

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Kodavimo teorijos A11 praktinės užduoties ataskaita

Darbą atliko: Tautvydas Mačiulis
Programų sistemos 4 kursas 3 grupė

Turinys

1. Realizuotos užduoties dalys.....	3
2. Trečiųjų šalių bibliotekos	3
3. Užduoties atlikimo trukmė	3
4. Programos paleidimas.....	4
5. Programos tekstų failai	4
6. Vartotojo sąsajos aprašymas.....	8
7. Programiniai sprendimai.....	14
8. Eksperimentai	15
9. Naudotos literatūros sąrašas	17

1. Realizuotos užduoties dalys

Programoje pilnai realizuotos visos užduoties dalys. Visi trys vartotojo scenarijai, kodavimas tiesiniu kodu, 2-naris simetrinis kanalas, dekodavimas žingsnis po žingsnio (angl. *step-by-step*) algoritmu.

2. Trečiųjų šalių bibliotekos

Programoje buvo naudota tik viena trečiųjų šalių biblioteka – „TornadoFX“. Ši biblioteka skirta programuoti grafinės vartotojo sąsajos (žemiau *GUI*) elementus.

3. Užduoties atlikimo trukmė

Deja, bet užduoties atlikimo trukmė nebuvo tiksliai fiksuojama, tad žemiau yra pateikti apytiksli laiko matavimai:

- Teorijos nagrinėjimas: ~3 val.
- Projektavimas: ~30 min.
- Programavimas: ~12 val.
- Klaidų ieškojimas ir taisymas: ~4 val.
- Ataskaitos ruošimas: ~2 val. 30 min.

Iš viso: ~22 val.

Nemažai laiko buvo skirta GUI programavimui, nes prieš tai niekada nenaudojau „TornadoFX“ bibliotekos, tad teko dalį laiko paskirti ją perprasti. Taip pat, nemažai laiko buvo sugaišta surandant būdą konvertuoti baitus į binarines 8 bitų eilutes (angl. *string*) taip, kad būtų įmanoma konvertuoti neigiamas baitų reikšmes į bitus ir vėliau būtų įmanoma lengvai juos vėl konvertuoti atgal į baitus.

4. Programos paleidimas

Programą galima paleisti tiesiog atidarius „run.bat“ failą, kuris yra pradiniam aplanke.

5. Programos tekstų failai

Visi programos tekstų failai yra „src/main/kotlin/com/coding“ aplanke. Programos tekstų failai yra susiskirstyti į tris aplankus pagal jų paskirtį:

- „ui“ – šiame aplanke yra visi failai susiję su vartotojo grafine sąsaja
- „util“ – šiame aplanke yra failai su pagalbiniomis GUI arba matematinių operacijų funkcijomis.
- „viewModel“ – šiame aplanke yra failai susiję su pagrindine verslo logika (angl. „business logic“), t.y. klasės, kuriuos atlieka skaičiavimus ir kitas svarbias operacijas.

Toks skirstymas pasirinktas dėl to, kad tvarkingai būtų atskirta GUI komponentai nuo skaičiavimų, logikos ir pan. atsižvelgus į MVVM programų sistemų architektūrą. Kiekvienas failas „src/main/kotlin/com/coding“ aplanke yra detaliau aprašytas žemiau esančiame sąrašė:

- „ui/general/ParametersView.kt“ – GUI klasė, skirta įvesti N , K ir P_e parametrus.
- „ui/general/GeneratorMatrixView.kt“ – GUI klasė, skirta įvesti generuojančios matricos stulpelius nuo K iki N . Pradinius K stulpelius negalima keisti, nes jie užpildyti vienetine matrica. Taip pat, vartotojas gali pasirinkti matricą uždildyti atsitiktinėmis reikšmėmis.
- „ui/general/ScenarioSelectorView.kt“ – GUI klasė, skirta pasirinkti vieną iš trijų scenarijų.

