleA(triangleAValue)
    }
```

Obrázok 44 Časť zdrojového kódu po stlačení tlačidla „Spustiť Experiment“

7.2.3 Štart experimentu

Na odštartovanie experimentu slúži metóda *start()*. Jej zdrojový kód je zobrazený na obrázku 45. V metóde sa najskôr resetujú premenné potrebné na výpočet (**pi_** = 0 a **progress_** v metóde *setProgress(0)*) a vyprázdnia sa všetky QVarianty v metóde *clearVariants()* pomocou metódy QVariantu *clear()*. Následne je na základe vybranej plochy a zvoleného počtu iterácií volaná jedna zo šiestich metód pre výpočet Pi.

```
void Calculations::start()
{
    clearVariants();
    setProgress(0);
    pi_ = 0;
    switch (static_cast<Surface>(surfaceSelected_)) {
    case Surface::Parallels:
        numberOfIterations_ == 1 ? calculatePiParallelsSingleIteration():
                calculatePiParallelsMultiIterations();
        return;
    case Surface::Rectangle:
        numberOfIterations_ == 1 ? calculatePiRectangleSingleIteration():
                calculatePiRectangleMultiIterations();
        return;
    case Surface::EquilTriangle:
        numberOfIterations_ == 1 ? calculatePiEquilTriangleSingleIteration():
                calculatePiEquilTriangleMultiIterations();
        return;
    }
}
```

Obrázok 45 Zdrojový kód metódy *start()*

7.2.4 Generovanie náhodných čísel

V každej metóde pre výpočet Pi je nutné pre každú ihlu náhodne vygenerovať súradnice stredu ihly a uhol rotácie ihly voči osy x. Pre účely generácie náhodných čísel sme použili triedy z knižnice *random.h*. Na obrázku 46 popisujeme deklaráciu použitých premenných pre generáciu náhodných čísel. Generátor potom generuje náhodné čísla z intervalu definovaného v premenných **distrX, distrY** a **distrAngle**. Pre vygenerovanie jedného čísla následne používame generátor takto: *distrAngle(generator)* a vygenerovaný uhol v radiánoch bude v intervale od 0 po 2*PI.

```
const double PI = 3.14159265358979323846264338327950;
std::random_device                  rand_dev;
std::mt19937                        generator(rand_dev());
std::uniform_real_distribution<double>  distrX(rangeFromX, rangeToX);
std::uniform_real_distribution<double>  distrY(rangeFromY, rangeToY);
std::uniform_real_distribution<double> distrAngle(0, 2*PI);
```

Obrázok 46 Deklarácia premenných pre generáciu náhodných čísel

7.2.5 Výpočet Pi – rovnobežky

Na výpočet jednej Pi pri zvolenej ploche rovnobežky a jednej iterácii je v metóde *štart()* volaná metóda *calculatePiParallelsSingleIteration()*. Prvým krokom v metóde je vytvorenie vektora rovnobežiek. Táto časť zdrojového kódu je zobrazená na obrázku 47. V cykle while sa do vektoru a taktiež do QVariantu **linesVariant_**, použitého na vykreslenie rovnobežiek na obrazovky v QML časti, ukladajú y súradnice s rozostupom rovnobežiek, ktorý je uložený v premennej **parallelRange_**.

```
QVector<double> lines;
int lineDistance = parallelRange_;
//save lines
while(lineDistance < canvasHeight_){
    lines.push_back(lineDistance);
    linesVariant_.append(lineDistance);
    lineDistance += parallelRange_;
}
```

Obrázok 47 Časť zdrojového kódu vektora rovnobežiek

Po vytvorení vektora rovnobežiek je nutné určiť hranice pre generáciu súradníc stredov ihiel. Aby vykreslená ihla nezasahovala mimo plochu obdĺžnika sme museli nastaviť hranice na veľkosť polovice ihly. V prípade, že by ihla bola rovnobežná s osou x alebo rovnobežná

s osou y, maximálna možná pozícia stredu by bola veľkosť plochy mínus polovica dĺžky ihly. V takomto prípade by sa ihla dotýkala okraja plochy a nepresiahla by mimo.

Ďalším krokom je samotný výpočet Pi po hodení ihly. Cyklus for iteruje cez všetky ihly a pre každú z nich generuje súradnice stredu **needleMiddleX** a **needleMiddleY**, a uhol **angle**. Súradnica x nie je pre samotný výpočet dôležitá. Generuje sa len pre zobrazenie ihly na obrazovku. Následne je pre zistenie pretnutia rovnobežky použitý algoritmus:

1. Metóda iteruje cez vektor súradníc y rovnobežiek
2. Súradnica y prvého krajného bodu ihly je *needleMiddleY – needleLength_/2 * sin(angle)*
3. Súradnica y druhého krajného bodu ihly je *needleMiddleY + needleLength_/2 * sin(angle)*
4. Ihla pretne rovnobežku práve vtedy keď:
    a. Súradnica y prvého krajného bodu je **menšia alebo rovná** ako súradnica iterovanej rovnobežky *lines[j]* a zároveň súradnica y druhého krajného bodu je **väčšia alebo rovná** ako súradnica iterovanej rovnobežky *lines[j]*
    b. Súradnica y prvého krajného bodu je **väčšia alebo rovná** ako súradnica iterovanej rovnobežky *lines[j]* a zároveň súradnica y druhého krajného bodu je **menšia alebo rovná** ako súradnica iterovanej rovnobežky *lines[j]*
5. V prípade pretnutia je iterácia vektorom rovnobežiek prerušená, keďže je jasné že ihla preťala rovnobežku a inkrementuje sa počet pretnutí

Po zistení či ihla pretla niektorú rovnobežku nasleduje samotný výpočet Pi podľa vzorca popísaného rovnicou [8]:

$$\pi = \frac{2 * needleLength * i}{parallelRange * crossings} \quad (13)$$

Kde *parallelRange* je vzdialenosť rovnobežiek, *needleLength* je veľkosť ihly, *i* je počet hodených ihiel a *crossings* je počet pretnutí.

Po vypočítaní Pi sa do korešpondujúcich QVariantov vložia *variant.append(value)* hodnoty pre vykreslenie ihly a hodnoty Pi. Modifikované QVarianty sú:

- crossingsVariant_ - pre nastavenie farby ihly
- xVariant_ a yVariant – pre súradnice stredu ihly

- rotationVariant_ - pre rotáciu ihly (v stupňoch)
- piVariant_ - pre hodnotu Pi po danom počte hodených ihiel

Po hodení všetkých ihiel je emitovaný signál *singleParallelsCalculationFinished()*. Po ukončení výpočtov sa začnú vykresľovať ihly na obrazovku a hodnota Pi bude aktualizovaná v závislosti na počte hodených ihiel.

7.2.6 Výpočet Pi – obdĺžniková mriežka

Na výpočet jednej Pi pri zvolenej ploche rovnobežky a jednej iterácii je v metóde *štart()* volaná metóda *calculatePiRectangleSingleIteration()*. Obdĺžniková mriežka rozširuje riešenie rovnobežiek o vertikálne rovnobežky. Prvým krokom je v metóde vycentrovanie obdĺžnikovej mriežky na stred plochy. Následne rovnakým spôsobom ako sme popísali v podkapitole 5.2.5 boli vytvorené dva vektory rovnobežiek. Jeden pre rovnobežky vertikálne a druhý pre rovnobežky horizontálne.

V ďalšom kroku prebieha iterácia cez všetky ihly a generácia súradníc rovnakým spôsobom ako v predchádzajúcej podkapitole. Algoritmus pretnutia je rozšírený o kontrolu pretnutia vertikálnych rovnobežiek. V prípade pretnutia horizontálnej rovnobežky je úspešne ukončený bez nutnosti skúmať vertikálne rovnobežky. Ak ihla nepretne ani jednu horizontálnu rovnobežku, vyhodnocuje sa pretnutie s niektorou z vertikálnych rovnobežiek takto:

1. Iterujeme cez všetky x súradnice vertikálnych rovnobežiek vo vektore
2. Súradnica y prvého krajného bodu ihly je *needleMiddleX – needleLength_/2 * cos(angle)*
3. Súradnica y druhého krajného bodu ihly je *needleMiddleX + needleLength_/2 * cos(angle)*
4. Ihla pretne rovnobežku práve vtedy keď
    a. Súradnica x prvého krajného bodu je **menšia alebo rovná** ako súradnica iterovanej rovnobežky *linesX[k]* a zároveň súradnica y druhého krajného bodu je **väčšia alebo rovná** ako súradnica iterovanej rovnobežky *linesX[k]*
    b. Súradnica y prvého krajného bodu je **väčšia alebo rovná** ako súradnica iterovanej rovnobežky *linesX[k]* a zároveň súradnica y druhého krajného bodu je **menšia alebo rovná** ako súradnica iterovanej rovnobežky *linesX[k]*

Ak ihla preťala ľubovoľnú rovnobežku, zvýši sa počet pretnutí a pristúpime k výpočtu Pi. Pri tomto variante riešenia rozlišujeme dva vzorce pre výpočet Pi.

- Prvý v prípade, že dĺžka ihly je väčšia ako strana B obdĺžnika. Tento vzorec je popísaný rovnicou [8]:

$$\pi = \frac{1}{rectangleB * rectangleA * crossings}\left(\left(rectangleB^2 + 2 * rectangleA * needleLength - 2 * rectangleA * \sqrt{needleLength^2 - rectangleB^2} + 2 * rectangleB * rectangleA * \cos^{-1}\left(\frac{rectangleB}{needleLength}\right)\right) * i\right)$$

(14)

Kde *rectangleB rectangleA* sú dĺžky strán obdĺžnika, *needleLength* je veľkosť ihly, *i* je počet hodených ihiel a *crossings* je počet pretnutí.

- Ak je strana B menšia alebo rovnaká ako veľkosť ihly, je vzorec pre výpočet Pi popísaný nasledovnou rovnicou [8]:

$$\pi = \frac{1}{rectangleB * rectangleA * crossings}\left((2 * rectangleB * needleLength + 2 * rectangleA * needleLength - needleLength^2) * i\right)$$

(15)

Po výpočte Pi sú uložené všetky potrebné dáta o ihle a samotná hodnota Pi do QVariantov. Metóda je opäť ukončená emitovaním signálu. Pre túto metódu sa emituje *singleRectangleCalculationFinished()*.

