Факултет инжењерских наука Универзитета у Крагујевцу

Сандра, С. Стефановић

НУМЕРИЧКИ МОДЕЛ СТАЦИОНАРНОГ И НЕСТАЦИОНАРНОГ ПРОВОЂЕЊА ТОПЛОТЕ
- ЦЕЛУЛАРНИ АУТОМАТИ (CA) -
- LATTICE BOLTZMANN МЕТОД (LB) -
- JAVA ПРОГРАМСКО РЕШЕЊЕ -

завршни рад

Крагујевац, 2018.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Програмски језици

Број индекса: 49/2015

Сандра, С. Стефановић

НУМЕРИЧКИ МОДЕЛ СТАЦИОНАРНОГ И НЕСТАЦИОНАРНОГ ПРОВОЂЕЊА ТОПЛОТЕ
- ЦЕЛУЛАРНИ АУТОМАТИ (CA) -
- LATTICE BOLTZMANN МЕТОД (LB) -
- JAVA ПРОГРАМСКО РЕШЕЊЕ -

завршни рад

Комисија за преглед и одбрану:

1. **Ред. проф. др Ненад Грујовић - ментор**
2. _____ Датум одбране:_____
3. _____ Оцена:_____

У оквиру завршног рада кандидат треба да примени методе Cellular Automata и Lattice Boltzmann у JAVA софтверском решењу за симулацију стационарног и нестационарног превођење топлоте, да омогући графички приказ резултата и провери и упореди добијена решења са решењима добијеним методом коначних елемената.

Препоручена литература:

[1] Burzynski M., Cudny W., Kosinski W., Cellular Automata: Structures and some applications, Journal of theoretical and applied mechanics, 42, 3, pp. 461-482, Warsaw 2004.

[2] Subhash C. Mishra, Bittagopal Mondal, Tanuj Kush, B. Siva Rama Krishna, Soloving transient heat conduction problems on uniform and non-uniform lattice using the latice Boltzmann method, Internacional Communications in Heat and Mass Transfer 36 (2009) 322-328.

[3] Miloš Kojić i dr. Metod konačnih elemenata I, Univerzitet u Kragujevcu Mašinski fakultet, Kragujevac, 2010.

Крагујевац, датум

ментор:

Др Ненад Грујовић, ред. проф.

_____

Резиме

Циљ рада је да се применом целуларних аутомата и Lattice Boltzmann-ове методе покаже расподела температуре при стационарном и нестационарном провођењу топлоте. Ради провере тачности добијени резултати се пореде са традиционалном методом за пренос топлоте, методом коначних елемената. Поред основних теоријских поставки, у раду је приказано и креирање JAVA софтвера у којима се врше неопходна израчунавања и дају прикази расподеле температуре. За креирање софтвера коришћено је развојно окружење NetBeans IDE 8.2.

Кључне речи: Стационарни и нестационарни пренос топлоте, примена целуларних аутомата при превођењу топлоте, имплементација нумеричке методе Lattice Boltzmann, JAVA софтверско решење за пренос топлоте, упоређивање резултата LBM и MKE

Summary of work

The goal of this work is to show the distribution of temperature in stationary and non-stationary heat conduction using Cellular Automata and Lattice Boltzmann's methods. In order to verify accuracy, the obitained results are compared with traditional heat transfer method, using finite element method. In addition to fundamental teoretical knowledge, the work also presents the creation of JAVA software that performs the necessary calculations and gives the displays of the temperature distribution. The NetBeans IDE 8.2 development environment was used to create software.

Key words: Stationary and non-stationary heat transfer, application of Cellular Automata in the translation of heat, implementation of the numerical method Lattice Boltzmann, JAVA software solution for heat transfer, comparison of LBM and FEM results

Садржај

Увод

Провођење топлоте представља процес у коме се врши расподела температуре у току времена, на неком растојању. Постоји три начина преноса топлоте и то су: кондукција, конвекција и радијација. При преносу топлоте јавља се температурно поље које зависи од времена и простора. Ово поље може бити стационарно и нестационарно, па и сам процес преноса топлоте може се идентично поделити. Карактеристично је да се у току времена количина протока при стационарном процесу не мења, док код нестационарног то није случај јер количина протока зависи од времена. За израчунавање расподеле температуре неопходно је познавати почетне и граничне услове.

Проблеми везани за прорачун могу се решити применом различитих метода. Најпознатија метода јесте метода коначних елемената код које се решавањем обичних и парцијалних диференцијалних једначина долази до траженог решења. За одређене проблеме овај начин решавања је доста сложен и компликован, па се уместо ове методе примењују неке друге методе.

Једна од најједноставнијих за имплементацију јесте метода ћелијских аутомата (Cellular Automata). Ова метода се заснива на процесу интеракције посматране ћелије са њој суседним ћелијама. Такође ова метода је итеративног карактера, па се са повећањем броја итерација добија све прецизнија вредност траженог решења.

На основу ове методе настала је и Lattice Boltzmann-ова метода. Обе методе имају широку примену у различитим областима. Принцип рада LB методе је сличан принципу рада целуларних аутомата. Код LB методе јављају се два процеса, то су процес судара и процес струјања.

Ова два процеса се примењују на све типове решетака и њених распореда. Проблеми који се јављају обично су 1-D, 2-D или 3-D проблеми и сваки од ових проблема има више различитих могућих распореда решетке. За сваки од ових проблема и распореда решетке користе се одговарајући параметри који фигуришу у једначини за прорачун. Основна једначина за прорачун представља Boltzmann-ову једначину и управо ова једначина приказује два процеса, односно две фазе претходно поменуте – струјање и судар.

Поред основних теоријских знања неопхоних за нумерички модел стационарног и нестационарног провођења топлоте, креирано је више врста програма и дати су примери рада ових метода.

Програми су креирани у програмском језику JAVA због свих предности које овај језик пружа, а као највећа предност истиче се могућност покретања креираних програма на било ком desktop уређају. На највећем броју уређаја JAVA је имплементирана, па из тог разлога није неопходна додатна платформа како би се програми покретали и користили. Имплементација различитих примера у овим програмима и упоређивање добијених резултата заправо представља кључни циљ овог рада. Преглед рада по поглављима дат је у наставку.

Прво поглавље се односи на температуру и топлоту, односно на начине како се температура проводи, какво може да буде температурно поље, и који су, и какви све могу да буду услови за провођење топлоте.

Друго поглавље намењено је кратком осврту на JAVA програмски језик и само окружење NetBeans IDE 8.2 које је коришћено за креирање свих софтвера поменутих у овом раду.

Треће поглавље даје дефиницију целуларних аутомата, њихове основне карактеристике, начин функционисања и примену.

Четврто поглавље односи се на Lattice Boltzmann-ов метод и све неопходне чињенице за коришћење овог модела у нумеричкој анализи.

Пето поглавље представља суштину овог рада. У њему су детаљно објашњене све верзије креираних програма, дати примери и упоређени са примерима из МКЕ. Графички су приказана сва упоређивања и дати су закључци до којих се дошло разматрајући различите проблеме и параметре.

У прилозима су дати комплетни кодови свих поменутих софтвера.

1. Температура и топлота

Температура представља меру унутрашње енергије тела, односно меру загрејаности тела. За два тела се може рећи да су у топлотном контакту ако могу размењивати топлоту, ово правило важи и за већи број тела. Када су тела у контакту и када је размена топлоте једнака нули, односно када тела имају исту температуру, постиже се термичка или топлотна равнотежа. Према томе **Нулти принцип (закон) термодинамике** говори о томе да уколико су два тела, свако појединачно, у топлотној равнотежи са трећим телом, онда су и та два тела међусобно у топлотној равнотежи [1]. Вредност температуре може се исказати коришћењем **температурних скала**: Келивинове, Целзијусове, Реомирове или Фаренхајтове. У званичној употреби су Келвинова и Целзијусова скала. За апсолутну термодинамичку температуру користи се апсолутна Келвинова температура. Претварање вредности температура из једне у другу скалу температура дато је у изразима од (1.1) до (1.4) [2].

$$T = 273{,}16 + t \tag{1.1}$$

$$t = \frac{5}{4}t_R = \frac{5}{9}(t_F - 32) \tag{1.2}$$

$$t_R = \frac{4}{5}t = \frac{4}{9}(t_F - 32) \tag{1.3}$$

$$t_F = \frac{9}{5}t + 32 = \frac{9}{4}t_R + 32 \tag{1.4}$$

T – температура по Келвиновој скали [K];
t – температура по Целзијусовој скали [^oC];
t_R – температура по Рихтеровој скали [R];
t_F – температура по Фаренхајтовој скали [F].

Топлота представља енергију насталу услед промене температуре између тела. Када се тела налазе у контакту и када између њих постоји размена топлоте, односно када тела имају различиту температуру, каже се да се таква тела не налазе у топлотној равнотежи. Између тела која имају различиту температуру, топлота прелази са тела више температуре на тело ниже температуре. Овај процес се одвија све до тренутка док се температуре тела не изједначе. Како је температура мера унутрашње енергије, тако **пренос топлоте** представља процес размене унутрашње енергије између тела, а унутрашња енергија која се том приликом размени представља **количину топлоте**. Поред ове количине топлоте, постоји и масена количина топлоте тзв. **специфична топлота** која представља топлоту коју треба довести телу јединичне масе како би му се температура променила за јединицу. **Топлотни капацитет** неког тела представља количину топлоте која се треба довести телу да би му се температура повисила за јединичну вредност.

Разликују се три начина преноса топлоте [3]:

- **Провођење** (*кондукција*);
- **Струјање** (*конвекција*);
- **Зрачење** (*радијација*).

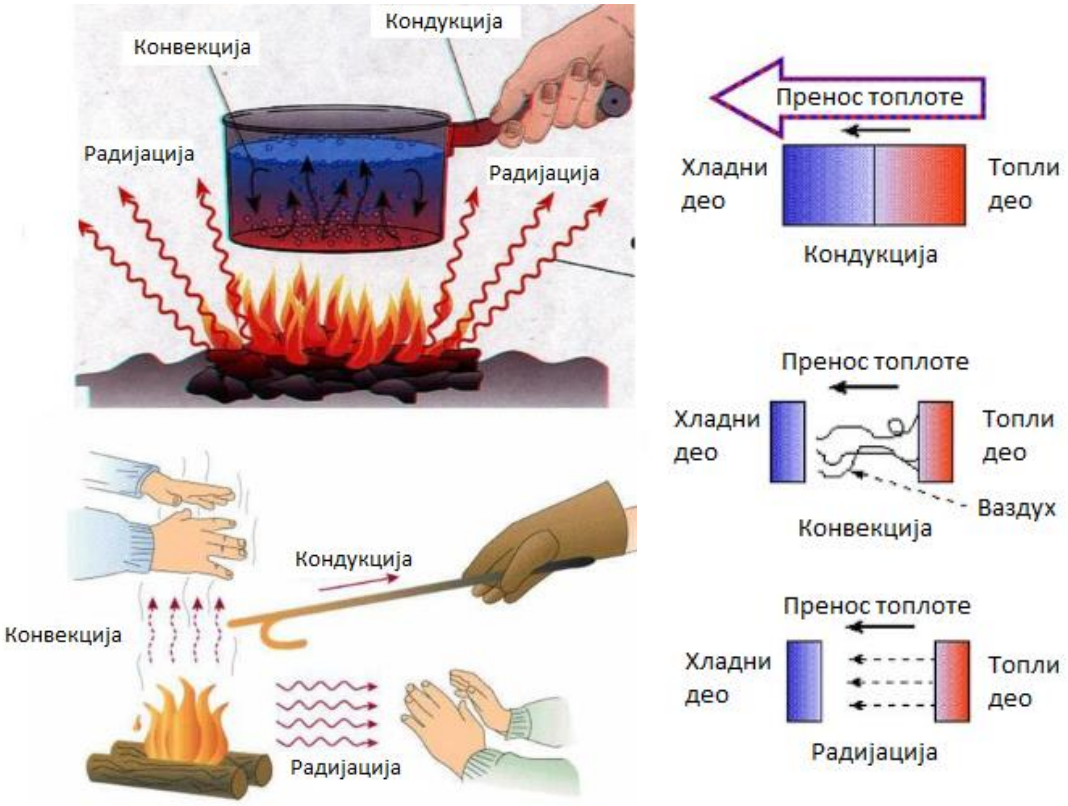
1.1. Пренос топлоте

Сва три начина преноса топлоте поменута у претходном поглављу, могу се одвијати истовремено, али је том приликом један од њих увек доминантнији у односу на остала два [4]. На слици 1 су дата сва три преноса топлоте.

- Када је реч о **преносу топлоте провођењем или кондукцијом** долази се до преноса топлоте са једног тела на друго без приметног кретања честица тела и то у правцу кретања топлотног флукса, односно са тела више температуре ка телу ниже температуре. Да би се овај процес одвијао тела морају да буду у непосредном контакту.

- Када је реч о **преносу топлоте струјањем или конвекцијом**, долази се до процеса који се одвија кретањем појединих делова тела приликом разлике у температурама.

- Када је реч о **преносу топлоте зрачењем или радијацијом**, долази се до преноса топлоте са једног тела на друго тело приликом кретања електромагнетних таласа у простору између та два тела. За пренос топлоте путем овог процеса тела се не налазе у непосредном додиру.



Слика 1: *Три начина преноса топлоте* [5]

1.1.1. Провођење или кондукција

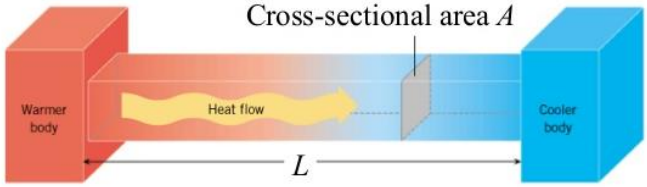
Пренос топлоте провођењем или кондукцином је процес у коме се топлота преноси директно кроз материјал и том приликом не долази до премештања саставних компоненти материјала. Тако да постоје материјали који проводе топлоту, тзв. **топлотни проводници**, и материјали који не проводе топлоту познати као **топлотни изолатори**. Овај процес се одвија интеракцијом између делића који врше термичко кретање док делови тела миријy, односно док се делови тела налазе у стационарном стању. Такође кондукција је процес који се одвија у микро размерама, што значи да се топлота предаје са једног молекула на други. Сударањем топлих честица са хладним долази до преноса топлоте. Процес превођења се одвија постепено и распростире се по читавом телу што доводи до изједначавања температуре по целој запремини материјала. Карактеристичан је за чврста тела, али се може јавити и код тела која се налазе у течном и гасовитом агрегатном стању [6]. Пренос топлоте превођењем од топлијих честица, од топлије стране тела, ка хладнијим честицама и хладнијој страни тела приказано је на слици 2.

Слика 2: *Пренос топлоте провођењем или кондукцијом* [5]

Количина топлоте Q која се кондукциојом преноси кроз материјал у облику штапа дата је једначином (1.5) [7]. На слици 3 приказан је пренос топлоте са топлијег тела на хладније. На основу једначине (1.5) долази се до закључка да се за дужи период провођења топлоте, при већој разлици температура на краћем растојању, као и при већем попречном пресеку кроз који се топлота преноси, заправо већа количина топлоте пренесе дуж тела.

$$Q = \lambda A \frac{\Delta T}{L} t \qquad (1.5)$$

Q - количина топлоте [J];
λ - коефицијент провођења топлоте $\left[\frac{W}{m\,°C}\right]$;
A - површина попречног пресека нормална на правац преноса топлоте [m^2];
ΔT - промена температуре између два краја [^{o}C];
L - дужина штапа [m];
t - време провођења топлоте [s].



Слика 3: *Пренос топлоте* [7]

Када би се количина топлоте посматрала као временски извод, добио би се **топлотни проток Q^*** који представља брзину преноса топлоте дуж штапа. Ова величина је дата следећом једначином (1.6) [4]:

$$Q^* = \frac{dQ}{dt} \tag{1.6}$$

Мерењем топлотног протока Q^* у зависности од дужине штапа L, утврђено је да се са повећањем дужине штапа топлотни проток смањује [4]. Ако се на растојању Δx између две тачке на штапу L који спаја топло и хладно тело, која се налазе у стању блиском термодинамичкој равнотежи, деси промена температуре ΔT, израз за топлотни проток постаје као у једначини (1.7).

$$Q^* = -\lambda A \frac{dT}{dx} \tag{1.7}$$

Q^* - топлотни проток [W];
$\frac{dT}{dx}$ - градијент температуре.

Коефицијент провођења топлоте λ означава количину топлоте изражену у џулима, коју у јединици времена пропусти слој неког материјала јединичне дебљине и то управно на његову јединичну површину, ако разлика температуре у стационарном стању између граничних површина материјала износи 1 °C [4]. Споменуто је већ да топлотна проводност зависи од температуре, односно да је она функција температуре, због тога, ако посматрамо општи случај, код чврстих тела и течности са повећањем температуре опада коефицијент провођења λ, док код гасова са повећањем температуре овај коефицијент расте.

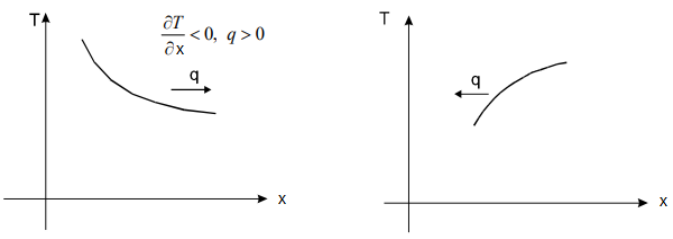
Проведена количина топлоте по јединици површине у јединици времена представља **топлотни флукс**, и дефинисана је једначином (1.8).

$$q = \frac{Q^*}{A} \tag{1.8}$$

q - топлотни флукс $\left[\frac{W}{m^2}\right]$.

Ако се једначина (1.8) замени у једначини (1.7), добија се једначина (1.9) која представља једнодимензионални облик **Фуријеовог закона за топлотну проводност**.

$$q = -\lambda \frac{dT}{dx} \tag{1.9}$$

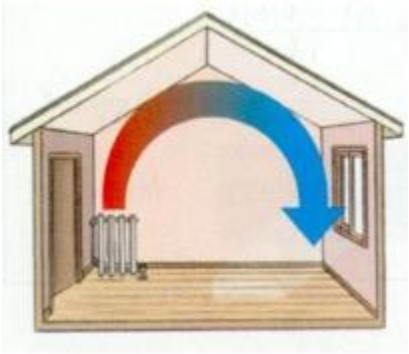


Знак "-" у једначини (1.9) даје информацију о смеру провођења топлоте, односно о смеру топлотног флукса q. Позитивна вредност значи да је његов смер у позитивном смеру x-осе, а негативна вредност значи да је у супротном смеру од позитивног смера x-осе (види слику 4).

Слика 4: *Смер топлотног флукса* [8]

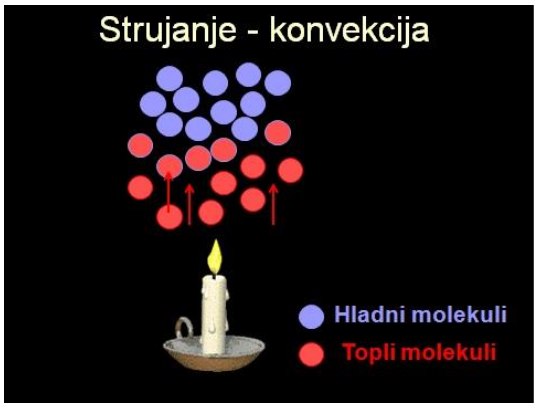
1.1.2. Струјање или конвекција

Струјање или конвекција представља пренос топлоте покретом или струјањем топлог материјала односно флуида у простору између тела. Пренос топлоте конвекцијом се најчешће остварује струјањем ваздуха (гасова), али је могуће остварити га и струјањем флуида (течности). Процес преноса топлоте путем ваздуха дат је на слици 5, док је на слици 6 дат истог овог процеса али путем флуида, док је одвијање овог процеса на нивоу молекула дато на слици 7.



Слика 5: *Конвекција путем гаса* [5]

Слика 6: *Конвекција путем флуида* [5]



Слика 7: *Пренос топлоте кондукцијом на нивоу молекула* [9]

Конвекција може да се јави у два облика:

- Природна конвекција
- Принудна конвекција

Природна конвекција настаје спонтано када топли материјал почне сам од себе да се креће, а што је последица појаве разлике у густини материјала, док **принудна конвекција** настаје под утицајем спољашњих фактора.

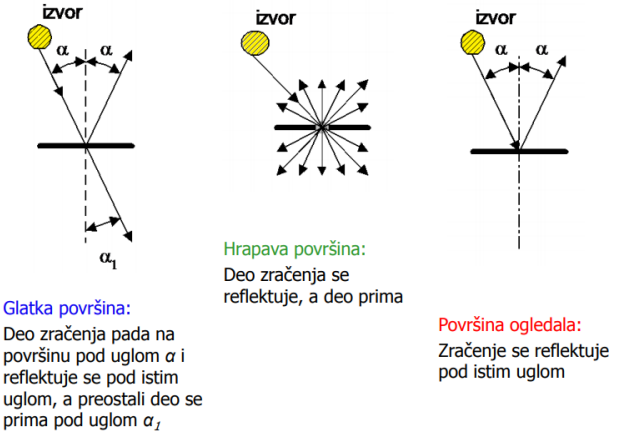
Размена количине топлоте код конвекције зависи и од додирне површине и од самог флуида или гаса. Количина топлоте Q коју површина A прими или преда конвекцијом у току времена t, а при разлици температура ΔT, дата је изразом (1.10) [1].

$$Q = h\,A\,\Delta T\,t \qquad (1.10)$$

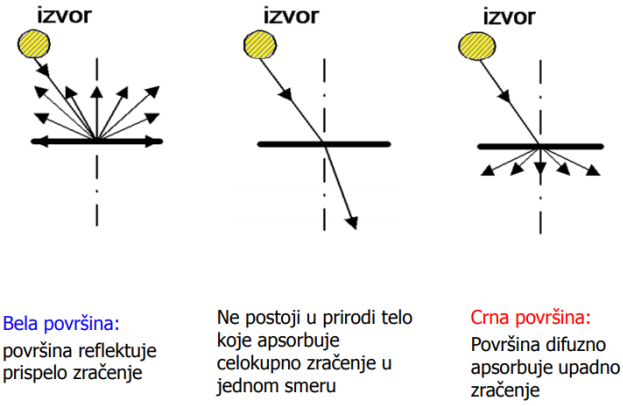
Q - количина топлоте [J];
h - коефицијент конвекције;
A - површина попречног пресека нормална на правац преноса топлоте [m^2];
ΔT - промена температуре између два краја [s];
t - време примања/предаје топлоте [s].

1.1.3. Зрачење или радијација

Пренос топлоте без директног контакта остварује се путем емитовања електромагнетног топлотног зрачења између тела. Код радијације најзначајнију улогу има површина тела које апсорбује топлоту, а што је објашњено на слици 8. У природи сва тела непрестано зраче, али и апсорбују електормагнетне таласе. Укупна енергија зрачења која се емитује на неком телу може да се делимично апсорбује, рефлектује или да прође кроз дато тело. На слици 9 приказано је рефлектовање и апсорбовање радијације. За тело које подједнако емитује и апсорбује енргију путем радијације каже се да је такво тело у термодинамичкој равнотежи са својом околином.

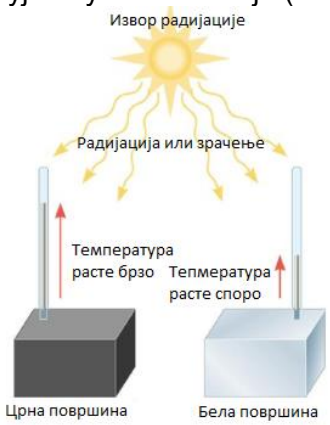


Слика 8: *Зависност радијације и површине тела које је изложено дејству радијације* [2]



Слика 9: *Апсорбција и рефлексија радијације тела у природи* [2]

На слици 10 дата је зависност између радијације, површине тела и температуре, а израз који описује ову зависност је (1.11).



Слика 10: *Зависност радијације, површине и температуре тела* [5]

$$a + r + t = 1 \qquad (1.11)$$

a - коефицијент апсорбције;
r - коефицијент рефлескије;
t - коефицијент транспаренције.

- Када је $\boldsymbol{a = 1}$ тело је апсолутно црно и апсорбује сву топлоту.
- Када је $\boldsymbol{r = 1}$ тело је апсолутно бело и рефлектује сву топлоту.
- Када је $\boldsymbol{t = 1}$ тело је апсолутно транспарентно и пропушта сву топлоту.

Количина топлоте Q која се путем зрачења емитује од стране апсолутно црног тела сразмерна је времену емитовања t, површини A и емисионој способности тела W_{ec} која је једнака производу Штефан-Болцманове константе $\sigma = 5{,}7 * 10^{-8}\ \left[\frac{W}{m^2\ K^4}\right]$ и T^4 (1.12) [1].

$$Q = \sigma\, T^4\, A\, t \qquad (1.12)$$

1.2. Температурно поље

Температурно поље је као и све физичке појаве дефинсано просторно и временски. Ово поље може да буде стационарно и нестационарно. За случај када је температура једнака у свим тачкама поља, за такво поље кажемо да је изотермно поље.

- **Стационарно температурно поље** је поље код кога се температура у посматраној тачки не мења у току времена, па температура зависи само од просторних координата (1.13).

$$T = f(x, y, z) \qquad \frac{\partial T}{\partial t} = 0 \tag{1.13}$$

- **Нестационарно температурно поље** је поље код кога се температура у посматраној тачки мења у току времена, па температура зависи и од просторних координата и од времена (1.14).

$$T = f(x, y, z, t) \tag{1.14}$$

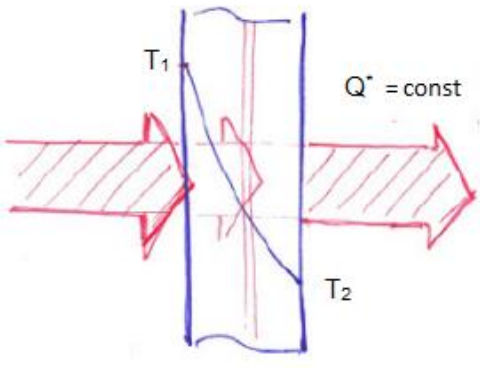
Поред стационарног и нестационарног температурног поља, постоји и стационарни и нестационарни пренос топлоте. Ови преноси топлоте дати су на сликама 11 и 12.

- **Стационарни пренос топлоте** је процес размене топлоте када у једнаким временским интервалима долази до преноса једнаке количине топлоте са једног тела на друго. У овом случају топлотни проток не зависи од времена, односно није функција времена (1.15).

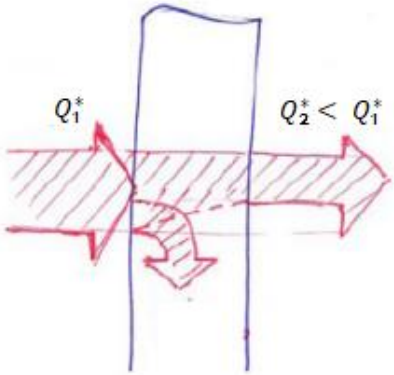
$$\frac{dQ}{dt} = const \tag{1.15}$$

- **Нестационарни пренос топлоте** је процес размене топлоте када се топлотни проток мења током времена, односно код нестационарног преноса топлоте долази до преноса различите количине топлоте у различитим временским тренутцима (1.16).

$$\frac{dQ}{dt} \neq const \tag{1.16}$$



Слика 11: *Стационарни пренос топлоте* [4]

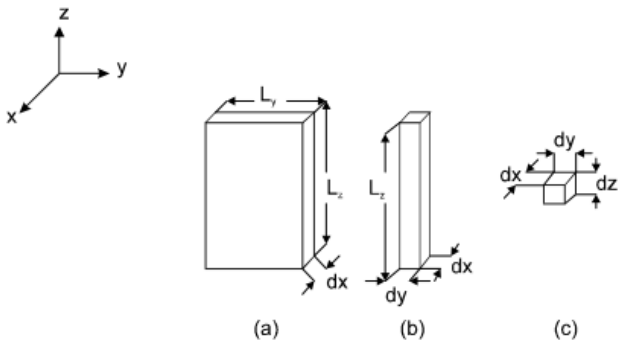


Слика 12: *Нестационарни пренос топлоте* [4]

1.2.1. Нестационарни пренос топлоте кондукцијом

Температура нестационарног система са расподељеним параметрима је функција времена и једне или више координата. Математички модели нестационарног преноса топлоте у систему са расподељеним параметрима, чијим се решавањем добија температурно поље, имају облик парцијалних диференцијалних једначина и могу се извести формулисањем енергетског биланса чија општа форма гласи (1.17) [10]:

$$\text{Улаз} - \text{Излаз} + \text{Генерисање у систему} = \text{Акумулација у систему} \qquad (1.17)$$

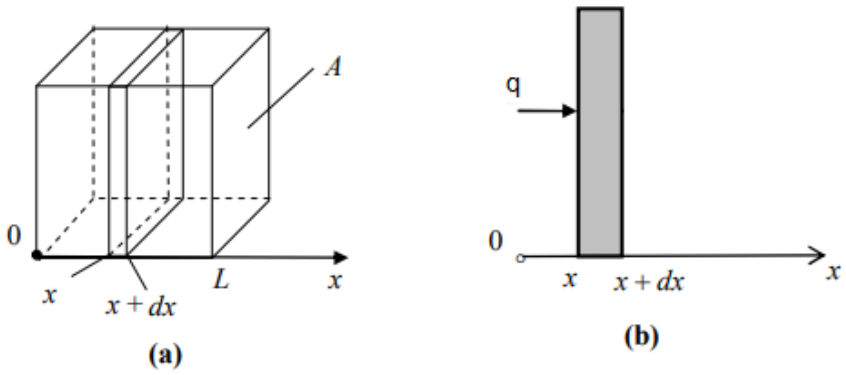


Горе поменута форма представља једначину за део система чије су димензије бесконачно мале у оним правцима дуж којих се температура мења, а у осталим правцима део се простире до граница система [10]. Пример са димензијама L_x x L_yx L_z је дат на слици 13.

Слика 13 (лево)*: Систем са расподељеним параметрима: а)Температура се мења само у x-правцу,* $T(x,t)$*; b) Температура се мења у x и y-правцу,* $T(x,y,t)$*; c) Температура се мења у сва три правца,* $T(x,y,z,t)$ [10]

Најједноставнији случај нестационарног преноса топлоте је **једнодимензионални пренос топлоте** и он подразумева пренос топлоте у току времена по једном од праваца координатног система. За случај правоугаоне геомертије код које се температура мења само у аксијалном правцу, односно по дужини x*-координате*, температура је дата као функција $T(x,t)$.

Правоугаоно тело, облика равног зида, по чијој дебљини се, у x*-правцу*, мења температурно поље, у зависности од флукса топлоте, дато је на слици 14 (а). Да би се извела једначина топлоте за ово тело, узима се слој бесконачно мале дебљине dx који је нормалан на правац преноса топлоте, а што је приказано на слици 14 (b).



Слика 15: *Пренос топлоте кроз раван зид: а) Тело правоуганог облика за које се прорачунава пренос топлоте; b) Бесконачно мали део нормалан на правац преноса топлоте* [4]

1.2.2. Почетни и гранични услови

За добијање партикуларног решења диференцијалних једначина преноса топлоте, односно за добијање температурног поља потребно је поставити и граничне услове. Како би се одредила промена температуре у току времена потребно је телу задати неке граничне вредности, односно потребно је истаћи какав је почетни температурни профил тела. За одређивање граничних услова користе се изводи по времену и просторним координатама. Односно потребно је поставити почетни гранични услов по времену и два гранична услова по просторној координати.

За посматрање промене температуре у телу, ако је посматрано тело раван зид, почетни гранични услов се дефинише формулом (1.17), у случају да је температурни профил тела униформан користи се формула (1.18), ове формуле важе у случају да је задата температура тела.

$$t = 0: \qquad \mathrm{T}(x, 0) = f(x) \tag{1.17}$$

$$t = 0: \qquad T(x, 0) = const \tag{1.18}$$

Поред ових, почетних услова, неопходно је познавати и граничне услове, односно услове на граничној површини тела са околином. Ови услови се код равног зида дефинишу на левој и десној граници. тј. за: $x = 0$ и $x = L$. Ове границе приказане су на слици 15 (а).

У погледу математичке форме, разликују се три типа граничних услова за обичне и парцијалне диференцијалне једначине [4]:

1. **Дирихлеов** (*Dirichlet*);
2. **Нојманов** (*Neuman*);
3. **Робинов** (*Robin*).

Дирихлеов услов даје вредност температуре на граници, код проблема преноса топлоте, овај услов се односи на то да је позната температура на додирној површини са околином. Једначином (1.19) приказан је Дирихлеов услов када је позната температура $T = T_1$ на левој страни зида за $x = 0$, а једначином (1.20) приказан је исти овај услов за познату темепературу, али на десној страни зида за $x = L$.

$$x = 0: \qquad \mathrm{T}(0, t) = \mathrm{T}_1 \qquad (t > 0) \tag{1.19}$$

$$x = L: \qquad \mathrm{T}(L, t) = \mathrm{T}_1 \qquad (t > 0) \tag{1.20}$$

Нојманов услов даје вредности извода температуре на граници. Пренос топлоте се јавља када се тело загрева или хлади под утицајем околине. Једначином (1.21) дат је Нојманов услов који представља математички услов екстремума:

$$x = 0: \qquad \frac{\partial T(x,t)}{\partial x} = 0 \qquad (t > 0) \tag{1.21}$$

Робинов гранични услов даје задату вредност у облику линеарне комбинације функције и њеног извода.

Што се почетних и граничних услова тиче поред задате температуре ови услови могу бити дати и помоћу задатог флукса, конвективног услова или радијационог услова. За све ове услове приказане су формуле карактеристичне за прорачун и оне су дате изразима од (1.22) до (1.29). На слици 16 дати су одговарајући зидови и почетни и гранични услови на левој и десној страни зида, односно на почетку и крају зида.