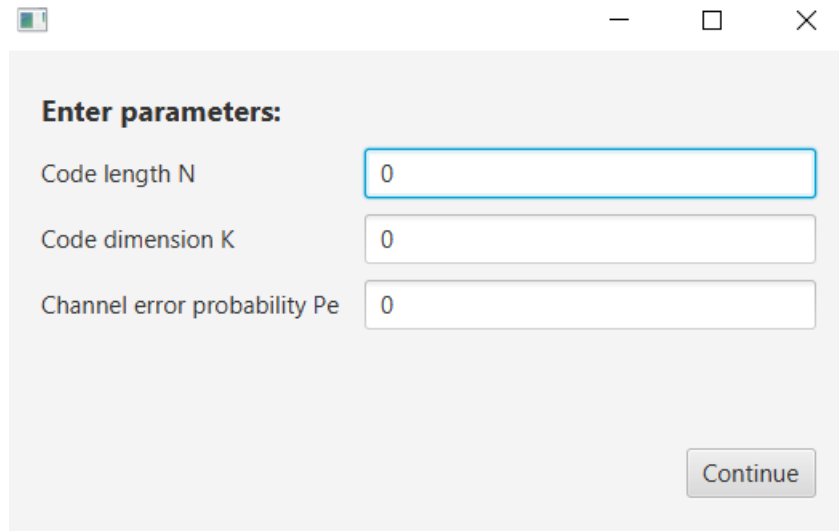
- „*ui/firstScenario/ChannelView.kt*“ – GUI klasė, skirta pavaizduoti pirmame scenarijuje užkoduotą vektorių išėjusį iš kanalo bei klaidas jame. Be to, yra leidžiama vartotojui jį modifikuoti, taip keičiant vektoriuje padarytas klaidas.
- „*ui/firstScenario/DecodeView.kt*“ – GUI klasė, skirta parodyti pirmame scenarijuje dekuotodą vektorių. Be to, papildomai yra parodomas pradinis pranešimo vektorius, bei užkoduotas vektorius išėjęs iš kanalo palyginimui.
- „*ui/firstScenario/EncodeView.kt*“ – GUI klasė, skirta įvesti K ilgio vektorių ir jį užkoduotį pirmame scenarijuje.
- „*ui/secondScenario/TextInputView.kt*“ – GUI klasė, skirta įvesti tekstą, kuris vėliau bus užkoduojamas/dekoduojamas antrame scenarijuje.
- „*ui/secondScenario/TextProcessingView.kt*“ – GUI klasė, kuri rodo progreso juostą tol, kol tekstas yra užkoduojamas ir dekoduojamas antrame scenarijuje.
- „*ui/secondScenario/TextProcessingResultsView.kt*“ – antro scenarijaus GUI klasė, kuri atvaizduoja tris tekstus: pradinį tekstą, tekstą, kuris buvo išsiųstas kanalu nekoduojant ir tekstą, kuris buvo užkoduotas, išsiųstas kanalu ir dekodotas.
- „*ui/thirdScenario/ImageSelectorView.kt*“ – GUI klasė, skirta pasirinkti *bmp* formato paveikslėlį, kuris bus naudojamas trečiame scenarijuje.
- „*ui/thirdScenario/ImageProcessingView.kt*“ – GUI klasė, kuri rodo progreso juostą tol, kol paveikslėlis yra užkoduojamas ir dekoduojamas trečiame scenarijuje.

- „*ui/thirdScenario/ImageProcessingResultsView.kt*“ – trečio scenarijaus GUI klasė, kuri atvaizduoja tris paveikslėlius: pradinį paveikslėlį, paveikslėlį, kuris buvo išsiųstas kanalu nekoduojant ir paveikslėlį, kuris buvo užkoduotas, išsiųstas kanalu ir dekodotas.
- „*ui/Events.kt*“ – GUI įvykių klasė, kuriems įvykstant yra atnaujinami tam tikri GUI elementai.
- „*util/Calculations.kt*“ – šiame faile saugomos pagalbinės matematinių operacijų funkcijos, tokios kaip matricos daugyba iš vektoriaus ir pan.
- „*util/GUI.kt*“ – šiame faile saugomos pagalbinės GUI funkcijos, kurios palengvina darbą kuriant GUI komponentus.
- „*viewModel/Channel.kt*“ – kanalo klasė, kuri yra atsakinga už pranešimo siuntimą dvinariu simetriniu kanalu su P_e tikimybe, kad kodo pozicijoje n bus padaryta klaida.
- „*viewModel/Decoder.kt*“ – dekodavimo klasė, kuri dekoduoja gautą pranešimo vektorius taikant žingsnis po žingsnio (angl. *step-by-step*) algoritmą.
- „*viewModel/Encoder.kt*“ – užkodavimo klasė, kuri užkoduoja pranešimo vektorius į padauginus iš generuojančios matricos.
- „*viewModel/FirstScenarioViewModel.kt*“ – pirmo scenarijaus logikos klasė, kuri saugo reikšmes naudojamas pirmo scenarijaus GUI klasių.
- „*viewModel/ProcessingViewModel.kt*“ – abstrakti klasė, kuri atsakinga už baitų konvertavimą į binarinę bitų eilutę ir pan. Šią klasę paveldi antro ir trečio scenarijų verslo logikos klasės aprašytos žemiau. Taip pat, ši klasė saugo užkodavimo, kanalo ir dekodavimo klases.

- „*viewModel/SecondScenarioViewModel.kt*“ – antro scenarijaus logikos klasė, kuri apdoroja vartotojo įvestą tekstą, naudojantis „*ProcessingViewModel.kt*“ metodais.
- „*viewModel/ThirdScenarioViewModel.kt*“ – trečio scenarijaus logikos klasė, kuri apdoroja vartotojo įvestą tekstą, naudojantis „*ProcessingViewModel.kt*“ metodais.

6. Vartotojo sąsajos aprašymas

Paleidus programą yra parodomas langas, kuriame vartotojas gali įvesti kodo parametrus N , K ir P_e .