7.2.7 Výpočet Pi – rovnostranný trojuholník

Poslednou plochou v aplikácii je rovnostranný trojuholník. Pred popisom implementácie je nutné poznamenať, že musí byť splnená podmienka popísaná rovnicou:

$$needleLength \leq triangleA * \frac{\sqrt{3}}{2}$$

(16)

Kde *triangleA* je strana trojuholníka a *needleLength* veľkosť ihly. Pre jednu iteráciu je v metóde štart volaná metóda *calculatePiEquilTriangleSingleIteration()*. Na rozdiel od rovnobežiek a obdĺžnikovej mriežky sme pretnutie ihly skúmali len pri jednom trojuholníku so stranou o veľkosti uloženej v premennej **triangleA_**. Prvým krokom je získanie súradnice y

spodnej strany trojuholníka. Vypočítali sme ju takto: *bottomTriangleLineY_ = (canvasHeight_ - triangleHeight) / 2 + triangleHeight;*, kde *canvasHeight* je výška podkladového obdĺžnika pre vykresľovanie ihiel a *triangleHeight* je výška trojuholníka. Výšku trojuholníka sme zistili použitím Pytagorovej vety v tvare odvodenom na rovnici:

$$triangleHeight = \sqrt{triangleA^2 - \left(\frac{triangleA}{2}\right)^2} \quad (17)$$

Body trojuholníka A, B, C sme za pomoci súradnice y spodnej strany trojuholníka, šírky podkladového obdĺžnika a výšky trojuholníka odvodili tak, ako je zobrazené na obrázku 48. Na tomto obrázku sa nachádza aj časť zdrojového kódu pre určenie obmedzení súradníc generácie stredov ihiel.

```
auto xA = canvasWidth_ / 2 - triangleA_ / 2;
auto yA = bottomTriangleLineY_;
auto xC = canvasWidth_ / 2 + triangleA_ / 2;
auto yC = bottomTriangleLineY_;
auto xB = canvasWidth_ / 2;
auto yB = bottomTriangleLineY_ - triangleHeight;

auto rangeFromX = xA;
auto rangeFromY = (yB - needleLength_/2) > 0 ? yB : 0 + needleLength_/2;
auto rangeToX = xC;
auto rangeToY = (yA + needleLength_/2 <= canvasHeight_) ? yA : canvasHeight_ - needleLength_/2;
```

Obrázok 48 Zdrojový kód súradníc bodov trojuholníka

Generátor náhodných čísel nedokáže vygenerovať súradnice stredu ihly tak, aby sa nachádzali vždy v trojuholníku. Preto bolo nutné použiť iný prístup. V cykle while bol vygenerovaný bod so súradnicami x a y, s obmedzeniami definovanými a obrázku 48. Takýto bod je s určitosťou v obdĺžniku ohraničenom hodnotami **rangeFromX**, **rangeFromY**, **rangeToX** a **rangeToY**. Vygenerovaný bod sa nachádza v trojuholníku v prípade, ak je splnená podmienka v metóde *isPointInTriangle(xA, yA, xB, yB, xC, yC, needleMiddleX, needleMiddleY)*. Zdrojový kód tejto metódy je zobrazený na obrázku 49. Prevodom na barycentrické súradnice sme zistili či bod leží v trojuholníku. Ericson [21] sa odkazuje na to, že ak všetky tri barycentrické súradnice **a**, **b** a **c** majú hodnotu v intervale <0,1> nachádza sa bod v trojuholníku. Pre výpočet súradníc sme použili pomocnú premennú **denominator**. Súčet troch barycentrických súradníc musí byť rovný 1. Preto po vyčíslení súradníc **a**, **b** sme súradnicu **c** získali nasledovne *c = 1 – a – b;*

```
auto denominator = (triangleBy - triangleCy) * (triangleAx - triangleCx) + (triangleCx - triangleBx) * (triangleAy - triangleCy);
auto a = ((triangleBy - triangleCy) * (pointX - triangleCx) + (triangleCx - triangleBx) * (pointY - triangleCy)) / denominator;
auto b = ((triangleCy - triangleAy) * (pointX - triangleCx) + (triangleAx - triangleCx) * (pointY - triangleCy)) / denominator;
auto c = 1 - a - b;

return ((0 <= a && a <= 1) && (0 <= b && b <= 1) && ((0 <= c && c <= 1)));
```

Obrázok 49 Zdrojový kód metódy *isPointInTriangle()*

Po získaní súradníc všetkých ihiel nasleduje ďalší krok, ktorým je samotná iterácia všetkými ihlami. Pre každú ihlu je odvodená súradnica x *borderPointX = xVariant_[i].toDouble() - (needleLength_/2) * cos(angle)* a súradnica y *borderPointY = yVariant_[i].toDouble() - (needleLength_/2) * sin(angle)* prvého krajného bodu. Súradnice sú dosadené do metódy *isPointInTriangle(xA, yA, xB, yB, xC, yC, borderPointX, borderPointY)* a výsledok je uložený. Rovnaký postup bol vykonaný aj pre druhý krajný bod. V prípade, že aspoň jeden z krajných bodov sa nachádza mimo trojuholníka, ihla pretla jednu zo strán a inkrementuje sa hodnota pretnutí. Následne je vypočítané Pi podľa rovnice [8]:

$$\pi = \frac{3 * \sqrt{3} * needleLength * (4 * triangleA\ -\ needleLength) * i}{(3 * triangleA^{\,2} + 2 * needleLength^2) * crossings} \quad (18)$$

Kde *triangleA* je dĺžka strany trojuholníka, *needleLength* je veľkosť ihly, *i* je počet hodených ihiel a *crossings* je počet pretnutí. Na záver každej iterácie sú uložené potrebné hodnoty pre vykreslenie ihly a hodnota aktuálneho Pi do QVariantov a je emitovaný signál *singleEquilTriangleCalculationFinished()*.

7.2.8 Výpočet Pi pre viac iterácií

Metódy na výpočet viacerých iterácií fungujú na rovnakom princípe ako metódy jednej iterácie. Výhodou je, že sa ihly ani plocha nevykresľujú, a preto nie je nutné ukladať ich do QVariantov. Výpočet jedného experimentu je vsunutý do cyklu for, ktorý sa nachádza na obrázku 50.

```
for (auto iteration = 0; iteration < numberOfIterations_; iteration++){
```

Obrázok 50 Cyklus for pre iterácie

Pred každou iteráciou cyklu je nutné vyčistiť QVariant **Pi**, hodnotu premennej **pi_** a počet pretnutí. Výsledkom experimentu s viacerými iteráciami je hodnota najhoršieho, najlepšieho a priemerného Pi. Preto je nutné po každej iterácii odložiť QVariant s uloženými hodnotami Pi, získanými počas iterácie. Pre tento účel sme vytvorili vektor QVariantov **multipleIterationsPiVariantVector_**. Do vektora je vložený **piVariant_** po skončení každej iterácie.

Po skončení experimentu sa v metóde *calculateBestWorstAveragePi()*, na obrázku 51 dopočítajú najlepší a najhorší experiment a hodnota priemerného experimentu. Priemerné Pi je vypočítané ako aritmetický priemer výsledkov všetkých experimentov. Experiment s najväčšou odchýlkou od reálnej hodnoty Pi je vyhodnotený ako najhorší a naopak, ten s odchýlkou najmenšou je najlepší. Všetky tri hodnoty sú následne uložené ako QVarianty pre využitie v obrazovke Výsledok Experimentu a na vykreslenie grafov.

```
void Calculations::calculateBestWorstAveragePi()
{
    bestPiVariant_ = multipleIterationsPiVariantVector_[0];
    worstPiVariant_ = multipleIterationsPiVariantVector_[0];
    averagePi_ = 0.0;
    averagePiVariant_.clear();
    const double PI = 3.14159265358979323846264338327795;
    for(auto i = 0; i < multipleIterationsPiVariantVector_.size(); i++){
        averagePi_ += multipleIterationsPiVariantVector_[i].back().toDouble();
        if( std::abs(PI - multipleIterationsPiVariantVector_[i].back().toDouble()) > std::abs(PI - worstPiVariant_.back().toDouble()))
            worstPiVariant_ = multipleIterationsPiVariantVector_[i];
        else if( std::abs(PI - multipleIterationsPiVariantVector_[i].back().toDouble()) < std::abs(PI - bestPiVariant_.back().toDouble()))
            bestPiVariant_ = multipleIterationsPiVariantVector_[i];
    }
    averagePi_ = averagePi_ / multipleIterationsPiVariantVector_.size();

    auto helpAverage = 0.0;
    for(auto length = 0; length <  multipleIterationsPiVariantVector_.size(); length++){
        helpAverage += multipleIterationsPiVariantVector_[length].back().toDouble();
        averagePiVariant_.append(helpAverage/(length+1));
    }
}
```

Obrázok 51 Zdrojový kód metódy *calculateBestWorstAveragePi()*

Na záver je potrebné emitovať príslušný signál pre pokračovanie aplikácie.

7.3 Teoretický úvod

Aplikáciu sme implementovali ako pomôcku pre študentov na pochopenie problematiky Buffonovej úlohy o ihle. Z tohto dôvodu sa v aplikácii nachádza aj časť obsahujúca teoretické informácie. Táto časť je prístupná z obrazovky Hlavné Menu po stlačení tlačidla „Teoretický úvod“. Obrazovku sme navrhli na princípe viacerých stránok, medzi ktorými sa dá jednoducho prepínať tlačidlami.

7.4 Videoukážka

V aplikácii je možnosť pozrieť si video tutoriál, ktorý ozrejmí funkčnosť a ovládanie aplikácie. Video sme nahrali pomocou windows aplikácie **Xbox Game Bar** [22]. Aplikácia umožňuje nahrať okno zvolenej aplikácie, v tomto prípade aplikácie Buffonovej úlohy. Nahrané video je vo formáte .mp4. Toto video sme pridali ako zdroj priamo do aplikácie. V adresárovej štruktúre je uložené v zložke \Gui\Videos. Video je možné prehrať po stlačení tlačidla „Videoukážka“ na obrazovke Hlavné Menu. Na zásobník obrazoviek je vložená

obrazovka Videoukážka. Skladá sa z nadpisu a troch tlačidiel. Tlačidlo „X“ pre návrat do obrazovky Hlavné Menu a tlačidlá „Spustiť“ a „Pauza“ slúžiacich na obsluhu videa. Pre funkciu videa sa v QML používa objekt **Video**, ktoré má ako zdroj nahrávku *source: "qrc:/Gui/Videos/videoTutorial.mp4"*. Video sa spúšťa pomocou funkcie *video.play()* a pozastavuje pomocou funkcie *video.pause()*.