Почетни и гранични услови када је задата [11]:

1. **Температура**

Леви услов: $T(0,t) = T_1$ (1.22)

Десни услов: $T(L,t) = T_2$ (1.23)

2. **Флукс**

Леви услов: $q = -\lambda \frac{dT(0,t)}{dx}$ (1.24)

Десни услов: $q = -\lambda \frac{dT(L,t)}{dx}$ (1.25)

3. **Конвективност**

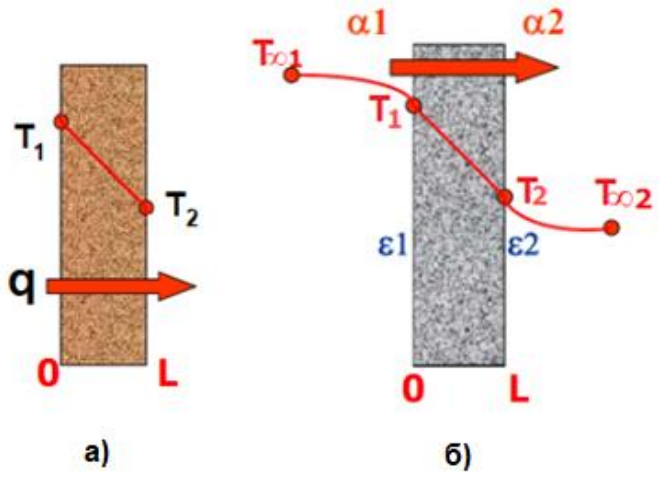
Леви услов: $-\lambda \frac{dT(0,t)}{dx} = \alpha_{1[T_{\infty 1} - T(0,t)]}$ (1.26)

Десни услов: $-\lambda \frac{dT(L,t)}{dx} = \alpha_{2[T(L,t) - T_{\infty 2}]}$ (1.27)

4. **Радијација**

Леви услов: $-\lambda \frac{dT(0,t)}{dx} = \varepsilon_1 \sigma [T_{\infty 1}^4 - T^4(0,t)]$ (1.28)

Десни услов: $-\lambda \frac{dT(L,t)}{dx} = \varepsilon_2 \sigma [T^4(L,t) - T_{\infty 1}^4]$ (1.29)



Слика 16: П*очетни и гранични услови температурног поља: а) Задата температура или флукс; б) Задат конвективни или радиојациони услов* [11]

2. Објектно орјентисано програмирање (ООП)

Код објектно орјентисаног програмирања све се састоји из **објеката**, односно из блокова неопходних за пројектовање рачунарских програма и апликација. Објектно орјентисано програмирање је програмирање код кога се може користити **велики број модула кода**, од којих сваки нуди специфичну функционалност за креирање апликација. Објектно орјентисано програмирање (ООП) представља тзв. **модуларно програмирање** које нам омогућава већу манипулацију и коришћење кода.

Основни елементи објектно орјентисаног програмирања су [12]:

- **Енкапсулација (*учауривање*)** – поступак обједињавања стања и понашања у једну целину и обезбеђивање контроле приступа. Крајњи резултат је класа.
- **Полиморфизам (*вишеобличност*)** – поступак помоћу кога можемо доделити променљиву основног типа променљивој изведеног типа.
- **Наслеђивање** – могућност хијерархијске организације класа, где када једна класа наследи другу, она задржава комплетан садржај класе коју је наследила.
- **Модуларност** – поступак разбијања програма на мање делове који могу самостално да функционишу.

2.1. JAVA програмски језик

JAVA је објектно орјентисани програмски језик и рачунарска платформа. Направљен је тако да подсећа на језик C++, али је једноставнији за употребу. Овај програмски језик развила је компанија Sun Microsystems, 1995. године и служи за израду апликација. Назив је добила по Индонежанском острву Јава [13].

Једна класа код писања JAVA кода налази се у једном фајлу уколико није реч о инутрашњим класама. Изворни код код овог програмског језика чува се у фајловима са наставком .java. Ови кодови се не компајлирају, већ се преводе у бајт-кодове, па се тако преведени фајлови чувају са екстензијом *.class*. Како би дошло до извршавања кода писаног у JAVA програмском језику, неопходно је имати **Java Виртуалну Машину** (***J**ava **V**irtual **M**achine*). Њеним коришћењем постиже се независност платформе, што омогућава да се исти програм може извршавати на различитим оперативним системима уколико се поседује JVM.

2.2. NetBeans IDE 8.2 радно окружење

JAVA програм за нестационарни пренос топлоте је настао у NetBeans IDE 8.2 радном окружењу. NetBeans је платформа за развој генеричких Swing апликација. Он пружа поуздану и флексибилну архитектуру апликације, омогућава аутоматско генерисање кода и једноставно повезивање ставки са алатима. Коришћење ове платформе штеди количину времена и рада [14].

За рад у овом програму најпре је неопходно са званичног сајта www.oracle.com преузети одговарајући JDK (*Java Development Kit*) и NetBeans. Приликом преузимања ове апликације потребно је водити рачуна о верзији оперативног сисема који се користи. За потребе креирања програма за прорачун нестационарног преноса топлоте преузета је верзија *Windows x64*, док је за JDK преузета верзија *jdk-8u161-nb-8_2-windows-x64.exe* [15].

3. Cellular Automata (CA)

Cellular Automata (*Ћелијски аутомат*) представља дискретни динамички систем чије понашање је потпуно специфично у погледу једноставних локалних односа. Такође представљају математички модел просторно распоређених процеса који могу довести до одговарајуће симулације сложених динамичких процеса. Резултати симулација Cellular Automata дају основу за предлагање опције Cellular Automata као алата у моделирању и решавању проблема комплексне природе који су некада недовољно познати. Cellular Automata описује понашање система са великим бројем дискретних степени слободе користећи диференцијалне једначине које су биле погодне за системе са малим бројем степени слободе који се стално развијају.

Реални модели, засновани на физичким законима, потичу из великих система нелинарних интегралних или парцијално-диференцијалних једначина. Као и сваки приступ и он има својих предности и мана. Највећи недостатак представља нумеричка непоузданост оваквих система због замагљивања основних физичких закона. Овај проблем се поједностављује употребом Cellular Automata, чији је приступ имитирање физичких закона низом једноставних правила која се лако израчунавају [16]. Основу приступа Cellular Automata чини следеће запажање: *„Не описивати сложене системе помоћу сложених једначина, али нека комплексност произилази из интеракције једноставних појединости које прате једноставана правила.“*

3.1. Дефинисање

Постоји велики број дефиниција Cellular Automata, овде су издвојене две:

Деф. 1. *„Ћелијски аутомати су динамички системи у којима су простор и време дискретни. Ћелијски аутомат се састоји од редовне мреже ћелија, од којих свака може бити у броју к могућих стања, ажурирана синхроно у дискретним временских корацима у складу са локалним, истоветним правилима интракције. Стање ћелије одређује претходно стање околног суседства ћелије.“*

Деф. 2. *„Ћелијски аутомат је (линеарни) низ ћелија које комуницирају са својим суседима. Низ може преузети било који број димензија, почевши од једнодимензионалног низа ћелија. Свака ћелија има свој скуп стања. Пријемом уноса из повезаних ћелија (или генерално, порука од својих суседа), ћелија користи сопствене скупове правила да би утврдила каква ће реакција бити. Ова реакција се манифестује као промена сопственог стања и такође може бити и покретач за слање порука суседима. Порука се преноси на суседне ћелије што њих доводи до тога да се понашају на исти начин.“*

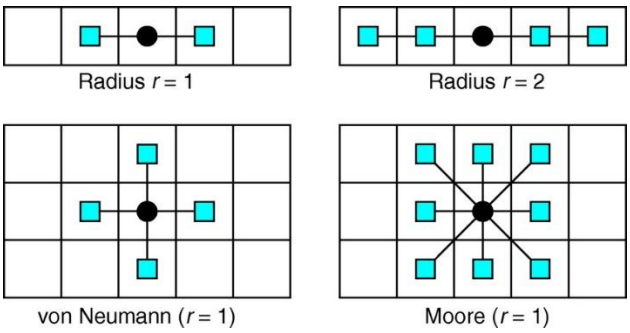
3.2. Карактеристике и примена

Главне карактеристике Cellular Automata одређују њихову структуру и понашање, и то су:

- **Паралелност** – појединачна ажурирања ћелија се врше међусобно независно, односно сва ажурирања се врше у исто време;
- **Локалитет** – приликом ажурирања стање ћелије зависи од претходног стања ћелије и њених суседа;
- **Хомогеност** – све ћелије се ажурирају по истим правилима, стање ћелије и њеног најближег суседства повезано је неком алгебарском или логичком формулом.

Примена Cellular Automata, новог приступа моделирању различитих природних феномена, може се представити решавањем следећих проблема: еколошки систем предатора и плена, стационарни и нестационарни пренос топлоте, ширење нафтне мрље, ток саобраћаја, ширење лека у ткиву, и др.

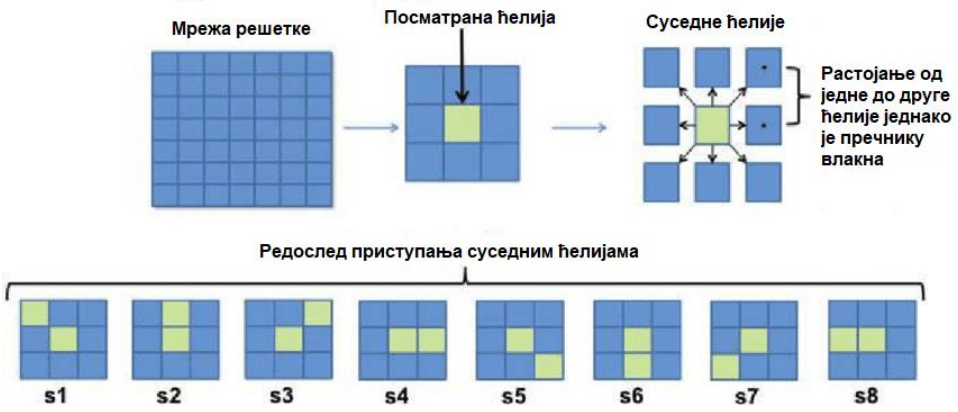
3.3. Начин функционисања



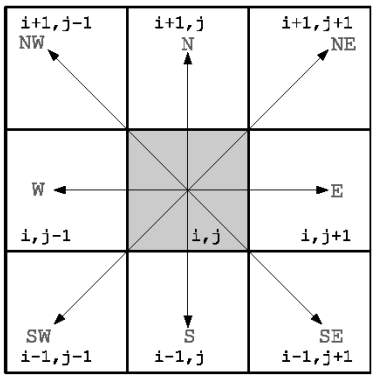
Функционисање ћелијских аутомата се занива на реакцији тренутне ћелије са њеним околним ћелијама. Овај поступак је итеративан и свако наредно стање ћелије зависи од претходног стања суседних ћелија, као и о временском кораку у коме се ћелијски аутомат налази. За добијање вредности у тренутној ћелији могу се применити различита правила по којима се дефиниши суседи ћелије и она су дата на слици 17.

Слика 17: *Правила за дефинисање суседних ћелија ћелијског аутомата* [17]

На слици 18 дат је приказ мреже, посматране ћелије и суседних ћелија, као и редослед приступања посматране (централне) ћелије њеним суседним ћелијама.



Слика 18: *Графички приказ ћелије и њених суседних ћелија* [18]



Употреба ћелијских аутомата је намењена за компијутерску примену, па је на слици 19 приказано како се у програмирању дефинишу посматрана ћелија и њене суседне ћелије. Посматрана ћелија се налази у центру и окружена је са осам страна различитим суседним ћелијама.

Слика 19 (лево): *Означавање ћелија у програмирању* [19]

4. The Lattice Boltzman-ов метод (LBM)

Lattice Boltzman-ов метод је метод који је базиран на принципу cellular automata (ћелијских аутомата). Овај метод ради са нормализованим временским корацима на домену где су све ћелије нормализоване у простору. Свака од ових ћелија садржи коначан број дискретних стања које се још називају и функције дистрибуције. Сва ова стања се на сваком појединачном временском кораку, у свим ћелијама домена, синхроно ажурирају, односно у сваком временском тренутку долази до синхоног ажурирања сваке функције дистрибуције. Правила ажурирања ових стања су детерминистичка и једнака дуж целог дискретног домена. Еволуција ћелија зависи само од коначних стања суседних ћелија [20]. Што значи да стање у тренутној ћелији зависи од тренутних стања њених околних (суседних) ћелија. Из тог разлога LBM представља алтрнативно решење традиционалним *Navier–Stokes решењима* у домену компијутерске динамике флуида (*Computational Fluid Dynamics* – CFD). Велика потреба за капацитетом моморије и рачунарским временом доводи до тога да се многи проблеми CFD-а не могу решити употребом само једног рачунара за одговарајуће време. Због тога је уведен LBM који омогућава паралелни приступ меморији, а што доводи до смањења времена потребног за решавање проблема.

Године 1988. McNamara и Zanetti су представили **Lattice Boltzman-ов метод**, као метод који превазилази недостатке Lattice Gas Cellular Automata (LGCA) – гасних ћелијских аутомата [21]. Од тада LBM се појављује као алтернативни метод за решавање проблема динамке флуида, као што су: CFD, Navier–Stokes једначине за израчунавање масе, импулса или енергије на дискретним чворовима, елементима или запреминама. Другим речима, нелинеарне парцијалне диференцијалне једначине се трансформишу у сет нелинеарних алгебарсих једначина које се решавају итеративно, по пролазима. Према LBM методи течност се апроксимира честицама које се сударају. Ове честице струје дуж одређених праваца кроз решетку мреже и сударају се на мрежи. Ова метода се може сматрати експлицитном. Процеси струјања и сударања се одвијају локално, па се може извршити природно програмирање за паралелну обраду машине.

4.1. Предности и недостаци

LBM је метод који поседује предности и макроскомпсик и микроскопских приступа, са подесивим рачунарским ресурсима, због чега се може рећи да Lattice Boltzmann-ов метод има много предности и мало недостатака.

Предности ове методе су [22]:

- Врло једноставан алгоритам за имплементацију и разумевање, који је у својој природи паралелан и могуће га је прилагодити за рад на GPU;
- Нема проблема са сложеним и неправилним границама;
- Главна формула је једноставна и брзо се израчунава;
- Може лако да симулира мезоскопска и микроскопска својства флуида.

Недостаци ове методе су [22]:

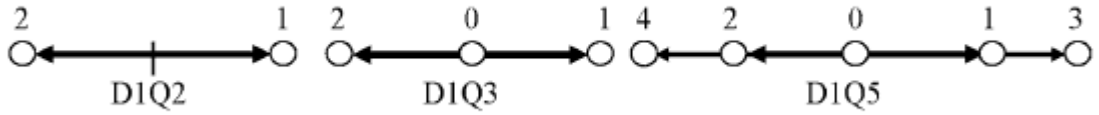
- Лоша скалабилност, јер се подаци чувају у мрежи;
- Строг, тј. мали временски корак може довести до нестабилности (због брзог израчунавања до овог недостатка најчешће не долази).

4.2. Распоред решетке

За представљање димензије проблема и броја брзина, у LBM методи се користи стандардни израз **DnQm**, код кога је са n – означена димензија проблема која се представља помоћу бројева: 1 – за једнодимензионалне, 2 – за дводимензионалне и 3 – за тродимензионалне проблеме, док *m* – представља број брзина модела, односно број веза. За све ове проблеме (D1, D2, D3), постоји одговарајуће решење решетке које се користи: D1Q2, D1Q3, D1Q5, D2Q4, D2Q5, D2Q9, D3Q15, D3Q19. У наставку су приказане неке од ових решетки.

4.2.1. 1-D решетка

За једнодимензионални проблем користе се решетке D1Q2, D1Q3, D1Q5. На слици 20 приказана су сва три примера решетке. Белим круговима је означени чворови, а црним стрелицама су означени правци, док су бројевима означен број брзина. Чвор у средини код D1Q3 и D1Q5, означава централни чвор, док остали чворови представљају суседне чворове. Струјање се одвија од централних чворова ка суседним чворовима одређеном брзином, што је и представљено стрелицама, а ова брзина се још назива и брзина решетке. За једнодимензионалне решетке број брзина може бити 2, 3 или 5.



Слика 20: *Распоред решетке за 1-D проблем* [21]

Код **D1Q2** распореда решетке постоје два вектора брзине (C_1 и C_2) за функције расподеле редом f_1 и f_2. Вредности за ове векторе брзина дате редом: 1 и -1, у случају када је $dx = dt$, у супротном потребно их је израчунати према формулама датим у једначини (4.1).

$$c_1 = \frac{\Delta x}{\Delta t}; \qquad c_2 = -\frac{\Delta x}{\Delta t}; \tag{4.1}$$

Δx – дужина корака;
Δt – временски корак;

Према овој расподели укупан број коришћених честица не сме бити већи од две честице. Две суседне честице се померају једна у леви, а друга у десни чвор за време процеса струјања. Тежински фактор или коефицијент w_i за f_1и f_2има вредност $\frac{1}{2}$. Брзина звука за овај распоред решетке износи $c_s = \frac{1}{\sqrt{2}}$.

Код **D1Q3** распореда решетке постоји три вектора брзине (C_0, C_1и C_2) за функције расподеле редом f_0, f_1и f_2. Вредности за ове векторе брзина дате редом: 0, 1 и -1, у случају када је $dx = dt$, у супротном потребно их је израчунати према формулама датим у једначини (4.2).

$$c_0 = 0; \qquad c_1 = \frac{\Delta x}{\Delta t}; \qquad c_2 = -\frac{\Delta x}{\Delta t}; \tag{4.2}$$

Према овој расподели укупан број коришћених честица не сме бити већи од три честице, а једина честица која се не креће јесте централна честица. Остале две, суседне честице се померају једна у леви, а друга у десни чвор за време процеса струјања. Тежински фактор или коефицијент w_i за f_0има вредност $\frac{4}{6}$, за f_1и f_2има вредност $\frac{1}{6}$. Брзина звука за овај распоред решетке износи $c_s = \frac{1}{\sqrt{3}}$.

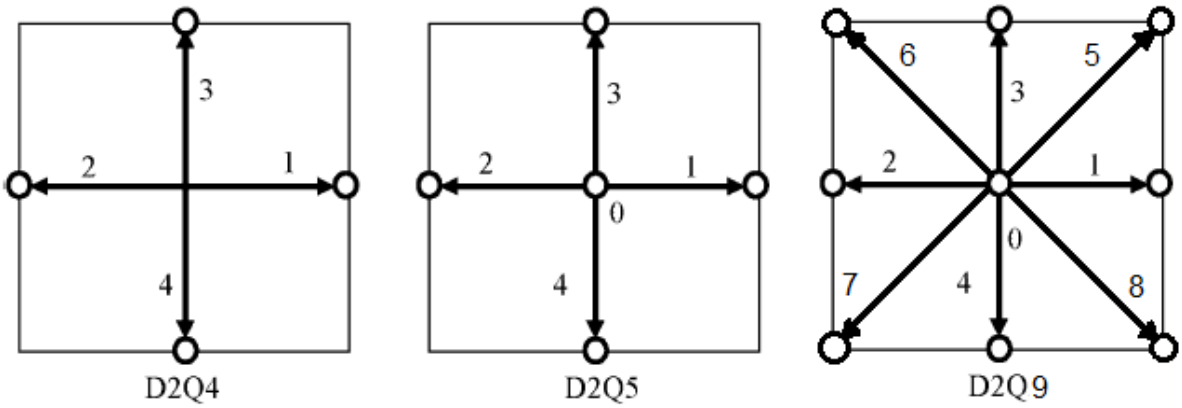
Код **D1Q5** распореда решетке постоји пет вектора брзине (C_0, C_1, C_2, C_3и C_4) за функције расподеле редом f_0, f_1, f_2, f_3и f_4. Вредности за ове векторе брзина потребно је израчунати према формулама датим у једначини (4.3).

$$c_0 = 0; \qquad c_1 = \frac{\Delta x}{\Delta t}; \qquad c_2 = -\frac{\Delta x}{\Delta t}; \qquad c_1 = 2\frac{\Delta x}{\Delta t}; \qquad c_2 = -2\frac{\Delta x}{\Delta t}; \qquad (4.3)$$

Према овој расподели укупан број коришћених честица не сме бити већи од пет честице, а једина честица која се не креће јесте централна честица. Остале четири, суседне честице се померају две за по једно место, а друге две за по два места: једна у леви, а друга у десни чвор, за време процеса струјања. Тежински фактор или коефицијент w_i за f_0има вредност $\frac{6}{12}$, за f_1и f_2има вредност $\frac{2}{12}$, а за f_3и f_4има вредност $\frac{1}{12}$. Брзина звука за овај распоред решетке износи $c_s = \frac{1}{\sqrt{3}}$, што је исто као и код D1Q3.

4.2.2. 2D решетка

За дводимензионални проблем користе се решетке D2Q4, D2Q5, D2Q9. На слици 21 приказана су сва три примера решетке. Белим круговима су означени чворови, а црним стрелицама су означени правци, док је бројевима означен број брзина. Чвор у средини код D2Q5 и D2Q9, означава централни чвор, док остали чворови представљају суседне чворове. Струјање се одвија од централних чворова ка суседним чворовима одређеном брзином, што је и представљено стрелицама, а ова брзина се још назива и брзина решетке. За дводимензионалне решетке број брзина може бити 4, 5 или 9.



Слика 21: *Распоред решетке за 2-D проблем* [21]

Код **D2Q4** распореда решетке постоји четири вектора брзине (C_1, C_2, C_3и C_4) за функције расподеле редом f_1, f_2, f_3и f_4. Вредности за ове векторе брзина дате су редом: $C_1(1,0)$, $C_2(-1,0)$, $C_3(0,1)$, $C_4(0,-1)$, у случају када је $\Delta x = \Delta y = dt$, у супротном потребно их је

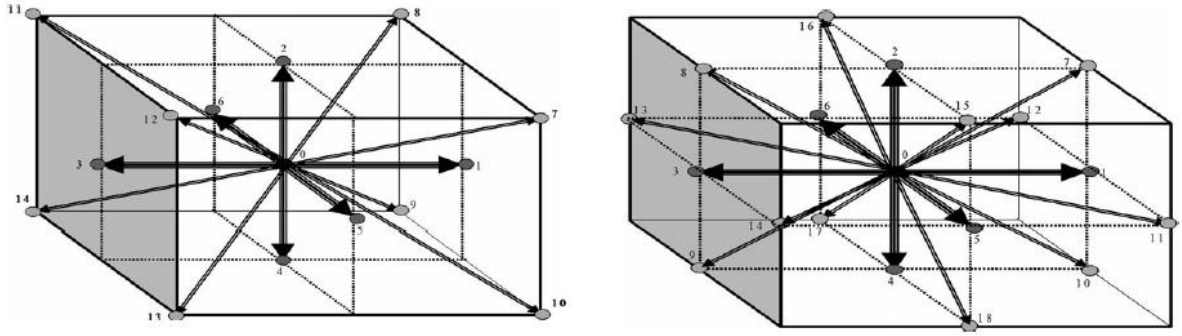
израчунати као што је проказано за 1-D проблем. Тежински коефицијент w_i за све f_1, f_2, f_3 и f_4има вредност $\frac{1}{4}$.

Код **D2Q5** распореда решетке постоји пет вектора брзине (C_0, C_1, C_2, C_3и C_4) за функције расподеле редом f_0, f_1, f_2, f_3и f_4. Вредности за ове векторе брзина дате су редом: $C_0(0,0)$, $C_1(1,0)$, $C_2(-1,0)$, $C_3(0,1)$, $C_4(0,-1)$, у случају када је $\Delta x = \Delta y = dt$, у супротном потребно их је израчунати као што је проказано за 1-D проблем. Тежински коефицијент w_i за f_0 има вредност $\frac{2}{6}$, за f_1, f_2, f_3и f_4 има вредност $\frac{1}{6}$.

Код **D2Q9** распореда решетке постоји девет вектора брзине (C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7и C_8) за функције расподеле редом f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7и f_8. Вредности за ове векторе брзина дате су редом: $C_0(0,0)$, $C_1(1,0)$, $C_2(-1,0)$, $C_3(0,1)$, $C_4(0,-1)$, $C_5(1,1)$, $6(-1,1)$, $C_7(-1,1)$, $C_8(-1,-1)$ у случају када је $\Delta x = \Delta y = dt$, у супротном потребно их је израчунати као што је проказано за 1-D проблем. Тежински коефицијент w_i за f_0има вредност $\frac{4}{9}$, за f_1, f_2, f_3и f_4 има вредност $\frac{1}{9}$, а за f_5, f_6, f_7и f_8има вредност $\frac{1}{36}$.

4.2.3. 3D решетка

За тродимензионални проблем користе се решетке D3Q15 и D3Q19. На слици 22 приказана су оба примера решетке. Сивим круговима су означени чворови, а црним стрелицама су означени правци, док је бројевима означен број брзина. Чвор у средини код D3Q15 и D3Q19, означава централни чвор, док остали чворови представљају суседне чворове. Струјање се одвија од централних чворова ка суседним чворовима одређеном брзином, што је и представљено стрелицама, а ова брзина се још назива и брзина решетке. За тродимензионалне решетке број брзина може бити 15 или 19.



Слика 22: *Распоред решетке за 3-D проблем* [21]

Код **D3Q15** распореда решетке постоји петнаест вектора брзине (C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, и C_{14}) за функције расподеле редом f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13} и f_{14}. Вредности за ове векторе брзина дате су редом: $C_0(0,0,0)$, $C_1(1,0,0)$, $C_2(0,1,0)$, $C_3(-1,0,0)$, $C_4(0,-1,0)$, $C_5(0,0,1)$, $C_6(0,0,-1)$, $C_7(1,1,1)$, $C_8(1,1,-1)$, $C_9(1,-1,-1)$, $C_{10}(1,-1,\ 1)$, $C_{11}(-1,1,-1)$, $C_{12}(-1,1,1)$, $C_{13}(-1,-1,1)$, $C_{14}(-1,-1,-1)$.
Тежински коефицијент w_i за f_0има вредност $\frac{16}{72}$, за f_1, f_2, f_3, f_4, f_5 и f_6има вредност $\frac{8}{72}$ и за f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13} и f_{14} има вредност $\frac{1}{72}$.

Код **D3Q19** распореда решетке постоји деветнаест вектора брзина (C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, C_{16}, C_{17} и C_{18}) за функције расподеле редом f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, f_{17} и f_{18}. Тежински коефицијент w_i за f_0има вредност $\frac{12}{36}$, за f_1, f_2, f_3, f_4, f_5 и f_6има вредност $\frac{2}{36}$ и за f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, f_{17} и f_{18} има вредност $\frac{1}{36}$.

4.3. The Lattice Boltzman-ове једначине

Boltzmann-ова једначина представља једначину која описује слободно кретање атома и његову промену насталу услед судара са другим атомима у току кретања. Када се посматра слободно кретање атома подразумева се да оно не долази у контакт са другим атомима, односно нема сударања са другим атомима у правцу његовог кретања. Одатле произилази да се функција расподеле у току времена не мења што је представљено изразом (4.4) [23].

$$\frac{df}{dt} = 0 \tag{4.4}$$

Функција расподеле f представља функцију која зависи од времена t, положаја x и брзине e, $f = f(t, x, e)$. Када се каже да се функција расподеле мења у току времена, тада се морају узети у обзир и судари, па претходна једначина са десне стране добија колизиони оператор. Овако промењена једначина представља Boltzmann-ову једначину која је дата изразом (4.5). Коначан облик ова једначина добиће у наставку овог поглавља.

$$\frac{df}{dt} = \Omega(f) \tag{4.5}$$

Лева страна ове једначине се за општи случај, у коме важи да је функција расподеле у ствари функција времена, положаја и брзине, када и положај x и брзина e зависе од промене времена, односно када су обе ове вредности функције времена, лева страна једначине добија се помоћу тоталног диференцијала. Пошто се посматра дејство интеракције између честица, приликом писања једначине тоталног диференцијала занемарује се дејство спољашњих сила и ова једначина има облик (4.6):

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + e\nabla f \tag{4.6}$$

Десна страна Boltzmann-ове једначине, као што је већ поменуто, представља колизиони оператор, односно колизиони интеграл. Овај интеграл је јако сложен и представља највећу препреку у решавању Boltzmann-ове једначине, због чега су и настали различити математички модели за решавање колизионих интеграла.

Модели који решавају колизионе интеграла називају се колизиони модели. Ови модели морају да задовољавају два основна услова, а то су:

- да поједностављују колизиони интеграл који мењају;
- да поседују главне особине колизионог интеграла који мењају.

Најпознатији колизиони модел је колизиони модел BGK кога су поставили Bhatnagar, Gross, и Krook, и по чијим је почетнм словима овај модел и добио назив. Такође за решавање

парцијалне диференцијалне једначине колизионог интеграла могуће је користити и Метод коначних разлика.

Bhatnagar, Gross и Krook су 1954. године увели једноставан оператор, који није правио значајне грешке, да апроксимира колизиони оператор. Исте године Welander је представио сличан оператор. Овај оператор приказан је формулом (4.7) [21].

$$\Omega = \omega(f^{eq} - f) = \frac{1}{\tau}(f^{eq} - f) \tag{4.7}$$

Коефицијент ω представља колизиону фреквенцију, док τ представља фактор релаксације. Коефицијент ω дат је изразом (4.8). Уколико промена у току времена није једнака јединици, односно уколико постоји промена времена Δt, коефицијен ω се израчунава према формули (4.9), а фактор τ представља време релаксације. Израз којим се може добити време релаксације дат је са (4.10).

$$\omega = \frac{1}{\tau} \tag{4.8}$$

$$\omega = \frac{\Delta t}{\tau} \tag{4.9}$$

$$\tau = \frac{\alpha}{\left|\vec{e_i}\right|^2} + \frac{\Delta t}{2} \tag{4.10}$$

α – термална дифузност

Топлотна дифузност α зависи од топлотне проводности k, густине ρ и специфичне топлоте c_p које представљају параметре материјала. Једначина према којој се израчунава топлотна дифузност дата је изразом (4.11).

$$\alpha = \frac{k}{\rho c_p} \tag{4.11}$$

Еквивалентна (локална) функција расподеле f^{eq} представља Maxwll-Boltzmann-ову функцију расподеле. Њена општа једначина за 1-D проблем дата је једначином (4.12). Ова функција расподеле одређује врсту проблема који се решава и њена највећа предност је то што је једноставна и може се применити на велики број проблема са малим изменама и једноставном имплементацијом.

$$f_i^{eq} = w_i T(x,t) \tag{4.12}$$

w_i – одговарајући тежински коефицијент

Тежински коефицијент w_i мора да буде једнак јединици, односно сума свих тежинских коефицијената који фигуришу у једначини мора да буде једнака јединици, што је и приказано изразом (4.13). За D1Q2 решетку ови коефицијенти су дати изразом (4.14).

$$\sum_{i=1}^{m} w_i = 1 \tag{4.13}$$

$$w_1 = \ w_2 = \frac{1}{2} = 0.5 \tag{4.14}$$

Изједначавањем леве и десне стране, односно изједначавањем једначине (4.6) са једначином (4.7) добија се коначан облик Boltzmann-ове једначине и он је дат изразом (4.15).

$$\frac{\partial f}{\partial t} + e\nabla f = \frac{1}{\tau}(f^{eq} - f) \tag{4.15}$$

У случају дискретизације овог метода претходна једначина постаје следећа једначина (4.16). Ова једначина представља Boltzmann-ову једначину која замењује Navier–Stokes-ове једначине. Дискретизацијом ова једначина добија облик (4.17).

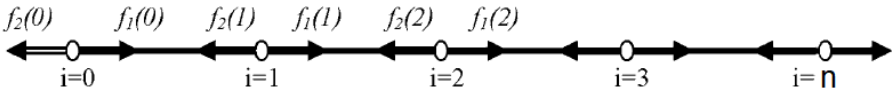
$$\frac{\partial f_i}{\partial t} + e_i\nabla f = \frac{1}{\tau}\left(f_i^{eq} - f_i\right) \tag{4.16}$$

$$f_i(r + e_i\Delta t, t + \Delta t) = f_i(r,t) + \frac{1}{\tau}\left[f_i^{eq}(r,t) - f_i(r,t)\right] \tag{4.17}$$

Примењена за 1-D проблем Boltzmann-ова једначина (4.17) добија нови облик који је дат једначином (4.18). На слици 23 приказано је како се шири функција дистрибуције.

$$f_i(x + e_i\Delta t, t + \Delta t) = f_i(x,t) + \frac{1}{\tau}\left[f_i^{eq}(x,t) - f_i(x,t)\right] \tag{4.18}$$

Вектор r се пројектује на три осе Декартовог координатног система и може бити представљен као $r = xi + yj + zk$, где су i, j и k јединични вектори у правцима x, y и z [21].



Слика 23: *Шематски приказ 1-D распореда решетке* [21]

Брзина ширења e_i је различита за различите димензије проблема и распореде решетке. За D1Q2 проблем брзина ширења је дата једначинама (4.19),при чему се вредност C добија према једначини (4.20).

$$e_1 = C \qquad e_2 = -C \tag{4.19}$$

$$C = \frac{\Delta x}{\Delta t} = \frac{\Delta y}{\Delta t} = \frac{\Delta z}{\Delta t} \tag{4.20}$$

У LB методи, домен решења се дели на решетке на чијим се чворовима налазе функције расподеле. Како се ове честице приликом свог кретања сударају са осталим честицама, долази до померања честица дуж одређених праваца до суседних чворова. Број праваца којима ће да се крећу честице, као и веза између честица, зависи од распореда решетке. Различити распореди решетке објашњени су у претходном поглављу.

У овом раду детаљније је обрађена примена Boltzmann-ове једначине за 1-D проблем, односно примена Boltzmann-ове једначине на D1Q2 решетки. За овај проблем једначина за пренос топлоте је изражена помоћу парцијалних извода и дата изразом (4.21).

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{4.21}$$

Температура поља се може израчунати тако што се сумирају све функције дистрибуције, у општем случају применом формулое (4.22). За 1-D проблем и распоред решетке Q2 ова једначина добија облик као што је приказно изразом (4.23).