Enter parameters:

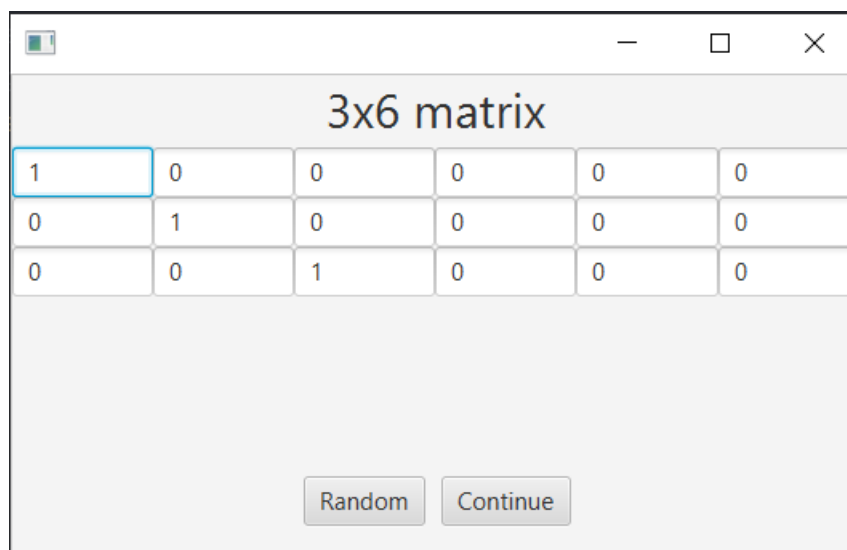
Code length N

Code dimension K

Channel error probability P_e

N ir K parametrų teksto laukai priima tik sveikas skaitines reikšmes, tuo tarpu tikimybės parametro laukas priima ir reikšmes su kableliu (angl. *Double*). Jei reikšmių formatas netinkamas, tai tada negalima pereiti į kitą langą.

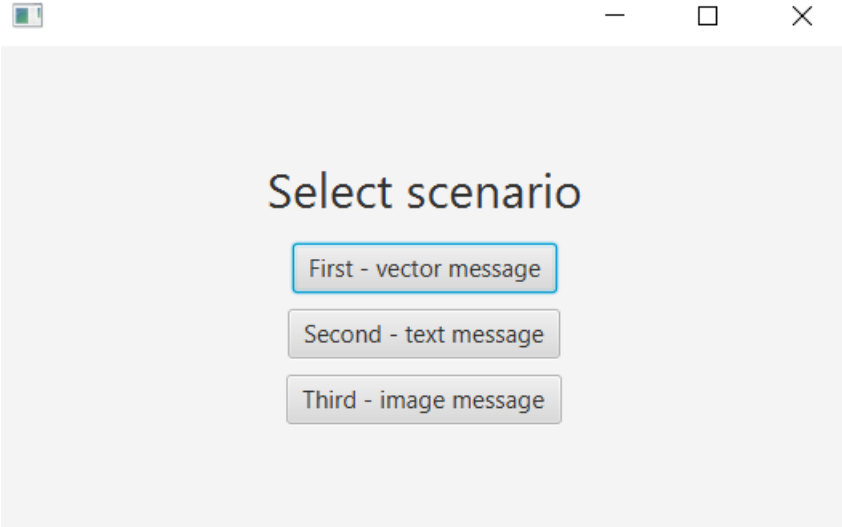
Teisingai įvedus parametrus yra parodomas langas, kuriame galima įvesti generuojančios matricos stulpelius nuo K iki N , pirmi K stulpeliai yra nekeičiami. Į langelius galima įvesti tik 1 ar 0. Be to, galima pasirinkti, kad programa įvestų atsitiktines reikšmes.



3x6 matrix

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0

Įvedus generuojančią matricą, vartotojas tuomet gali pasirinkti vieną iš trijų scenarijų:



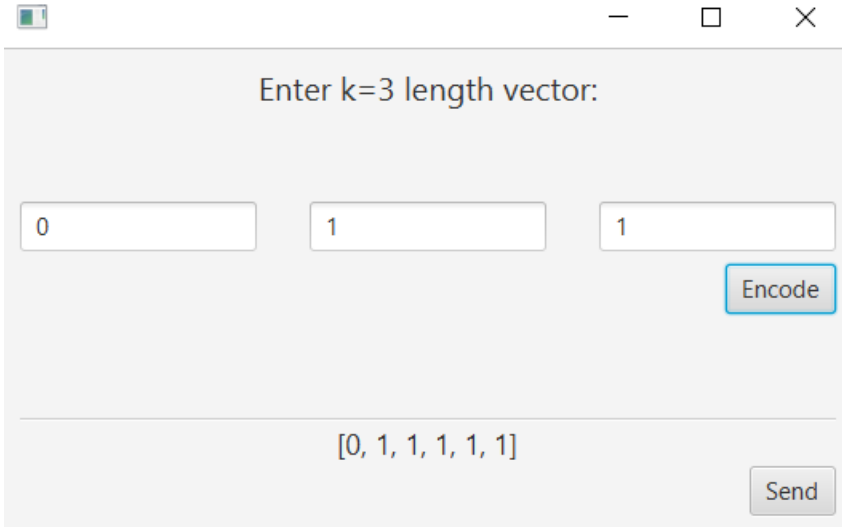
Select scenario

First - vector message

Second - text message

Third - image message

Pasirinkus pirmą scenarijų, vartotojas tuomet gali įvesti pradinį K ilgio pranešimo vektorių ir jį užkoduoti. Šiuo atveju galima įvesti vėl tik 1 ar 0 į langelius.