7.5 Vytvorenie aplikácie a prenositeľnosť na iné zariadenia

Všetky potrebné informácie k zostaveniu aplikácie sa nachádzajú v projektovom **.pro** súbore, vytvorenom spolu s projektom. Qt Creator je schopný automaticky modifikovať projektový súbor. Po pridaní novej triedy cez grafické rozhranie je hlavičkový aj zdrojový súbor automaticky pridaný do .pro súboru so zachovanou štruktúrou. V aplikácii Buffonovej ihly nebolo nutné robiť takmer žiadne úpravy tohto súboru. Jedinou zmenou, ktorá nie je povinná, bolo pridanie ikony ihly pomocou parametra RC_ICONS. Ako ikonu sme zvolili obrázok ihly, preformátovaný do formátu .ico. [23]

7.5.1 Qmake

V Qt aplikáciách nie je nutné vytvárať takzvané makefily. Tento proces je automatizovaný a o všetky náležitosti sa stará utilita qmake. Čerpá z informácií v projektovom súbore .pro. [24]

7.5.2 Prenositenosť

Pre spustiteľnosť aplikácie bežiacej pod frameworkom Qt sú nutné dynamicky linkované knižnice **dll** (dynamic linked libraries). Tieto knižnice sú uložené v adresári inštalácie Qt. Jednou z možností ako umožniť spustenie aplikácie na inom zariadení je nainštalovať na zariadení rovnakú verziu Qt a pridať ju do systémovej premennej prostredia **Path**. V takom prípade budú dll knižnice načítané priamo z adresára Qt. Avšak nie všetky zariadenia majú možnosť inštalácie Qt. Z tohto dôvodu sme zvolili prístup nakopírovania všetkých potrebných knižníc priamo do zložky k spustiteľnému **.exe** súboru aplikácie. Pre tento prístup má Qt framework vytvorenú vlastnú utilitu **windeployqt**. Táto utilita nakopíruje všetky potrebné závislosti priamo do zložky .exe súboru. Následne je možné celú zložku preniesť na iné zariadenie a spustiť aplikáciu. Utilitu windeployqt sme spúšťali v nasledovnom formáte: *windeployqt.exe --qmldir "\qml_cesta" "\súbor.exe"*.

8 EXPERIMENTY NA VÝPOČET LUDOLFOVHO ČISLA

V ôsmej kapitole rozoberieme výsledky experimentov realizovaných na vytvorenej aplikácii. Experimenty sme realizovali na všetkých troch podporovaných plochách. Cieľom experimentov bolo overiť a dokázať funkčnosť aplikácie a následne parametre najlepšieho experimentu zvoliť ako defaultné hodnoty.

8.1 Experimenty na ploche rovnobežky

8.1.1 Experimenty podobnosti výsledku v prípade rovnosti pomerov parametrov

Z testovania pri vývoji aplikácie sme zistili, že samotný pomer vzdialenosti rovnobežiek a veľkosti voči výške plochy nemá na experiment žiaden vplyv. Dôvodom je zvýšenie počtu rovnobežiek (väčšia šanca pretnutia) pri zmenšení ich vzdialenosti a naopak. To znamená, že experiment s parametrami:

- Vzdialenosť rovnobežiek = 100
- Veľkosť ihly = 100
- Počet ihiel = 5000

A experiment s parametrami:

- Vzdialenosť rovnobežiek = 40
- Veľkosť ihly = 40
- Počet ihiel = 5000

dosahovali približne rovnaké výsledky. V tabuľkách 1 a 2 sme porovnávali výsledok desiatich experimentov pri zadaných parametroch a overovali pravdivosť tvrdenia.

Tabuľka 1 Experimenty na ploche rovnobežky rovnosť paramerov číslo 1

Experiment číslo	**Vzdialenosť rovnobežiek**	**Veľkosť ihly**	**Počet ihiel**	**Výsledné Pi**	**Odchýlka**
1	100	100	5000	3,14861465	0,223517%
2	100	100	5000	3,19386768	1,663966%
3	100	100	5000	3,15457416	0,413214%
4	100	100	5000	3,12500000	0,528161%
5	100	100	5000	3,09693408	1,421527%
6	100	100	5000	3,12109852	0,652349%
7	100	100	5000	3,10655475	1,115291%
8	100	100	5000	3,10752010	1,084563%
9	100	100	5000	3,13873196	0,091059%
10	100	100	5000	3,15955758	0,571841%
Priemerné Pi/Odchýlka				**3,135245348**	**0,7765488%**

Tabuľka 2 Experimenty na ploche rovnobežky rovnosť paramerov číslo 2

Experiment číslo	**Vzdialenosť rovnobežiek**	**Veľkosť ihly**	**Počet ihiel**	**Výsledné Pi**	**Odchýlka**
1	40	40	5000	3,10752010	1,084563%
2	40	40	5000	3,11817908	0,745277%
3	40	40	5000	3,13185096	0,310088%
4	40	40	5000	3,13479614	0,216340%
5	40	40	5000	3,15258503	0,349898%
6	40	40	5000	3,14564323	0,128934%
7	40	40	5000	3,07503080	2,118729%
8	40	40	5000	3,13577915	0,185050%
9	40	40	5000	3,09501696	1,482550%
10	40	40	5000	3,11138773	0,961453%
Priemerné Pi/Odchýlka				**3,120778918**	**0,7582882%**

Výsledkami experimentov sme dokázali, že pri konštantnej rovnakej experiment s rovnakým pomerom veľkosti ihly a vzdialenosti rovnobežiek je možné bez straty presnosti hodnoty Pi zmenšovať/zväčšovať oba parametre o rovnakú veľkosť.

8.1.2 Experimenty s malou ihlou

Ďalšími experimentami sme dokázali, že v prípade zvolenia čo najmenšej možnej veľkosti ihly a najväčšej vzdialenosti rovnobežiek, bol výsledok s veľkou odchýlkou. Parametre experimentov sme zvolili nasledovne:

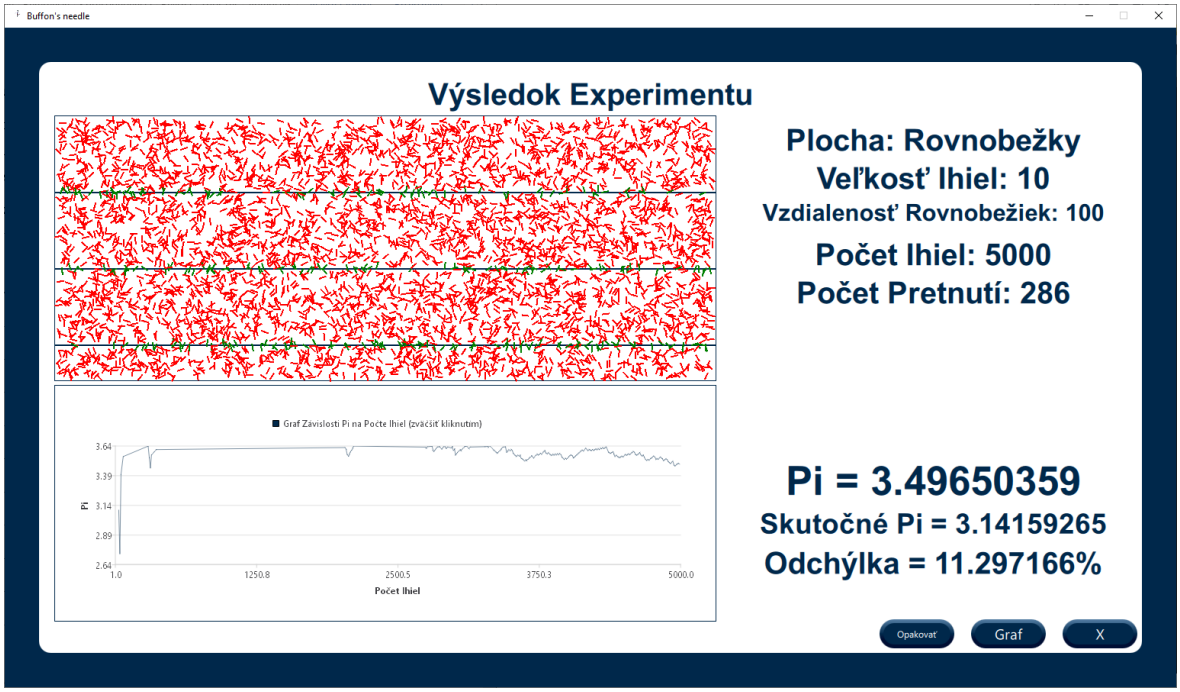
- Vzdialenosť rovnobežiek = 100
- Veľkosť ihly = 10
- Počet ihiel = 5000

Výsledky experimentov je možné vidieť v nasledujúcej tabuľke.

Tabuľka 3 Experimenty na ploche rovnobežky malá ihla

Experiment číslo	**Vzdialenosť rovnobežiek**	**Veľkosť ihly**	**Počet ihiel**	**Výsledné Pi**	**Odchýlka**
1	100	10	5000	3,53356886	12,476990%
2	100	10	5000	3,43642616	9,384842%
3	100	10	5000	3,38983059	7,901659%
4	100	10	5000	3,55871892	13,277541%
5	100	10	5000	3,59712219	14,499956%
6	100	10	5000	3,57142854	13,682101%
7	100	10	5000	3,49650359	11,297166%
8	100	10	5000	3,3333325	6,103293%
9	100	10	5000	3,52112675	12,080945%
10	100	10	5000	3,52112675	12,080945%
Priemerné Pi/Odchýlka				**3,495918485**	**11,2785438%**

Priemerná odchýlka po desiatich opakovaniach experimentu vyšla **11,2785438%**. Môžeme konštatovať, že experiment s takýmto rozdielom hodnôt parametrov nemá pre odvodenie hodnoty Ludolfovho čísla zmysel. Výsledok experimentu číslo 7 je možné vidieť na obrázku 52.