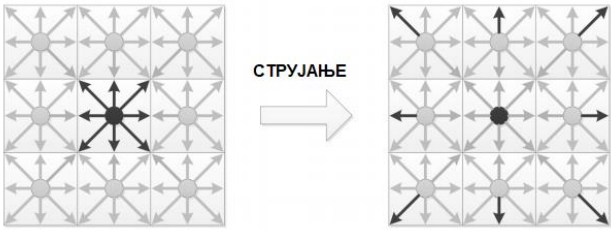
$$T(r,t) = \sum_{i=1}^{n} f(r,t) = f_1 + f_2 + \cdots + f_n \tag{4.22}$$

$$T(x,t) = \sum_{i=1}^{2} f(x,t) = f_1 + f_2 \tag{4.23}$$

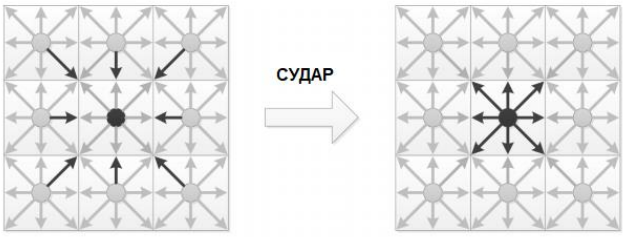
Сваки временски корак је подељен на два процеса, фазу струјања (*streaming*) и фазу сударања (*collision*). Струјање подразумева кретање честица од једног чвора ка другом, суседном чвору, и том приликом долази до размене честица. Након ове фазе следи фаза сударања, где долази до размене количине кретања између суседних четица приликом њиховог сударања. У сваком временском кораку примењују се правила која важе за ова два процеса. Комбиновањем ових фаза врши се напредовање процеса кроз простор и време и тим путем долази до ажурирања једначине којом се врши прорачун нестационарног преноса топлоте. У Boltzmann-овој једначини, на слици 24, приказано је који се део једначине односи на који процес (фаза судара и фаза струјања), и ови процеси су засебно илистровани на сликама 25 и 26.

$$\underbrace{f_i(\vec{x} + \vec{e}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t)}_{\textbf{\textit{Faza strujanja}}} \underbrace{- \frac{1}{\tau}\left[f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t)\right]}_{\textbf{\textit{Faza sudara}}}$$

Слика 24: *Boltzmann-ова једначина и две фазе* [22]



Слика 25: *Фаза струјања* [27]



Слика 26: *Фаза судара* [27]

Редослед остваривања ових процеса није битан. Односно, да ли се прво дешава фаза струјања или фаза судара, није од пресудног значаја за прорачун.

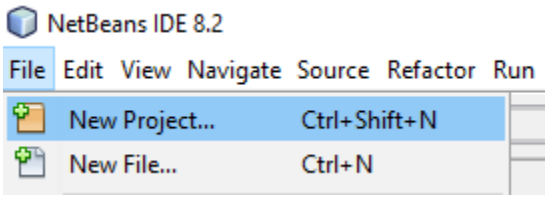
5. Креирање програма

У овом поглављу је детаљно приказан поступак креирања програма за стационарни и нестационарни пренос топлоте. Оба програма су писана у програмском језику JAVA у радном окружењу NetBeans IDE 8.2.

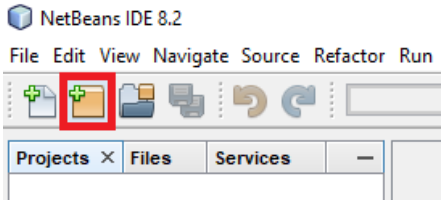
5.1. Креирање JAVA апликације за стационарни прорачун топлоте

JAVA пројекат се у NetBeans радном окружењу може креирати на три начина, и то:

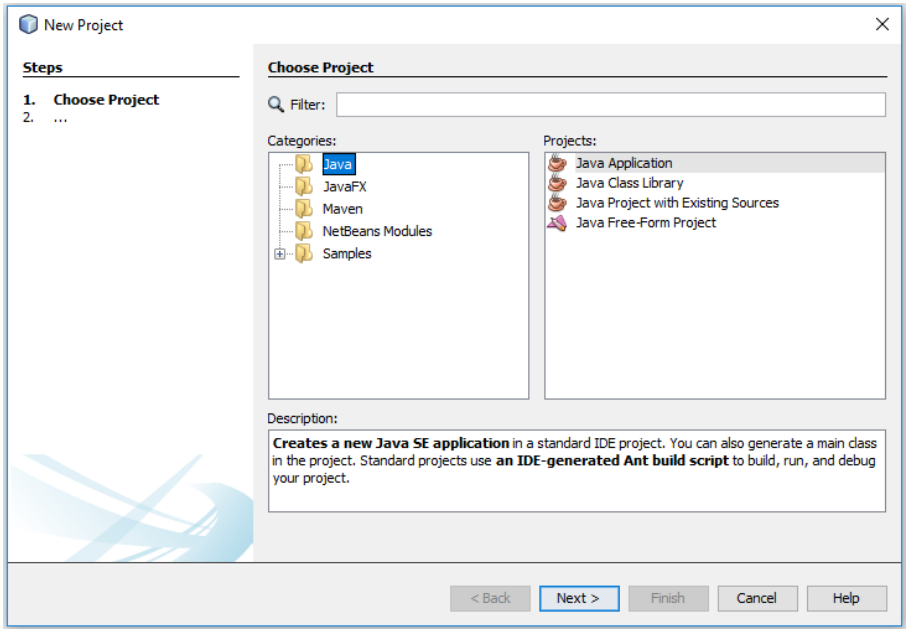
1. Кликом на **File -> New Project** (слике 27)
2. Пречицом **Ctrl+Shift+N**
3. Кликом на иконицу (слика 28)



Слика 27: *Креирање пројекта*



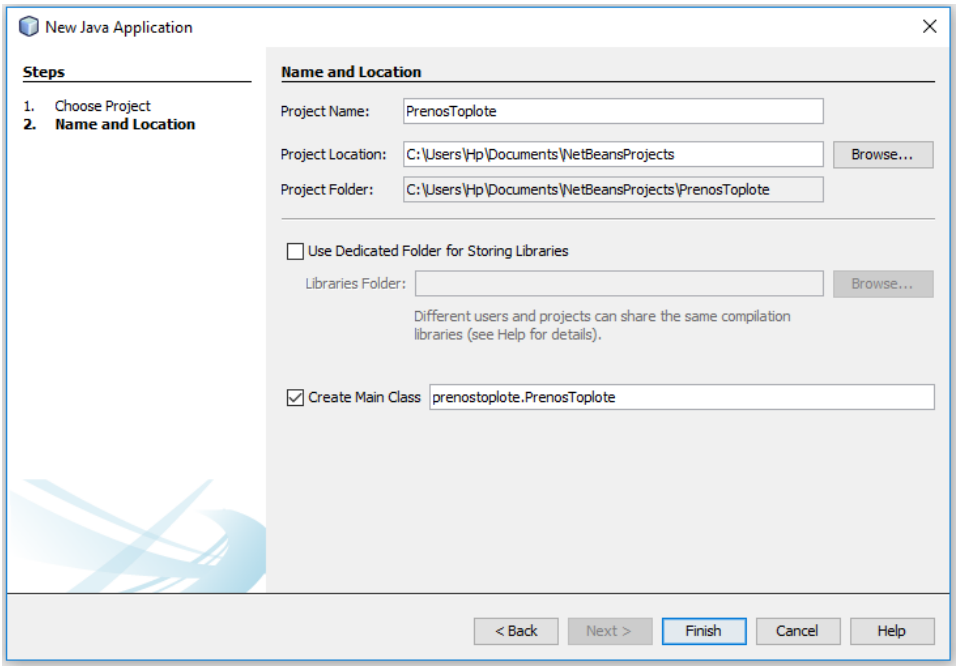
Слика 28: *Иконица за креирање пројект*



На слици 29 у пољу **Categories:** бирамо **Java**, а у пољу **Projects:** бирамо **Java Application**, а затим кликнемо на дугме **Next**.

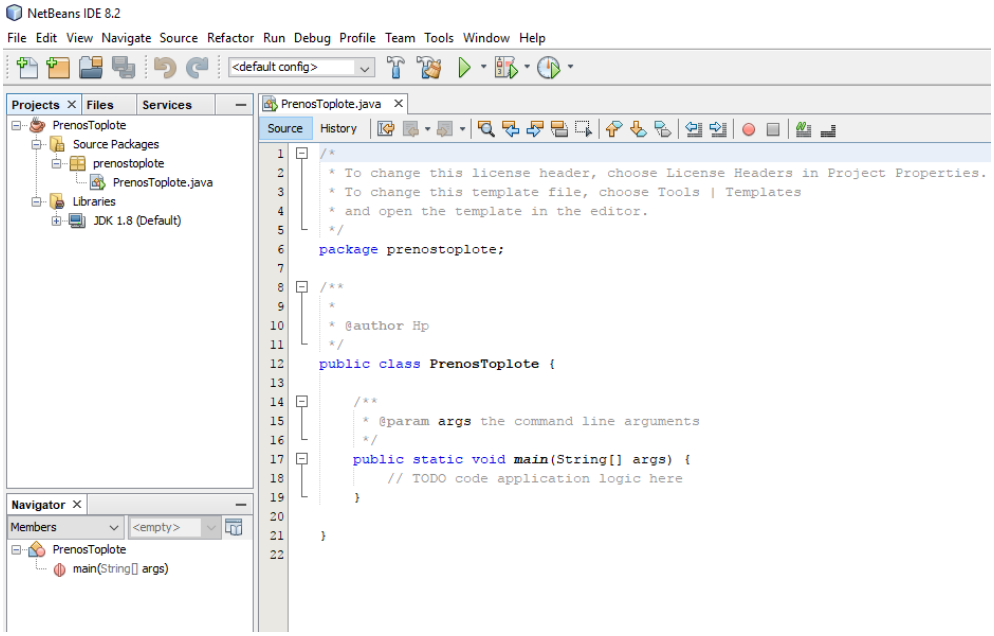
Слика 29: *Креирање Java апликације*

На слици 30 у пољу **Project Name:** уноси се име пројекта у овом случају то је **PrenosToplote**, а у пољу **Project Location:** уноси се место на коме се чува апликација. Такође потребно је и креирати основну класу без које програм не би радио, а то је *класа Main*. Ова класа се креира тако што се чекира последњи ред који каже: **Create Main Class**. Када су унесени сви неопходни параметри кликом на дугме **Finish** врши се креирање апликације.



Слика 30: *Одабир имена и места чувања апликације*

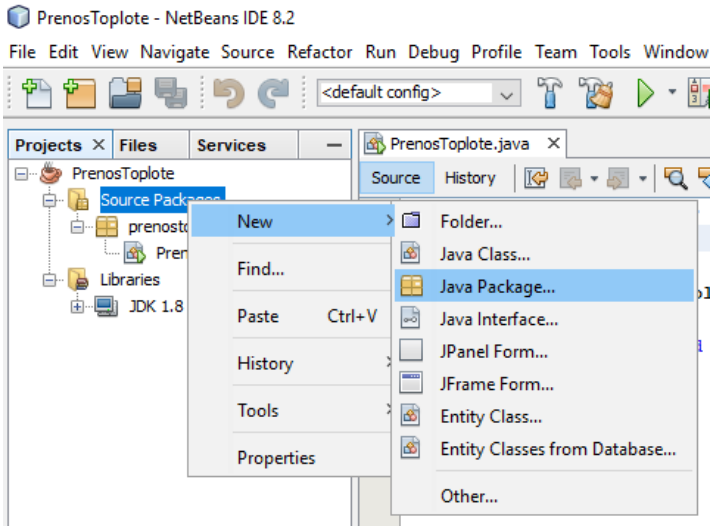
Када је апликација креирана отвориће се прозор као на слици 31. Линије кода су обележене бројевима од 1 до 22 и ту је приказан аутоматски генерисан код са пакетом, јавном класом **PrenosToplote** коју смо ми креирали и основном *класом main* која је јавна статичка класа која као резултат не враћа ништа. Такође креирани су у коду и коментари који се могу обрисати. Са леве стране на слици 31 може се видети хијерархија пројекта, као и навигатор у коме се појављују све коришћене променљиве, класе, методе и конструктори.



Слика 31: *Аутоматски генерисан код приликом креирања пројекта*

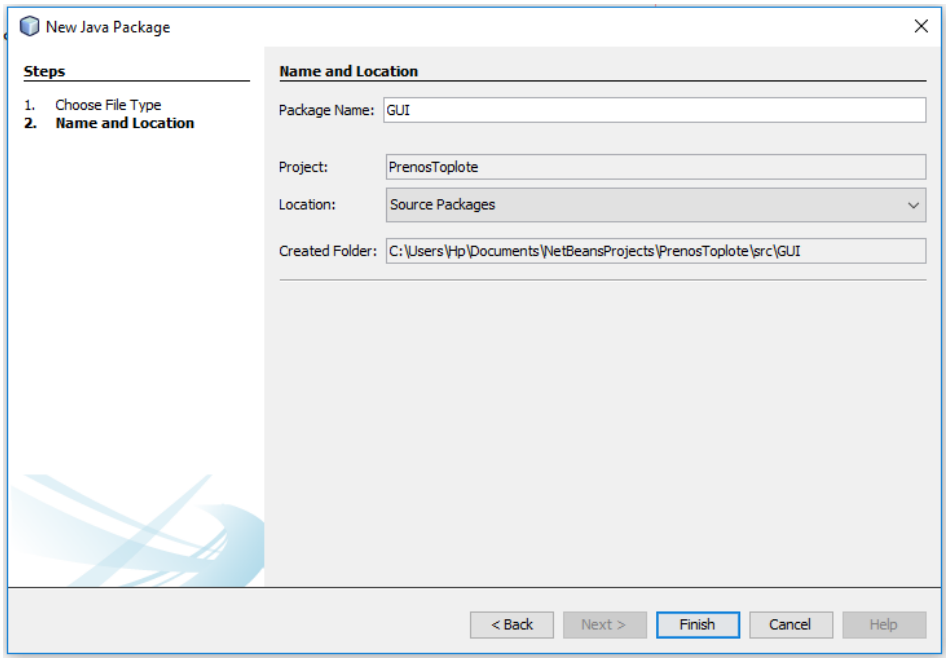
5.1.1. Креирање Java Windows display-а

Најпре креирамо још један пакет тако што кликнемо **десним кликом миша на Source Packages -> New -> Java Package…**, као на слици 32.



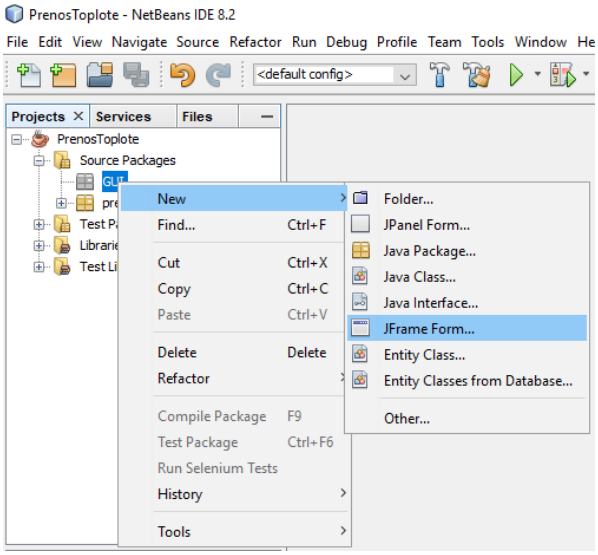
Слика 32: *Креирање новог пакета*

Све што је сада потребно је унети назив новог пакета у пољу **Package Name:** у нашем случају то је име **GUI** (Слика 33). Све остало је већ аутоматски подешено и сада га нећемо мењати. Кликом на дугме **Finish** завршавамо наше креирање.



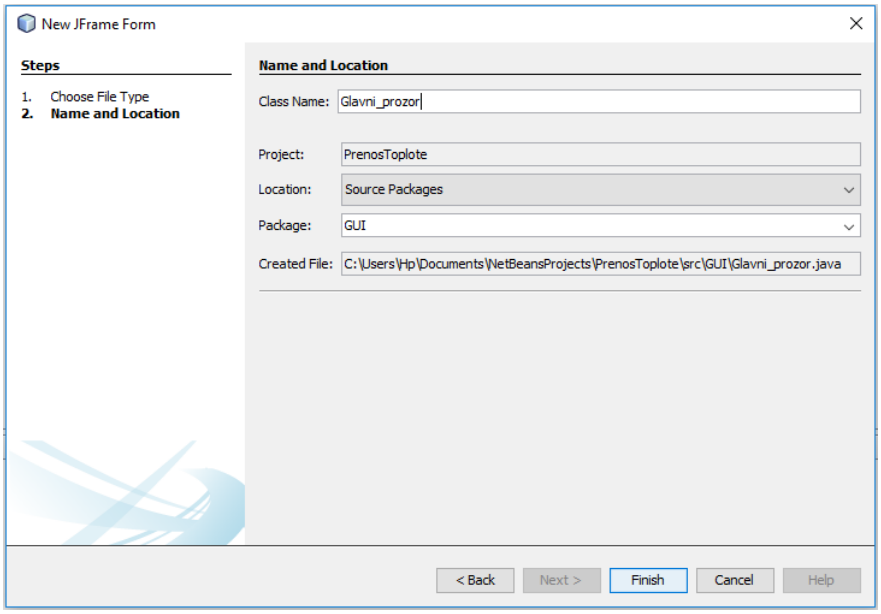
Слика 33: *Додела имена и локације новом пакету*

Даље је потребно креирати форму у којој ће се креирати Windows display. Креирање форме је приказано на слици 34.



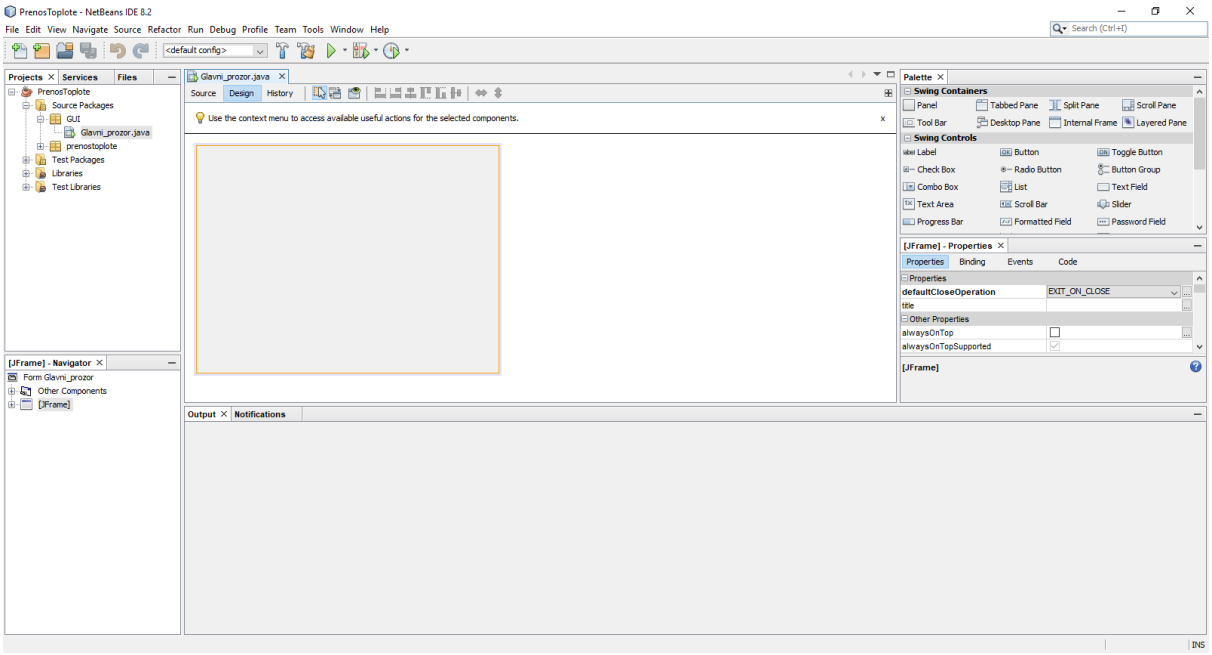
Слика 34: *Креирање форме*

Остало је још унети назив класе форме **Class Name:** у овом случају то је **Glavni_prozor**, све остало оставићемо како је аутоматски генерисано, а затим ћемо кликнути на дугме **Finish** (слика 35).



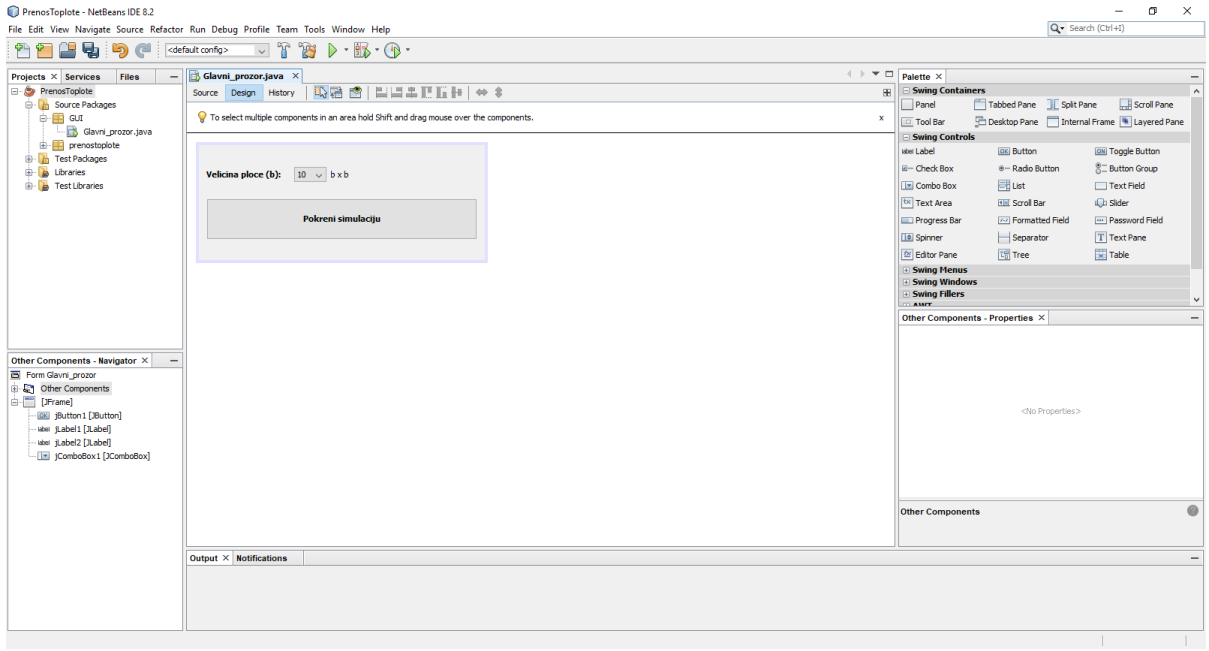
Слика 35: *Додељивање имена и локације форме*

По завршетку креирања форме радно окружење изгледа као на слици 36. Лева страна основног радног окружења NetBeans-a је задржала свој изглед и функцију (архитектура пројекта - картица **Projects**), уз додатак картице форме – **[JFrame] Navigator**, где ће бити смештани сви алати који се користе за Windows display. Са десне стране може се видети палета са свим поменутим алатима – картица **Palette** као и својства сваког алата који поставимо на форму. Приступање својствима се врши тако што се на прозору кликне алат чије својство желимо да мењамо, а затим са десне стране у картици **Properties** мењамо својства.



Слика 36: *Изглед радног окружења*

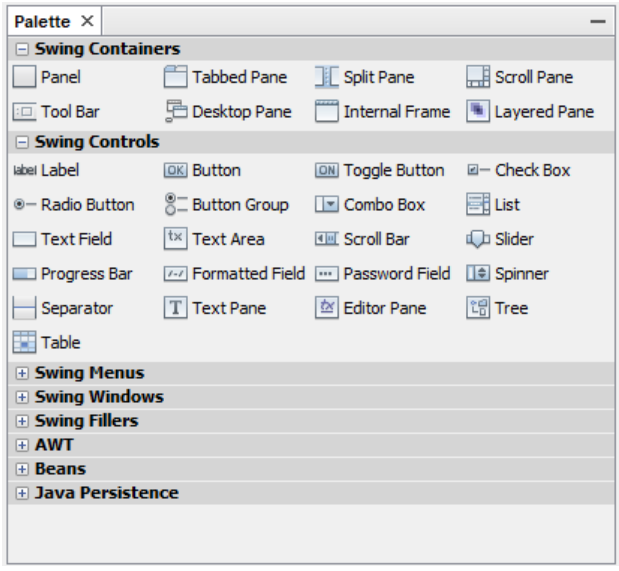
Да би радно окружење, а и сама форма изгледали као на слици 37 мора се извршити уређење форме. Најпре се врши додавање и позиционирање компоненти, а затим и подешавање својстава.



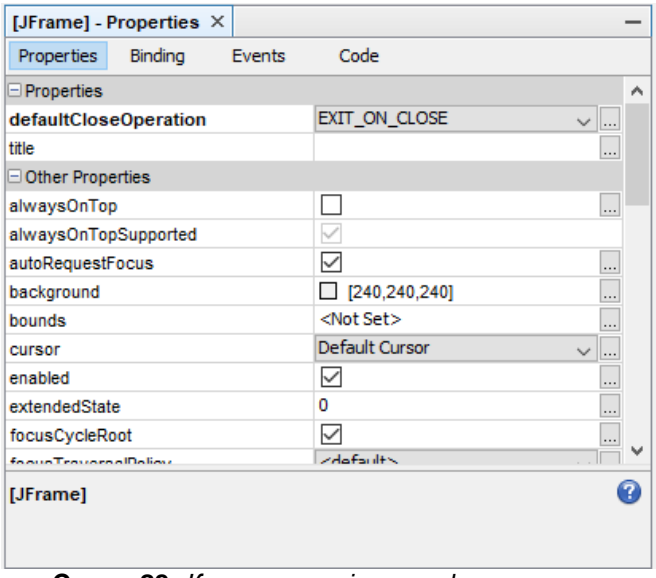
Слика 37: *Изглед радног окружења са додатим компонентама*

5.1.2. Уређивање форме

Додавање и позиционирање компоненти врши се превлачењем из картице **Palette** (слика 38) на форму на средини екрана. Подешавање својства компоненте се врши тако што се кликне на компоненту чија својства желимо да мењамо, а затим се у картици **Properties** врши мењање одговарајућих параметара. Картице својстава (Properties) се разликују у зависности од компоненте која се селектује. На слици 39 је приказана картица својстава форме.



Слика 38: *Картица компоненти*



Слика 39: *Картица својстава форме*

Подешавања својстава осталих компоненти са слике 37 из претходног поглавља могу се видети на сликама од 40 до 43. Све промене заокружене су црвеним правоугаоницима. Промене се врше двоструким кликом миша у десној колони жељеног параметара. Том приликом може доћи и до отварања нових прозора за подешавање.

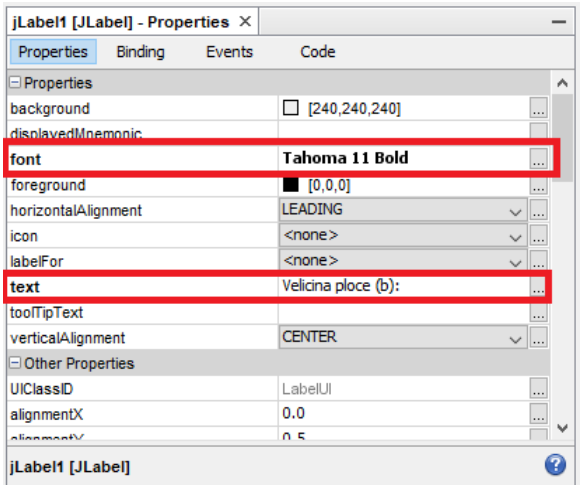
Параметри који су мењани у овом пројекту:

1. За лабеле су: **text** и **font**. Постоје две лабеле, и код једне и код друге font је Tahoma 11, разлика је само што је jLabel1 Bold, док је jLabel2 Plain, тј. код jLabel1 смо мењали оба параметра и text и font, док смо код jLabel2 мењали само параметар text.

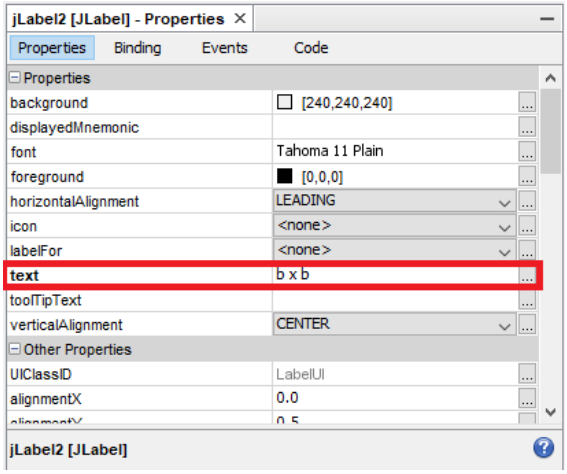
2. За дугме су: **text** и **font**, као и код jLabel1.

3. За падајући мени је: **model**. Ту се наводе ставке по редоследу по коме желимо да се оне налазе у падајућем менију. Свака ставка је од следеће одвојена запетама. Након последње ставке не пише се ништа.

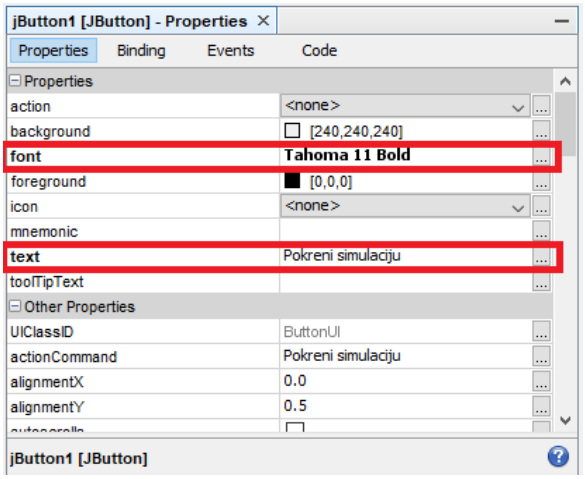
4. За форму: Што се форме тиче она је једина задржала своја својства без промена.



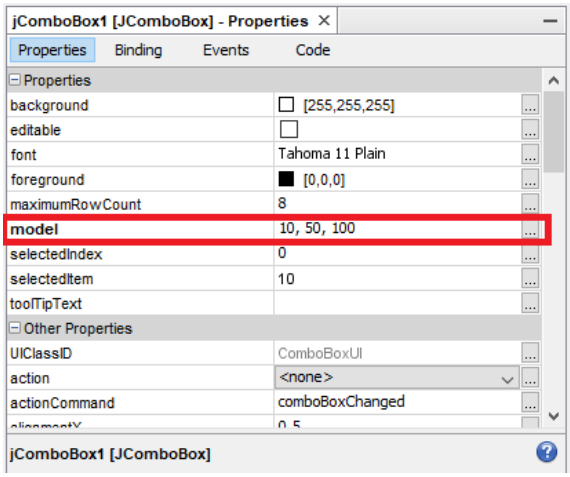
Слика 40: *Својства jLabel1*



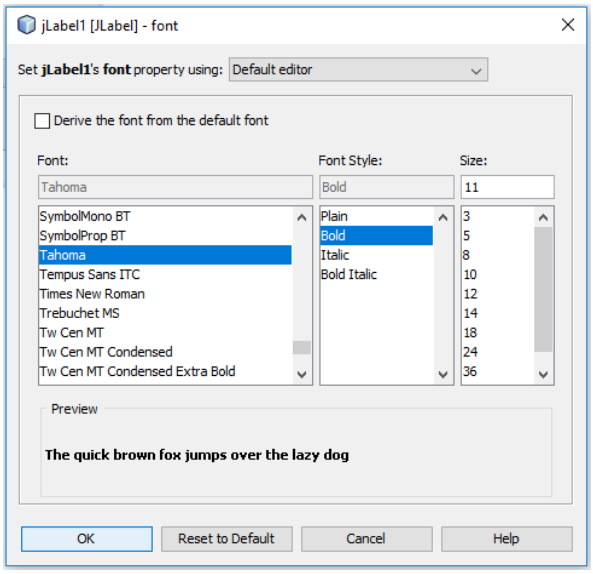
Слика 41: *Својства jLabel2*



Слика 42: *Својства jButton1*



Слика 43: *Својства jComboBox1*



Пример новог прозора са параметрима за подешавање који се отвара приликом двоструког клика мишем на параматар **font** дат је на слици 44.

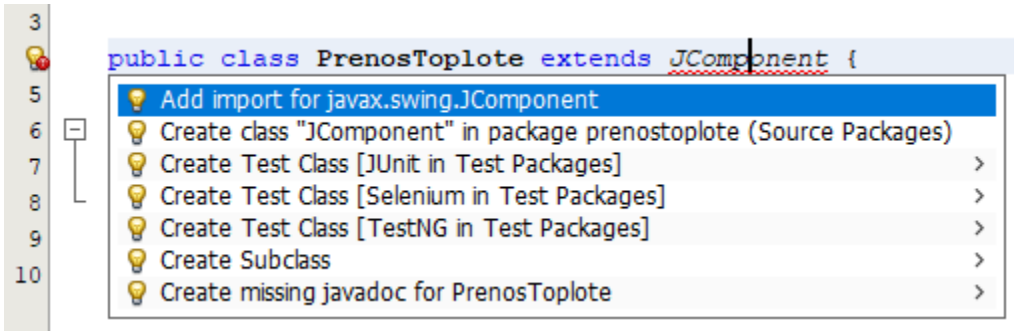
Том приликом могуће је подесити Font, Font Style и Size, односно стил и величину текста.

Слика 44 (лево): *Подешавање font-a*

5.1.3. Програмирање форме и главног програма

Да би дугме и падајући мени радили неопходно је извршити њихово програмирање. Програмирање се врши двоструким кликом миша на компоненту коју желимо да програмирамо, након чега се врши аутоматско пребацивање у део радног окружења у коме се пише код. Пре него што се почне са програмирањем форме, најпре се мора извршити унос података који ће се користити у програму и то у главној класи *PrenosToplote.java*.

Како би се креирали графички прикази неопходно је класи додати *extends JComponent*. Том приликом јавља се грешка и понуђена су решења као на слици 45, одлучујемо се за прво наведено решење.