Enter k=3 length vector:

0 1 1

Encode

[0, 1, 1, 1, 1, 1]

Send

Vartotojui paspaudus „Send“ mygtuką, programa pereina į kitą langą, kuriame yra parodomas iš kanalo gautas užkoduotas vektorius bei kuriose vietose buvo padarytos klaidos (jei tokių buvo). Vartotojas gali modifikuoti vektorių, taip keičiant kiek ir kur įvyko klaidos (tai iš karto yra atvaizduojama).

Vector from channel

0	0	1	0	1	1
-	X	-	X	-	-

X - mistake
- - no mistake

Decode

Paspaudus mygtuką „*Decode*“ programa parodo langą, kuriame parodomas dekodotas vektorius. Taip pat yra parodomas pradinis pranešimo vektorius prieš užkodavimą bei užkodotas vektorius, kuris išėjo iš kanalo palyginimui.

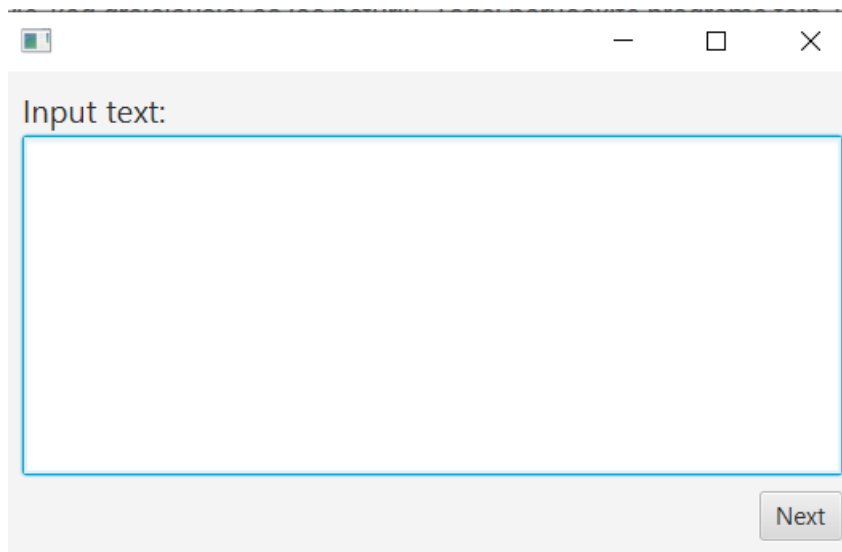
Decoding results

Original vector: [0, 1, 1]
 From channel vector: [0, 1, 0, 1, 1, 1]
 Decoded vector: [0, 1, 1]

Other scenario New parameters

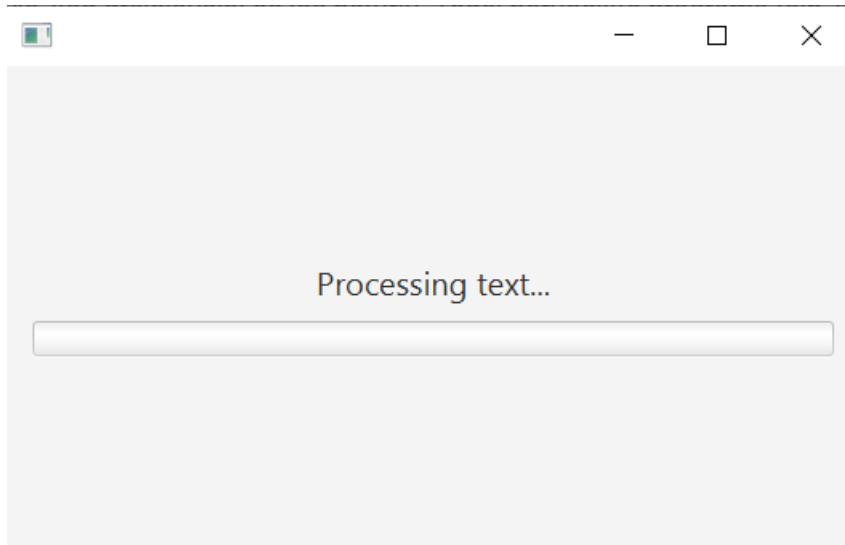
Šiame lange vartotojas gali pasirinkti grįžti atgal į scenarijaus pasirinkimo langą arba pradėti programą vėl nuo pradžių įvedant naujus parametrus.

Jei scenarijaus pasirinkimo lange buvo pasirinktas antras scenarijus, tai yra parodomas teksto įvesties langas.