Obrázok 52 Výsledok experimentu s malou veľkosťou ihly

8.1.3 Experimenty s polovičnou hodnotou veľkosti ihly

V podkapitole 6.1.2 sme dokázali, že experiment s malou ihlou je veľmi nepresný. Ako ďalší experiment sme preto vybrali ihly s polovičnou veľkosťou ako vzdialenosť rovnobežiek. Experimenty sú s rôznym počtom ihiel. Výsledky experimentov sa nachádzajú v tabuľke 4. Parametre experimentov sme zvolili nasledovne:

- Vzdialenosť rovnobežiek = 60
- Veľkosť ihly = 30
- Počet ihiel = 2050

Tabuľka 4 Experimenty na ploche rovnobežky polovičná ihla číslo 1

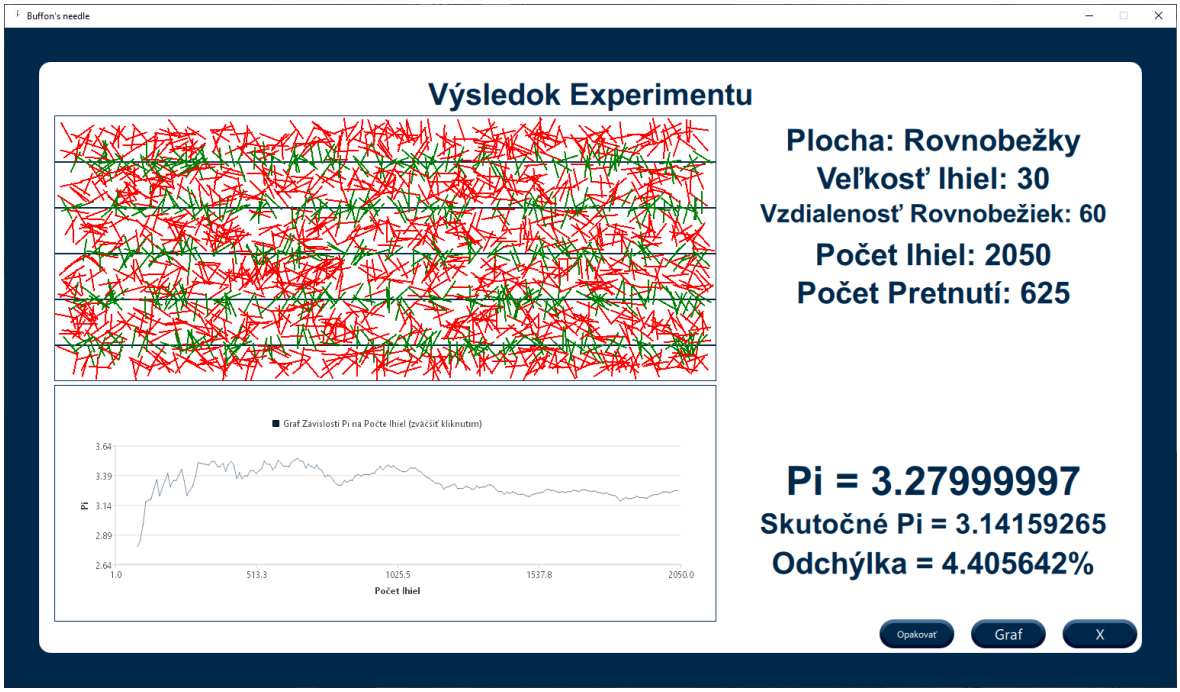
Experiment číslo	**Vzdialenosť rovnobežiek**	**Veľkosť ihly**	**Počet ihiel**	**Výsledné Pi**	**Odchýlka**
1	60	30	2050	3,38283825	7,679086%
2	60	30	2050	3,48639464	10,975388%
3	60	30	2050	3,6617398	7,148646%
4	60	30	2050	3,38842964	7,857065%
5	60	30	2050	3,27999997	4,405642%
6	60	30	2050	3,37726521	7,501697%
7	60	30	2050	3,37171054	7,324880%
8	60	30	2050	3,30112720	5,078142%
9	60	30	2050	3,34420872	6,449470%
10	60	30	2050	3,18322992	1,325355%
Priemerné Pi/Odchýlka				**3,377694389**	**6,5745371**%

Pre ďalšiu sériu experimentov sme zvýšili počet ihiel na 7000. Výsledky je možné vidieť v nasledujúcej tabuľke.

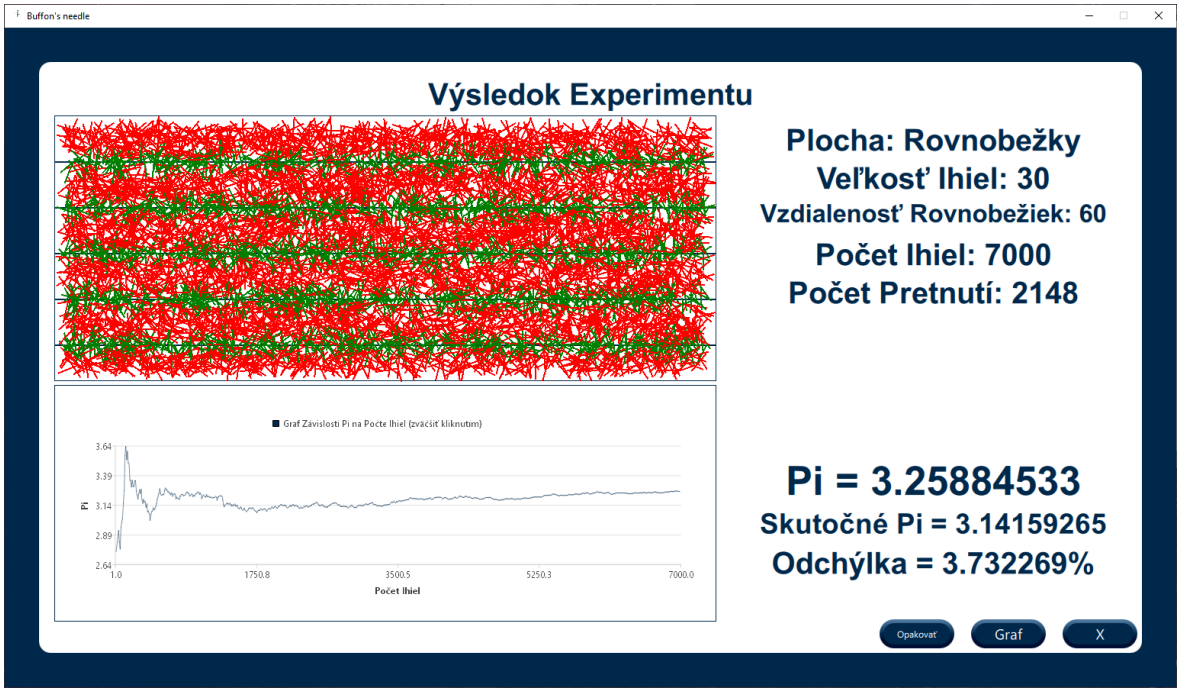
Tabuľka 5 Experimenty na ploche rovnobežky polovičná ihla číslo 2

Experiment číslo	**Vzdialenosť rovnobežiek**	**Veľkosť ihly**	**Počet ihiel**	**Výsledné Pi**	**Odchýlka**
1	60	30	7000	3,25884533	3,732269%
2	60	30	7000	3,34128881	6,356526%
3	60	30	7000	3,34448171	6,458159%
4	60	30	7000	3,28022504	4,412806%
5	60	30	7000	3,38818979	7,849431%
6	60	30	7000	3,43305540	9,277547%
7	60	30	7000	3,41796875	8,797324%
8	60	30	7000	3,33016181	6,002343%
9	60	30	7000	3,30969262	5,350788%
10	60	30	7000	3,25884533	3,732269%
Priemerné Pi/Odchýlka				**3,336275459**	**6,1969462**%

Z tabuliek 4 a 5 môžeme vidieť, že počet ihiel nijak výrazne nezlepšil výsledok experimentov. Pre porovnanie podobnosti výsledkov sme vytvorili obrázky 53 a 54, na ktorých sa nachádzajú výsledky vybraného experimentu pri počte ihiel 2050 a 7000. Experimenty s polovičnou hodnotou veľkosti ihiel voči vzdialenosti rovnobežiek priniesli viditeľné zlepšenie oproti experimentom s malou ihlou.



Obrázok 53 Výsledok experimentu pri 2050 ihlách



Obrázok 54 Výsledok experimentu pri 7000 ihlách

8.1.4 Experimenty 2/3 veľkosťou ihly

Pre posledné optimalizačné experimenty sme zvolili ihlu s veľkosťou dvoch tretín veľkosti vzdialenosti rovnobežiek. Experimenty s výsledkami v tabuľke 6 sme vykonali pre 7000 ihiel a ostatné parametre sme definovali nasledovne:

- Vzdialenosť rovnobežiek = 60
- Veľkosť ihly = 30

Tabuľka 6 Experimenty na ploche rovnobežky 2/3 ihla

Experiment číslo	Vzdialenosť rovnobežiek	Veľkosť ihly	Počet ihiel	Výsledné Pi	Odchýlka
1	60	40	7000	3,22841001	2,763482%
2	60	40	7000	3,26454473	3,913686%
3	60	40	7000	3,19634700	1,742885%
4	60	40	7000	3,24412012	3,263551%
5	60	40	7000	3,23624587	3,012906%
6	60	40	7000	3,12748766	0,448976%
7	60	40	7000	3,13176179	0,312926%
8	60	40	7000	3,05759406	2,673758%
9	60	40	7000	3,13497496	0,210648%
10	60	40	7000	3,10839844	1,056605%
Priemerné Pi/Odchýlka				**3,172988464**	**1,9399423**%

8.1.5 Experimenty s použitím viacerých iterácií

Parametre pre experimenty s viacerými iteráciami sme zvolili na základe výsledkov experimentov v prechádzajúcich podkapitolách. Pre prvú sériu experimentov, ktorých výsledky sa nachádzajú v tabuľke 7, sme zvolili nasledujúce parametre:

- Vzdialenosť rovnobežiek = 40

- Veľkosť ihly = 40
- Počet ihiel = 5000
- Počet iterácií = 1000

Tabuľka 7 Experimenty na ploche rovnobežky viac iterácií

Experiment číslo	**Počet iterácií**	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	1000	3,1416902542	0,00310672%	3,00571084	3,11118149
2	1000	3,1416902542	0,00310672%	2,99132513	3,11215543
3	1000	3,1416902542	0,00310672%	3,01841235	3,11056900
4	1000	3,1416902542	0,00310672%	3,00480771	3,11200857
5	1000	3,1416902542	0,00310672%	3,00661444	3,11423945
Priemerné Pi/Odchýlka		**3,1416902542**	**0,00310672**%	3,00537409	3,11203079

Môžeme vidieť, že pri vybraných parametroch a 1000 iteráciách sme dokázali nájsť najlepšie možné Pi. V ďalších experimentoch sme skúsili znížiť počet iterácií na 108. Výsledok experimentov sa nachádza v tabuľke 8 a 192 iterácií v tabuľke 9.

Tabuľka 8 Experimenty na ploche rovnobežky 108 iterácií

Experiment číslo	**Počet iterácií**	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	108	3,1407034397	0,02830455%	3,02571868	3,10875415
2	108	3,1397173404	0,05969307%	3,02755069	3,10882878
3	108	3,1407034329	0,0283045%	3,13150234	3,11355447
4	108	3,1416902542	0,00310672%	3,03766703	3,10584499
5	108	3,1416902542	0,00310672%	3,04599452	3,11365032
Priemerné Pi/Odchýlka		**3,1409009443**	**0,02450311**%	3,05368665	3,11012654

Tabuľka 9 Experimenty na ploche rovnobežky 192 iterácií

Experiment číslo	**Počet iterácií**	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	192	3,1416902542	0,00310672%	3,01841235	3,11222767
2	192	3,1416902542	0,00310672%	3,03766703	3,11244130
3	192	3,1416902542	0,00310672%	3,00571084	3,10919833
4	192	3,1416902542	0,00310672%	3,01477241	3,10986304
5	192	3,1416902542	0,00310672%	3,03030300	3,11314582
Priemerné Pi/Odchýlka		**3,1416902542**	**0,00310672**%	3,02137313	3,11137523

Z experimentov sme zistili, že najlepšie možné Pi dokážeme s určitosťou určiť už pri približne 200 iteráciách. Priemerné Pi sa s počtom iterácií zlepšuje. Hodnota najhoršieho Pi sa s pribúdajúcimi iteráciami zhoršuje.