Слика 45: *Укључивање extends JComponent у програм*

Затим се додају све променљиве које ће се користити у програму (слика 46). Прва наведена променљива *public static int broj_kvadratica=0;* је променљива којој је неопходно да се приступи и из друге класе, тј. из класе у којој вршимо програмирање дугмета и падајућег менија, зато је неопходно навести да је та променљива јавна, иницијално јој је додељена вредност 0 на почетку.

int MAX_PROLAZ = 1000; је променљива која каже да се врши 1000 итерација, односно пролаза, приликом израчунавања температуре у сваком квадратицу.

int i, j, k; су променљиве које служе за пролаз по матрици.

```
1   package prenostoplote;
2
3   import javax.swing.JComponent;
4
5
6   public class PrenosToplote extends JComponent {
7       public static int broj_kvadratica = 0;
8       int MAX_PROLAZ = 1000;
9       int i, j, k;
10      double T, GORE, DOLE, LEVO, DESNO;
11      double GORE_LEVO, GORE_DESNO, DOLE_LEVO, DOLE_DESNO;
12      double[][] oldT = new double[100][100];
13      double[][] newT = new double[100][100];
14
15      public static void main(String[] args) {
16
17      }
18
19  }
```

Слика 46: *Унос променљивих у коду*

double T, GORE, DOLE, LEVO, DESNO; double GORE_LEVO, GORE_DESNO, DOLE_LEVO, DOLE_DESNO; јесу променљиве које служе за приступање вредностима суседних квадратица (поља) како би се израчунала температура *T* у једном пољу.

double[][] oldT = new double[100][100]; double[][] newT = new double[100][100]; представљају матрице старе и нове температуре по којима се врши промена вредности температуре, тј. пренос топлоте.

Како би се приступило класи у којој се програмира форма, неопходно је навести get-ер и set-ер, као што је приказано на слици 47.

get-ер треба да враћа вредност која је изабрана за *broj_kvadratica* тачније да обезбеди да се овој променљивој може приступити ван ове класе, док set-ер треба да додели вредност, која је задата у другој класи, променљивој која се налази у овој класи.

```
6   public class PrenosToplote extends JComponent {
7       public static int broj_kvadratica = 0;
8       int MAX_PROLAZ = 1000;
9       int i, j, k;
10      double T, GORE, DOLE, LEVO, DESNO;
11      double GORE_LEVO, GORE_DESNO, DOLE_LEVO, DOLE_DESNO;
12      double[][] oldT = new double[100][100];
13      double[][] newT = new double[100][100];
14
15      //Metodi za pristup GUI
16      public int getBroj_kvadratica() {
17          return broj_kvadratica;
18      }
19      public void setBroj_kvadratica(int broj_kvadratica) {
20          this.broj_kvadratica = broj_kvadratica;
21      }
```

Слика 47: *Методи за приступ GUI-у*

Да би обе класе биле повезане (*PrenosToplote.java* и *Glavni_prozor.java*) и у другој класи је неопходно извршити увођење одговарајућих променљивих, односно дозвољава се приступ свим јавним променљивама. Увоз се врши кодом приказаним на слици 48.

```
1   package GUI;
2
3   import prenostoplote.PrenosToplote;
```

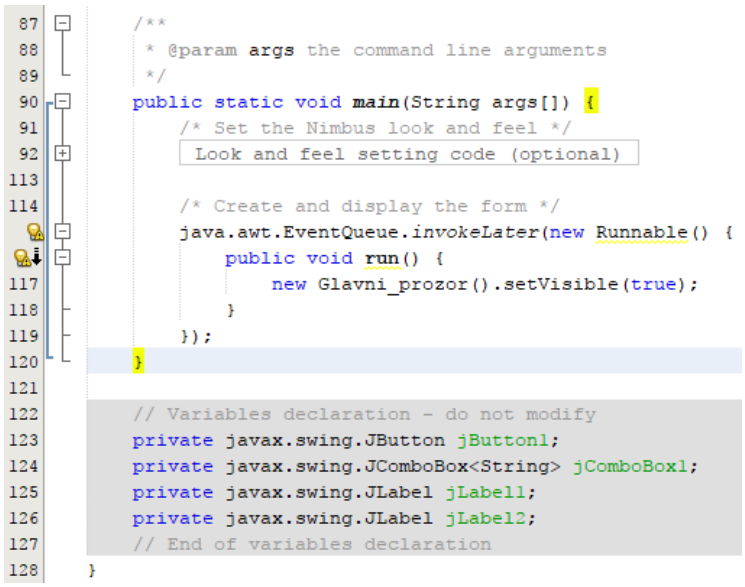
Слика 48: *Увоз класе*

```
1   package GUI;
2
3   public class Glavni_prozor extends javax.swing.JFrame {
4
5       public Glavni_prozor() {
6           initComponents();
7       }
8
9       /**
10      * This method is called from within the constructor to initialize the form.
11      * WARNING: Do NOT modify this code. The content of this method is always
12      * regenerated by the Form Editor.
13      */
14      @SuppressWarnings("unchecked")
15      Generated Code
72
73      private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
74          // TODO add your handling code here:
75      }
76
77      /**
78      * @param args the command line arguments
79      */
80      public static void main(String args[]) {
81          /* Set the Nimbus look and feel */
82          Look and feel setting code (optional)
103
104         /* Create and display the form */
105         java.awt.EventQueue.invokeLater(new Runnable() {
106             public void run() {
107                 new Glavni_prozor().setVisible(true);
108             }
109         });
110     }
```

Слика 49: *Аутоматски генерисан код*

На слици 49 приказан је изглед кода *Glavni_prozor.java* на самом почетку када смо извршили постављање и распоређивање свих компоненти на форму.

Креирање и приказивање форме која ће бити видљива кориснику врши се следећим кодом изнад дефинисања променљивих (слика 50).



```
87      /**
88       * @param args the command line arguments
89       */
90      public static void main(String args[]) {
91          /* Set the Nimbus look and feel */
92          Look and feel setting code (optional)
113
114         /* Create and display the form */
115         java.awt.EventQueue.invokeLater(new Runnable() {
116             public void run() {
117                 new Glavni_prozor().setVisible(true);
118             }
119         });
120     }
121
122     // Variables declaration - do not modify
123     private javax.swing.JButton jButton1;
124     private javax.swing.JComboBox<String> jComboBox1;
125     private javax.swing.JLabel jLabel1;
126     private javax.swing.JLabel jLabel2;
127     // End of variables declaration
128 }
```

Слика 50: *Креирање и приказивање форме*

Што се програмирања *Glavni_prozor.java* тиче, остало је још само испрограмирати дешавање након што корисник кликне на дугме. Део кода за тај део може се видети на слици 51. Овај код се пише у делу који је аутоматски генерисан *private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { }*.

```
16      @SuppressWarnings("unchecked")
17      Generated Code
74
75      private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
76
77          int vrednost_mreze = Integer.parseInt(jComboBox1.getSelectedItem().toString());
78
79          prenostoplote.PrenosToplote p = new PrenosToplote();
            p.main(null);
81          p.setBroj_kvadratica(vrednost_mreze);
82      }
83
84      /**
85       * @param args the command line arguments
86       */
```

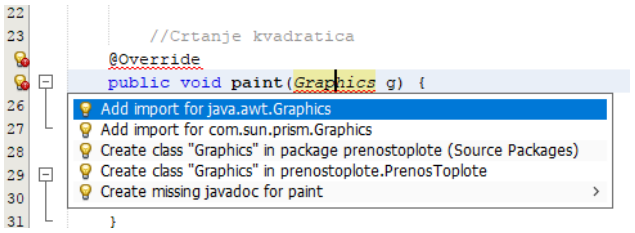
Слика 51: *Код за програмирање дугмета*

int vrednost_mreze = Integer.parseInt(jComboBox1.getSelectedItem().toString()); променљива *vrednost_mreze* је локална променљива која се користи у класи *Glavni_prozor.java* за дефинисање величине плоче. Код са друге стране знака једнакости говори о томе да се из падајућег менија врши селектовање вредности величине плоче, како су вредности у падајућем менију подаци типа *String*, потребно их је пребацити у целобројне променљиве. Управо то се врши кодом са друге стране знака једнакости. Вредност из падајућег менија се претвара у број који се затим додељује променљивој *vrednost_mreze*.

prenostoplote.PrenosToplote p = new PrenosToplote(); креира се конструктор преко кога ће се вршити комуникација са класом *PrenosToplote.java*.

p.setBroj_kvadratica(vrednost_mreze); приступа се методу у класи *PrenosToplote.java* и каже, пошто овај метод враћа променљиву *broj_kvadratica*, она добија вредност коју смо изабрали помоћу променљиве *vrednost_mreze*.

Сада када је програмирање форме завршено можемо наставити са програмирањем класе *PrenosToplote.java*. Додајемо нови метод за цртање квадратица и његову библиотеку као што је приказан на слици 52.



Слика 52: *Метод за цртање квадратица*

int sirina = 500/broj_kvadratica; локална променљива која дефинише колика је ширина квадратица који ће се приказати на екрану.

Што се самог прорачуна температуре тиче, најпре се постављају граничне вредности температуре писањем кода који је приказан на слици 53. Прва *for* петља креће се дуж првог реда матрице и додељује квадратићима вредност температуре 100 целзијуса. Друга *for* петља креће се дуж прве колоне матрице и додељује квадратићима вредност температуре 100 целзијуса. Трећа *for* петља креће се дуж задње колоне матрице и додељује квадратићима вредност температуре 100 целзијуса. Доња гранична вредност или четврта *for* петља која би се кретала по последњем реду није наведена међу граничним вредностима у коду јер се вредност нула аутоматски додељује.

```
25      @Override
26      public void paint(Graphics g) {
27          int sirina = 500/broj_kvadratica;
28
29          //Proracun
30          //Inicijalizacija temperature u svakom polju
31          //Gornja granicna temperatura
32          for(j=0; j<broj_kvadratica; j++){
33              oldT[0][j]=100;
34          }
35          //Leva granicna temperatura
36          for(i=0; i<broj_kvadratica; i++){
37              oldT[i][0]=100;
38          }
39          //Desna granicna temperatura
40          for(i=0; i<broj_kvadratica; i++){
41              oldT[i][broj_kvadratica-1]=100;
42          }
43      }
```

Слика 53: *Граничне вредности температуре*

На слици 54 дат је наставак прорачуна температуре по броју пролаза и чување резултата [28]. Прва *for* петља врши пролазак по итерацијама. Друге две *for* петље врше израчунавање температуре најпре у околним пољима, а затим и самог поља које је на реду. Такође имамо и чување нове вредности и испис сваке вредности по пролазу.

Вредности последње итерације се чувају у текстуалном документу у форми матрице где су вредности међу собом одвојене размаком [29].

На самом крају имамо и испис која је итерација по реду завршена.

```
45                  //Izracunavanje temperature po broju prolaza
46              for(k=0; k<MAX_PROLAZ; k++){
47                  //Proracun temperature
48                  for(i=1; i<broj_kvadratica-1; i++){
49                      for(j=1; j<broj_kvadratica-1; j++){
50                          GORE = oldT[i-1][j];
51                          DOLE = oldT[i+1][j];
52                          LEVO = oldT[i][j-1];
53                          DESNO = oldT[i][j+1];
54                          GORE_LEVO = oldT[i-1][j-1];
55                          GORE_DESNO = oldT[i-1][j+1];
56                          DOLE_LEVO = oldT[i+1][j-1];
57                          DOLE_DESNO = oldT[i+1][j+1];
58                          T=(4*(GORE+DOLE+LEVO+DESNO)+GORE_LEVO+GORE_DESNO+DOLE_LEVO+DOLE_DESNO)/20;
59                          newT[i][j]=T;
60                          //Cuvanje rezultata po broju prolaza
61                          oldT[i][j]=newT[i][j];
62                          //Stampanje rezultata po broju prolaza
63                          System.out.println(oldT[i][j]);
64                      }
65                  }
66                  //Cuvanje rezultata u tekstualnom dokumentu
67                  String Temperature = "Temperature.txt";
68                  try {
                        PrintWriter Rezultat = new PrintWriter(Temperature);
70                      for(i=0; i<broj_kvadratica; i++){
71                          for(j=0; j<broj_kvadratica; j++){
72                                  Rezultat.print(oldT[i][j]);
73                                  Rezultat.print(" ");
74                                  Rezultat.print(" ");
75                          }
76                          Rezultat.println("");
77                          Rezultat.println("");
78                      }
79                      Rezultat.close();
80                  }
81                  catch (FileNotFoundException ex) {
82                  }
83                  System.out.println("");
84                  System.out.println("Prolaz k = " + k);
85                  System.out.println("");
86              }
87      }
```

Слика 54: *Прорачун температуре и чување резултата*

Што се тиче бојења квадратица (поља), потребно је одредити опсеге и боје за сваки опсег. Део кода који приказује бојење дат је на слици 55.

Бојење се врши тако што се на основу вредности последње итерације одређује у ком опсегу се налази вредност и том квадратицу у матрици се додељује одговарајућа боја и тако редом [30]. Када су додељене боје свим квадратицима врши се исцртавање ивица односно мреже која ће раздвајати визуално квадратиће (поља) [31].

Сваки пут је потребно set-овати боју и обојити квадратић [32]. По квадратићима се креће по хоризонтали односно боје се редом ређају с' лева на десно. Када се заврши један ред прелази се на следећи и тако редом. Исто важи и приликом свих осталих прорачуна у програму.

```
88          //Bojenje kvadratica
89          for(i=0; i<broj_kvadratica; i++){
90              for(j=0; j<broj_kvadratica; j++){
91                  if(oldT[i][j]<=10){
92                      g.setColor(Color.BLUE);
93                      g.fillRect(j*sirina, i*sirina, sirina*sirina, sirina);
94                  }
95                  else if(oldT[i][j]>10 && oldT[i][j]<=20){
96                      g.setColor(Color.CYAN);
97                      g.fillRect(j*sirina, i*sirina, sirina, sirina);
98                  }
99                  else if(oldT[i][j]>20 && oldT[i][j]<=30){
100                     g.setColor(Color.GRAY);
101                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
102                 }
103                 else if(oldT[i][j]>30 && oldT[i][j]<=40){
104                     g.setColor(Color.LIGHT_GRAY);
105                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
106                 }
107                 else if(oldT[i][j]>40 && oldT[i][j]<=50){
108                     g.setColor(Color.GREEN);
109                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
110                 }
111                 else if(oldT[i][j]>50 && oldT[i][j]<=60){
112                     g.setColor(Color.YELLOW);
113                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
114                 }
115                 else if(oldT[i][j]>60 && oldT[i][j]<=70){
116                     g.setColor(Color.ORANGE);
117                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
118                 }
119                 else if(oldT[i][j]>70 && oldT[i][j]<=80){
120                     g.setColor(Color.PINK);
121                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
122                 }
123                 else if(oldT[i][j]>80 && oldT[i][j]<=90){
124                     g.setColor(Color.MAGENTA);
125                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
126                 }
127                 else if(oldT[i][j]>90 && oldT[i][j]<=100){
128                     g.setColor(Color.RED);
129                     g.fillRect(j*sirina, i*sirina, sirina, sirina);
130                 }
131                 g.setColor(Color.BLACK);
132                 g.drawRect(j*sirina, i*sirina, sirina, sirina);
133             }
134         }
```

Слика 55: *Бојење квадратица*

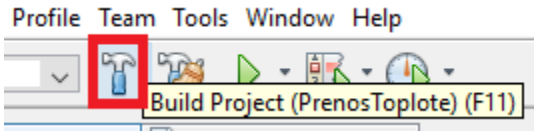
Да би све ово било приказано потребно је у главној *main* класи написати следећи код (слика 56) [33].

```
138     public static void main(String[] args) {
139         //Kreiranje prozora i ploce
140         JFrame window = new JFrame();
141         window.setTitle("Prenos toplote");
142         window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
143         window.setBounds(30, 30, 516, 539);
144         window.getContentPane().add(new PrenosToplote());
145         window.setVisible(true);
146     }
```

Слика 56: *Креирање прозора и плоче*

Овим кодом дефинише се наслов програма у насловној линији, дугме за прекид и излазак из програма, димензије прозора, као и могућност минимизирања програма. Комплетан код са коментарима дат је у прилогу 1 и 2.

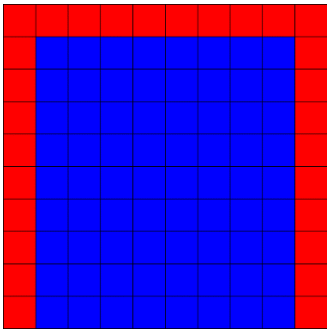
Како би се омогућио директан приступ апликацији, без потребе да се инсталира радно окружење NetBeans, по завршетку пројекта потребно је претиснути дугме *Build Project*, као на слици 57.



Слика 57: *Изградња независне апликације*

Овако изграђена апликација може се отворити директно из фолдера *dist* који се креира у матичном фолдеру нашег пројекта. Када отворимо фолдера *dist* преко Jave инсталиране на било ком уређају можемо извршити покретање апликације.

5.1.4. Пример примене CA у раду овог програма



У овом раду приказан је пример примене Cellular Automata за пренос топлоте на дводимензионалној квадратној плочи са густином мреже од 10x10, 50x50 и 100x100. На почетку процеса три граничне температуре су 100 °C (горња, лева и десна обојено црвеном бојом), док је једна 0 °C (доња). Почетна температура унутрашњих поља плоче је такође 0 °C. Поља која имају температуру 0 °C обојена су плавом бојом. Плоча са постављеним граничним вредностима температуре на почетку програма приказана је на слици 58.

Слика 58: *Граничне вредности температуре*

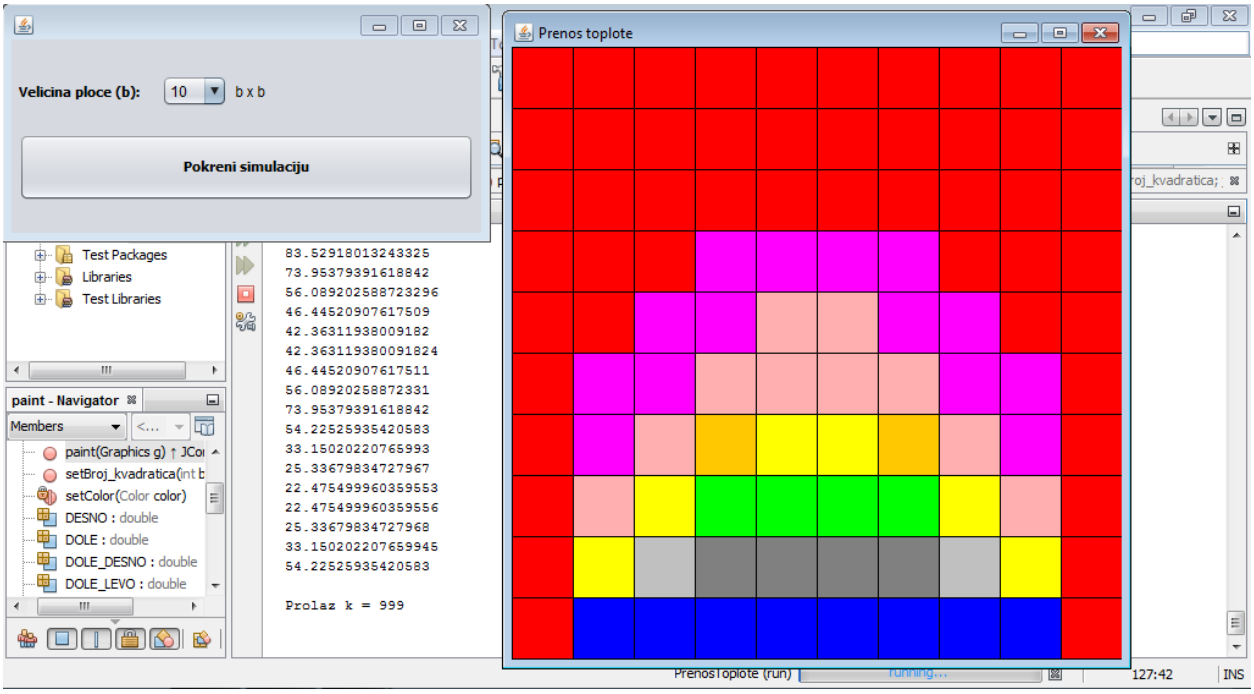
Када су постављене граничне вредности температура, унутрашња поља се рачунају према формули (5.1). За унутрашње поље T дата су околна поља, на слици 59, од којих зависи његова вредност.

$$T = \frac{4*(GORE+DOLE+LEVO+DESNO)+(GORE\_LEVO+GORE\_DESNO+DOLE\_LEVO+DOLE\_DESNO)}{20} \qquad (5.1)$$

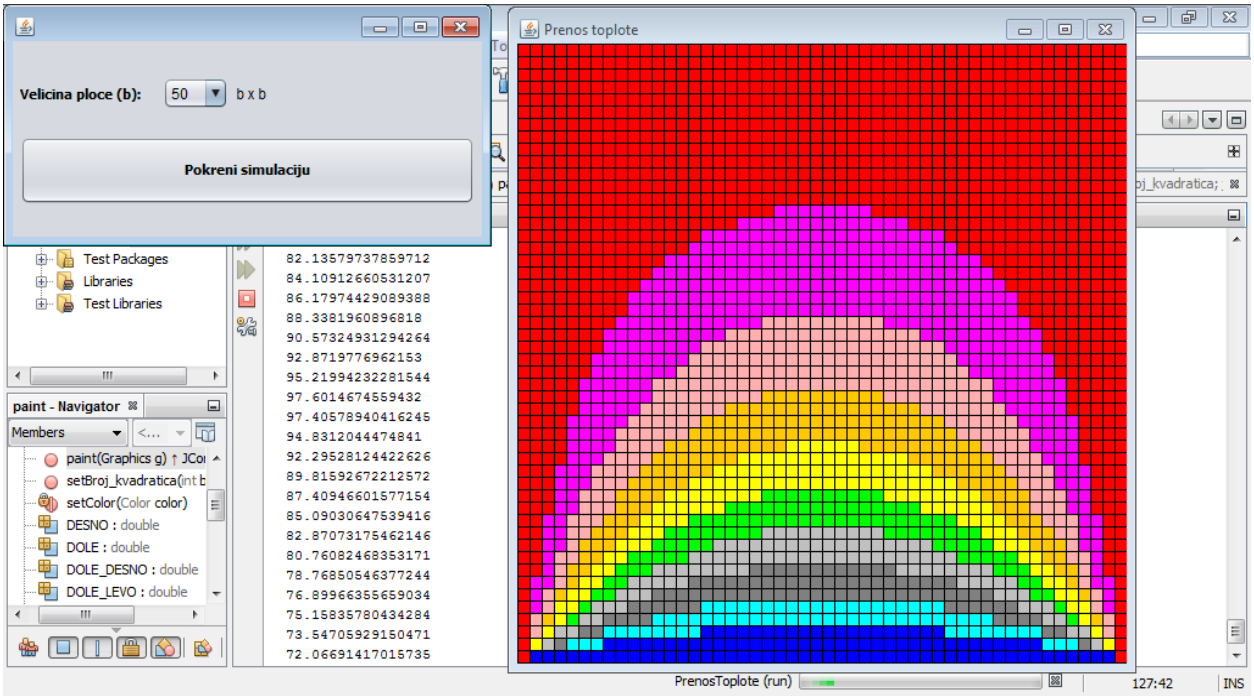
Број итерација, односно пролаза по квадратној дводимензионалној матрици је 1000. Са повећењем броја итерација достиже се нека коначна вредност преноса топлоте кроз плочу. Треба напоменути да се при мањем броју итерација добија временси краћи процес при коме долази до бржег преноса топлоте. По истој аналогији ако се користи мања густина мреже долази до бржег преноса топлоте. Што би значило да је за плочу са мрежом 10x10 потребно најмање времена да се постигне нека коначна температура, док је за плочу са мрежом 100x100 потребно више времена.

Слика 59: *Поља*

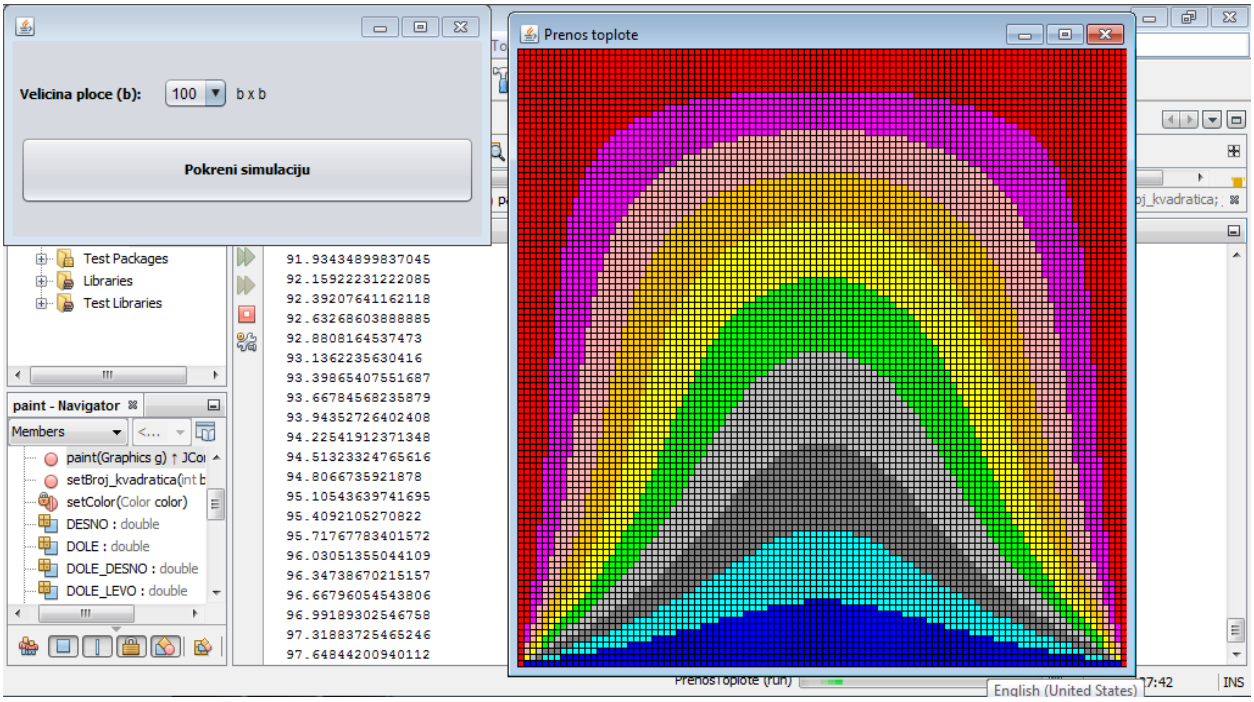
За прорачун температуре унутрашњих поља плоче коришћена је једначина (5.1). Прорачуном су добијени нумерички и графички резултати преноса топлоте на плочама са мрежама 10x10, 50x50 и 100x100 при максималном броју пролаза, односно за 1000 итерација. На сликама 60, 61 и 62 могу се видети графички резултати у радном окружењу.



Слика 60: *Пренос топлоте на плочи 10x10*



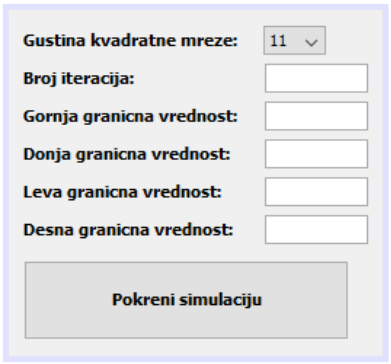
Слика 61: *Пренос топлоте на плочи 50x50*



Слика 62: *Пренос топлоте на плочи 100x100*

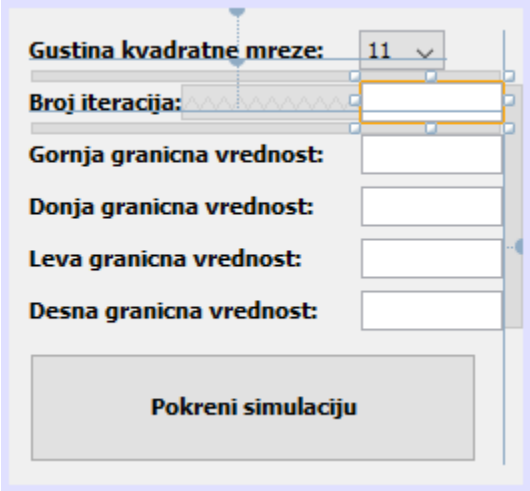
5.1.5. Друга верзија програма за стационарни прорачун топлоте

Поред горе наведене верзије, *Програм за пренос топлоте* је урађен у још једној верзији. У другој верзији овог програма омогућено је кориснику да сам уноси граничне вредности температуре, број итерација, а густина мреже је проширена за по једну врсту и једну колону, тј. врши се креирање поља: 11x11, 51x51 и 101x101 ради праћења промене температуре у централној тачки плоче. Изглед нове форме која је креирана за ову верзију програма приказана је на слици 63.

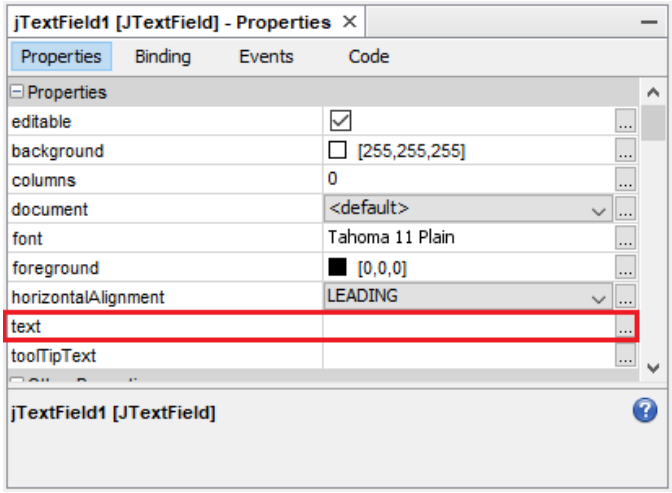


Слика 63: *Изглед форме друге верзије програма*

Осим измене интерфејса било је неопходно извршити и промене у самом коду програма. Потребно је само извршити кратку анализу за нове елементе постављене на форми, а то су празна поља у којима корисник уноси жељене вредности. На форму се превлачењем додаје пет поља (**JTextField** - слика 64) из **Palette** са алатима. У **Properties** се од подешавања врши само брисање текста у пољу **text** (означено црвено - слика 65). Комплетан код ове верзије програма дат је у прилозима Прилог 3 и Прилог 4 на крају овог рада.



Слика 64: *Селектовано празно поље*

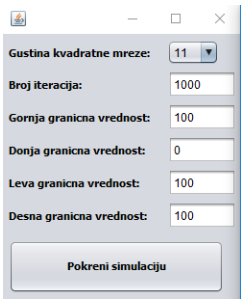


Слика 65: *Својства JTextField*

Овако модификован програм коришћен је за решавање четири примера анализе стационарног линеарног провођења топлоте, уз варијацију температура, граничних услова, густине мреже и максималног броја итерација. Сви ови резултати тестирани су и у MathLab-у и упоређени са добијеним резултатима у нашем програму. Програм у MathLab-у је урађен помоћу методе коначних елемената [34].

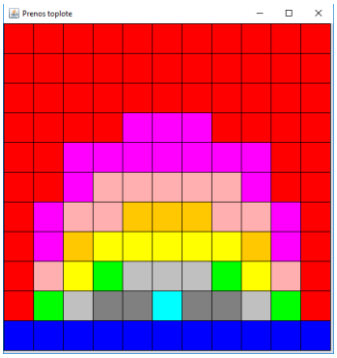
5.1.5.1. Први пример

Први пример представља пример приказан у претходном поглављу решен модификованим програмом.

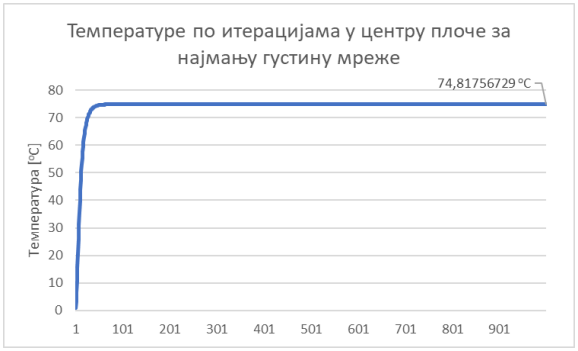


При анализи најмање густине мреже 11x11, при 1000 итерација (слика 66), можемо приметити да се већ при 291-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 74,81756728978453 °C. Распоред температура након последње итерације приказан је на слици 67, док је на слици 68 приказан дијаграм промене температуре у централној тачки по итерацијама.

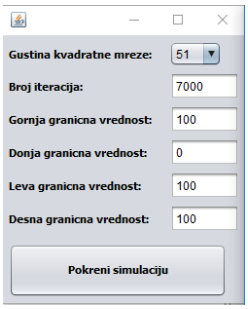
Слика 66 (лево): *Почетни услови мерења*



Слика 67: *Расподела температуре*

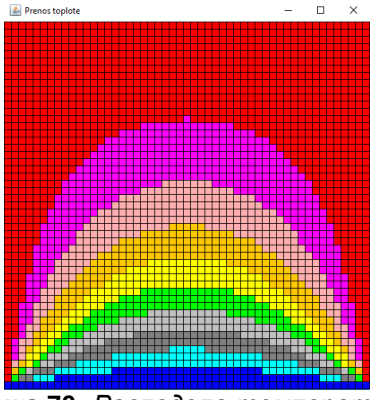


Слика 68: *Дијаграм промене температуре центра плоче*



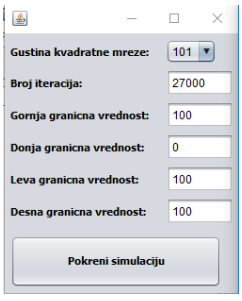
Приликом анализи средње густине мреже 51x51, при 7000 итерација (слика 69), можемо приметити да се већ при 6441-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 74,99270519758741 ºC. Распоред температура након последње итерације приказан је на слици 70, док је на слици 71 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 69 (лево): *Почетни услови мерења*



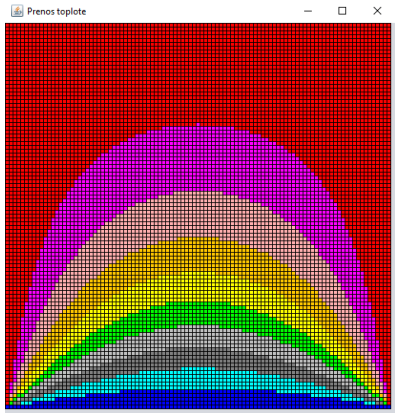
Слика 70: *Расподела температуре*

Слика 71: *Дијаграм промене температуре центра плоче*



При анализе највеће густине мреже 101x101, при 27000 итерација (слика 72), можемо приметити да се већ при 25883-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 74,99817630025272 ºC. Распоред температура након последње итерације приказан је на слици 73, док је на слици 74 приказан дијаграм промене температуре у централној тачки по итерацијама.