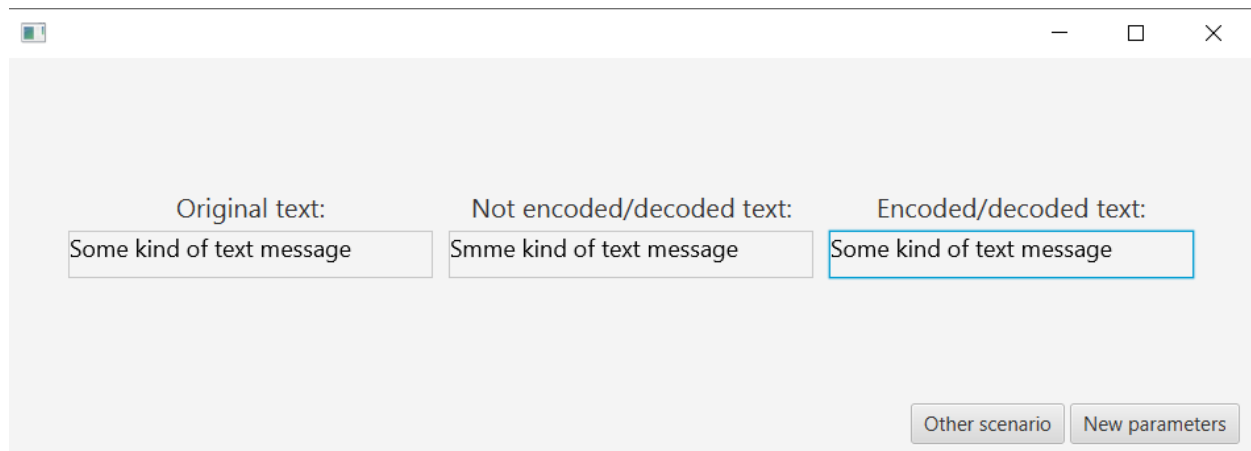
A screenshot of a graphical user interface window. The window has a standard title bar with a small icon on the left and three control buttons (minimize, maximize, close) on the right. The main content area is light gray. At the top left of the content area, the text "Input text:" is displayed. Below this text is a large, empty rectangular text input field with a blue border. In the bottom right corner of the content area, there is a small, rectangular button with the text "Next" on it.

Vartotojui įvedus tekstą ir paspaudus mygtuką „*Next*“, programa rodo langą su progreso juostą tol, kol yra apdorojamas (verčiamas į baitus, vėliau į bitus, užkoduojamas ir t.t.) įvestas tekstas.



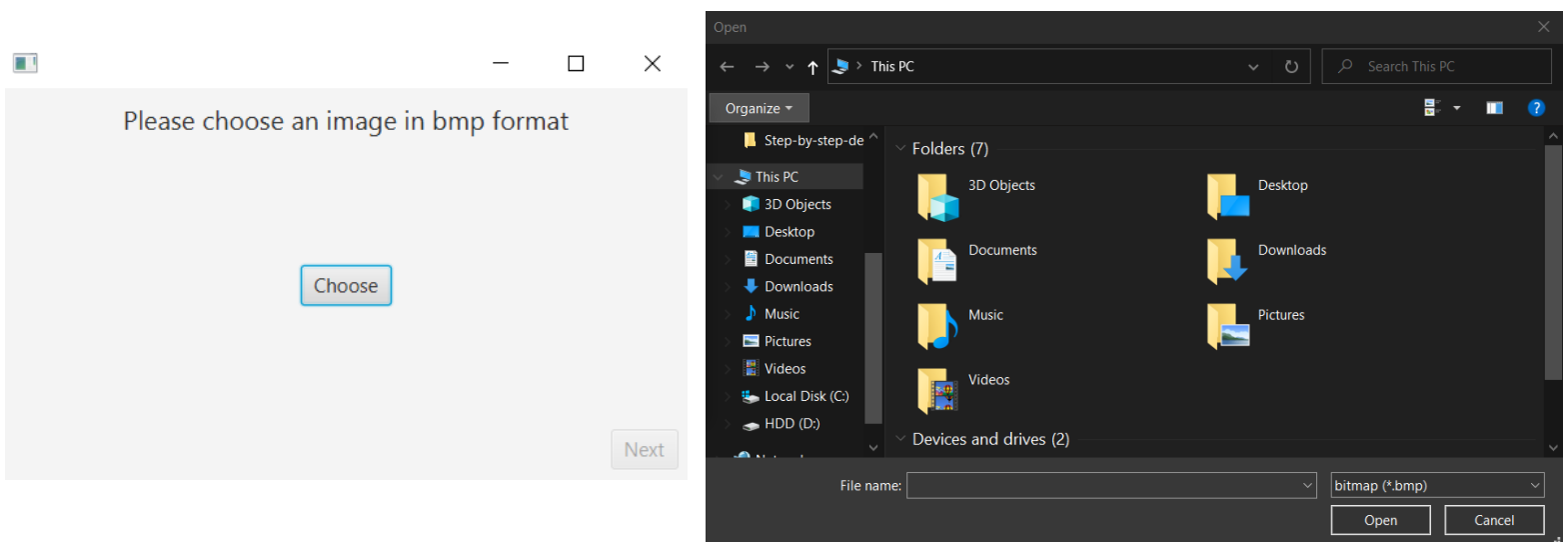
A screenshot of a graphical user interface window. The window has a standard title bar with a small icon on the left and three control buttons (minimize, maximize, close) on the right. The main content area is light gray. In the center of the content area, the text "Processing text..." is displayed. Below this text is a horizontal progress bar, which is currently empty and has a light gray fill.

Programai baigus apdoroti tekstą yra automatiškai parodomas kitas langas, kuriame yra pavaizduoti originalus tekstas prieš apdorojimą, tekstas, kuris buvo išsiųstas kanalu nekoduojant ir tekstas, kuris buvo užkoduotas, išsiųstas kanalu ir dekodotas.