8.1.6 Experimenty pre získanie najlepšej hodnoty Pi

V podkapitole 6.1.5 sme zistili, že pri počte iterácií 1000 s určitosťou nájdeme najlepšie možné Pi pri daných parametroch. V nasledujúcom experimente sme našli najlepšiu možnú hodnotu Ludolfovho čísla pri vzdialenosti rovnobežiek 100 a postupnom znižovaní veľkosti ihly. Výsledky experimentov sú zaznamenané v tabuľke 10 a najpresnejšia hodnota Pi je vyznačená.

Tabuľka 10 Experimenty na ploche rovnobežky najlepšie Pi

Experiment číslo	Počet iterácií	Vzdialenosť Rovnobežiek	Veľkosť ihly	Najlepšie Pi	Odchýlka Najlepšieho
1	1000	100	100	3,1416902542	0,00310672%
2	1000	100	99	3,1418597698	0,00850257%
3	1000	100	98	3,1420326232	0,01400467%
4	1000	100	97	3,1411917209	0,01276208%
5	1000	100	96	3,1413612365	0,00736623%
6	1000	100	95	3,1415343284	0,00185654%
7	1000	100	94	3,1417112350	0,00377456%
8	1000	100	93	3,1418919563	0,00952710%
9	1000	100	92	3,1420764923	0,01540106%
10	1000	100	91	3,1411805155	0,01311876%

Najlepší experiment s odchýlkou len 0,00185654% a hodnotou Pi presnou na 4 desatinné miesta má parametre:

- Vzdialenosť rovnobežiek = 100
- Veľkosť ihly = 95
- Počet ihiel = 5000
- Počet iterácií = 1000

Pridaním počtu ihiel na 10000 a zachovaním ostatných parametrov nám vyšiel výsledok rovnaký.

8.2 Experimenty na ploche obdĺžniková mriežka

V podkapitole 6.1 sme vykonali experimenty pri jednej ako aj pri viacerých iteráciách. Presnejšie výsledky sú podľa očakávania pri väčšom počte iterácií. Preto v tejto podkapitole sme pre experimenty využívali možnosť väčšieho počtu iterácií. Dôkaz podobnosti nie je nutný, keďže sme ho dokázali už v podkapitole 6.1.1.

Na rozdiel od plôch rovnobežky a rovnostranný trojuholník vyžadujú experimenty na obdĺžnikovej mriežke dva parametre plochy, stranu A a stranu B obdĺžnika. Podmienky spustiteľnosti experimentu sú nasledovné:

- Strana A musí byť väčšia alebo rovná ako strana B
- Veľkosť ihly musí byť menšia ako strana A

8.2.1 Experimenty s malou ihlou

V úvodných experimentoch sme dokázali veľkú chybovosť pri zvolení najväčších veľkostí strán a naopak čo najmenšej veľkosti ihly. Výsledok experimentov s hodnotami:

- Strana A = 100
- Strana B = 100
- Veľkosť ihly = 10
- Počet ihiel = 10000
- Počet iterácií = 1000

Môžeme vidieť v nasledujúcej tabuľke.

Tabuľka 11 Experimenty na ploche obdĺžnik malá ihla číslo 1

Experiment číslo	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	3,4031412601	8,32535071%	4,05827283	3,72696685
2	3,4090909957	8,51473668%	4,14452695	3,72730636
3	3,3562822341	6,83378159%	4,08376979	3,73729324
4	3,4090909957	8,51473668%	4,41012718	3,73205709
5	3,4240562915	8,99109684%	4,11826801	3,73347759
Priemerné Pi/Odchýlka	**3,4003323554**	**8,2359405**%	4,16299295	3,73142023

Odchýlka najlepšieho Pi v experimentoch sa pohybovala v hodnotách okolo 8%, čo je pomerne nepresný výsledok. Odchýlka najhoršieho Pi sa pohybovala v rozmedzí 28-30% a odchýlka priemerného Pi vyšla približne 18%. Experimenty sme preto vyhodnotili ako nepresné.

V druhej sade experimentov s malou ihlou sme menili hodnotu veľkosti strany B a zisťovali zlepšenie či zhoršenie výsledku. Konštantné parametre experimentov sme zvolili takto:

- Strana A = 100
- Veľkosť ihly = 10
- Počet ihiel = 10000
- Počet iterácií = 1000

V tabuľke 12 môžeme porovnávať výsledok pri meniacej sa hodnote parametra veľkosti strany B.

Tabuľka 12 Experimenty na ploche obdĺžnik malá ihla číslo 2

Experi-ment číslo	Strana B	Najlepšie Pi	Odchýlka Najlepšieho	Najhoršie Pi	Priemerné Pi
1	100	3,3942558765	8,04252017%	4,13135576	3,73360061
2	90	3,3074104785	5,27814530%	3,96825385	3,61368727
3	80	3,3473603725	6,54978992%	3,99908590	3,63259410
4	70	3,2179424762	2,43029033%	3,73260951	3,46704673
5	60	3,1952173709	1,70692776%	3,72238230	3,43793964
6	50	3,1453361511	0,11915922%	3,64550590	3,35013914
7	40	3,1380753517	0,11195919%	3,56765317	3,33099937
8	30	3,1422824859	0,02195804%	3,46356320	3,27540659
9	20	3,1412181854	0,01191969%	3,36848282	3,22033739
10	10	3,1413612365	0,00736623%	3,23475050	3,14795851

Z experimentov, kde sme menili parameter strana B sme zistili, že pri menšej veľkosti strany B sa spresňuje výsledok experimentu. Najpriaznivejšiu zmenu sme pozorovali pri zmene veľkosti zo 60 na 50.

8.2.2 Experimenty s polovičnou ihlou

Ako ďalšiu sadu experimentov sme zvolili konštantné parametre nasledovne:

- Strana A = 100
- Veľkosť ihly = 50
- Počet ihiel = 10000
- Počet iterácií = 1000

V tabuľke 13 môžeme sledovať zmeny hodnoty Pi v závislosti na upravovaní veľkosti strany B. Predpokladali sme, že výsledky mali vychádzať o niečo presnejšie ako v experimentoch s malou ihlou.

Tabuľka 13 Experimenty na ploche obdĺžnik polovičná ihla

Experiment číslo	Strana A	Strana B	Najlepšie Pi	Odchýlka Najlepšieho	Najhoršie Pi	Priemerné Pi
1	100	100	3,2371439937	3,04149360%	3,44623875	3,34061574
2	100	90	3,1414210796	0,00546136%	3,28319001	3,17672729
3	100	80	3,1887755393	1,50187790%	3,36956524	3,26665210
4	100	70	3,1413838863	0,00664526%	3,23964428	3,17481684
5	100	60	3,1415805816	0,000384261%	3,21566390	3,15252733
6	100	50	3,1414928436	0,00317704%	3,19815778	3,15081715
7	100	40	3,1415903568	0,000073108%	3,10134148	3,14080643
8	100	30	3,1417117118	0,00378974%	3,18714094	3,14857959
9	100	20	3,1414330000	0,00508191%	3,17472243	3,14894343
10	100	10	3,1415128707	0,00253956%	3,15266776	3,14124202

Experimentami s polovičnou ihlou na ploche obdĺžniková mriežka sa nám podarilo dosiahnuť presnosť Pi na 5 desatinných miest. Parametre konkrétneho experimentu sú nasledujúce:

- Strana A = 100
- Strana A = 40
- Veľkosť ihly = 50
- Počet ihiel = 10000
- Počet iterácií = 1000

V ďalších experimentoch presnosť oproti experimentu s polovičnou ihlou klesala, a preto sme ich v práci neuviedli.

8.3 Experimenty na ploche rovnostranný trojuholník

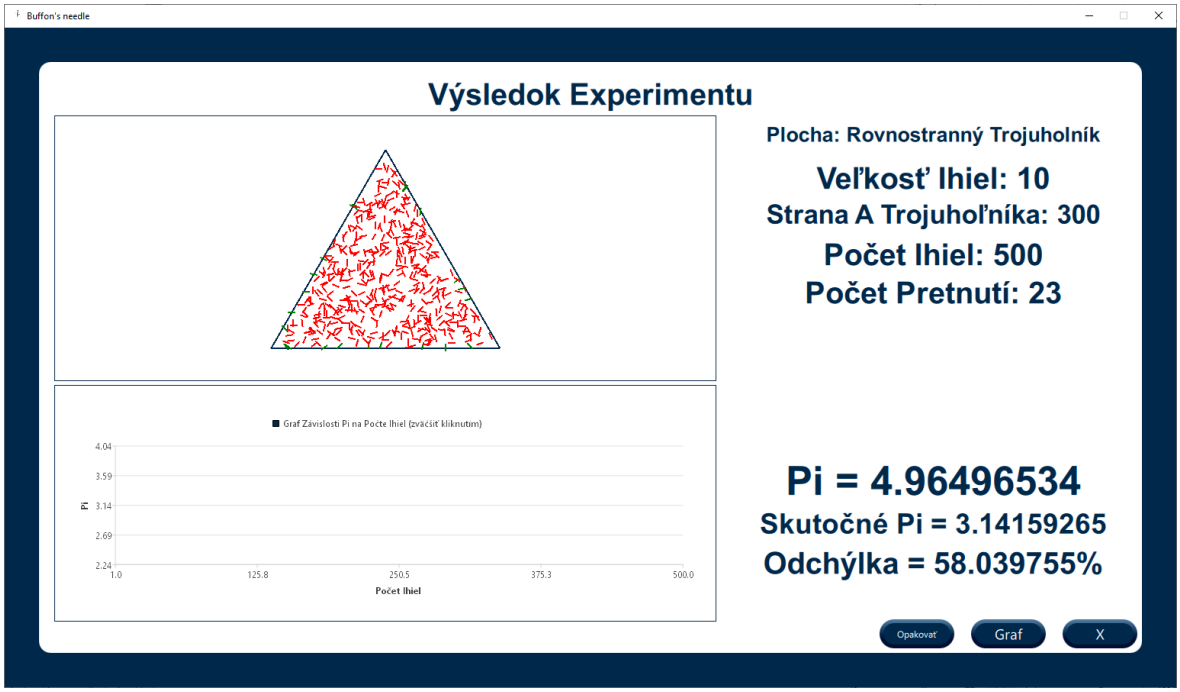
Posledným typom experimentov, ktoré sme skúmali boli experimenty na ploche rovnostranný trojuholník. Plocha je špecifická tým, že na rozdiel od prechádzajúcich plôch, je vykreslený len jeden objekt plochy. Maximálnym počtom ihiel je 1000. Experimenty na dokázanie podobnosti výsledkov v prípade rovnosti pomerov parametrov je nutné upraviť, z dôvodu obmedzenia maximálnej hodnoty veľkosti ihly, teda veľkosti ťažnice trojuholníka (ihla musí byť menšia ako strana A krát odmocnina z 3). Modifikované tvrdenie stále platí rovnako ako sme dokázali v podkapitole 6.1.1, a preto nebolo nutné experimenty opakovať.