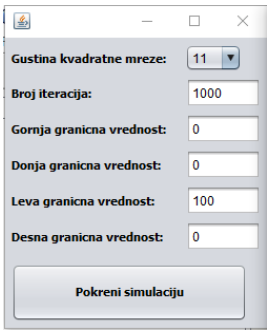
Слика 72 (лево): *Почетни услови мерења*



Слика 73: *Расподела температуре*

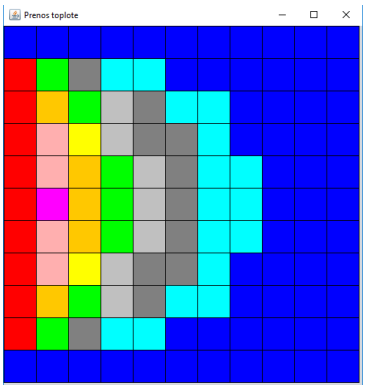
Слика 74: *Дијаграм промене температуре центра плоче*

5.1.5.2. Други пример:



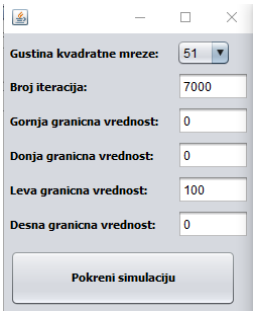
Приликом анализе најмање густине мреже 11x11, при 1000 итерација (слика 75), можемо приметити да се већ при 298-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 24,817567289784655 °C. Распоред температура након последње итерације приказан је на слици 76, док је на слици 77 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 75 (лево): *Почетни услови мерења*



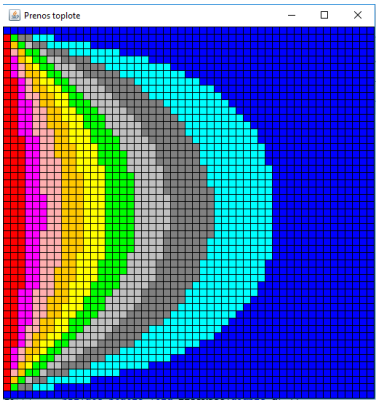
Слика 76: *Расподела температуре*

Слика 77: *Дијаграм промене температуре центра плоче*



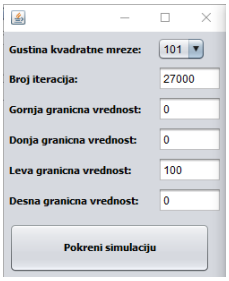
При анализи средње густине мреже 51x51, при 7000 итерација (слика 78), можемо приметити да се већ при 6789-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 24,992705197589313 °C. Распоред температура након последње итерације приказан је на слици 79, док је на слици 80 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 78 (лево): *Почетни услови мерења*



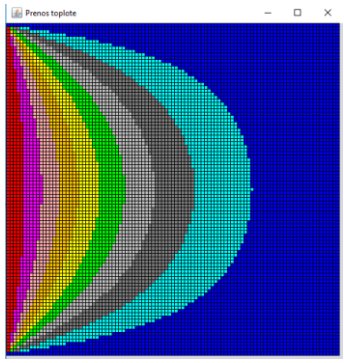
Слика 79: *Расподела температуре*

Слика 80: *Дијаграм промене температуре центра плоче*



Приликом анализе највеће густине мреже 101x101, при 27000 итерација (слика 81), можемо приметити да се већ при 26364-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 24,998176300259846 °C. Распоред температура након последње итерације приказан је на слици 82, док је на слици 83 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 81 (лево): *Почетни услови мерења*

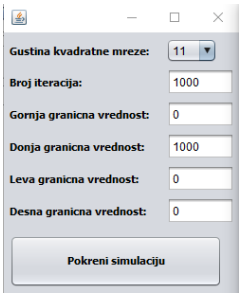


Слика 82: *Расподела температуре*

Слика 83: *Дијаграм промене температуре центра плоче*

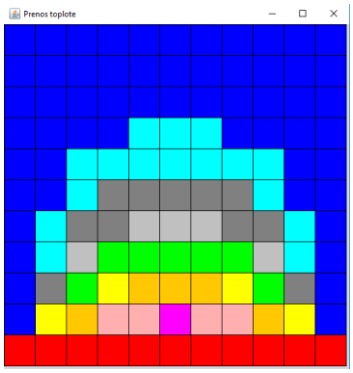
Трећа и четврта верзија овог програма представљају модификацију друге верзије, где су извршене промене граничних вредности температуре. У трећој верзији распон температура се креће од 0 до 1000 °C. Док се у четвртој верзији распон температура креће од 1000 до 1500 °C.

5.1.5.3. Трећи пример:



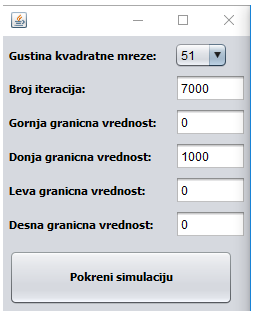
Прилоком анализе најмање густине мреже 11x11, при 1000 итерација (слика 84), можемо приметити да се већ при 193-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 251,8243271 °C. Распоред температура након последње итерације приказана је на слици 85, док је на слици 86 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 84 (лево): *Почетни услови мерења*



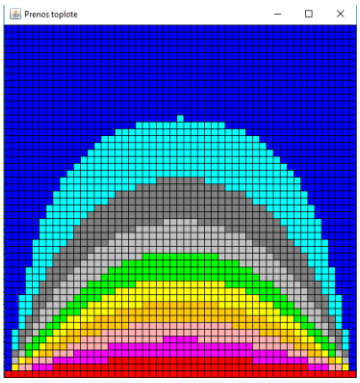
Слика 85: *Расподела температуре*

Слика 86: *Дијаграм промене температуре центра плоче*



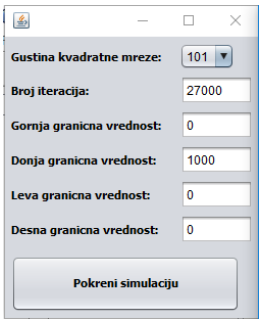
При анализи средње густине мреже 51x51, при 7000 итерација (слика 87), можемо приметити да се већ при 4744-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 250,072948 °C. Распоред температура након последње итерације приказан је на слици 88, док је на слици 89 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 87 (лево): *Почетни услови мерења*



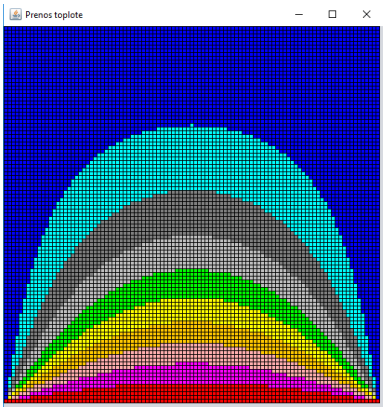
Слика 88: *Расподела температуре*

Слика 89: *Дијаграм промене температуре центра плоче*

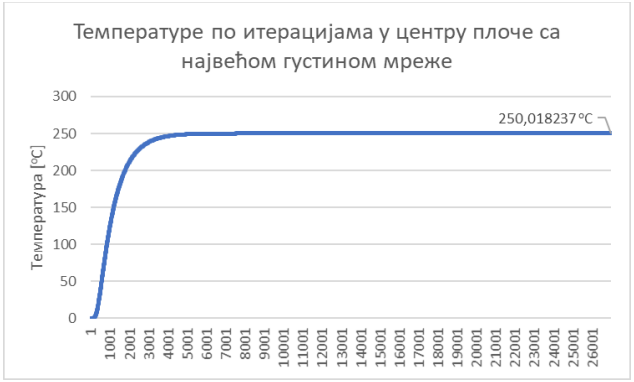


Приликом анализе највеће густине мреже 101x101, при 27000 итерација (слика 90), можемо приметити да се већ при 19333-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 250,018237 °C. Распоред температура након последње итерације приказан је на слици 91, док је на слици 92 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 90 (лево): *Почетни услови мерења*

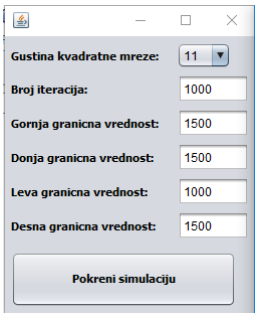


Слика 91: *Расподела температуре*



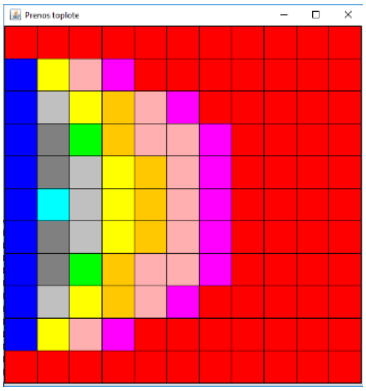
Слика 92: *Дијаграм промене температуре центра плоче*

5.1.5.4. Четврти пример:

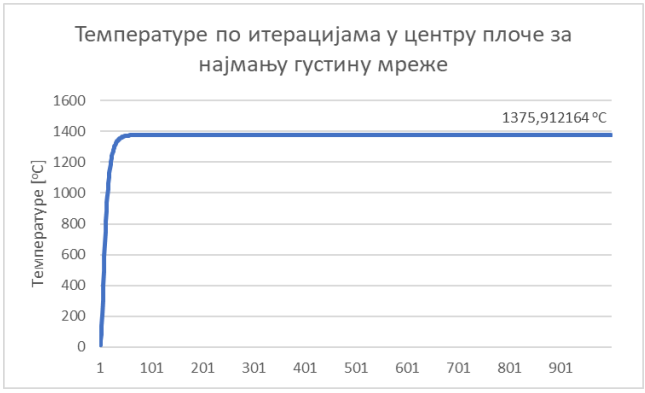


При анализи најмање густине мреже 11x11, при 1000 итерација (слика 93), можемо приметити да се већ при 206-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 1375,912164 °C. Распоред температура након последње итерације приказан је на слици 94, док је на слици 95 приказан дијаграм промене температуре у централној тачки по итерацијама.

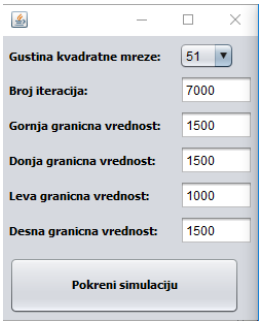
Слика 93 (лево): *Почетни услови мерења*



Слика 94: *Расподела температуре*

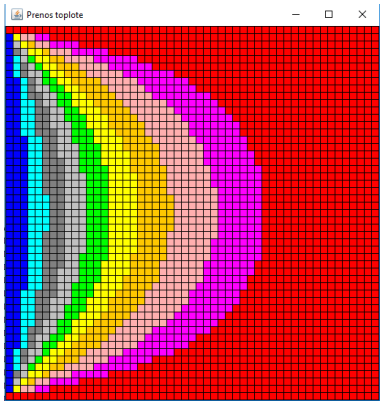


Слика 95: *Дијаграм промене температуре центра плоче*



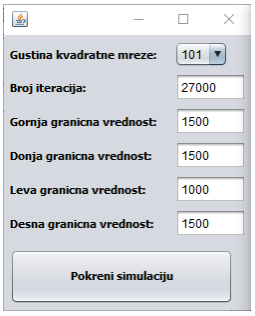
Приликом анализе средње густине мреже 51x51, при 7000 итерација (слика 96), можемо приметити да се већ при 4685-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 1375,036474 °C. Распоред температура након последње итерације приказан је на слици 97, док је на слици 98 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 96 (лево): *Почетни услови мерења*



Слика 97: *Расподела температуре*

Слика 98: *Дијаграм промене температуре центра плоче*



При анализи највеће густине мреже 101x101, при 27000 итерација (слика 99), можемо приметити да се већ при 18176-ој итерацији постиже равнотежна температура у централној тачки плоче која износи 1375,009118 °C. Распоред температура након последње итерације приказан је на слици 100, док је на слици 101 приказан дијаграм промене температуре у централној тачки по итерацијама.

Слика 99 (лево): *Почетни услови мерења*

Слика 100: *Расподела температуре*

Слика 101: *Дијаграм промене температуре центра плоче*

5.1.6. Упоређивање резултата

Посматрањем графика, дијаграма и добијених вредности дошли смо до следећих запажања. При малом броју итерација модификовани програм брже постиже стационарно стање, тј. брже конвергира. За случај већих вредности итерација брзина постизања стационарног стања варира од примера до примера. Посматрајући вредности температуре у централном пољу плоче долази се до закључка да нема већих одступања у току итерисања.

Ради провере тачности програма, упоредићемо резултате добијене у нашем програму према формули (5.1), са резултатима добијеним прорачуном у MATLAB-у према формули (5.2), за аналитичко решавање дводимензионалне топлотне једначине помоћу Фуријеових редова [34], за два примера граничних температура, где су све граничне температуре 0 °C, док је разлика у доњој граничној температури.

У првом примеру доња гранична температура је 100 °C, а у другом 1000 °C. Резултати по итерацијама и величини мреже приказани су на слици 102 (доња граница 100 °C) и слици 105 (доња граница 1000 °C). Грешка по итерацијама која се јавља приказана је на слици 103 (доња граница 100 °C) и слици 106 (доња граница 1000 °C).

$$\mathrm{T}[i,j] = \frac{4T_i}{\pi}\sum_{n=1}^{\infty}\frac{\sin\left[(2n-1)\frac{\pi x}{a}\right]\sinh\left[(2n-1)\frac{\pi y}{a}\right]}{(2n-1)\sinh\left[(2n-1)\frac{\pi b}{a}\right]} \qquad (5.2)$$

Такође, објашњено је како број децимала утиче на број итерација неопходан да се постигне коначна вредност температуре у центру плоче.

Потребно је још истаћи да је у оба примера приликом тестирања у MATLAB-у направљена грешка за величину мреже 10x10. Није постављено да се доња гранична вредност не мења по итерацијама, зато се јавља велика грешка у поређењу резултата.

Добијена вредност	100			У MATLAB-у вредност	100		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	296	25,18243271	10x10	5	3	20,396
50x50	7000	6700	25,0072948	50x50	5	3	24,149
100x100	27000	26220	25,0018237	100x100	5	4	24,578
Добијена вредност	100			У MATLAB-у вредност	100		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	296	25,18243271	10x10	50	3	20,396
50x50	7000	6700	25,0072948	50x50	50	3	24,149
100x100	27000	26220	25,0018237	100x100	50	4	24,578

Добијена вредност	100			У MATLAB-у вредност	100		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	98	25,182	10x10	5	3	20,396
50x50	7000	2509	25,007	50x50	5	3	24,149
100x100	27000	9146	25,001	100x100	5	4	24,578
Добијена вредност	100			У MATLAB-у вредност	100		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	98	25,182	10x10	50	3	20,396
50x50	7000	2509	25,007	50x50	50	3	24,149
100x100	27000	9146	25,001	100x100	50	4	24,578

Слика 102: *Табеле упоредних вредности коначних температура T=100 °C*

Посматрајући леви (плави) део прве и друге табеле на слици 102, долазимо до закључка да је у зависности од величине мреже потребан различити број итерација за постизање коначне вредности температуре. Смањивањем броја децимала, за које се посматра достизање коначне вредности температуре у центру плоче, постиже се бржа конвергенција, односно потребан је око три пута мањи број итерација, чиме је и време конвергенције знатно смањено. Ако сада посматрамо десни (наранџасти) део ових табела, закључујемо да се итерација у којој се постиже коначна вредност температуре у зависности од величине мреже готово не мења.

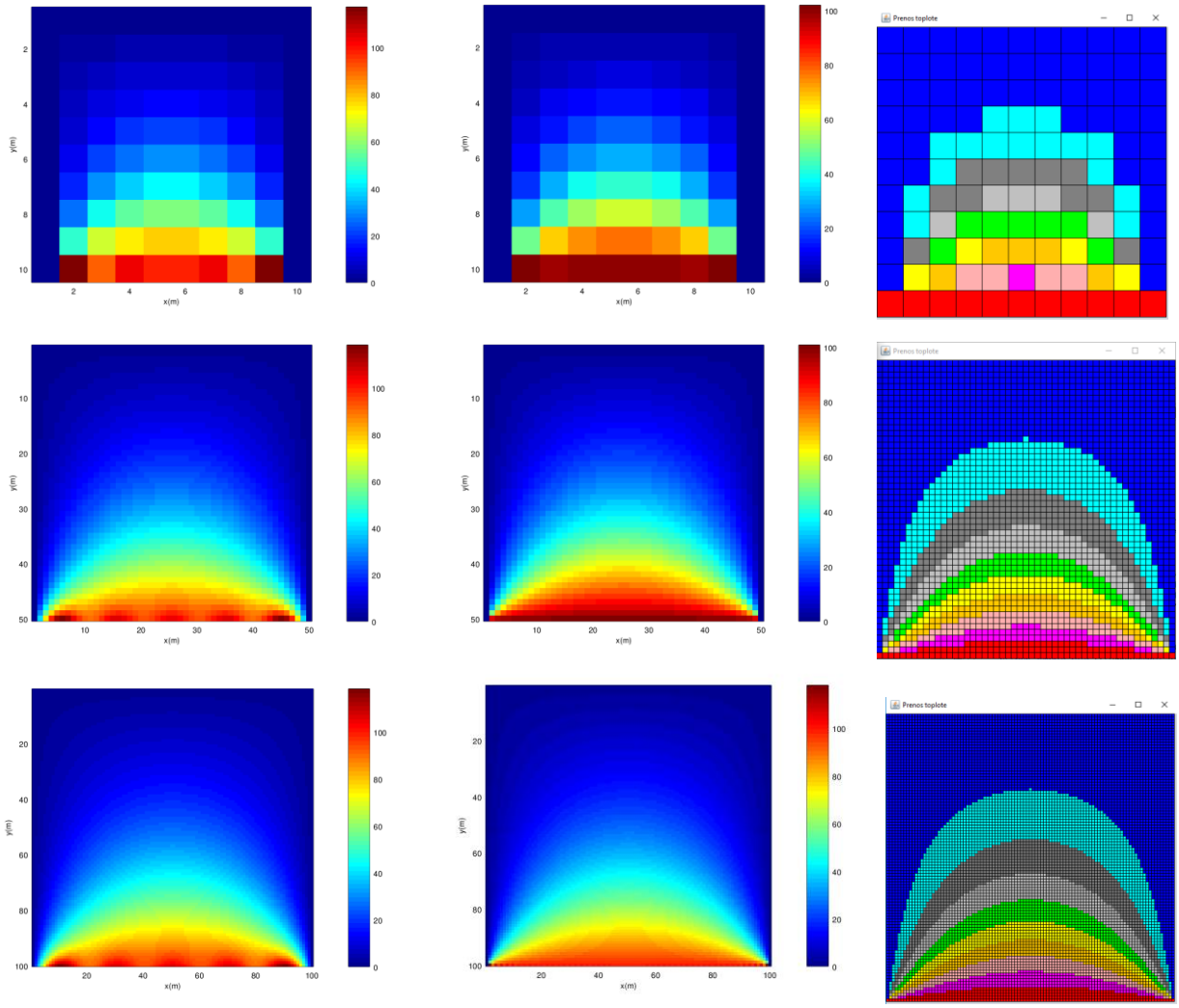
Грешка по итерацијама		Грешка по итерацијама	
Поређење по итерацијама за мрежу	100	Поређење по итерацијама за мрежу	100
10x10	19,0070306760858%	10x10	19,0056389484552%
50x50	3,4321777273011%	50x50	3,4310393089935%
100x100	1,6951711396078%	100x100	1,6919323227071%
Грешка по итерацијама		Грешка по итерацијама	
Поређење по итерацијама за мрежу	100	Поређење по итерацијама за мрежу	100
10x10	19,0070306760858%	10x10	19,0056389484552%
50x50	3,4321777273011%	50x50	3,4310393089935%
100x100	1,6951711396078%	100x100	1,6919323227071%

Слика 103: *Грешка по итерацијама за T=100 ^{o}C*

На слици 103 приказане су грешке за поједине густине мреже и доњу граничну температуру од T=100 ^{o}C у случајевима када је достизање коначне температуре посматрано по тринаестој децимали (табела лево) и по трећој децимали (табела десно). Долазимо до закључка да у оба случаја проценат грешке се не разликује у зависности од децималног места које се узима као референтно за коначну вредност температуре.

Такође, примећујемо да се са повећањем густине мреже постиже и мања грешка приликом итерисања. За прорачун температуре са нултим граничним вредностима и доњом граничном вредношћу од T=100 ^{o}C, у оба случаја се температура у центру плоче креће у опсезима блиским 25 ^{o}C. Температура која се добија прорачуном према формули (5.1) се не разликује много од температуре која се добија прорачуном према формули (5.2).

Графички приказ расподеле температуре за овај пример у MATLAB-у (слике лево и у средини) и у нашем програму (слике десно) приказани су у колони једна испод друге на слици 104, док слике у сваком реду представљају мреже густине од 10x10, 50x50 и 100x100. Слике лево представљају слике у MATLAB-у за 5 итерација, слике у средини су за 50 итерација, док су слике са десне стране графички прикази из нашег програма редом за 1000, 7000 и 27000 итерација. На графичким приказима уочава се да се добијене вредности у нашем програму са малим процентом грешке поклапају са вредностима добијеним у теорији.



Слика 104: *Упоредни графички приказ преноса топлоте*

Сада посматрамо други пример расподеле температуре код кога је доња гранична вредност температуре T=1000 ^{o}C. На слици 105 приказане су температуре по итерацијама и густини мреже добијне у нашем програму и у MATLAB-у. На слици 106 приказане су грешке по итерацијама за сваку мрежу у процентима.

Ако сада упоредимо овај пример са претходним, долазимо до закључка да са већом температуром, у нашем прорачуну неопходан је и већи број итерација за постизање коначне вредности, односно потребно је 0,6% више итерација за постизање коначне температуре. Сви закључци које смо донели у вези првог примера односе се и на други пример, па их није потребно посебно наводити. Тиме смо доказали да за различите вредности граничних температура, различити број итерација и различите густине мреже постиже приближно иста вредност коначне температуре у посматраном пољу плоче.

За крај дати су и графички прикази преноса топлоте овог примера на слици 107. Први ред односи се на MATLAB, а други ред приказује вредности добијене у нашем програму. Прва је за прорачун по формули (5.2), а друга према (5.1). По колонама приказане су различите густине мреже и то редом 10x10, 50x50 и 100x100.

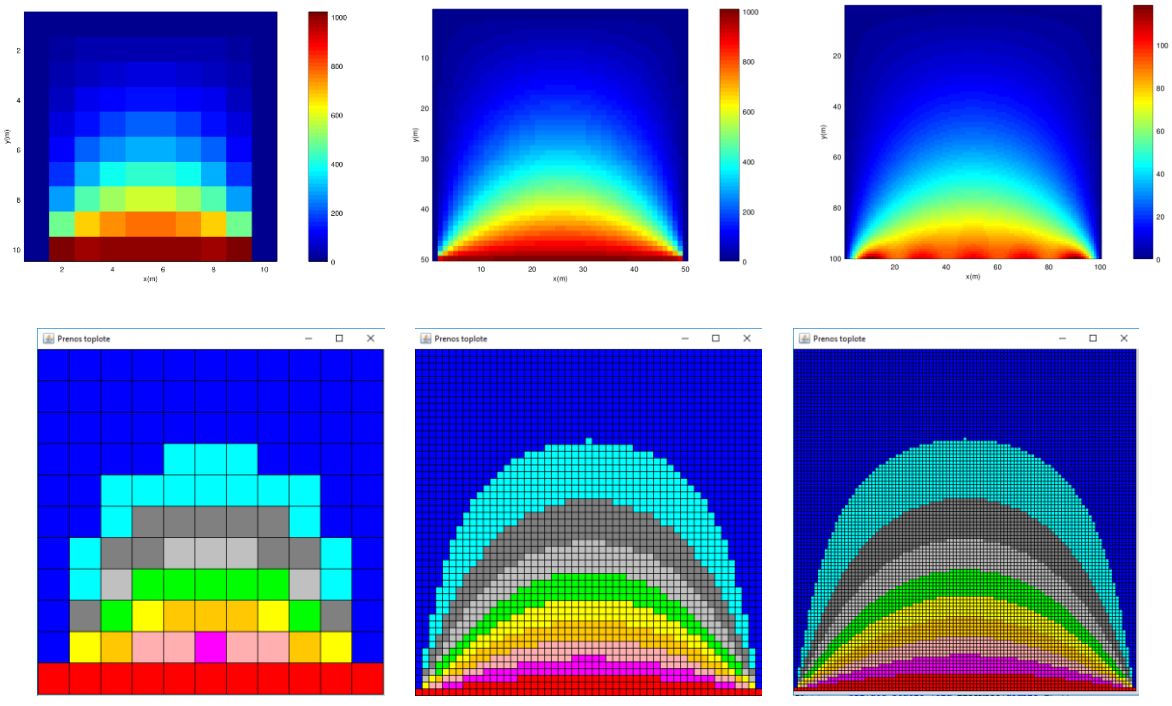
Добијена вредност	1000			У MATLAB-у вредност	1000		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	283	251,8243271	10x10	5	3	203,958
50x50	7000	6741	250,072948	50x50	5	4	241,485
100x100	27000	25933	250,018237	100x100	5	4	245,785
Добијена вредност	1000			У MATLAB-у вредност	1000		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	283	251,8243271	10x10	50	3	203,958
50x50	7000	6741	250,072948	50x50	50	4	241,485
100x100	27000	25933	250,018237	100x100	50	4	245,785

Добијена вредност	1000			У MATLAB-у вредност	1000		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	120	251,824	10x10	5	3	203,958
50x50	7000	2749	250,072	50x50	5	4	241,485
100x100	27000	12142	250,018	100x100	5	4	245,785
Добијена вредност	1000			У MATLAB-у вредност	1000		
Величина мреже	Максималан број итерација	Број итерација	Температура	Величина мреже	Максималан број итерација	Број итерација	Температура
10x10	1000	120	251,824	10x10	50	3	203,958
50x50	7000	2749	250,072	50x50	50	4	241,485
100x100	27000	12142	250,018	100x100	50	4	245,785

Слика 105: *Табеле упоредних вредности коначних температура T=1000 °C*

Грешка по итерацијама		Грешка по итерацијама	
Поређење по итерацијама за мрежу	1000	Поређење по итерацијама за мрежу	1000
10x10	19,0078248805308%	10x10	19,0077196772349%
50x50	3,4341771438875%	50x50	3,4338110624140%
100x100	1,6931712854923%	100x100	1,6930780983769%
Грешка по итерацијама		Грешка по итерацијама	
Поређење по итерацијама за мрежу	1000	Поређење по итерацијама за мрежу	1000
10x10	19,0078248805308%	10x10	19,0077196772349%
50x50	3,4341771438875%	50x50	3,4338110624140%
100x100	1,6931712854923%	100x100	1,6930780983769%

Слика 106: *Грешка по итерацијама за T=1000 °C*



Слика 107: *Упоредни графички приказ преноста топлоте*

5.2. Креирање JAVA апликације за нестационарни пренос топлоте

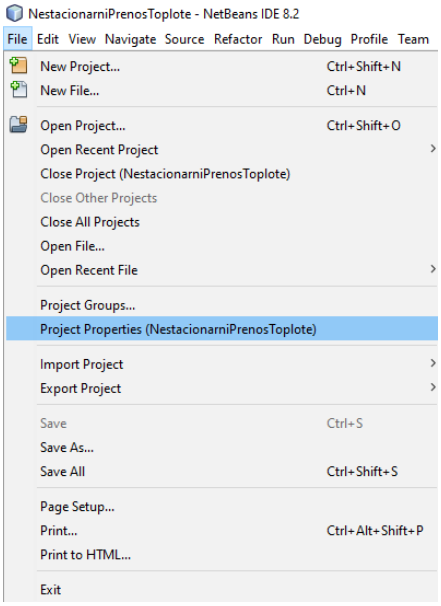
У овом поглављу приказан је поступак креирања две верзије програма за нестационарни пренос топлоте, јер се јављају два случаја задатих почетних и граничних услова, у првој верзији је задата температура, док је у другој дат топлотни флукс. У претходном поглављу је детаљно описано како се креира програм у радном окружењу NetBeans IDE 8.2, па сходно томе у овом поглављу није детаљно приказан поступак креирања програма, већ је акценат стављен на разликама које се јављају при креирању програма за стационарни и нестационарни пренос топлоте.

Креирање програма за нестационарни пренос топлте одвија се на исти начин као и код стационарног преноса топлоте, а што је објашњено у уводном делу поглавља 5.1. и приказано на сликама од 27 до 30. Разлика која постоји приликом креирања програма за нестационарни пренос топлоте је описана је у наставку овог поглавља. Ако се посматра слика 30. из поглавља 5.1., у пољу **ProjectName:** потребно је унети назив **NestacionarniPrenosToplote**. Исти назив уноси се приликом креирања обе верзије програма и до краја се понавља идентичан поступак као у претходном поглављу.

5.2.1. Креирање Java Windows display-a

Код програма за нестационарни преноос топлоте, за разлику од програма за стационарни пренос топлоте, није креиран још један пакет, већ је у оквиру постојећег пакета, аутоматски креираног са *main класом*, креирана форма за Windows display. Креирање ове форме приказано је у поглављу 5.1.1. на слици 34 и 35. Приликом попуњавања екрана који је дат на слици 35 уноси се следећи податак у пољу **Class Name: GUI**, овај податак се унос и приликом креирања друге верзије програма а затим се кликне дугме **Finish**. Изглед прозора и алати који се том приликом појаве на екрану су исти као и при креирању у поглављу 5.1.1 и приказани су на слици 36 из поменутог поглављa.

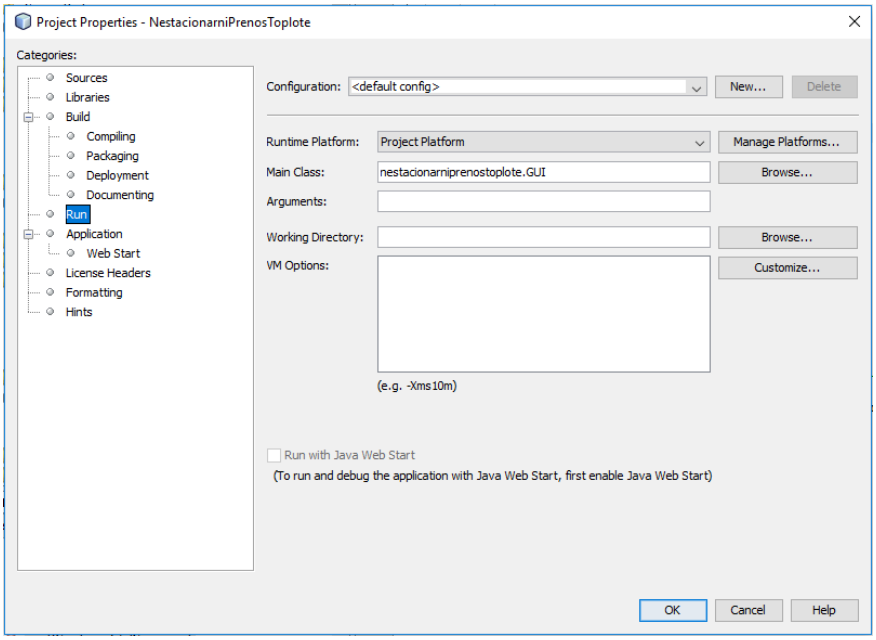
5.2.2. Дефинисање прве класе



Која класа ће се прва отворити при покретању програма, зависи од тога како се дефинише у програму. По аутоматизму уколико корисник креира апликацију са *main класом*, ова класа је дефинисана као прва класа и при покретању програма најпре се врши њено покретање. У случају да корисник не креира *класу main*, прва класу коју креира када заврши са делом креирања апликације постаје прва класе. Уколико корисник не жели да му једна од поменутих класа буде прва класа, или жели да замени тренутну прву класу неком новом класом коју је креирао, то може урадити на следећи начин који је приказан на сликама од 108 до 110.

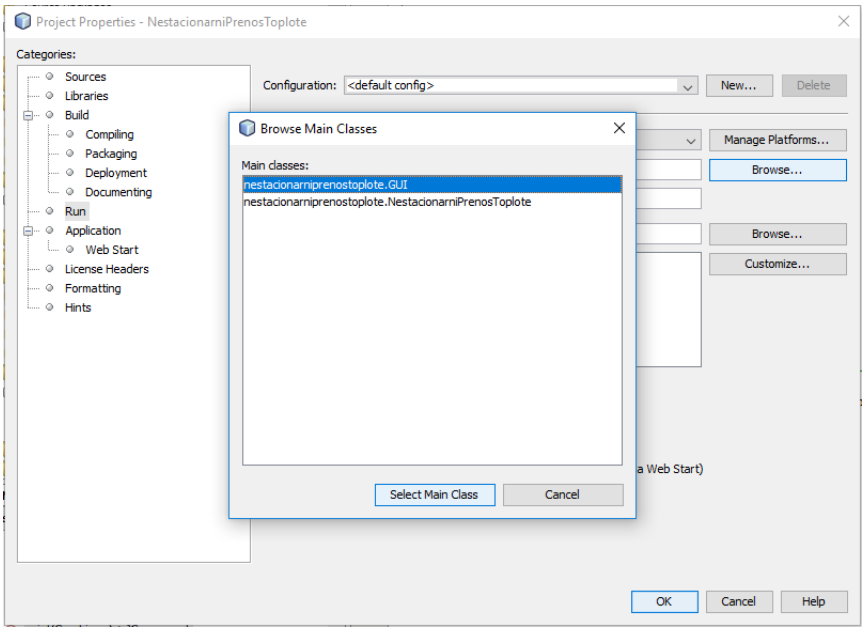
Слика 108 (лево): *Својства пројекта*

За мењање својства пројекта, односно у овом случају за дефинисање прве класе мора се отворити прозор у коме се ово својство може променити. То се може урадити кликом на падајући мени **File – Project Properties (NestacionarniPrenosToplote)**, а што је приказно на слици 108. Након тога отвара се нови прозор прозор на чијој левој страни се из дела **Categories** врши одабир онога што се мења. У овом случају да би се заменила прва форма мора се изабрати део који се односи на покретање програма, а то је у листи означено са **RUN**. Кликом на RUN изглед прозора постаје као на слици 109.



Слика 109: *Дефинисање покретања програма*

На датом екрану у правцу дела где пише **Main Class:** кликне се на дугме **Browse...** након чега се отвара нови прозор као на слици 110.