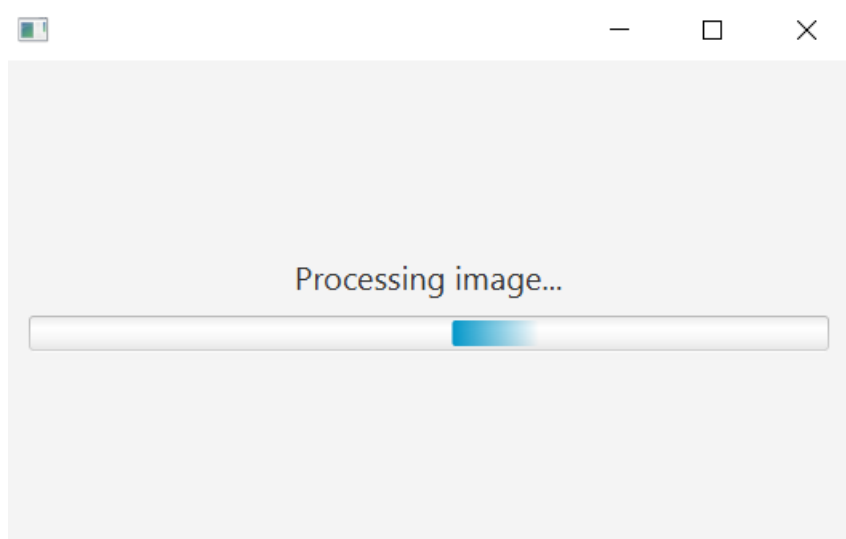
Šiame lange vartotojas vėl gali pasirinkti kitą scenarijų su tais pačiais parametrais arba pradėti programą nuo pradžių pasirenkant kitus parametrus.

Jei scenarijaus pasirinkimo lange vartotojas pasirenka trečią scenarijų, tai programa parodo langą, kuriame vartotojas gali pasirinkti *bmp* formato paveikslėlį, paspaudus „Choose“ mygtuką, kuris atidaro failų pasirinkimo langą.

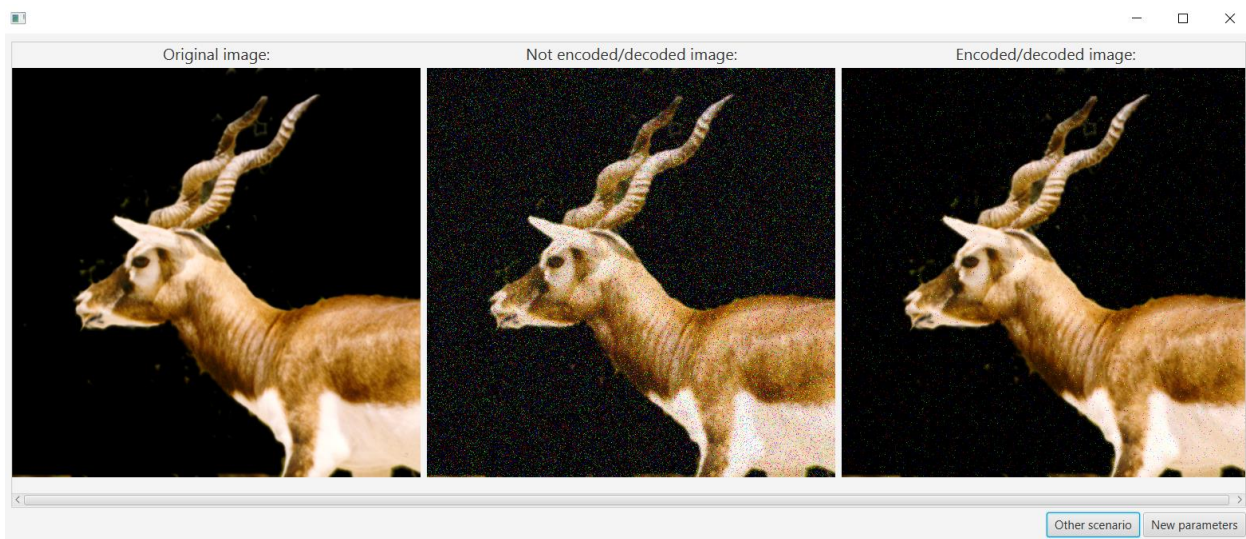


Vartotojui yra leidžiama pasirinkti tik *bmp* formato failus. Pasirinkus failą, vartotojui leidžiama spausti „next“ mygtuką. Paspaudus šį mygtuką, programa

pradedama apdoroti paveikslėlį (konvertuoti į baitus, vėliau į bitus, užkoduoti ir t.t.) bei parodo langą su progreso juosta.



Programai baigus apdoroti paveikslėlį yra parodomas langas su trimis paveikslėliais: originalus paveikslėlis, paveikslėlis, kuris buvo išsiųstas kanalu nekoduojant ir paveikslėlis, kuris buvo užkoduotas, išsiųstas kanalu ir dekodotas.



Vartotojas vėl gali pasirinkti kitą scenarijų arba grįžti į programos pradžią ir įvesti naujus parametrus.