8.3.1 Experimenty s malou ihlou

Predpokladaným výsledkom experimentov s malou ihlou bola pomerne vysoká nepresnosť. Toto tvrdenie sme overili v tabuľke 14. Vstupné parametre experimentov boli nasledovné:

- Vzdialenosť strany A = 300
- Veľkosť ihly = 10
- Počet ihiel = 500
- Počet iterácií = 1000

Výsledok jedného experimentu pre lepšiu vizualizáciu čitateľa možno vidieť na obrázku 55. Počet pretnutí bol v porovnaní s celkovým počtom ihiel veľmi malý.



Obrázok 55 Experiment malej ihly na ploche rovnostranný trojuholník

Hodnota Pi vyšla v experimente s vysokou odchýlkou.

Tabuľka 14 Experimenty na ploche trojuhulník malá ihla

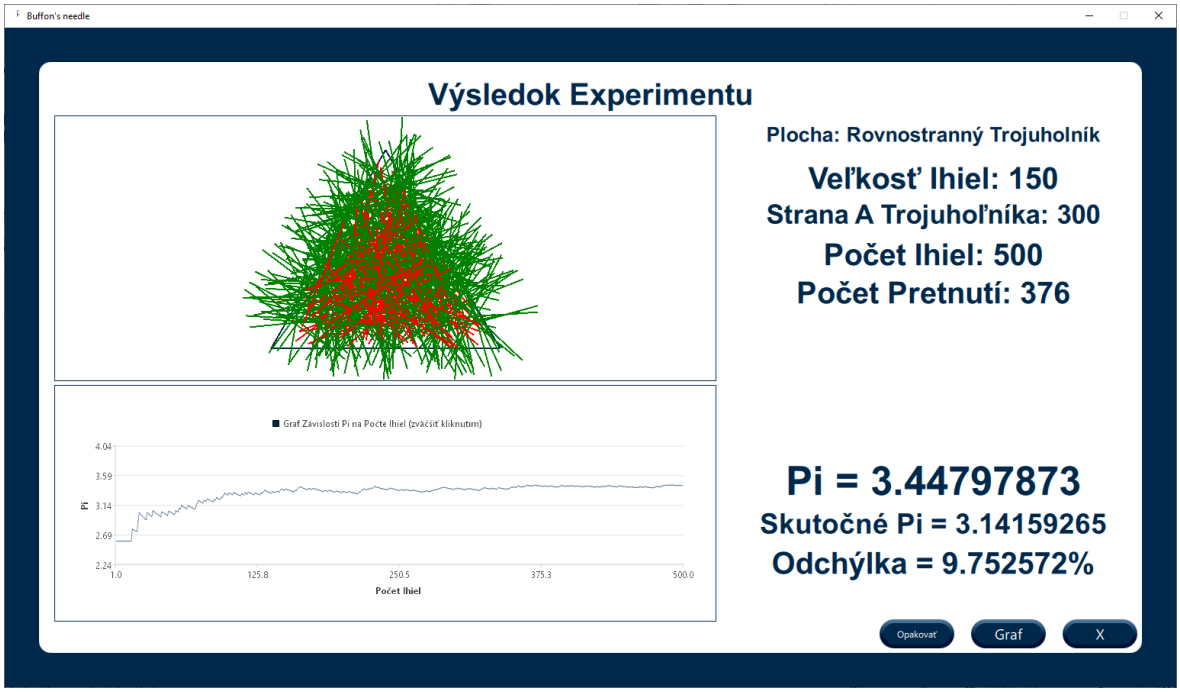
Experiment číslo	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	3,1720612049	0,96984411%	6,34412240	3,47197818
2	3,1720612049	0,96984411%	6,34412240	3,45143985
3	3,1720612049	0,96984411%	8,78416919	3,46062636
4	3,1720612049	0,96984411%	7,61294651	3,44925141
5	3,1720612049	0,96984411%	7,13713741	3,48896074
Priemerné Pi/Odchýlka	**3,1720612049**	**0,96984411**%	7,24449958	3,46445131

Odchýlka najlepšieho Pi bola pod hodnotou jedného percenta. Avšak odchýlka priemerného Pi sa pohybovala v rozmedzí 10-11%. Hodnota najhoršieho Pi dosiahla veľmi zlé hodnoty s odchýlkou presahujúcou 100%. Predpoklad vysokej nepresnosti sme potvrdili.

8.3.2 Experimenty s polovičnou ihlou

Pre lepšiu vizuálnu predstavu sa na obrázku 56 nachádza experiment s parametrami použitými v nasledujúcich experimentoch. Parametre boli nasledujúce:

- Vzdialenosť strany A = 300
- Veľkosť ihly = 150
- Počet ihiel = 500



Obrázok 56 Experiment polovičnej ihly na ploche rovnostranný trojuholník

Výsledkom jedného experimentu bolo Pi s odchýlkou takmer 10%. Predpokladom bolo, že odchýlka priemerného a najhoršieho Pi bude výrazne menšia ako pri malej ihle. Zároveň sme očakávali aj zlepšenie odchýlky najlepšieho Pi. V tabuľke 15 je možné vidieť výsledky experimentov pri počte iterácií 1000.

Tabuľka 15 Experimenty na ploche trojuhulník polovičná ihla

Experiment číslo	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	3,1390800476	0,07997873%	3,74693655	3,44895100
2	3,1314976215	0,32133485%	3,88155698	3,44700455
3	3,1466991901	0,16254610%	3,72540235	3,44677186
4	3,2169728279	2,39942547%	3,80187702	3,44900631
5	3,1775491237	1,14452999%	3,76872110	3,44453191
Priemerné Pi/Odchýlka	**3,1623597622**	**0,82156303%**	3,7848988	3,44725313

Z výsledkov experimentov môžeme konštatovať, že odchýlka priemerného Pi sa veľmi nezmenila, náš predpoklad bol chybný. V prípade najhoršieho Pi bol predpoklad správny. Hodnota najhoršieho Pi sa pohybovala okolo 20%. Zaujímavým bolo ale najlepšie Pi. To sa síce priemerne zlepšilo, ale z experimentov sme zistili, že pri počte iterácií 1000 nie je možné v každom experimente nájsť najlepší možný výsledok.

8.3.3 Experimenty s najväčšou možnou ihlou

Ďalšími experimentami sme overili presnosť hodnoty Pi pri najväčšej možnej ihle. Výsledky experimentov je možné vidieť v tabuľke 16. Zvolené parametre pre experimenty boli tieto:

- Vzdialenosť strany A = 300
- Veľkosť ihly = 259
- Počet ihiel = 500
- Počet iterácií = 1000

Tabuľka 16 Experimenty na ploche trojuhulník najväčšia ihla

Experiment číslo	**Najlepšie Pi**	**Odchýlka Najlepšieho**	**Najhoršie Pi**	**Priemerné Pi**
1	3,1396915912	0,06051269%	3,21060872	3,15102243
2	3,1396915912	0,06051269%	3,19747734	3,15084910
3	3,1396915912	0,06051269%	3,21060872	3,15126758
4	3,1396915912	0,06051269%	3,21060872	3,15133476
5	3,1396915912	0,06051269%	3,19747734	3,15176248
Priemerné Pi/Odchýlka	**3,1396915912**	**0,06051269%**	3,20535617	3,15124727

Z výsledku experimentov môžeme konštatovať, že pri veľkých ihlách boli experimenty najpresnejšie. V poslednej sade experimentov sme sa pokúsili zistiť hodnotu najlepšieho Pi. Veľkosť strany A bola 300, počet iterácií (1000) a počet ihiel (500) ostal nezmenený, a veľkosť ihly sme znižovali, až kým sa odchýlka nezačala výraznejšie zhoršovať. V tabuľke 17

je možné vidieť výsledky experimentov a vyznačený experiment s najlepšou hodnotou Ludolfovho čísla.

Tabuľka 17 Experimenty na ploche trojuhulník najlepšia veľkosť ihly

Experiment číslo	Počet ite-rácií	Veľkosť strany A	Veľkosť ihly	Najlepšie Pi	Odchýlka Najlepšieho
1	1000	300	259	3,1396915912	0,06051269%
2	1000	300	258	3,1389234066	0,08496477%
3	1000	300	257	3,1444084644	0,08963004%
4	1000	300	256	3,1435158252	0,06121645%
5	1000	300	255	3,1425611972	0,03082970%
6	1000	300	254	3,1415441036	0,00154539%
7	1000	300	253	3,1404640674	0,03592401%
8	1000	300	252	3,1393206119	0,07232133%
9	1000	300	251	3,1444399356	0,09063180%
10	1000	300	250	3,1431655883	0,05006806%
11	1000	300	249	3,1418263912	0,00744009%
12	1000	300	248	3,1404218673	0,03726728%
13	1000	300	247	3,1389515399	0,08406925%
14	1000	300	246	3,1437530517	0,06876760%
15	1000	300	245	3,1421463489	0,01762460%

Najlepší experiment s odchýlkou len 0,00154539% a hodnotou Pi presnou na 4 desatinné miesta mal parametre:

- Veľkosť strany A = 300
- Veľkosť ihly = 254
- Počet ihiel = 500
- Počet iterácií = 1000

Pridaním počtu ihiel na 1000 a zachovaním ostatných parametrov sa však experiment zhoršil. V nasledujúcej tabuľke sme porovnávali výsledky pri meniacom sa počte ihiel a veľkosti ihly 254.

Tabuľka 18 Experimenty na ploche trojuhulník najlepší počet ihiel

Experiment číslo	**Počet iterácií**	**Veľkosť strany A**	**Počet Ihiel**	**Najlepšie Pi**	**Odchýlka Najlepšieho**
1	1000	300	50	3,1289527416	0,40234089%
2	1000	300	100	3,1289527416	0,40234089%
3	1000	300	201	3,1446762084	0,09815260%
3	1000	300	250	3,1415696144	0,00073335%
4	1000	300	301	3,1494174098	0,06924015%
5	1000	300	400	3,1447157859	0,09941239%
6	1000	300	500	3,1415441036	0,00154539%
7	1000	300	600	3,1394350528	0,6867856%
8	1000	300	750	3,1415357589	0,00181101%
9	1000	300	1000	3,1415314674	0,00194761%

Zistili sme, že pri počte ihiel 250 bol výsledok experimentu ešte presnejší. Odchýlka bola len 0,00073335% a hodnota Pi vyšla 3,1415696144, presnosť na 4 desatinné miesta.