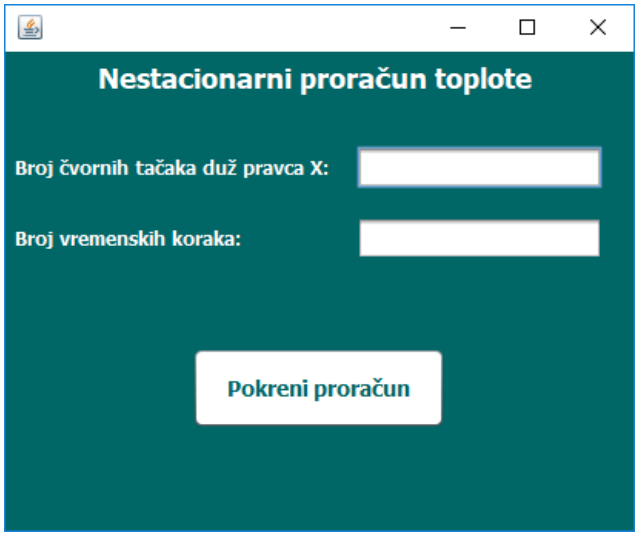


Слика 110: *Одабир прве класе*

Аутоматски се генеришу све класе које су креиране у оквиру овог програма. Корисник бира коју класу жели да постави као прву класу односно као *main класу*. У случају овог програма то је **nestacionarniprenostoplote.GUI**. Кликне се на жељену класу, затим се кликне дугме **Select Main Class**, а одмах након тога дугме **OK**. На овај начин постављена је нова класа која ће се прва приказивати при покретању програма.

5.2.3. Уређивање форме прве верзије програма

За уређивање форме користе се картица **Palette** и **Properties** које су детаљно објашњене у поглављу 5.1.2. На слици 111. приказан је изглед форме креиран за потребе овог програма.

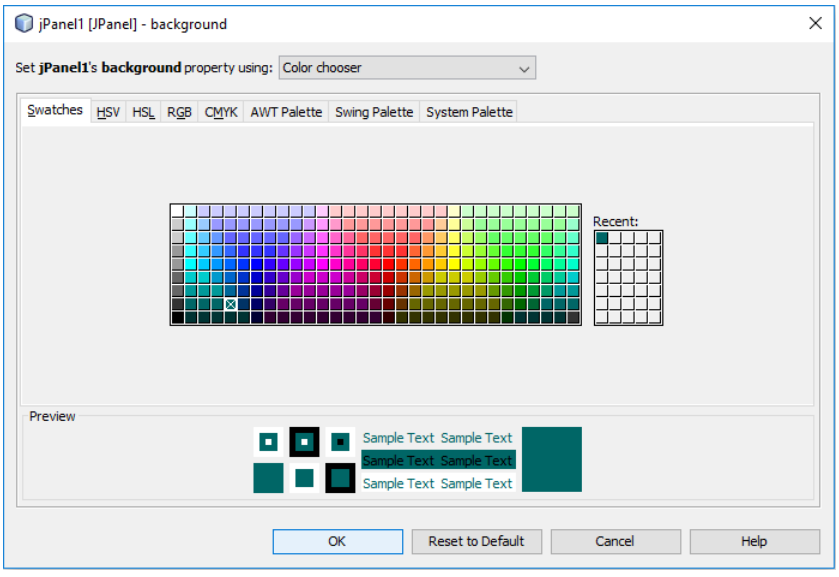


Слика 111 (лево): *Изглед форме програма за нестационарни пренос топлоте када су задате почетне и граничне температуре*

За креирање ове форме из картице **Palette** коришћене су следеће компоненте: **Panel**, **Label** (лабела), **Text Field** (празно поље за унос текста) и **Button** (дугме). У картици **Properties** извршене су промене својстава ових компоненти и то:

1. За Panel: **background**;
2. За Label: **text**, **font** и **foreground**;
3. За Text Field: **text** и **background**;
4. За Button: **text**, **font, foreground** и **background**.

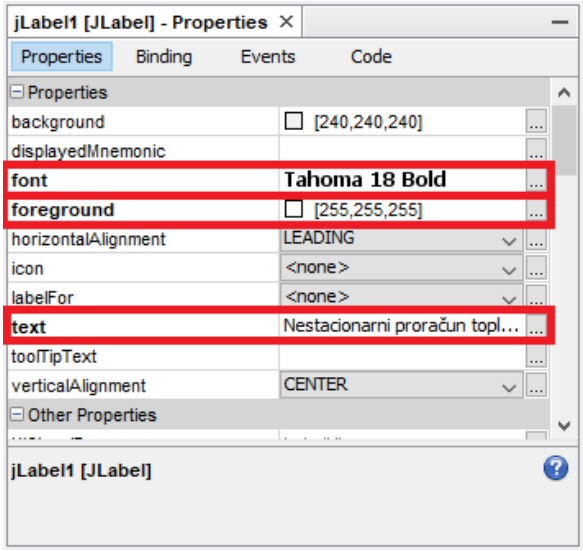
За потребе креирања овог програма од компоненти су примењени: по један Panel и Button, три Label-а и два Text Field-а.



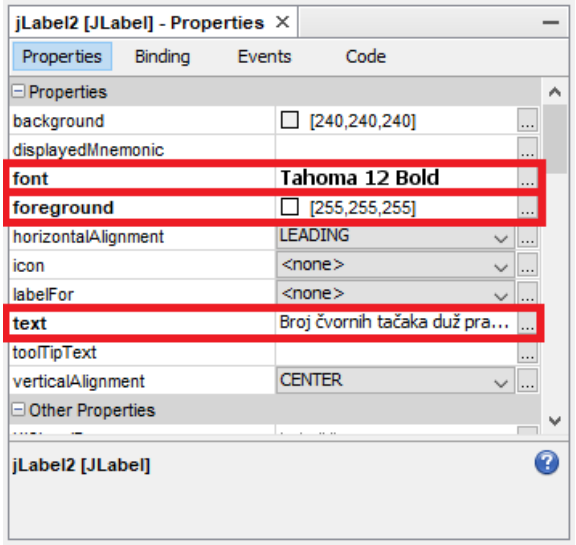
На слици 112 приказан је одабир боје за позадину панела односно background. На приказаној палети кликне се на одговарајућу боју, а затим се потврди кликом на дугме **OK**.

Слика 112 (лево): *Одабир боје за позадину панела*

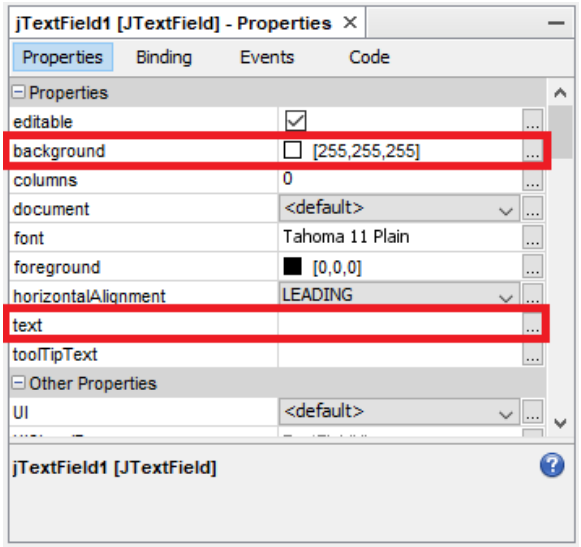
На сликама од 113 до 116 приказана су подешавања која су извршена за све компоненте коришћене за креирање ове верзије програма.



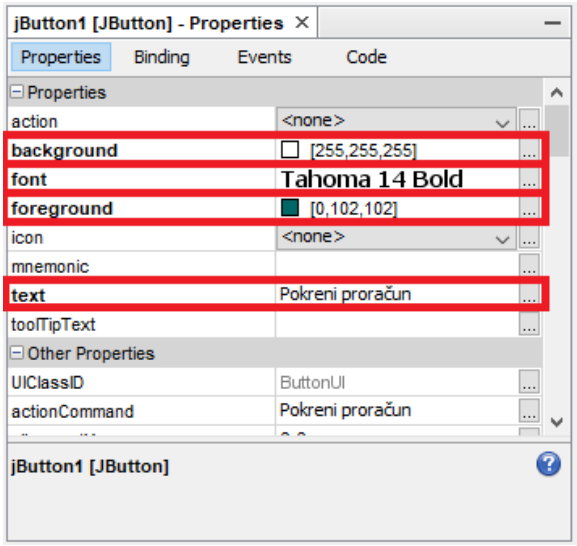
Слика 113: *Подешавање лабеле наслова*



Слика 114: *Подешавање осталих лабела*



Слика 115: *Подешавање празних поља*



Слика 116: *Подешавање дугмета*

5.2.4. Програмирање форме и главног програма прве верзије програма

Да би компоненте постављене на форми функционисале неопходно је извршити њихово програмирање. Програмирање форме и главног програма се врше на исти начин као и код програма за стационарни пренос топлоте и то је детаљно објашњено у поглављу 5.1.3. У овом поглављу укратко је објашњен код форме, док је код главног програма детаљно објашњен.

5.2.4.1. Програмирање форме

Што се програмирања форме тиче овде је истакнуто само програмирање дешавања у случају када корисник кликне дугме. Том приликом све податке које је корисник унео у празна поља програм прикупља, обрађује, претвара у променљиве одговарајућег типа и шаље главном програму који даље те вредности користи за израчунавања. На слици 117 приказан је код којим се програмира дугме. Значење свих линија кода је исто као што је објашњено код стационарног преноса топлоте.

```
119     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
120         int broj_cvornih_tacaka = Integer.parseInt(jTextField1.getText());
121         int broj_vremenskih_koraka = Integer.parseInt(jTextField2.getText());
122 
123         nestacionarniprenostoplote.NestacionarniPrenosToplote p = new NestacionarniPrenosToplote();
            p.main(null);
125         p.setNx(broj_cvornih_tacaka);
126         p.setNt(broj_vremenskih_koraka);
127     }
```

Слика 117: *Програмирање дугмета*

5.2.4.2. Програмирање главног програма

Креирање пакета (слика 118).

```
1   package nestacionarniprenostoplote;
```

Слика 118: *Креирање пакета*

Позивање коришћених готових библиотека приказано је на слици 119.

```
3   import java.awt.Color;
4   import java.awt.Graphics;
5   import java.io.FileNotFoundException;
6   import java.io.PrintWriter;
7   import java.text.DecimalFormat;
8   import javax.swing.JComponent;
9   import javax.swing.JFrame;
```

Слика 119: *Коришћене библиотеке*

Креирање главне јавне класе са применом графичких компоненти (слика 120).

```
11  public class NestacionarniPrenosToplote extends JComponent {
```

Слика 120: *Главна класа*

Иницијализација јавних променљива приказана је на слици 121.

```
13      public static int nx = 0;
14      public static int nt = 0;
```

Слика 121: *Дефинисање јавних променљива*

Креирање get-ера и set-ера за приступ променљивама из друге класе (слика122 и 123).

```
16      public int getNx() {
17          return nx;
18      }
19
20      public void setNx(int nx) {
22          this.nx = nx;
        }
```

Слика 122: *Креирање get-ера и set-ера за променљиву Nx*

```
24      public int getNt() {
25          return nt;
26      }
27
28      public void setNt(int nt) {
30          this.nt = nt;
        }
```

Слика 123: *Креирање get-ера и set-ера за променљиву Nt*

Креирање методе за исцртавање графике (слика 124).

```
32      //Kreiranje grafike
        public void paint(Graphics g) {
```

Слика 124: *Креирање графике*

Иницијализација локалних променљивих приказана је на слици 125.

```
35          //Inicijalizacija promenljivih
36          int x, t, i;
37          double lambda = 0.5;
38          double[][][] f = new double[100][2][100];
39          double[][] T = new double[100][100];
40          double[][] feq = new double[2][100];
41          DecimalFormat df = new DecimalFormat("0.0000");
```

Слика 125: *Локалне променљиве*

Дефинисање почетних услова приказано је на слици 126, са *f[0][0][0]* је означен леви чвор првог поља, а са *f[0][1][0]* је означен десни чвор првог поља. Њиховим сабирањем добија се температура у центру поља. Исто важи и за ове функције у *for* петљи само што су оне дефинисане за последње поље штапа.

```
43          f[0][0][0] = 0.5;
44          f[0][1][0] = 0.5;
45
46          for (x = 1; x <= nx; x++) {
47              f[0][0][x] = 1.5;
48              f[0][1][x] = 1.5;
49          }
```

Слика 126: *Почетни услови*

Петља по временском кораку (слика 127).

```
51          for (t = 0; t <= nt; t++) {
```

Слика 127: *Петља по временском кораку*

Петља по пољима штапа, односно дуж чворних тачака (слика 128).

```
52              for (x = 0; x <= nx; x++) {
```

Слика 128: *Петља по пољима*

Прорачун промене температуре дат је следећим кодом приказаним на слици 129. За прорачун је коришћена LB метода за 1-D проблем преноса топлоте.

```
53              //Temperatura pre sudara
54              T[x][t] = f[t][0][x] + f[t][1][x];
55              for (i = 0; i < 2; i++) {
56                  feq[i][x] = 0.5 * T[x][t];
57                  f[t][i][x] = f[t][i][x] + lambda * (feq[i][x] - f[t][i][x]);
58              }
59              //Temperatura posle sudara
60              T[x][t] = f[t][0][x] + f[t][1][x];
```

Слика 129: *Прорачун промене температуре*

Пропагација је приказана на слици 130.

```
63              //Propagacija
64              for (x = 1; x <= nx; x++) {
65                  f[t + 1][0][x - 1] = f[t][0][x];
66              }
67
68              for (x = 0; x <= nx - 1; x++) {
69                  f[t + 1][1][x + 1] = f[t][1][x];
70              }
```

Слика 130: *Пропагација*

Дефинисање граничних услова дато је на слици 131.

```
72              //Granicni uslovi
73              f[t + 1][0][0] = 0.5;
74              f[t + 1][1][0] = 0.5;
75              f[t + 1][0][nx] = 1.5;
76              f[t + 1][1][nx] = 1.5;
```

Слика 131: *Дефинисање граничних услова*

Штампање температуре у одређеном формату у конзоли приказано је на слици 132.

```
79          System.out.println("MATRICA T");
80          for (t = 0; t < nt; t++) {
81              for (x = 0; x < nx; x++) {
82                  System.out.print(df.format(T[x][t]) + "  " + "  ");
83                  //System.out.print(T[t][x] + "  ");
84              }
85              System.out.println("");
86          }
```

Слика 132: *Форматирано штампање у конзоли*

Чување резултата температуре по временским корацима у текстуалном фајлу (слика133).

```
88          //Cuvanje rezultata u tekstualnom dokumentu
89          String Temperature = "Temperature.txt";
90          try {
91              PrintWriter Rezultat = new PrintWriter(Temperature);
92              for (t = 0; t < nt; t++) {
93                  for (x = 0; x < nx; x++) {
94                      Rezultat.print(df.format(T[x][t]) + "  " + "  ");
95                  }
96                  Rezultat.println("");
97                  Rezultat.println("");
98              }
99              Rezultat.close();
100         } catch (FileNotFoundException ex) {
101         }
```

Слика 133: *Чување резултата у текстуалном документу*

Дефинисање величине поља које ће се графички приказивати (слика 134).

```
103             int sirina = 20;
104             if (nx < 10) {
                    sirina = sirina;
106             } else if (nx > 20) {
107                 sirina = 500 / nx;
108             }
```

Слика 134: *Дефинисање величине приказиваног поља*

Креирање графике у зависности од вредности израчунате температуре дато је на слици 135.

```
110         for (t = 0; t < nt; t++) {
111             for (x = 0; x < nx; x++) {
112                 if (T[x][t] <= 0.75) {
113                     g.setColor(Color.BLUE);
114                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
115                 } else if (T[x][t] > 0.75 && T[x][t] <= 1) {
116                     g.setColor(Color.CYAN);
117                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
118                 } else if (T[x][t] > 1 && T[x][t] <= 1.25) {
119                     g.setColor(Color.GRAY);
120                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
121                 } else if (T[x][t] > 1.25 && T[x][t] <= 1.5) {
122                     g.setColor(Color.LIGHT_GRAY);
123                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
124                 } else if (T[x][t] > 1.5 && T[x][t] <= 1.75) {
125                     g.setColor(Color.GREEN);
126                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
127                 } else if (T[x][t] > 1.75 && T[x][t] <= 2) {
128                     g.setColor(Color.YELLOW);
129                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
130                 } else if (T[x][t] > 2 && T[x][t] <= 2.25) {
131                     g.setColor(Color.ORANGE);
132                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
133                 } else if (T[x][t] > 2.25 && T[x][t] <= 2.5) {
134                     g.setColor(Color.PINK);
135                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
136                 } else if (T[x][t] > 2.5 && T[x][t] <= 2.75) {
137                     g.setColor(Color.MAGENTA);
138                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
139                 } else if (T[x][t] > 2.75 && T[x][t] <= 3) {
140                     g.setColor(Color.RED);
141                     g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
142                 }
143                 g.setColor(Color.BLACK);
144                 g.drawRect(x * sirina, t * 2 * sirina, sirina, sirina);
145             }
146         }
```

Слика 135: *Креирање графике*

Креирање прозора у коме се исцртава графика (слика 136).

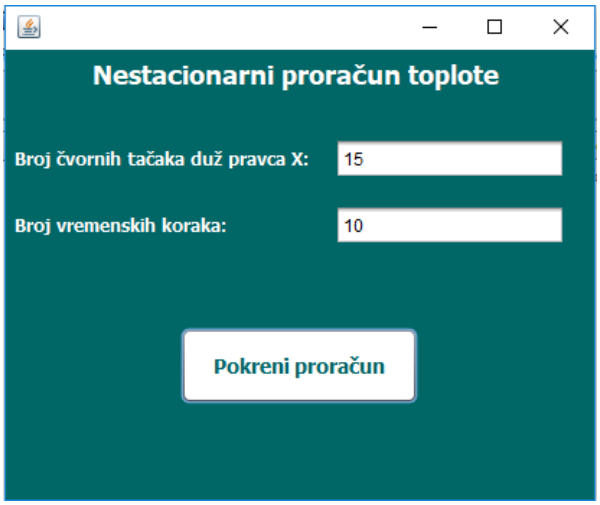
```
149     /**
150      * @param args the command line arguments
151      */
152     public static void main(String[] args) {
153         //Kreiranje prozora
154         JFrame window = new JFrame();
155         window.setTitle("Prenos toplote");
156         window.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
157         window.setBounds(10, 10, 600, 600);
158         window.getContentPane().add(new NestacionarniPrenosToplote());
159         window.setVisible(true);
160     }
161 }
```

Слика 136: *Креирање прозора за приказ графике*

Детаљна објашњења сваке линије кода су иста као и у програму за стационарни пренос топлоте и нису поново наведена. Комплетани кодови за обе класе дати су у прилогу 5 и 6.

5.2.5. Пример примене LBM у раду прве верзије програма

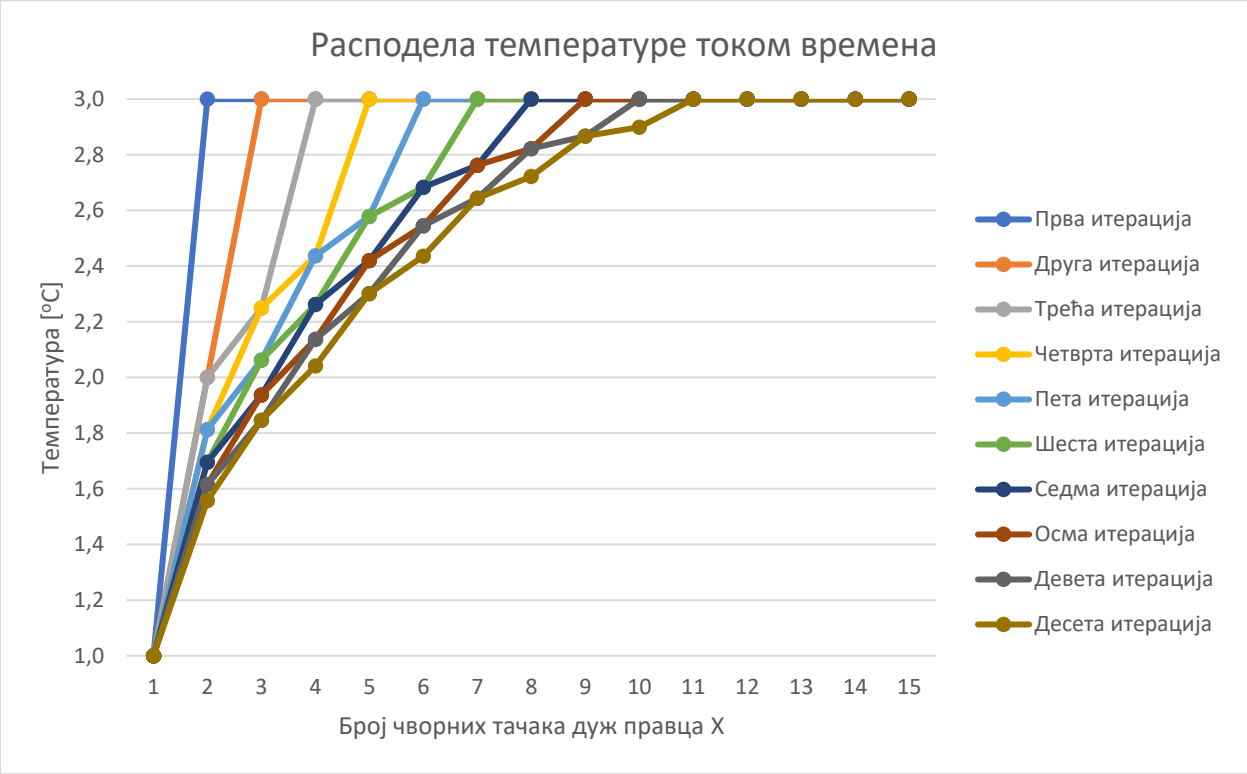
Дати програм решава проблем када су задате почетне и граничне температуре применом LB методе. Задати параметри при покреању програма су следећи: температура у првом пољу износ 1°C, а у осталим пољима ова температура је 3°C, вредност λ износи 0.5 $\frac{W}{m\,°C}$. Реч је о једнодимензионалном проблему, па се за прорачун преноса топлоте користи формула (4.18). Приликом креирања програма усвојене су следеће вредности: $\Delta t = 1$, $e_1 = 1$ и $e_2 = -1$, па је $\left|\underset{e_i}{\rightarrow}\right|^2 = \sqrt{1^2 + (-1)^2}^2 = 2$, $\alpha = 3$, па према формули (4.10) добија се да је $\tau = 2$, за w_i за D1Q2 проблем примењена је вредност дата изразом (4.14). Када су дефинисани сви параметри може се прећи на примену програма. На слици 137 приказан је прозор са унетим параметрима, притиском на дугме „*Pokreni proračun*“, врши се прорачунавање температуре по временским корацима и графички приказ ширења топлоте дуж штапа по временским корацима, а што је приказано на слици 138.



Слика 137: *Главни прозор*

Слика 138: *Графички приказ ширења топлоте*

На основу добијених резултата, који су достављени у посебном excel документу, добија се расподела температуре током времена, а што је приказано дијаграмом на слици 139.



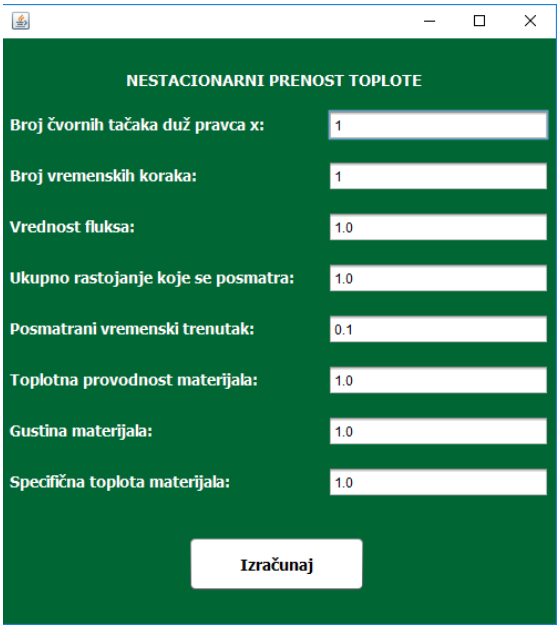
Слика 139: *Расподела температуре током времена*

Са графика се види да при константном граничном услову на почетку и крају штапа, са повећањем броја пролаза, задата почетна температура штапа се све више и више смањује како се иде од почетка ка крају штапа, односно долази се до закључка да се при непромењеним граничним условима на почетку и крају штапа, крећући се по временским корацима, у сваком наредном временском кораку, посматрајући од почетка ка крају штап, температура штапа опада.

У овом програму акценат је стављен на промени број чворова, односно броја поља, и промену временског корака за које се рачуна расподела температуре. Када се врши промена броја чворова примећује се да се применом LB методе са поратом броја чворова температура задата у првом пољу, спорије шири дуж целог штапа при не промењеном броју временских корака. Са повећањем временског корака повећава се и расподела температуре дуж штапа.

У овом поглављу било је речи о примени LB методе и закључцима до којих се долази на основу добијених резултата када се ова метода примени. У наредном поглављу приказана је друга верзија програма у којој се кориснику дозвољава да уноси већи број параметара и извршена је упоредна анализа примене овог програма коришћењем LBM-а и МКЕ (**М**етоде **К**оначних **Е**лемената). Такође је дат и скраћени поступак креирања овог програма уз анализу кода.

5.2.6. Уређивање форме друге верзије програма



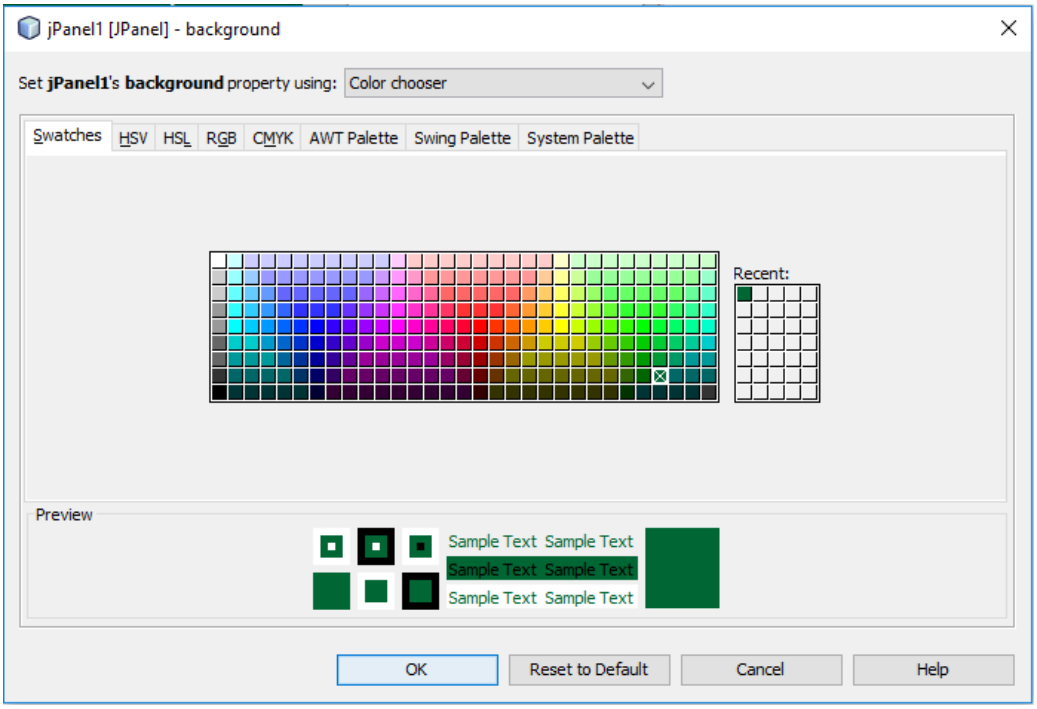
На слици 140. приказан је изглед форме креиран за потребе овог програма. За креирање ове форме из картице **Palette** коришћене су следеће компоненте: **Panel**, **Label** (лабела), **Text Field** (празно поље за унос текста) и **Button** (дугме). У картици **Properties** извршене су промене својстава ових компоненти и то:

5. За Panel: **background**;
6. За Label: **text**, **font** и **foreground**;
7. За Text Field: **text**;
8. За Button: **text**, **font** и **background**.

За потребе креирања овог програма од компоненти су примењени: по један Panel и Button, девет Label-а и осам Text Field-а.

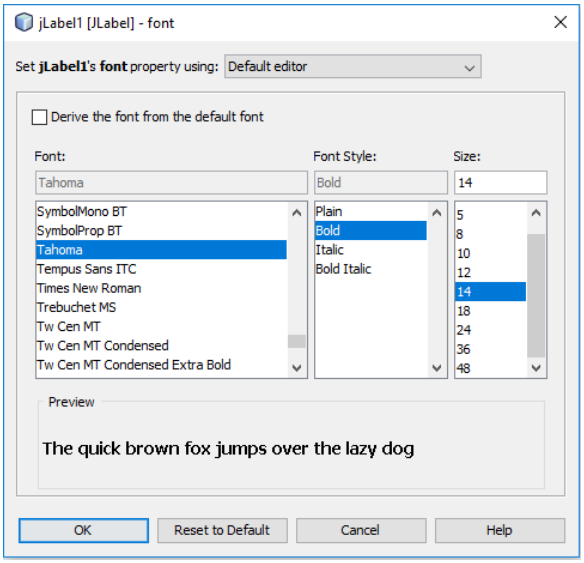
Слика 140 (лево): *Изглед форме програма за нестационарни пренос топлоте*

На слици 141 приказан је одабир боје за позадину панела односно background. На приказаној палети кликне се на одговарајућу боју, а затим се потврди кликом на дугме **OK**.

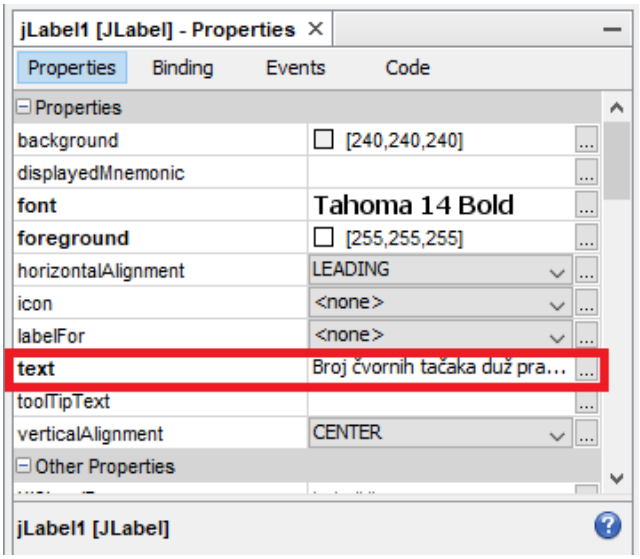


Слика 141: *Одабир боје за позадину панела*

На слици 142 приказан је одабир font-а који се примењује на свих девет лабела, док је на слици 143 приказано поље text-а попуњено за прву лабелу. Подразумева се да се то поље мења, односно да се за сваку лабелу у том пољу уписује друга вредност.

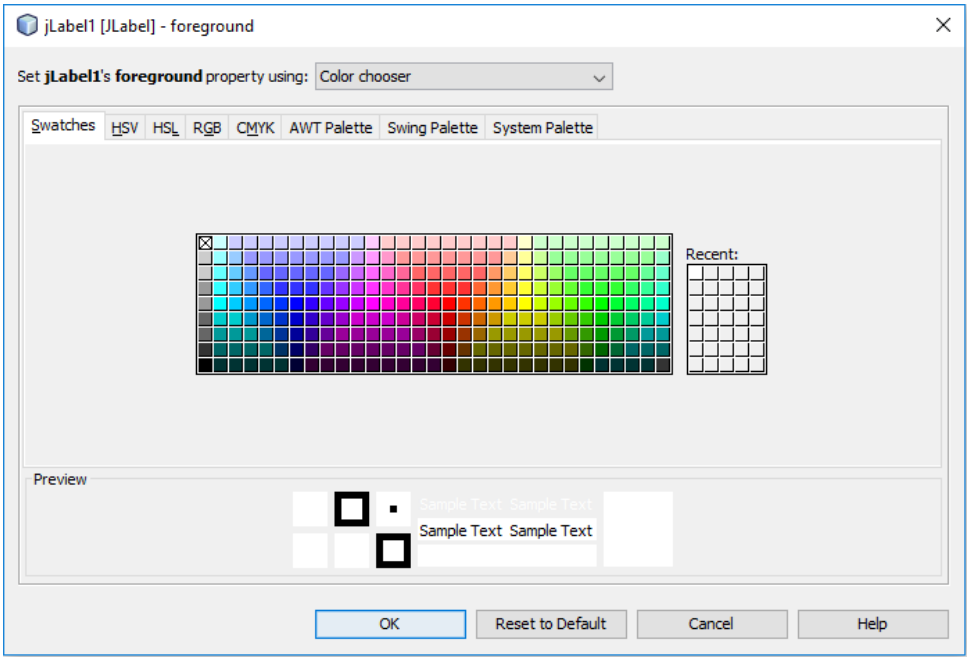


Слика 142: *Одабир font-a*



Слика 143: *Поље за унос text-a*

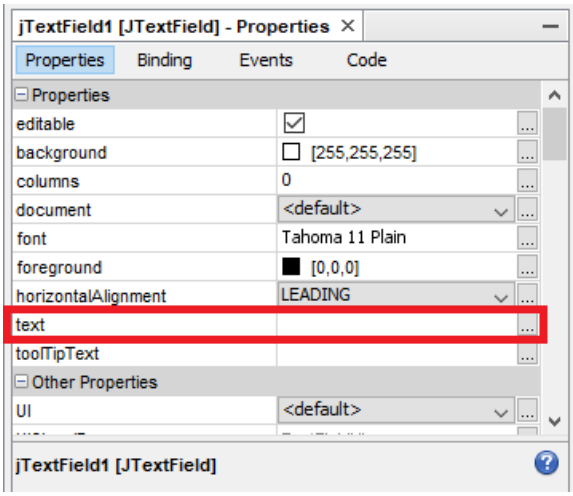
Поред промене font-a и text-a, код лабела се мења и боја слова којима се вредности лабеле приказују на форми односно foreground. Одабир боје је приказан на слици 144, а правила за одабир боје су иста као и код одабира боје за позадину панела.