7. Programiniai sprendimai

Antro ir trečio scenarijaus metu tekstas ar nuotrauka yra konvertuojami į baitų masyvą. Šie baitai tuomet yra konvertuojami į dvejetainių bitų teksto eilutę (angl. *String*). Tai atliekama baitą konvertavus į *Integer* duomenų tipą bei atliekant loginę AND operaciją su 0xFF reikšme. Šitaip mes konvertuojam neigiamus baitus į be ženklo baitus. Vėliau prie gautos reikšmės yra pridama 0x100 tam, kad jei reikia būtų pridama nulių prie gautos reikšmės, jog iš viso būtų 8 bitai. Atlikus šiuos žingsnius, pavyzdžiui, baitas 11111111 10101010 tampa 10101010 teksto eilute. Galiausiai bitų teksto eilutė yra suskaidoma į K ilgio pranešimo vektorius, kurie yra saugomi kaip *Integer* reikšmių sąrašas. Kiekvieno sąrašo elementas gali būti arba 1 arba 0.

Jei suskaidžius teksto eilutę į pranešimų vektorius yra negaunamas pilnas vektorius, tuomet jis yra papildomas 0 iki tol, kol vektoriaus ilgis tampa K. Informacija kiek nulių buvo prirašyta yra saugoma „*ProcessingViewModel*“ klasėje, kurią paveldi antro ir trečio scenarijaus „*ViewModel*“ klasės. Ši informacija nėra siunčiama kanalu.

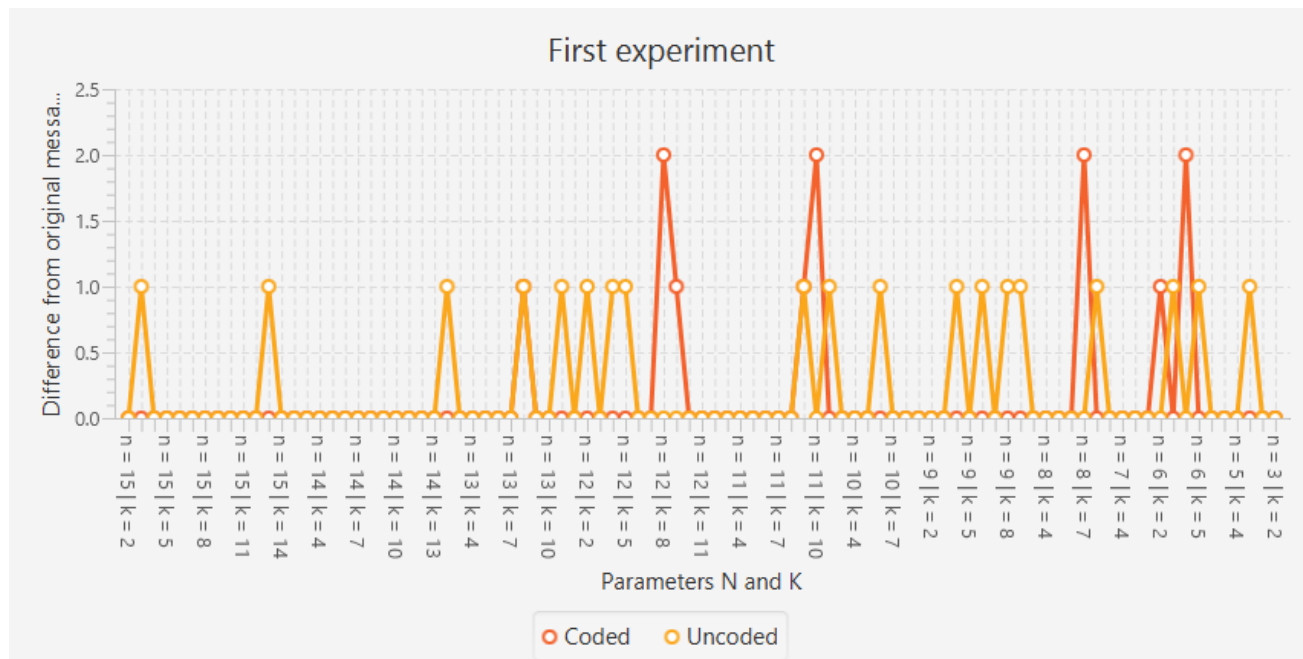
Kiekvienas užkoduotas pranešimo vektorius yra siunčiamas po vieną taikant „*map*“ operaciją ant vektorių sąrašo. Gavus vektorius iš kanalo ir juos dekodavus, šie yra konvertuojami atgal į baitus ir vėliau į tekstą ar paveikslėlį atitinkamai antrame ar trečiame scenarijuje.

Trečiame scenarijuje dirbant su *bmp* formato paveikslėliais pirmi 54 baitai yra saugomi atskirai kaip tarnybinė informacija „*ThirdScenarioViewModel*“ klasėje, nes šie baitai sudaro paveikslėlio antraštės informaciją (angl. *Header Info*). Šie baitai nėra siunčiami kanalu.