9 VÝSLEDOK EXPERIMENTOV A POROVNANIE S EXISTUJÚCIMI APLIKÁCIAMI

V poslednej kapitole našej bakalárskej práce zhodnotíme dosiahnuté výsledky vytvorenej aplikácie a porovnáme aplikáciu s existujúcimi variantami riešenia problematiky Buffonovej úlohy o ihle ako celok.

9.1 Zhodnotenie výsledkov aplikácie

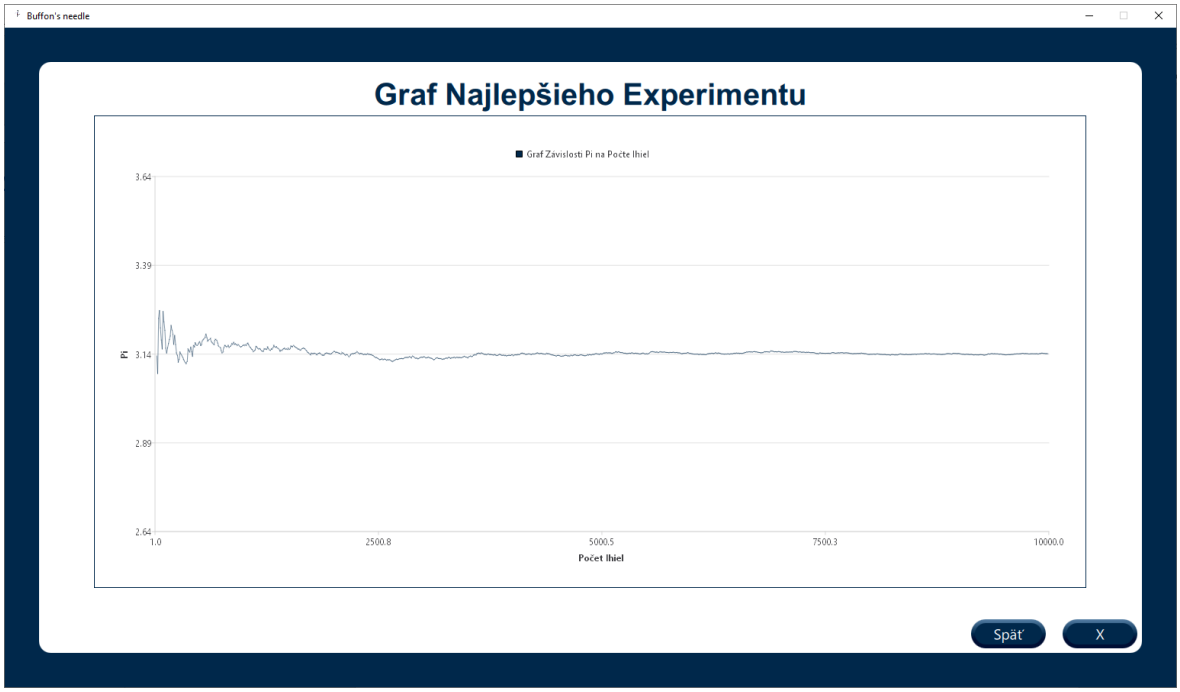
Po vykonaní experimentov na získanie optimálneho nastavenia parametrov pre jednotlivé plochy môžeme konštatovať, že najpresnejší výsledok Ludolfovho čísla sme dosiahli pri ploche obdĺžniková mriežka. Dokonca aj experimenty, v ktorých sme predpokladali nepresné výsledky vyšli s maximálne 10% chybou, čo je oproti najväčším chybám v zvyšných dvoch plochách oveľa menšia odchýlka.

Pri experimentoch nám testovanie uľahčila možnosť vykonania viacerých iterácií naraz. Pri vyššom počte sme spoľahlivo dokázali nájsť najlepší možný výsledok pri konkrétnych parametroch.

Výsledkom najlepšieho experimentu je hodnota 3,1415903568. Táto hodnota sa zhoduje so skutočným Pi na päť desatinných miest s odchýlkou 0,000073108%. Tento výsledok sme dosiahli pri ploche obdĺžniková mriežka a pri parametroch:

- Strana A = 100
- Strana B = 40
- Veľkosť ihly = 50
- Počet ihiel = 10000
- Počet iterácií = 1000

Na obrázku 57 môžeme vidieť priebeh grafu najlepšieho experimentu. Z grafu vidíme, že hodnota Pi sa ustáli na hodnotu okolo osi 3,14 približne po 3500 hodoch ihly.



Obrázok 57 Graf najlepšieho experimentu

Pri ďalších dvoch plochách nám vyšli výsledky experimentov o niečo horšie. Avšak presnosť na 4 desatinné miesta považujeme za rovnako úspešný výsledok. Najlepšie experimenty plôch rovnobežky a rovnostranný trojuholník boli tieto:

Hodnota Pi najlepšieho experimentu na ploche rovnobežky vyšla 3,1415343284, pri zadaných parametroch:

- Vzdialenosť rovnobežiek = 100
- Veľkosť ihly = 95
- Počet ihiel = 5000
- Počet iterácií = 1000

Hodnota najlepšieho Pi na ploche rovnostranný trojuholník sa rovná 3,1415696144, pri parametroch:

- Veľkosť strany A = 300
- Veľkosť ihly = 254
- Počet ihiel = 250
- Počet iterácií = 1000

Z výsledkov najlepších experimentov sme v aplikácii nastavili defaultné parametre na hodnotu týchto experimentov. Používateľ nebude musieť experimentálne skúšať, aké parametre

sú vhodné. Tie najlepšie získané s našich experimentov sú mu k dispozícii práve použitím tlačidla pre defaultné parametre.

V tejto podkapitole spomenieme aj negatívne výsledky. Pri všetkých troch plochách sme dokázali, že experimenty s malou ihlou sú dosť nepresné, a preto ich neodporúčame vykonávať. Postupným zvyšovaním veľkosti ihly sa zlepšuje aj výsledok experimentu.

Na záver je nutné poznamenať, že experimenty sme nevykonali pre každú kombináciu parametrov, a preto je možné, že existujú presnejšie výsledky.

9.2 Porovnanie s existujúcimi aplikáciami

9.2.1 GUI

Keďže Buffonova úloha o ihle je triviálny problém, nebola v existujúcich aplikáciách hlavnou úlohou grafická časť. V našej aplikácii sme na grafickú časť prikladali väčší dôraz. Aplikáciu sme navrhli ako pomôcku pre študentov. Z tohto dôvodu bolo pre nás dôležité, aby bolo pre používateľa príjemné a intuitívne ju ovládať. Avšak dávali sme si pozor, aby nebola aplikácia prekombinovaná a bol zachovaný jej hlavný zámer – experiment Buffonovej úlohy. Pre väčšiu prehľadnosť sme, na rozdiel od existujúcich aplikácií, oddelili časť pre výber parametrov a časť pre samotnú animáciu experimentu. Medzi ďalšie vylepšenia, oproti existujúcim aplikáciám, patrí pridanie grafov závislosti hodnoty Ludolfovho čísla na počte ihiel/iterácií daného experimentu. V neposlednom rade môžu byť nápomocné aj možnosti nastavenia predvolených parametrov, opakovanie experimentu či videoukážka pre lepšie pochopenie funkcionality aplikácie.

9.2.2 Počet iterácií

Aplikácia, ktorú sme implementovali v rámci zadania práce podporuje možnosť opakovať experiment až 1000 krát. Táto funkcionalita výrazne uľahčila experimentálnu časť. Vďaka uskutočneniu niekoľko tisíc experimentov, čo by pri jednej iterácii nebolo reálne, sme mohli čo najpresnejšie určiť hodnotu defaultných parametrov pre jednotlivé plochy. V existujúcich aplikáciách sa možnosť vykonania viacerých experimentov naraz nevyskytuje. Túto nadstavbu považujeme za veľký prínos a vylepšenie oproti existujúcim aplikáciám.

9.2.3 Rozšírenia

Ďalšou súčasťou aplikácie je možnosť výberu troch plôch pre hádzanie ihly. Ako môžeme vidieť v podkapitole 7.1, je to práve plocha obdĺžniková mriežka, kde experimenty vyšli s najpresnejším výsledkom. Rozšírenia Buffonovej úlohy sú popísané teoreticky v práci od Hledíka [8]. Avšak žiadna dostupná aplikácia na simuláciu tejto problematiky neumožňuje zmeniť plochu a pracujú so základným problémom vrhania ihiel na rovnobežky. Toto považujeme za najväčší funkcionálny pokrok oproti existujúcim aplikáciám.

9.2.4 Presnosť experimentov

Existujúce aplikácie fungujú na princípe jednej iterácie. Dôsledkom tohto obmedzenia je logicky znížená presnosť. Zo skúmaných aplikácií sme zistili, že najpresnejší je webový simulátor Buffonovej úlohy [14] s presnosťou na 2-3 desatinné miesta. V porovnaní s nami implementovanou aplikáciou, s presnosťou na 4-5 desatinných miest, všetky existujúce výrazne zaostávajú v presnosti.

ZÁVER

Cieľom práce bolo vytvorenie užívateľsky prívetivej aplikácie, ktorá bude riešiť problematiku Buffonovej úlohy o ihle. Aplikáciu sme sa snažili navrhnúť tak, aby bola čo najintuitívnejšia, plynulá a čo najrýchlejšia. Daný cieľ sme splnili za pomoci frameworku Qt a jeho editoru Qt Creator v programovacích jazykoch C++ a QML. Proces vykonania experimentu v aplikácii pozostáva z dvoch častí. Prvou je výber parametrov, s ktorými sme experiment realizovali a druhou je samotný experiment hádzania ihly na zvolenú plochu v podobe animácie jeho priebehu. Výsledkom experimentu je hodnota Ludolfovho čísla a graf závislosti hodnoty Pi na počte vrhnutých ihiel.

Aplikácia bola rozšírená o výber plochy a možnosť vykonania viacerých iterácií v jednom experimente. Pridanie možnosti viacerých iterácií výrazne uľahčilo experimentálnu časť, ktorej výsledkom bolo získanie optimálnych parametrov pre jednotlivé plochy. Pridanie výberu plôch a následné experimenty, ktoré sme vykonali na plochách obdĺžniková mriežka a rovnostranný trojuholník viedli k zlepšeniu výsledku práce. Presnosť Ludolfovho čísla na 5 desatinných miest považujeme za výrazný pokrok oproti doposiaľ existujúcim aplikáciám.

Aplikácia by mohla byť prínosom pre výučbu problematiky pravdepodobnosti. Študenti ako aj iní používatelia aplikácie by mali možnosť experimentálne overiť funkčnosť a snažiť sa nájsť najefektívnejšie nastavenie parametrov pre čo najlepší výsledok hodnoty Ludolfovho čísla.

SEZNAM POUŽITÉ LITERATURY

[1] PŁOCKI, Adam. *O náhodě a pravděpodobnosti: pro účastníky matematické olympiády*. Praha: Mladá fronta, 1982. Škola mladých matematiků.