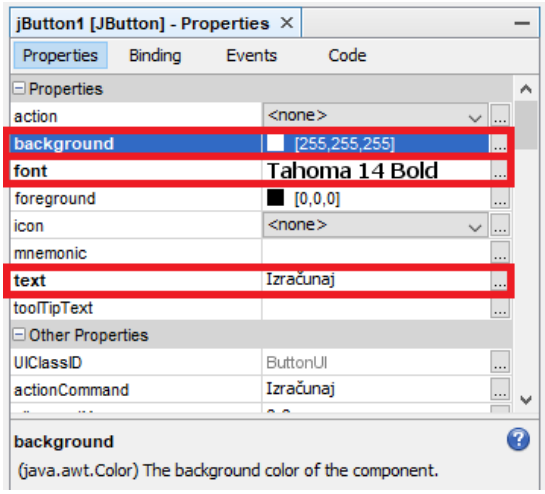
Слика 144: *Одабир боје слова*

Код Text Field-a у пољу text кориснику су већ унете неке вредности како би био упознат са типом податка који је потребно да унесе ради прецизнијег израчунавања. Ово поље за Text Field приказано је на слици 145, а оно се односни и на осталих седам Text Field-ова.

На слици 146 приказана су сва поља која се мењају при постављању дугмета на форму. Односно приказана су сва својства компоненте Button која се мењају.



Слика 145: *Празно поље text*



Слика 146: *Својства дугмета*

5.2.7. Програмирање форме друге верзије програма

Што се програмирања форме тиче истакнуто је само програмирање дешавања у случају када корисник кликне дугме. На слици 147 приказан је код којим се програмира дугме.

```
193     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
194         int broj_cvornih_tacaka = Integer.parseInt(jTextField1.getText());
195         int broj_vremenskih_koraka = Integer.parseInt(jTextField2.getText());
196         double fluks = Double.parseDouble(jTextField3.getText());
197         double ukupno_posmatrano_rastojanje = Double.parseDouble(jTextField4.getText());
198         double posmatrani_vremenski_trenutak = Double.parseDouble(jTextField5.getText());
199         double duzina_vremenskog_koraka = Double.parseDouble(jTextField6.getText());
200         double toplotna_provodnost = Double.parseDouble(jTextField7.getText());
201         double gustina = Double.parseDouble(jTextField8.getText());
202         double specificna_toplota = Double.parseDouble(jTextField9.getText());
203
204
205         nestacionarniprenostoplote.NestacionarniPrenosToplote p = new NestacionarniPrenosToplote();
            p.main(null);
207         p.setNx(broj_cvornih_tacaka);
208         p.setNt(broj_vremenskih_koraka);
209         p.setq(fluks);
210         p.setux(ukupno_posmatrano_rastojanje);
211         p.setvt(posmatrani_vremenski_trenutak);
212         p.setdelta_t(duzina_vremenskog_koraka);
213         p.setk(toplotna_provodnost);
214         p.setro(gustina);
215         p.setCp(specificna_toplota);
216     }
```

Слика 147: *Код за програмирање дугмета*

Значење свих линија кода је исто као што је објашњено код стационарног преноса топлоте, једино што је додато јесу променљиве типа *double* које имју исто значење као и објашњење за променљиве типа int, осим што се ове променљиве односе на децималне реалне променљиве са двоструком прицизношћу. Објашњење које следи односи се и на све остале променљиве овог типа.

double fluks = Double.parseDouble(jTextField1.getText()); променљива *fluks* је локална променљива која се користи у класи *GUI.java* за дефинисање вредности флукса. Код са друге стране знака једнакости говори о томе да се вредности које корисник уноси као текст претварају у променљиву типа double, односо у децималну реалну променљиву двоструке прецизности. Управо то се врши кодом са друге стране знака једнакости. Вредност коју корисник унесе претвара се у децималан број који се затим додељује променљивој *fluks*.

5.2.8. Програмирање главног програма друге верзије

Приликом креирања програма креира се аутоматки се генерише пакет (слика 148).

```
1    package nestacionarniprenostoplote;
```

Слика 148: *Дефинисање пакета*

Иницијализација библиотека које су коришћене у програму (слика 149).

```
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.io.FileNotFoundException;
6  import java.io.PrintWriter;
7  import java.text.DecimalFormat;
8  import javax.swing.JComponent;
9  import javax.swing.JFrame;
```

Слика 149: *Позивање библиотека*

Класа у којој је смештен комплетан код овог програма, у којој се врше сва израчунавања и која даје графички приказ израчунатих вредности (слика 150).

```
11    public class NestacionarniPrenosToplote extends JComponent {
```

Слика 150: *Главна класа*

Иницијализација свих јавних променљивих чије се вредности дефинишу у другој класи (слика 151).

```
13        public static int nx = 1;
14        public static int nt = 1;
15        public static double q = 1.0;
16        public static double ux = 1.0;
17        public static double vt = 0.1;
18        public static double k = 1.0;
19        public static double ro = 1.0;
20        public static double Cp = 1.0;
```

Слика 151: *Иницијализација променљивих*

Креирање get-ера и set-ера за преузимање вредности променљиве из друге класе и додељивање те вредности променљивој у овој класи (слика 152 и 153). Ови get-ери и set-ери се на истом принципу креирају и за остале променљиве које су иницијализоване на слици 151, зато нису и они посебно приказивани.

```
23      public int getNx() {
24          return nx;
25      }
26
27      public void setNx(int nx) {
            this.nx = nx;
29      }
```

Слика 152: *Get-ер и set-ер променљиве nx*

```
31      public int getNt() {
32          return nt;
33      }
34
35      public void setNt(int nt) {
            this.nt = nt;
37      }
```

Слика 153: *Get-ер и set-ер променљиве nt*

Креира се метод у оквиру главне класе који служи за креирање графичког приказа (слика 154).

```
86          //Kreiranje grafike
87          @Override
            public void paint(Graphics g) {
```

Слика 154: *Метод за креирање графике*

Иницијализација локалних променљивих (слика 155).

```
90          //Inicijalizacija promenljivih
91          double[] f1 = new double[1000];
92          double[] f2 = new double[1000];
93          double[] rho = new double[1000];
94          double[] feq = new double[1000];
95          double[] T = new double[1000];
96          double[] x = new double[1000];
97          double[] vreme = new double[1000];
98          int i, t;
99          double dx = ux / nx;
100         double dt = vt / nt;
```

Слика 155: *Иницијализација локалних променљивих*

Дефинисање растојања и времена (слика 156), иницијализација почетне вредности и увећање које се добија са сваким наредним кораком.

```
102         //Definisanje rastojanja x
103         x[0] = 0.0;
104         for (i = 1; i <= nx; i++) {
105             x[i] = x[i - 1] + dx;
106         }
107
108         //Definisanje rastojanja t
109         vreme[0] = 0.0;
110         for (t = 1; t <= nt; t++) {
111             vreme[t] = vreme[t - 1] + dt;
112         }
```

Слика 156: *Дефинисање ратојања i vremena*

Дефинисање осталих параметара који фигуришу у овом програму, а од значаја су сам прорачун нестационарног преноса топлоте (слика 157).

```
114         //Definisanje parametara
115         double csq = dx * dx / (dt * dt);
116         double ei = dt * csq;
117         double alfa = k / (ro * Cp);
118         double tau = (alfa / ei) + 0.5;
119         double omega = 1.0 / tau;
```

Слика 157: *Дефинисање параметара*

Дефинисање почетних услова, односно постављање почетних вредности температуре по пољима (слика 158).

```
124         //Pocetni uslovi
125         for (i = 0; i <= nx; i++) {
126             rho[i] = 0.0;
127             f1[i] = 0.5 * rho[i];
128             f2[i] = 0.5 * rho[i];
129         }
```

Слика 158: *Иницијализација почетних вредности температуре*

Главна петља по временским корацима (слика 159).

```
131         //Proracun po vremenskim koracima
132         for (t = 1; t <= nt; t++) {
```

Слика 159: *Петља по временским корацима*

Петља за прорачун вредности температуре у процесу судара (слика 160).

```
134             for (i = 0; i <= nx; i++) {
135                 rho[i] = f1[i] + f2[i];
136                 feq[i] = 0.5 * rho[i];
137                 f1[i] = (1.0 - omega) * f1[i] + omega * feq[i];
138                 f2[i] = (1.0 - omega) * f2[i] + omega * feq[i];
139             }
```

Слика 160: *Прорачун температуре при процесу судара*

Петља за прорачун вредности температуре у процесу струјања (слика 161).

```
140             //Proces strujanja
141             for (i = 1; i <= nx - 1; i++) {
142                 f1[nx - i] = f1[nx - i - 1];
143                 f2[i - 1] = f2[i];
144             }
```

Слика 161: *Прорачун температуре при процесу струјања*

Постављање граничних услова при прорачуну нестационарног преноса топлоте (162).

```
145             //Granicni uslovi
146             f1[0] = f1[1] + f2[1] - f2[0] + q * dx / k;
147             f1[nx] = f1[nx - 1];
148             f2[nx] = f2[nx - 1];
149
150             T[t] = rho[0];
```

Слика 162: *Дефинисање граничних услова*

Дефинисање величине (ширине и висине) поља која се исцртава на екрану при графичком представљану вредности температуре (слика 163).

```
152             //Velicina polja koja se iscrtava
153             int sirina = 20;
154             if (nx < 10) {
                    sirina = sirina;
156             } else if (nx >= 10 || nx < 30) {
157                 sirina = 250 / nx;
158             } else if (nx > 30) {
159                 sirina = 100 / nx;
160             }
```

Слика 163: *Дефинисање величине поља*

Петља у на основу које се врши исцртавање и бојење поља температуре (слика 164).

```
163                 for (i = 0; i < nx; i++) {
164                     if (rho[i] <= 0.025) {
165                         g.setColor(Color.BLUE);
166                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
167                     } else if (rho[i] > 0.025 && rho[i] <= 0.05) {
168                         g.setColor(Color.CYAN);
169                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
170                     } else if (rho[i] > 0.05 && rho[i] <= 0.1) {
171                         g.setColor(Color.GRAY);
172                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
173                     } else if (rho[i] > 0.1 && rho[i] <= 0.2) {
174                         g.setColor(Color.LIGHT_GRAY);
175                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
176                     } else if (rho[i] > 0.2 && rho[i] <= 0.3) {
177                         g.setColor(Color.GREEN);
178                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
179                     } else if (rho[i] > 0.3 && rho[i] <= 0.4) {
180                         g.setColor(Color.YELLOW);
181                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
182                     } else if (rho[i] > 0.4 && rho[i] <= 0.5) {
183                         g.setColor(Color.ORANGE);
184                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
185                     } else if (rho[i] > 0.5 && rho[i] <= 0.6) {
186                         g.setColor(Color.PINK);
187                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
188                     } else if (rho[i] > 0.6 && rho[i] <= 0.7) {
189                         g.setColor(Color.MAGENTA);
190                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
191                     } else if (rho[i] > 0.7 && rho[i] <= 0.8) {
192                         g.setColor(Color.RED);
193                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
194                     } else if (rho[i] > 0.8) {
195                         g.setColor(Color.RED);
196                         g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
197                     }
198                     g.setColor(Color.BLACK);
199                     g.drawRect(i * sirina, t * 2 * sirina, sirina, sirina);
200                 }
```

Слика 164: *Графички приказ поља температуре*

Штампање дужине поља (слика 165) и штампање температуре (слика 166) у конзоли. Штампање температуре у зависности од растојања је исто као на слици 165 само што се уместо *x[i]* пише *rho[i]*, док је штампање температуре у времену исто као на слици 166 где се уместо *vreme[t]* пише *T[t]*.

```
202                 //Stampanje duzine x u konzoli
203                 for (i = 0; i <= nx; i++) {
204                     System.out.print(x[i]);
205                     System.out.println();
206                 }
207                 System.out.println();
208                 System.out.println();
209                 System.out.println();
```

Слика 165: *Штампање дужине поља у конзоли*

```
226                 //Stampanje vremena u konzoli
227                 for (t = 0; t <= nt; t++) {
228                     System.out.print(vreme[t]);
229                     System.out.println();
230                 }
231                 System.out.println();
232                 System.out.println();
233                 System.out.println();
```

Слика 166: *Штампање vremena у конзоли*

Креирање формата за чување добијене вредности температире и растојања (слика 167).

```
221        //Definisanje broja decimalnih mesta
222        DecimalFormat df = new DecimalFormat("0.00000000");
```

Слика 167: *Дефинисање формата*

Чување резултата у текстуалном документу (слика 168).

```
238        //Cuvanje rezultata u tekstualnom dokumentu
239        String Temperature = "Temperature_vreme_rastojanje.txt";
240        try {
               PrintWriter Rezultat = new PrintWriter(Temperature);
242            Rezultat.println("Rastojanja: ");
243            for (i = 0; i <= nx; i++) {
244                Rezultat.print(df.format(x[i]) + "    ");
245            }
246            Rezultat.println("");
247            Rezultat.println("");
248
249            Rezultat.println("Temperatura i rastojanje: ");
250            for (i = 0; i <= nx; i++) {
251                Rezultat.print(df.format(rho[i]) + "    ");
252            }
253            Rezultat.println("");
254            Rezultat.println("");
255
256            Rezultat.println("Temperatura i vreme: ");
257            for (t = 1; t <= nt; t++) {
258                Rezultat.print(df.format(T[t]) + "    ");
259            }
260            Rezultat.println("");
261            Rezultat.println("");
262
263            Rezultat.println("Vreme: ");
264            for (t = 1; t <= nt; t++) {
265                Rezultat.print(df.format(vreme[t]) + "    ");
266            }
267
268            Rezultat.close();
269        } catch (FileNotFoundException ex) {
270        }
```

Слика 168: *Чување резултата у текстуалном документу*

Креирање прозора у коме се графички представљају резултати прорачуна температуре (слика 169).

```
244    /**
245     * @param args the command line arguments
246     */
247    public static void main(String[] args) {
248        //Kreiranje prozora i graficki prikaz rezultata
249        JFrame window = new JFrame();
250        window.setTitle("Prenos toplote");
251        window.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
252        window.setBounds(10, 10, 300, 600);
253        window.getContentPane().add(new NestacionarniPrenosToplote());
254        window.setVisible(true);
255    }
256 }
```

Слика 169: *Креирање прозора*

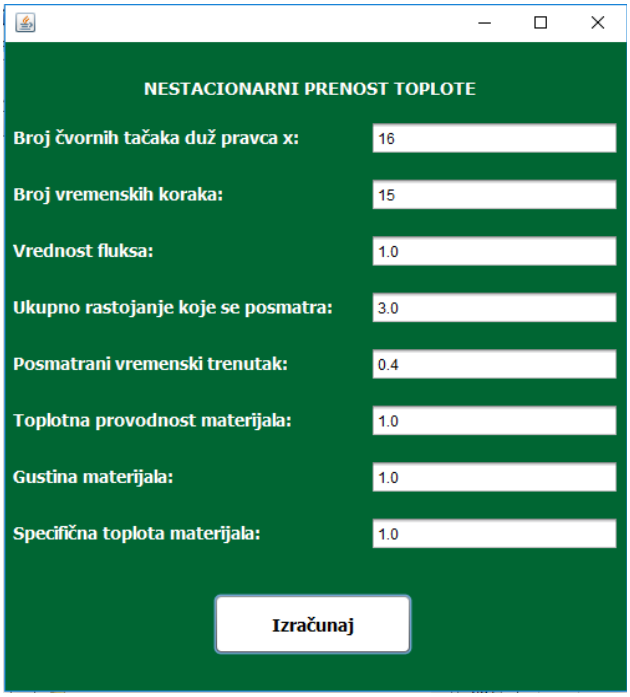
Детаљна објашњења сваке линије кода су иста као и у програму за стационарни пренос топлоте и нису поново наведена. Комплетани кодови за обе класе дати су у прилогу 7 и 8.

5.2.9. Пример примене LBM у раду друге верзије програма

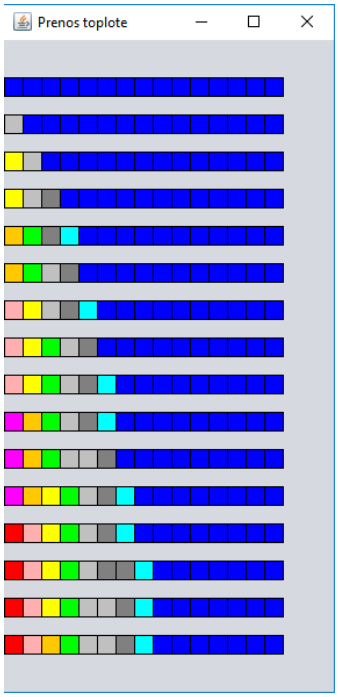
У овом раду приказан је пример примене Lattice Boltzmann-овог Метода за нестационарни пренос топлоте на једнодимензионалном штапу са следећим познатим подацима: број чворних тачака износи 15, број временских корака је 16, вредност флука износи $1\frac{W}{m^2}$, укупно посматрано растојање (мисли се на дужину штапа која се посматра у случају када имамо бесконачно дуги штап) је 3.0m, временски тренутаци који се посматрају су 0.4s, 0.2s, и 0.1s, дужина временског тренутка односно трајање једног временског корака је 0.025s, карактеристике материјала су следеће: топлотна проводност износи $1\frac{W}{m^oC}$, густина је $1\frac{kg}{m^3}$ и специфична топлота износи $1\frac{J}{kg^oC}$.

Како је у питању бесконачно дуг проводник посматра се само један његов део. Почетни услови су следећи: штап нема почетну температуру, односно његова почетна температура је 0 °C, на почетку штапа (са леве стране) делује флукс, тако се штап загрева под дејством топлоте која настаје од флукса, односно под дејством топлотног флукса.

На сликама 170, 172 и 174 приказани су изгледи програма са унетим подацима, а на сликама 171, 173 и 175 дати су графички прикази нестационарног преноса топлоте по итерацијама за посматране временске тренутке редом t=0.4s, t=0.2s и t=0.1s.

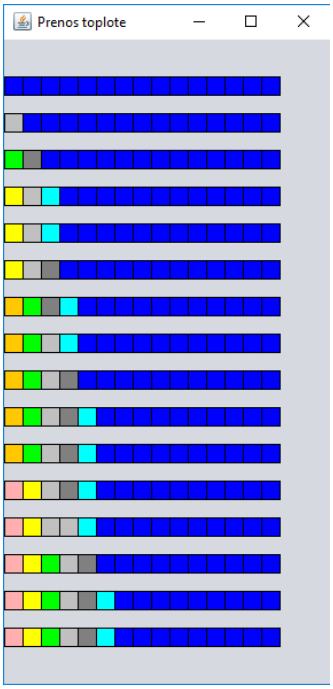


Слика 170: *Почетни прозор програма за нестационарни пренос топлоте*

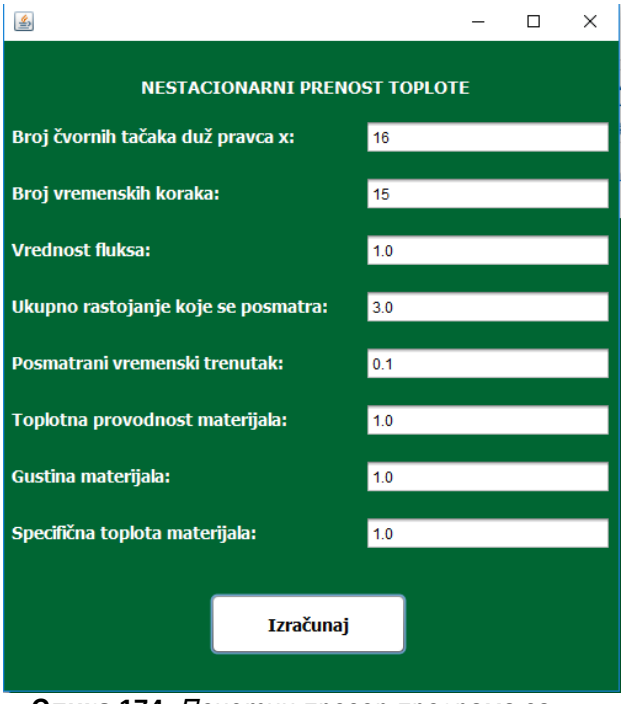


Слика 171: *Графички приказ нестационарног преноса топлоте за t=0.4s*

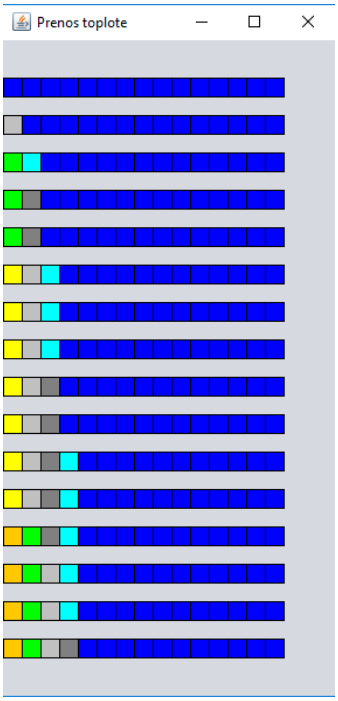
Слика 172: *Почетни прозор програма за нестационарни пренос топлоте*



Слика 173: *Графички приказ нестационарног преноса топлоте за t=0.2s*

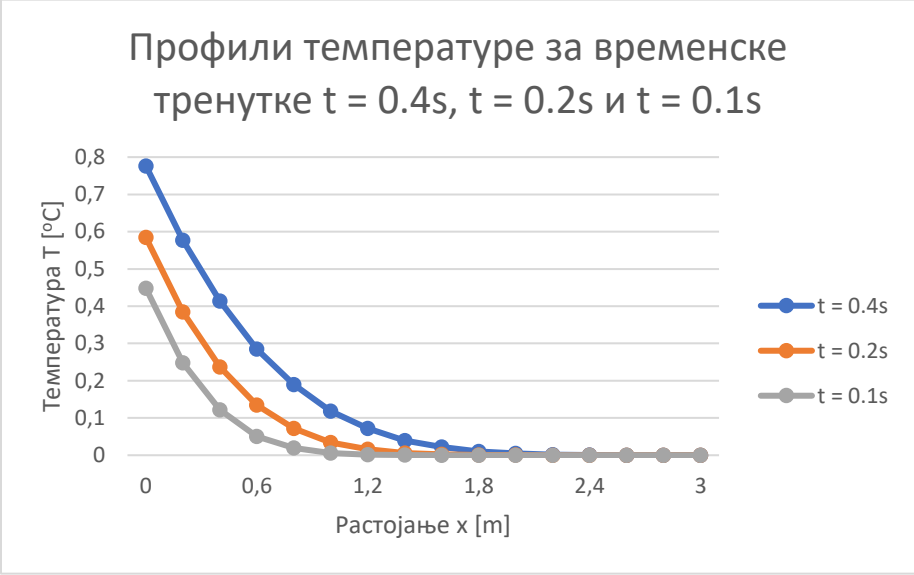


Слика 174: *Почетни прозор програма за нестационарни пренос топлоте*



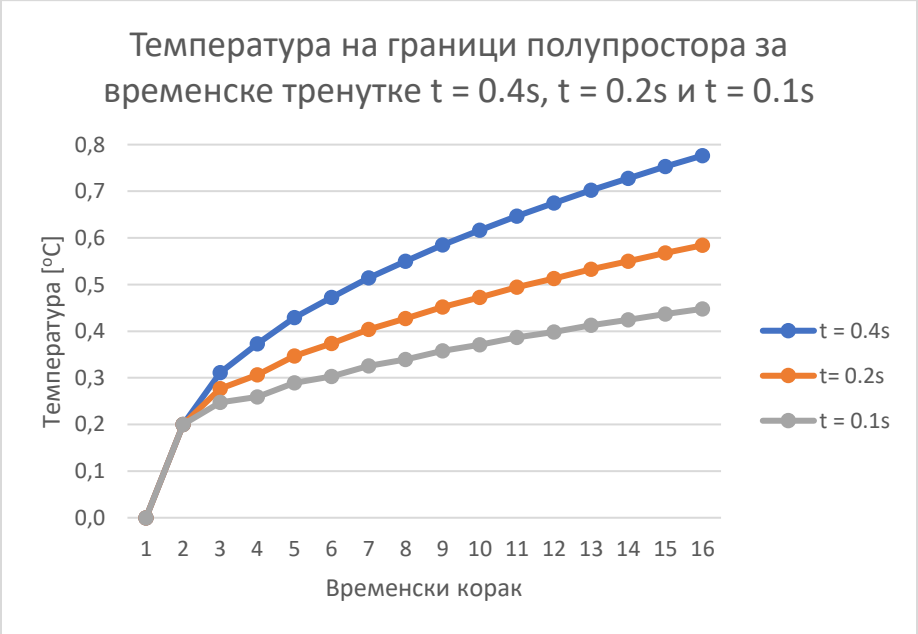
Слика 175: *Графички приказ нестационарног преноса топлоте за t=0.2s*

Како се шири топлота са порастом растојања за различите временске тренутке приказано је на слици 176. Са графика се може видети да са дужим временским тренутком температура са порастом растојања спорије опада, док у случају када је краћи временски тренутак, у коме се посматра, температура брже опада са порастом растојања.



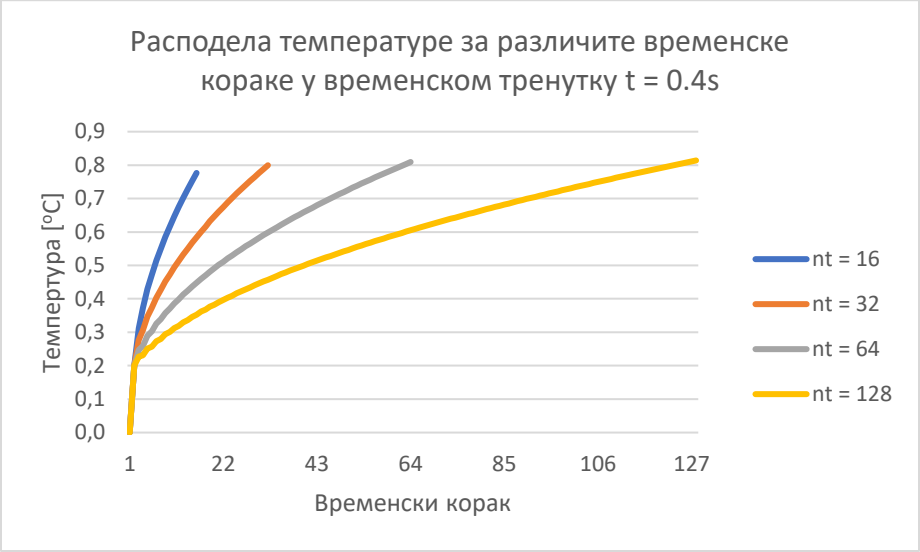
Слика 176: *Профили температура за различите временске тренутке*

За исте временске тренутке приказан је и дијаграм температура на граници полупростора (слика 177). Посматрајући дијаграм долази се до закључка да за дужи временски период температура на граници полупростора има већу вредност у односу на исту посматрану границу полупростора за краће временске тренутке. У пракси би то значило да што дуже загревамо крај штапа, то се температура на граници полупростора са поратом времена загревања све више повећава.



Слика 177: *Температуре на граници полупростора за различите временске тренутке*

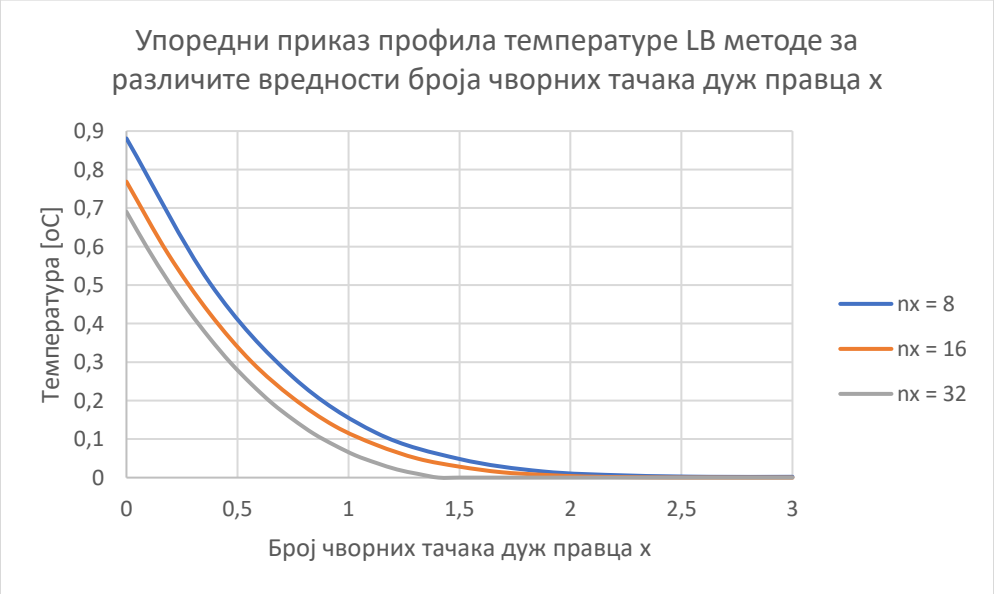
Помоћу претходних дијаграма објашњено је шта се дешава са температуром када се мења временски корак. Следећим дијаграмом (слика 178) приказана је расподела температуре за различите временске кораке у временском тренутку t = 0.4s.



Слика 178: *Расподела температуре за различите временске кораке*

Посматрајући слику 178 закључује се да при мањем броју временских корака температура брже расте, док при већем броју временских корака и истом временском тренутку та вредност спорије расте. Потребно је истаћи да се са повећањем броја временских корака повећава и тачност расподеле температуре.

На следећем графику (слика 179) приказани су профили температура када се мења број чворних тачака дуж правца x. За добијање ових резултата коришћени су идентични параметри као и у поставци са слике 170, параметару који се мења (nx) додељују се следеће вредност: nx = 8, nx = 16 и nx = 32. За додељене вредности параметра nx са графика се види да са порастом броја чворних тачака температура у временском тренутку t = 0.4s брже опада, односно брже тежи коначној расподели.

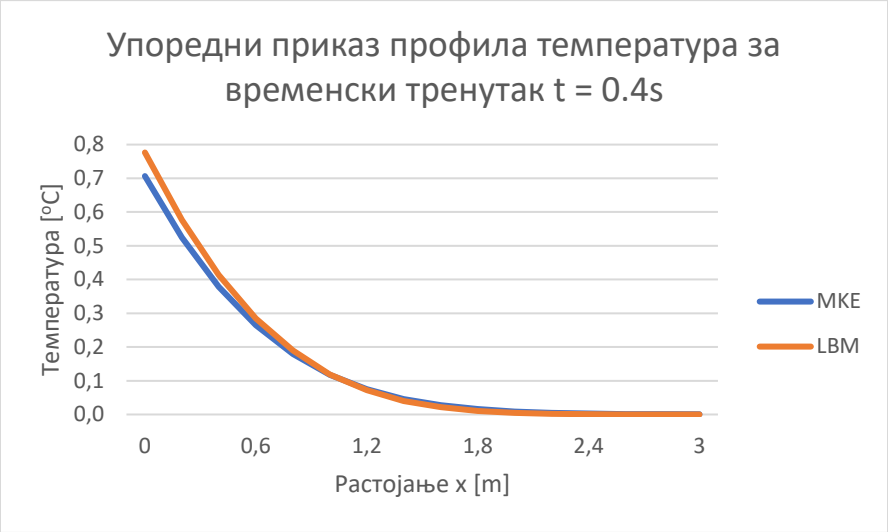


Слика 179: *Профил температуре за различите вредности чворних тачака дуж правца x*

5.2.10. Упоређивање резултата LBM и МКЕ

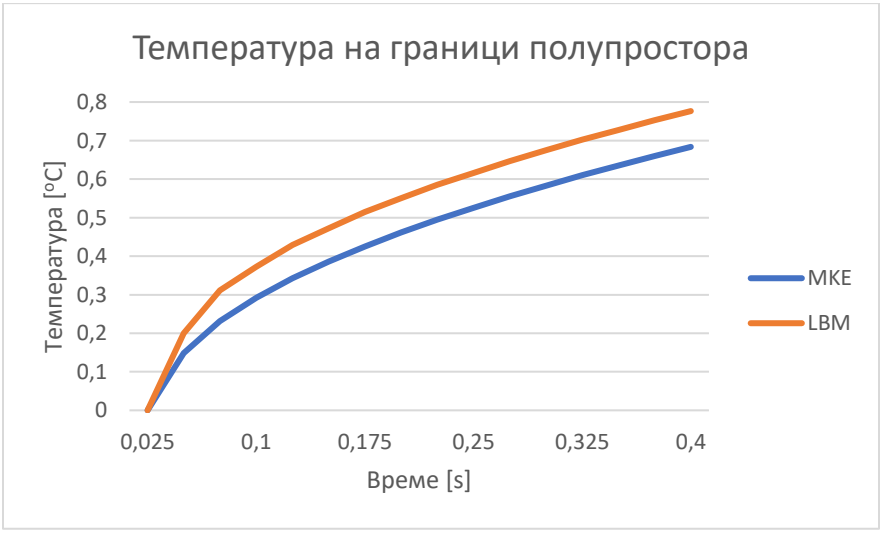
У овом поглављу упоређени су резултати добијени Lattice Boltzmann-овом методом са резултатима добијеним методом коначних елемената. Коришћени су идентични параметри као у претходном поглављу за анализу временског тренутка t = 0.4s.

На слици 180 дат је дијаграм упоредног приказа профила температуре у LBM и МКЕ за временски тренутак t = 0.4s. Са дијаграма се види да постоји да LB метод при краћем растојању даје већу вредност температуре него код МКЕ, док са порастом растојања температуре ових метода се изједначавају.