8. Eksperimentai

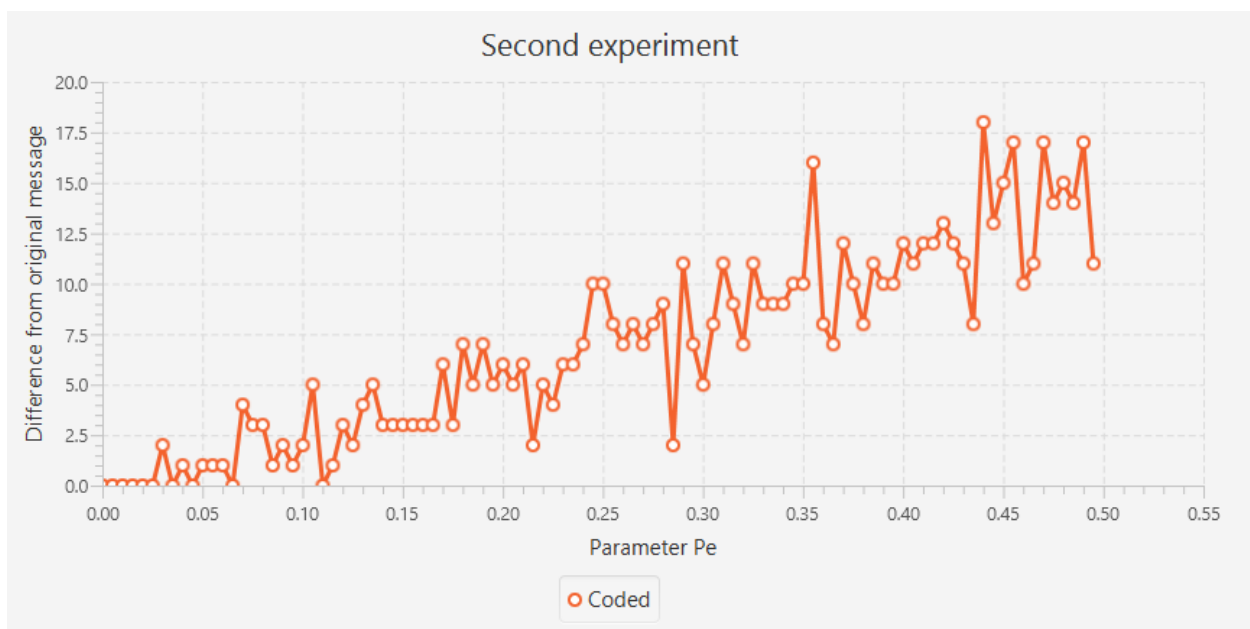
Iš viso buvo atlikti du eksperimentai. Pirmame eksperimente buvo nagrinėjama kaip skirtingi N ir K parametrai keičia kodo efektyvumą palyginus su nekoduotais pranešimais. Antrame eksperimente buvo nagrinėjama kaip skirtingos klaidos tikimybės keičia kodo efektyvumą. Abejuose eksperimentuose pranešimo vektorius buvo **1010101110111110110100001**.

Pirmame eksperimente buvo keičiami N ir K parametrai. N parametro reikšmės buvo iš intervalo $[3;15]$, o K reikšmės iš $[2;N-1]$. Kiekvienos iteracijos metu pasirinkus kurią nors N reikšmę, su šia reikšme yra prasukami ciklai su skirtingomis K reikšmėmis nuo 2 iki $N-1$. Kiekvienos iteracijos metu generuojanti matrica būdavo parenkama atsitiktinai, o kanalo trikdžio tikimybė P_e būdavo lygi 0.01. Kiekvieną kartą parinkus N ir K reikšmes, užkodavus pranešimo vektorių, prasiuntus jį kanalą ir dekodavus, būdavo išsaugoma kiek dekodotas pranešimas skirdavosi nuo pradinio, t.y. kiek bitų būdavo skirtingi. Taip pat, būdavo palyginamas ir kanalu išsiųstas nekoduotas pranešimo vektorius. Gauti eksperimento rezultatai yra pavaizduoti grafike apačioje.



Iš grafiko matosi, kad koduotas pranešimas dažnu atveju nesiskiria nuo pradinio, kitaip nei nekoduotas. Taip pat galima teigti, kad esant panašioms N ir K reikšmėms kodas sunkiau aptinka ir ištaiso klaidas.

Antrame eksperimente buvo nagrinėjama kaip klaidos tikimybė P_e paveikia kodo efektyvumą. Tikimybės reikšmės buvo naudojamos iš intervalo $[0,0001;0,5]$, pasibaigus kiekvienai iteracijai ją padidinus prie jos pridėjus 0,005. Viso eksperimento metu N ir K reikšmės buvo fiksuotos: $N = 5$, $K = 3$. Generuojanti matrica kiekvienos iteracijos metu būdavo parenkama atsitiktinai. Po kiekvienos iteracijos dekodotas pranešimas būdavo palyginimas su pradiniu pranešimu tokiu pat būdu kaip ir pirmame eksperimente. Eksperimento rezultatai pateikti grafike apačioje.



Iš grafiko galima matyti, kad esant labai mažoms klaidos tikimybėms kodo efektyvumas nenukenčia – dekodotas pranešimas nesiskiria nuo pradinio. Tačiau didėjant klaidos tikimybei, dekodotas pranešimas pradeda vis labiau skirtis nuo pradinio – maždaug nuo 0,07 dekodotas pranešimas pradeda skirtis nuo pradinio dviem ar daugiau bitų.

9. Naudotos literatūros sąrašas

- [\[VO89, §3.7, p. 78–81; ir p. 73\]](#)
- [Kodavimo teorijos paskaitų konspektai](#)
- [„TornadoFX“ dokumentacija](#)