[2] HYKŠOVÁ, Magdalena: *Filosofická pojetí pravděpodobnosti v pracích českých myslitelů.* (Czech). Praha: Matfyzpress, vydavatelství Matematicko-fyzikální fakulty Univerzity Karlovy v Praze, 2011. ISBN 978-80-7378-192-7.

[3] PILASTRO, Eleonora. *New value of pi calculated by Swiss university at over 62 trillion digits* [online]. 2022 [cit. 2022-05-18]. Dostupné z: https://www.guinnessworldrecords.com/news/2022/3/new-value-of-pi-calculated-by-swiss-university-at-over-62-billion-digits-694748

[4] O'CONNOR, J J a E F ROBERTSON. *Ludolph Van Ceulen* [online]. 2009 [cit. 2022-05-18]. Dostupné z: https://mathshistory.st-andrews.ac.uk/Biographies/Van_Ceulen/

[5] BECKMANN, Petr. *Historie čísla π*. Vydání druhé, revidované. Praha: Academia, 2021. Galileo. ISBN 978-802-0031-334.

[6] O'CONNOR, J J a E F ROBERTSON. *Georges Louis Leclerc Comte de Buffon* [online]. 2004 [cit. 2022-05-18]. Dostupné z: https://mathshistory.st-andrews.ac.uk/Biographies/Buffon/

[7] MATHAI, A. M. *An Introduction to Geometrical Probability: Distributional Aspects with Applications*. Amsterdam: Gordon and Breach Science Publishers, 1999. ISBN 978-90-5699-681-9.

[8] HLEDÍK, Jakub. *Buffonova úloha o jehle a její zobecnění*. 2018. Bakalářská práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra pravděpodobnosti a matematické statistiky.

[9] SAMUELYHUNTER. *Buffon* [online]. 2019 [cit. 2022-05-18]. Dostupné z: https://github.com/samuelyhunter/Buffon

[10] DAHM, Gunnar. *Monte Carlo Simulation of Buffon's Needle* [online]. 2019 [cit. 2022-05-18]. Dostupné z: https://github.com/GunnarDahm/buffon_monte_carlo_sim

[11] MO-MO-. *Buffon's Needle App* [online]. 2018 [cit. 2022-05-18]. Dostupné z: https://github.com/mo-mo-666/Buffon

[12] VETTER, Keith. *Buffon's Needle* [online]. 2003 [cit. 2022-05-18]. Dostupné z: https://wiki.tcl-lang.org/page/Buffon%27s+Needle

[13] DANÍČEK, Ladislav. *Využití geometrické pravděpodobnosti při experimentálním ověření Ludolfova čísla*. Bakalářská práce. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005, 37 s., 7s. příloh.

[14] MARTINEZ, Andrew. *Pi by Buffon's Needle* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://www.khanacademy.org/computer-programming/pi-by-buffons-needle/6695500989890560

[15] QT WIKI CONTRIBUTORS. *About Qt* [online]. 2022 [cit. 2022-05-18]. Dostupné z: https://wiki.qt.io/About_Qt

[16] THE QT COMPANY LTD. *Contribute to Qt* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://www.qt.io/community/contribute-to-qt

[17] THE QT COMPANY LTD. *The future is written with Qt* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://doc.qt.io

[18] THE QT COMPANY LTD. *Loader QML Type* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://doc.qt.io/qt-5/qml-qtquick-loader.html

[19] THE QT COMPANY LTD. *StackView QML Type* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://doc.qt.io/qt-5/qml-qtquick-controls2-stackview.html

[20] THE QT COMPANY LTD. *ChartView QML Type* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://doc.qt.io/qt-5/qml-qtcharts-chartview.html

[21] ERICSON, Christer. *Real-time collision detection*. Amsterdam: Elsevier, c2005. ISBN 978-1558607323.

[22] MICROSOFT. *Oboznámenie sa s panelom Xbox Game Bar vo Windowse 10* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://support.xbox.com/sk-SK/help/games-apps/game-setup-and-play/get-to-know-game-bar-on-windows-10

[23] THE QT COMPANY LTD. *Creating Project Files* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://doc.qt.io/qt-5/qmake-project-files.html

[24] THE QT COMPANY LTD. *Qmake Manual* [online]. © 2022 [cit. 2022-05-18]. Dostupné z: https://doc.qt.io/qt-5/qmake-manual.html

ZOZNAM OBRÁZKOV

Obrázok 1 Graf závislosti x na y 14
Obrázok 2 Výsledok experimentu pomocou Monte Carlo simulácie [9] 22
Obrázok 3 Aplikácia pre výpočet Pi v jazyku python [11]....23
Obrázok 4 Aplikácia v jazyku Tool Command Language [12]....24
Obrázok 5 Aplikácia v Matlabe [13] 24
Obrázok 6 Aplikácia s animáciou [14] 25
Obrázok 7 Grafické rozhranie Qt Creatoru....29
Obrázok 8 Dokumentácia objektu Rectangle v Qt Creatore....30
Obrázok 9 Štruktúra zdrojového kódu....30
Obrázok 10 Emit signálu 32
Obrázok 11 Prepojenie slotu so signálom....32
Obrázok 12 Ukážková trieda 32
Obrázok 13 Funkcia main pre QGui aplikácie 33
Obrázok 14 Main.qml....34
Obrázok 15 Sprístupnenie triedy pre qml súbory 34
Obrázok 16 Flow diagram aplikácie....36
Obrázok 17 Aktivované, stlačené a zakázané tlačidlo Spustiť Experiment 38
Obrázok 18 StackView [19] 39
Obrázok 19 Animácia prechodu obrazoviek 40
Obrázok 20 Základná obrazovka 41
Obrázok 21 Zdrojový kód obrázku ihly s animáciou rotácie....41
Obrázok 22 Obrazovka Výber Plochy 43
Obrázok 23 Obrazovka Výber Parametrov....43
Obrázok 24 Zdrojový kód posúvača....44
Obrázok 25 Vyskakovacie okno 45
Obrázok 26 Časť zdrojového kódu funkcie *areAllParametersFilledCorrectly()*....46
Obrázok 27 Prebiehajúci experiment....47
Obrázok 28 Obrazovka Výsledok Experimentu 1 iterácia 48
Obrázok 29 Obrazovka Výsledok Experimentu 771 iterácií....48
Obrázok 30 Grafy experimentu s viacerými iteráciami....49
Obrázok 31 Graf závislosti priemerného Pi od počtu iterácií....50
Obrázok 32 MyRectangle 51

Obrázok 33 Časť kódu funkcie *addLines()* pre plochu rovnobežky. 52
Obrázok 34 Časť kódu funkcie *addLines()* pre plochu obdĺžniková mriežka. 52
Obrázok 35 Časť kódu funkcie *addLines()* pre plochu rovnostranný trojuholník. 53
Obrázok 36 Kód funkcie *addNeedle()* pre vykreslenie ihly. 54
Obrázok 37 Kód funkcie *removeRectangles()* 54
Obrázok 38 Zdrojový kód kostry grafu 55
Obrázok 39 Zdrojový kód funkcie *addToGraph(index)* 56
Obrázok 40 Zdrojový kód funkcie *addToGraph()* 56
Obrázok 41 Kód enum classy Surface 57
Obrázok 42 Zdrojový kód metódy *maximumNeedleLength()* 58
Obrázok 43 Zdrojový kód metódy *onDefaultButtonClicked()* 58
Obrázok 44 Časť zdrojového kódu po stlačení tlačidla „Spustiť Experiment“ 59
Obrázok 45 Zdrojový kód metódy *start()* 59
Obrázok 46 Deklarácia premenných pre generáciu náhodných čísel 60
Obrázok 47 Časť zdrojového kódu vektora rovnobežiek 60
Obrázok 48 Zdrojový kód súradníc bodov trojuholníka 64
Obrázok 49 Zdrojový kód metódy *isPointInTriangle()* 65
Obrázok 50 Cyklus for pre iterácie 65
Obrázok 51 Zdrojový kód metódy *calculateBestWorstAveragePi()* 66
Obrázok 52 Výsledok experimentu s malou veľkosťou ihly 72
Obrázok 53 Výsledok experimentu pri 2050 ihlách 75
Obrázok 54 Výsledok experimentu pri 7000 ihlách 75
Obrázok 55 Experiment malej ihly na ploche rovnostranný trojuholník 84
Obrázok 56 Experiment polovičnej ihly na ploche rovnostranný trojuholník 86
Obrázok 57 Graf najlepšieho experimentu 91

ZOZNAM TABULIEK

Tabuľka 1 Experimenty na ploche rovnobežky rovnosť paramerov číslo 1.....69
Tabuľka 2 Experimenty na ploche rovnobežky rovnosť paramerov číslo 2.....70
Tabuľka 3 Experimenty na ploche rovnobežky malá ihla.....71
Tabuľka 4 Experimenty na ploche rovnobežky polovičná ihla číslo 1.....73
Tabuľka 5 Experimenty na ploche rovnobežky polovičná ihla číslo 2.....74
Tabuľka 6 Experimenty na ploche rovnobežky 2/3 ihla.....76
Tabuľka 7 Experimenty na ploche rovnobežky viac iterácií.....77
Tabuľka 8 Experimenty na ploche rovnobežky 108 iterácií.....78
Tabuľka 9 Experimenty na ploche rovnobežky 192 iterácií.....78
Tabuľka 10 Experimenty na ploche rovnobežky najlepšie Pi.....79
Tabuľka 11 Experimenty na ploche obdĺžnik malá ihla číslo 1.....81
Tabuľka 12 Experimenty na ploche obdĺžnik malá ihla číslo 2.....82
Tabuľka 13 Experimenty na ploche obdĺžnik polovičná ihla.....82
Tabuľka 14 Experimenty na ploche trojuhulník malá ihla.....85
Tabuľka 15 Experimenty na ploche trojuhulník polovičná ihla.....86
Tabuľka 16 Experimenty na ploche trojuhulník najväčšia ihla.....87
Tabuľka 17 Experimenty na ploche trojuhulník najlepšia veľkosť ihly.....88
Tabuľka 18 Experimenty na ploche trojuhulník najlepší počet ihiel.....89

ZOZNAM PRÍLOH

Príloha P 1: Obsah priloženého CD

PRÍLOHA P I: OBSAH PRILOŽENÉHO CD

Priložené CD obsahuje:

- Text bakalárskej práce – fulltext.pdf
- Zdrojový kód a spustiteľná aplikácia – prilohy.zip
    - zložka Buffon_needle – obsahujúca zdrojové kódy
    - zložka build-Buffon_needle-Desktop_Qt_6_2_2_MinGW_64_bit-Release obsahujúca skompilovanú aplikáciu so všetkými potrebnými súbormi pre spustenie
        - .exe súbor sa nachádza v zložke release - **Buffon_needle.exe**