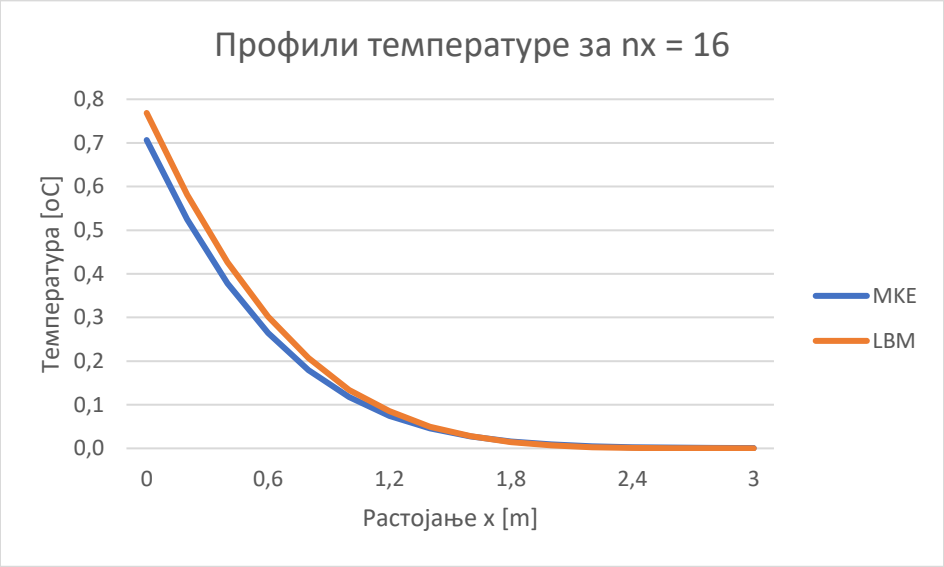
Слика 180: *Упоредни приказ профила температуре за временски тренутак t = 0.4s*

На слици 181 дат је дијаграм упоредног приказа температура на граници полупростора у LBM и МКЕ, односно зависност расподеле температуре од времена на граници полупростора. Као што се може приметити температура на граници полупростора код LB методе достиже већу вредност од МКЕ.



Слика 181: Упоредни приказ т*емпература на граници полупростора*

На слици 182 дат је дијаграм упоредног приказа расподеле температура по дужини када је број чворних тачака дуж правца x једнак 16 за обе претходно поменуте методе. Са дијаграма се може видети да са порастом растојања температуре ових метода достижу приближно исту вредност, а да при мањем растојању LBM има већу температуру.



Слика 182: *Упоредни приказ профили температура за број чворних тачака nx = 16*

Упоређивањем добијених резултата Lattice Boltzmann-овом методом и методом коначних елемената, добијено је да грешка која настаје при прорачуну расподеле температура нестационарног преноса топлоте износи 2,7%.

Закључак

Креирана софтверска решења нумеричке методе за прорачун расподеле температуре стационарног и нестационарног провођења топлоте веома су једноставна. Променом различитих параметара могу се пратити промене које се јављају при стационарном и нестационарном преносу топлоте. Софтвери нису захтевни, лако се користе и није их потребно инсталирати на рачунару. Сваки од ових софтвера ради на било којој Windows платформи, који поседује JAVA-у. Потребно је напоменути да је за рад ових апликација неопходно поседовање JDK 8.

При коришћењу Cellular Automata и/или Lattice Boltzmann-ове методе за прорачун расподеле температуре у простору и/или времену, појављује се нумеричка грешка која се разликује за случај стационарног и нестационарног процеса преноса топлоте. Ови процеси се суштински разликују у томе што код стационарног процеса пренос топлоте не зависи од времена. Стационарни пренос топлоте са повећањем броја итерација постиже коначну вредност расподеле температуре у времену. Што се нестационарног преноса топлоте тиче, расподела температуре у овом процесу јесте функција од времена. За разлику од стационарног процеса код нестационарног преноса топлоте важну улогу имају и параметри који фигуришу у прорачуну. Разлог настанка грешке јесте нумеричко упрошћавање сложених обичних и парцијалних диференцијалних једначина које се решавају применом методе коначних елемената.

Сва софтверска решења могу се побољшати додавањем већег броја опција кориснику, бољим графичким приказом резултата, процес израчунавања се може убрзати ако се уклоне из програма сви делови кода који се односе на штампање резултата у конзоли, код нестационарног прорачуна може се задати већи број параметара, док се код оба процеса могу применити и друге врсте проблема, пре свега имплементација вишедимензионалних проблема расподеле температуре 2D и 3D.

Литература

[1] https://www.tehnikum.edu.rs/predmeti/0003/Predavanje13.pdf, приступљено: 22.09.2018.

[2] Термодинамика – Једначине стања, скрипта са предавања из предмета Термодинамика, Факултет инжењерских наука Универзитета у Крагујевцу, 2016.

[3] http://www.vpps.edu.rs/Literatura/Tehnoloske%20operacije/TO_Predavanje_06_Prenos_toplote.pdf, приступљено: 22.09.2018.

[4] http://www.df.uns.ac.rs/files/200/kristina_fodor_-_diplomski_rad_(d-601).pdf, 22.09.2018.

[5] https://slideplayer.com/slide/11545080/#, приступљено: 22.09.2018.

[6] https://www.scribd.com/presentation/157071602/2-PROSTIRANJE-TOPLOTE, приступљено: 22.09.2018.

[7] https://www.gsl.cz/services-products/assessment/lab-services/glass-properties-measurement/gp-measurements-heat-transfer/, приступљено: 23.09.2018.

[8] http://tf.uns.ac.rs/~omorr/radovan_omorjan_003_mmtp/Brzina%20prenosa%20toplote%20i%20mase%20i%20hem_%20reakcije.pdf, приступљено: 23.09.2018.

[9] https://svejefizika.files.wordpress.com/2016/05/tp3.jpg, приступљено: 23.09.2018.

[10] http://www.tf.uns.ac.rs/~omorr/radovan_omorjan_003_mpi/MMPI/4_%20Nestacionaran%20prenos%20toplote%20kondukcijom.pdf, приступљено: 23.09.2018.

[11] https://www.scribd.com/document/135203063/prenos, приступљено: 27.09.2018.

[12] Karli Watson i dr. Od početka C#, CET Computer Equipment and trade Beograd, 2002.

[13] https://java.com/en/download/faq/whatis_java.xml, приступљено: 27.09.2018.

[14] https://netbeans.org/features/platform/index.html, приступљено, приступљено: 27.09.2018.

[15] http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html, приступљено: 27.09.2018.

[16] Burzynski M., Cudny W., Kosinski W., Cellular Automata: Structures and some applications, Journal of theoretical and applied mechanics, 42, 3, pp. 461-482, Warsaw 2004.

[17] https://www.google.rs/imgres?imgurl=https%3A%2F%2Fascelibrary.org%2Fcms%2Fattachment%2F8b40f958-dd19-45fd-aec7-dadac1512017%2F2.jpg&imgrefurl=https%3A%2F%2Fascelibrary.org%2Fdoi%2Fabs%2F10.1061%2F(ASCE)ST.1943-541X.0000307&docid=c45J0c-kFCPg2M&tbnid=9S-df1f2tIyKKM%3A&vet=10ahUKEwjUlp2x_u_dAhVj_CoKHZMcB8gQMwh2KDMwMw..i&w=846&h=435&bih=758&biw=1600&q=cellular%20automata%20heat%20&ved=0ahUKEwjUlp2x_u_dAhVj_CoKHZMcB8gQMwh2KDMwMw&iact=mrc&uact=8, 05.10.2018.

[18] https://www.researchgate.net/figure/Cellular-automata-model-description-with-possible-daughterstate-positions-The-distance_fig1_262787426, 05.10.2018.

[19] https://www.semanticscholar.org/paper/Cellular-Automata-Based-Simulation-for-Smoke-and-in-Curiac-Banias/bc75da3cd14682ef78ea683e7cd8608141cccf56/figure/0, 05.10.2018.

[20] https://www2.informatik.uni-erlangen.de/teaching/thesis/download/i2B00401.pdf, приступљено: 30.09.2018.

[21] Mohamad A.A., Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Codes, Springer, London, Dordrecht, Heidelberg, New York 2011.

[22] https://bib.irb.hr/datoteka/519123.Animacija_modela_vodenih_povrsina.pdf, приступљено: 06.10.2018.

[23] https://www.dmi.uns.ac.rs/site/dmi/download/master/primenjena_matematika/MilanaPavic.pdf, приступљено: 03.10.2018.

[24] Lattice- Boltzmann-ov metod – материјал достављен од стране проф. др Ненада Грујовића

[25] Subhash C. Mishra, Bittagopal Mondal, Tanuj Kush, B. Siva Rama Krishna, Soloving transient heat conduction problems on uniform and non-uniform lattice using the latice Boltzmann method, Internacional Communications in Heat and Mass Transfer 36 (2009) 322-328.

[26] Grażyna Kałuża, The numerical solution of the transient heat conduction problem using the lattice Boltzmann method, Scientific Research of the Institute of Mathematics and Computer Science, 2012, Volume 11, Issue 1, pages 23-30.

[27] https://brage.bibsys.no/xmlui/bitstream/handle/11250/238308/566322_FULLTEXT01.pdf?sequence=3, приступљено: 06.10.2018.

[28] https://stackoverflow.com/questions/5061912/printing-out-a-2-d-array-in-matrix-format, 28.01.2018.

[29] https://www.youtube.com/watch?v=WEZRc0GoP3E, 28.01.2018.

[30] http://www.otherwise.com/Lessons/ColorsInJava.html, 28.01.2018.

[31] https://www.ntu.edu.sg/home/ehchua/programming/java/J4b_CustomGraphics.html, 28.01.2018

[32] https://mathbits.com/MathBits/Java/Graphics/GraphingFill.htm, 28.01.2018

[33] http://www.java2s.com/Tutorial/Java/0261__2D-Graphics/DrawRectangle.htm, 28.01.2018.

[34] Miloš Kojić i dr. Metod konačnih elemenata I, Univerzitet u Kragujevcu Mašinski fakultet, Kragujevac, 2010.

Прилог 1 – *Glavni_prozor.java*

```
package GUI;

import prenostoplote.PrenosToplote;

public class Glavni_prozor extends javax.swing.JFrame {

    public Glavni_prozor() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jComboBox1 = new javax.swing.JComboBox<>();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        jButton1.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jButton1.setText("Pokreni simulaciju");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jLabel1.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel1.setText("Velicina ploce (b):");

        jLabel2.setText("b x b");

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "10", "50", "100" }));

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jButton1,  javax.swing.GroupLayout.Alignment.TRAILING,  javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel1)
                        .addGap(18, 18, 18)
                        .addComponent(jComboBox1,                                    javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(6, 6, 6)
                        .addComponent(jLabel2)
                        .addGap(0, 171, Short.MAX_VALUE)))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addGap(29, 29, 29)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(jLabel2)
```

```
                .addComponent(jComboBox1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(22, 22, 22)
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 55,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(26, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        int vrednost_mreze = Integer.parseInt(jComboBox1.getSelectedItem().toString());

        prenostoplote.PrenosToplote p = new PrenosToplote();
        p.main(null);
        p.setBroj_kvadratica(vrednost_mreze);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Glavni_prozor().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    // End of variables declaration
}
```

Прилог 2 – *PrenosToplote.java*

```
package prenostoplote;

import java.awt.Color;
```

```
import java.awt.Graphics;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import javax.swing.JComponent;
import javax.swing.JFrame;

public class PrenosToplote extends JComponent{
    public static int broj_kvadratica = 0;
    int MAX_PROLAZ = 1000;
    int i, j, k;
    double T, GORE, DOLE, LEVO, DESNO;
    double GORE_LEVO, GORE_DESNO, DOLE_LEVO, DOLE_DESNO;
    double[][] oldT = new double[100][100];
    double[][] newT = new double[100][100];

    //Metodi za bojenje kvadratica
    private static void setColor(Color color) {}
    private static void fillRect(double d) {}

    //Metodi za pristup GUI
    public int getBroj_kvadratica() {
        return broj_kvadratica;
    }
     public void setBroj_kvadratica(int broj_kvadratica) {
        this.broj_kvadratica = broj_kvadratica;
    }

    //Crtanje kvadratica
    @Override
    public void paint(Graphics g) {
        int sirina = 500/broj_kvadratica;

        //Proracun
        //Inicijalizacija temperature u svakom polju
        //Gornja granicna temperatura
        for(j=0; j<broj_kvadratica; j++){
            oldT[0][j]=100;
        }
        //Leva granicna temperatura
        for(i=0; i<broj_kvadratica; i++){
            oldT[i][0]=100;
        }
        //Desna granicna temperatura
        for(i=0; i<broj_kvadratica; i++){
            oldT[i][broj_kvadratica-1]=100;
        }
         //Izracunavanje temperature po broju prolaza
        for(k=0; k<MAX_PROLAZ; k++){
            //Proracun temperature
            for(i=1; i<broj_kvadratica-1; i++){
                for(j=1; j<broj_kvadratica-1; j++){
                    GORE = oldT[i-1][j];
                    DOLE = oldT[i+1][j];
                    LEVO = oldT[i][j-1];
                    DESNO = oldT[i][j+1];
                    GORE_LEVO = oldT[i-1][j-1];
                    GORE_DESNO = oldT[i-1][j+1];
                    DOLE_LEVO = oldT[i+1][j-1];
                    DOLE_DESNO = oldT[i+1][j+1];
                    T=(4*(GORE+DOLE+LEVO+DESNO)+GORE_LEVO+GORE_DESNO+DOLE_LEVO+DOLE_DESNO)/20;
                    newT[i][j]=T;
                    //Cuvanje rezultata po broju prolaza
                    oldT[i][j]=newT[i][j];
                    //Stampanje rezultata po broju prolaza
                    System.out.println(oldT[i][j]);
                }
            }
            //Cuvanje rezultata u tekstualnom dokumentu
            String Temperature = "Temperature.txt";
            try {
```

```
            PrintWriter Rezultat = new PrintWriter(Temperature);
            for(i=0; i<broj_kvadratica; i++){
               for(j=0; j<broj_kvadratica; j++){
                     Rezultat.print(oldT[i][j]);
                     Rezultat.print(" ");
                     Rezultat.print(" ");
               }
               Rezultat.println("");
               Rezultat.println("");
            }
            Rezultat.close();
         }
         catch (FileNotFoundException ex) {
         }
         System.out.println("");
         System.out.println("Prolaz k = " + k);
         System.out.println("");
      }
      //Bojenje kvadratica
      for(i=0; i<broj_kvadratica; i++){
         for(j=0; j<broj_kvadratica; j++){
            if(oldT[i][j]<=10){
               g.setColor(Color.BLUE);
               g.fillRect(j*sirina, i*sirina, sirina*sirina, sirina);
            }
            else if(oldT[i][j]>10 && oldT[i][j]<=20){
               g.setColor(Color.CYAN);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>20 && oldT[i][j]<=30){
               g.setColor(Color.GRAY);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>30 && oldT[i][j]<=40){
               g.setColor(Color.LIGHT_GRAY);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>40 && oldT[i][j]<=50){
               g.setColor(Color.GREEN);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>50 && oldT[i][j]<=60){
               g.setColor(Color.YELLOW);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>60 && oldT[i][j]<=70){
               g.setColor(Color.ORANGE);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>70 && oldT[i][j]<=80){
               g.setColor(Color.PINK);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>80 && oldT[i][j]<=90){
               g.setColor(Color.MAGENTA);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>90 && oldT[i][j]<=100){
               g.setColor(Color.RED);
               g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            g.setColor(Color.BLACK);
            g.drawRect(j*sirina, i*sirina, sirina, sirina);
         }
      }
   }

   public static void main(String[] args) {
      //Kreiranje prozora i ploce
      JFrame window = new JFrame();
```

```
        window.setTitle("Prenos toplote");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setBounds(30, 30, 516, 539);
        window.getContentPane().add(new PrenosToplote());
        window.setVisible(true);
    }
}
```

Прилог 3 – *Glavni_prozor.java*

```
package GUI;

import prenostoplote.PrenosToplote;

public class Glavni_prozor extends javax.swing.JFrame {

    public Glavni_prozor() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jComboBox1 = new javax.swing.JComboBox<>();
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jTextField5 = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        jButton1.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jButton1.setText("Pokreni simulaciju");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "11", "51", "101" }));

        jLabel1.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel1.setText("Gustina kvadratne mreze:");

        jLabel2.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel2.setText("Broj iteracija:");

        jLabel3.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel3.setText("Gornja granicna vrednost:");

        jLabel4.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel4.setText("Donja granicna vrednost:");

        jLabel5.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
```

```
        jLabel5.setText("Leva granicna vrednost:");

        jLabel6.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel6.setText("Desna granicna vrednost:");

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel2)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jTextField1,                javax.swing.GroupLayout.PREFERRED_SIZE,                71,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jButton1,                javax.swing.GroupLayout.PREFERRED_SIZE,                224,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(0, 0, Short.MAX_VALUE))
                    .addGroup(layout.createSequentialGroup()
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel1)
                            .addComponent(jLabel3)
                            .addComponent(jLabel4)
                            .addComponent(jLabel5)
                            .addComponent(jLabel6))
                        .addGap(18, 18, 18)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jComboBox1,                javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addGap(0, 0, Short.MAX_VALUE))
                            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                                .addGap(0, 1, Short.MAX_VALUE)
                                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jTextField2,                javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 71, javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField3,                javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 71, javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField4,                javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 71, javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField5,                javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 71, javax.swing.GroupLayout.PREFERRED_SIZE))))))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(jComboBox1,                javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel4)
```

```
                .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel5)
                .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel6)
                .addComponent(jTextField5, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(jButton1,                javax.swing.GroupLayout.PREFERRED_SIZE,                55,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        int vrednost_mreze = Integer.parseInt(jComboBox1.getSelectedItem().toString());
        int broj_iteracija = Integer.parseInt(jTextField1.getText());
        int gornja_granica = Integer.parseInt(jTextField2.getText());
        int donja_granica = Integer.parseInt(jTextField3.getText());
        int leva_granica = Integer.parseInt(jTextField4.getText());
        int desna_granica = Integer.parseInt(jTextField5.getText());

        prenostoplote.PrenosToplote p = new PrenosToplote();
        p.main(null);
        p.setBroj_kvadratica(vrednost_mreze);
        p.setBroj_iteracija(broj_iteracija);
        p.setGornja(gornja_granica);
        p.setDonja(donja_granica);
        p.setLeva(leva_granica);
        p.setDesna(desna_granica);

    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(Glavni_prozor.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
```

```
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Glavni_prozor().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    // End of variables declaration
}
```

Прилог 4 – *PrenosToplote.java*

```
package prenostoplote;

import java.awt.Color;
import java.awt.Graphics;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import javax.swing.JComponent;
import javax.swing.JFrame;

public class PrenosToplote extends JComponent{
    public static int broj_kvadratica = 0;
    public static int MAX_PROLAZ = 1;
    int i, j, k;
    double T, GORE, DOLE, LEVO, DESNO;
    double GORE_LEVO, GORE_DESNO, DOLE_LEVO, DOLE_DESNO;
    double[][] oldT = new double[101][101];
    double[][] newT = new double[101][101];
    public static int GO = 0, DO = 0, LE = 0, DE = 0;
    double[] CT = new double[1000001];

    //Metodi za bojenje kvadratica
    private static void setColor(Color color) {}
    private static void fillRect(double d) {}

    //Metodi za odredjivanje gustine mreze
    public int getBroj_kvadratica() {
        return broj_kvadratica;
    }
     public void setBroj_kvadratica(int broj_kvadratica) {
        this.broj_kvadratica = broj_kvadratica;
    }
    //Metodi za odredjivanje broja iteracija
    public int getBroj_iteracija() {
        return MAX_PROLAZ;
    }
    public void setBroj_iteracija(int broj_iteracija) {
        this.MAX_PROLAZ = broj_iteracija;
    }
    //Metodi za odredjivanje gornje granicne vrednosti
    public int getGornja() {
        return GO;
    }
```

```
public void setGornja(int gore) {
    this.GO = gore;
}
//Metodi za odredjivanje donje granicne vrednosti
public int getDonja() {
    return DO;
}
public void setDonja(int dole) {
    this.DO = dole;
}
//Metodi za odredjivanje leve granicne vrednosti
public int getLeva() {
    return LE;
}
public void setLeva(int levo) {
    this.LE = levo;
}
//Metodi za odredjivanje desne granicne vrednosti
public int getDesna() {
    return DE;
}
public void setDesna(int desno) {
    this.DE = desno;
}

//Crtanje kvadratica
@Override
public void paint(Graphics g) {
    int sirina = 0;
    if(broj_kvadratica == 11){
        sirina = 515/broj_kvadratica;
    }
    else if(broj_kvadratica == 51 ){
        sirina = 511/broj_kvadratica;
    }
    else if(broj_kvadratica == 101){
        sirina = 505/broj_kvadratica;
    }


    //Proracun
    //Inicijalizacija temperature u svakom polju
    //Gornja granicna temperatura
    for(j=0; j<broj_kvadratica; j++){
        oldT[0][j] = GO;
    }
    //Donja granicna temperatura
    for(j=0; j<broj_kvadratica; j++){
        oldT[broj_kvadratica-1][j] = DO;
    }
    //Leva granicna temperatura
    for(i=1; i<broj_kvadratica-1; i++){
        oldT[i][0] = LE;
    }
    //Desna granicna temperatura
    for(i=1; i<broj_kvadratica-1; i++){
        oldT[i][broj_kvadratica-1] = DE;
    }
     //Izracunavanje temperature po broju prolaza
    for(k=0; k<MAX_PROLAZ; k++){
        //Proracun temperature
        for(i=1; i<broj_kvadratica-1; i++){
            for(j=1; j<broj_kvadratica-1; j++){
                GORE = oldT[i-1][j];
                DOLE = oldT[i+1][j];
                LEVO = oldT[i][j-1];
                DESNO = oldT[i][j+1];
                GORE_LEVO = oldT[i-1][j-1];
                GORE_DESNO = oldT[i-1][j+1];
                DOLE_LEVO = oldT[i+1][j-1];
```

```
            DOLE_DESNO = oldT[i+1][j+1];
            T=(4*(GORE+DOLE+LEVO+DESNO)+GORE_LEVO+GORE_DESNO+DOLE_LEVO+DOLE_DESNO)/20;
            newT[i][j]=T;
            //Cuvanje rezultata po broju prolaza
            oldT[i][j]=newT[i][j];
            //Stampanje rezultata po broju prolaza
            //System.out.println(oldT[i][j]);
        }
    }
    //Cuvanje rezultata u tekstualnom dokumentu
    String Temperature = "Temperature.txt";
    try {
        PrintWriter Rezultat = new PrintWriter(Temperature);
        for(i=0; i<broj_kvadratica; i++){
            for(j=0; j<broj_kvadratica; j++){
                Rezultat.print(oldT[i][j]);
                Rezultat.print(" ");
                Rezultat.print(" ");
            }
            Rezultat.println("");
            Rezultat.println("");
        }
        Rezultat.close();
    }
    catch (FileNotFoundException ex) {
    }
    System.out.println("");
    System.out.println("Prolaz k = " + k);
    //System.out.println("Temperatura u centru ploce T = " + oldT[broj_kvadratica/2][broj_kvadratica/2]);
    //System.out.println("");

    CT[k] = oldT[broj_kvadratica/2][broj_kvadratica/2];
    System.out.println("Centar = "  + CT[k]);
}
//Bojenje kvadratica
for(i=0; i<broj_kvadratica; i++){
    for(j=0; j<broj_kvadratica; j++){
        if(oldT[i][j]<=10){
            g.setColor(Color.BLUE);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>10 && oldT[i][j]<=20){
            g.setColor(Color.CYAN);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>20 && oldT[i][j]<=30){
            g.setColor(Color.GRAY);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>30 && oldT[i][j]<=40){
            g.setColor(Color.LIGHT_GRAY);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>40 && oldT[i][j]<=50){
            g.setColor(Color.GREEN);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>50 && oldT[i][j]<=60){
            g.setColor(Color.YELLOW);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>60 && oldT[i][j]<=70){
            g.setColor(Color.ORANGE);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>70 && oldT[i][j]<=80){
            g.setColor(Color.PINK);
            g.fillRect(j*sirina, i*sirina, sirina, sirina);
        }
        else if(oldT[i][j]>80 && oldT[i][j]<=90){
```

```
                g.setColor(Color.MAGENTA);
                g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>90 && oldT[i][j]<=100){
                g.setColor(Color.RED);
                g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            else if(oldT[i][j]>100){
                g.setColor(Color.RED);
                g.fillRect(j*sirina, i*sirina, sirina, sirina);
            }
            g.setColor(Color.BLACK);
            g.drawRect(j*sirina, i*sirina, sirina, sirina);
        }
      }
      //Cuvanje Centralne tacke
      String Tacka = "Centralna_tacka.txt";
         try {
            PrintWriter Rezultat = new PrintWriter(Tacka);
            for(k=0; k<MAX_PROLAZ; k++){

            Rezultat.println(CT[k]);
            }
            Rezultat.close();
         }
         catch (FileNotFoundException ex) {
         }
 }

   public static void main(String[] args) {
      //Kreiranje prozora i ploce
      JFrame window = new JFrame();
      window.setTitle("Prenos toplote");
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setBounds(30, 30, 527, 550);
      window.getContentPane().add(new PrenosToplote());
      window.setVisible(true);
   }
}
```

Прилог 5 – *NestacionarniPrenosToplote.java*

```
package nestacionarniprenostoplote;

import java.awt.Color;
import java.awt.Graphics;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import javax.swing.JComponent;
import javax.swing.JFrame;

public class NestacionarniPrenosToplote extends JComponent {

   public static int nx = 0;
   public static int nt = 0;

   public int getNx() {
      return nx;
   }

   public void setNx(int nx) {
      this.nx = nx;
   }

   public int getNt() {
      return nt;
   }
```

```
public void setNt(int nt) {
    this.nt = nt;
}

//Kreiranje grafike
public void paint(Graphics g) {

    //Inicijalizacija promenljivih
    int x, t, i;
    double lambda = 0.5;
    double[][][] f = new double[100][2][100];
    double[][] T = new double[100][100];
    double[][] feq = new double[2][100];
    DecimalFormat df = new DecimalFormat("0.0000");

    f[0][0][0] = 0.5;
    f[0][1][0] = 0.5;

    for (x = 1; x <= nx; x++) {
        f[0][0][x] = 1.5;
        f[0][1][x] = 1.5;
    }

    for (t = 0; t <= nt; t++) {
        for (x = 0; x <= nx; x++) {
            //Temperatura pre sudara
            T[x][t] = f[t][0][x] + f[t][1][x];
            for (i = 0; i < 2; i++) {
                feq[i][x] = 0.5 * T[x][t];
                f[t][i][x] = f[t][i][x] + lambda * (feq[i][x] - f[t][i][x]);
            }
            //Temperatura posle sudara
            T[x][t] = f[t][0][x] + f[t][1][x];
        }

        //Propagacija
        for (x = 1; x <= nx; x++) {
            f[t + 1][0][x - 1] = f[t][0][x];
        }

        for (x = 0; x <= nx - 1; x++) {
            f[t + 1][1][x + 1] = f[t][1][x];
        }

        //Granicni uslovi
        f[t + 1][0][0] = 0.5;
        f[t + 1][1][0] = 0.5;
        f[t + 1][0][nx] = 1.5;
        f[t + 1][1][nx] = 1.5;
    }

    System.out.println("MATRICA T");
    for (t = 0; t < nt; t++) {
        for (x = 0; x < nx; x++) {
            System.out.print(df.format(T[x][t]) + "  " + "  ");
            //System.out.print(T[t][x] + "  ");
        }
        System.out.println("");
    }

    //Cuvanje rezultata u tekstualnom dokumentu
    String Temperature = "Temperature.txt";
    try {
        PrintWriter Rezultat = new PrintWriter(Temperature);
        for (t = 0; t < nt; t++) {
            for (x = 0; x < nx; x++) {
                Rezultat.print(df.format(T[x][t]) + "  " + "  ");
            }
            Rezultat.println("");
```

```
            Rezultat.println("");
        }
        Rezultat.close();
    } catch (FileNotFoundException ex) {
    }

    int sirina = 20;
    if (nx < 10) {
        sirina = sirina;
    } else if (nx > 20) {
        sirina = 500 / nx;
    }

    for (t = 0; t < nt; t++) {
        for (x = 0; x < nx; x++) {
            if (T[x][t] <= 0.75) {
                g.setColor(Color.BLUE);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 0.75 && T[x][t] <= 1) {
                g.setColor(Color.CYAN);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 1 && T[x][t] <= 1.25) {
                g.setColor(Color.GRAY);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 1.25 && T[x][t] <= 1.5) {
                g.setColor(Color.LIGHT_GRAY);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 1.5 && T[x][t] <= 1.75) {
                g.setColor(Color.GREEN);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 1.75 && T[x][t] <= 2) {
                g.setColor(Color.YELLOW);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 2 && T[x][t] <= 2.25) {
                g.setColor(Color.ORANGE);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 2.25 && T[x][t] <= 2.5) {
                g.setColor(Color.PINK);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 2.5 && T[x][t] <= 2.75) {
                g.setColor(Color.MAGENTA);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            } else if (T[x][t] > 2.75 && T[x][t] <= 3) {
                g.setColor(Color.RED);
                g.fillRect(x * sirina, t * 2 * sirina, sirina, sirina);
            }
            g.setColor(Color.BLACK);
            g.drawRect(x * sirina, t * 2 * sirina, sirina, sirina);
        }
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    //Kreiranje prozora
    JFrame window = new JFrame();
    window.setTitle("Prenos toplote");
    window.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    window.setBounds(10, 10, 600, 600);
    window.getContentPane().add(new NestacionarniPrenosToplote());
    window.setVisible(true);
}
}
```

Прилог 6 – *GUI.java*

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package nestacionarniprenostoplote;

/**
 *
 * @author Hp
 */
public class GUI extends javax.swing.JFrame {

    /**
     * Creates new form GUI
     */
    public GUI() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setBackground(new java.awt.Color(0, 102, 102));

        jLabel1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
        jLabel1.setForeground(new java.awt.Color(255, 255, 255));
        jLabel1.setText("Nestacionarni proračun toplote");

        jLabel2.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
        jLabel2.setForeground(new java.awt.Color(255, 255, 255));
        jLabel2.setText("Broj čvornih tačaka duž pravca X:");

        jLabel3.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
        jLabel3.setForeground(new java.awt.Color(255, 255, 255));
        jLabel3.setText("Broj vremenskih koraka:");

        jButton1.setBackground(new java.awt.Color(255, 255, 255));
        jButton1.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
        jButton1.setForeground(new java.awt.Color(0, 102, 102));
        jButton1.setText("Pokreni proračun");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(60, 60, 60)
                        .addComponent(jLabel1))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addContainerGap()
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel2)
                            .addComponent(jLabel3))
                        .addGap(18, 18, 18)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                            .addComponent(jTextField1)
                            .addComponent(jTextField2, javax.swing.GroupLayout.DEFAULT_SIZE, 160, Short.MAX_VALUE)))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(121, 121, 121)
                        .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 165,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(20, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)
                .addGap(33, 33, 33)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(57, 57, 57)
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 53,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(66, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        );

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        int broj_cvornih_tacaka = Integer.parseInt(jTextField1.getText());
        int broj_vremenskih_koraka = Integer.parseInt(jTextField2.getText());

        nestacionarniprenostoplote.NestacionarniPrenosToplote p = new NestacionarniPrenosToplote();
        p.main(null);
        p.setNx(broj_cvornih_tacaka);
        p.setNt(broj_vremenskih_koraka);
    }

    /**
     * @param args the command line arguments
     */
```

```
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new GUI().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    // End of variables declaration
}
```

Прилог 7 – *NestacionarniPrenosToplote.java*

```
package nestacionarniprenostoplote;

import java.awt.Color;
import java.awt.Graphics;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import javax.swing.JComponent;
import javax.swing.JFrame;

public class NestacionarniPrenosToplote extends JComponent {

    public static int nx = 1;
    public static int nt = 1;
    public static double q = 1.0;
    public static double ux = 1.0;
    public static double vt = 0.1;
    public static double k = 1.0;
    public static double ro = 1.0;
    public static double Cp = 1.0;

    public int getNx() {
        return nx;
```

```
}

public void setNx(int nx) {
    this.nx = nx;
}

public int getNt() {
    return nt;
}

public void setNt(int nt) {
    this.nt = nt;
}

public double getq() {
    return q;
}

public void setq(double q) {
    this.q = q;
}

public double getux() {
    return ux;
}

public void setux(double ux) {
    this.ux = ux;
}

public double getvt() {
    return vt;
}

public void setvt(double vt) {
    this.vt = vt;
}

public double getk() {
    return k;
}

public void setk(double k) {
    this.k = k;
}

public double getro() {
    return ro;
}

public void setro(double ro) {
    this.ro = ro;
}

public double getCp() {
    return Cp;
}

public void setCp(double Cp) {
    this.Cp = Cp;
}

//Kreiranje grafike
@Override
public void paint(Graphics g) {

    //Inicijalizacija promenljivih
    double[] f1 = new double[1000];
    double[] f2 = new double[1000];
    double[] rho = new double[1000];
```

```
double[] feq = new double[1000];
double[] T = new double[1000];
double[] x = new double[1000];
int i, t;
double dx = ux / nx;
double dt = vt / nt;

//Definisanje rastojanja x
x[0] = 0.0;
for (i = 1; i <= nx; i++) {
    x[i] = x[i - 1] + dx;
}

//Definisanje parametara
double csq = dx * dx / (dt * dt);
double ei = dt * csq;
double alfa = k / (ro * Cp);
double tau = (alfa / ei) + 0.5;
double omega = 1.0 / tau;

//Pocetni uslovi
for (i = 0; i <= nx; i++) {
    rho[i] = 0.0;
    f1[i] = 0.5 * rho[i];
    f2[i] = 0.5 * rho[i];
}

//Proracun po vremenskim koracima
for (t = 1; t <= nt; t++) {
    //Proces sudara
    for (i = 0; i <= nx; i++) {
        rho[i] = f1[i] + f2[i];
        feq[i] = 0.5 * rho[i];
        f1[i] = (1.0 - omega) * f1[i] + omega * feq[i];
        f2[i] = (1.0 - omega) * f2[i] + omega * feq[i];
    }
    //Proces strujanja
    for (i = 1; i <= nx - 1; i++) {
        f1[nx - i] = f1[nx - i - 1];
        f2[i - 1] = f2[i];
    }
    //Granicni uslovi
    f1[0] = f1[1] + f2[1] - f2[0] + q * dx / k;
    f1[nx] = f1[nx - 1];
    f2[nx] = f2[nx - 1];

    T[t] = rho[0];

    //Velicina polja koja se iscrtava
    int sirina = 20;
    if (nx < 10) {
        sirina = sirina;
    } else if (nx >= 10 || nx < 30) {
        sirina = 250 / nx;
    } else if (nx > 30) {
        sirina = 100 / nx;
    }

    //Crtanje i bojenje polja temperature
    for (i = 0; i < nx; i++) {
        if (rho[i] <= 0.025) {
            g.setColor(Color.BLUE);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.025 && rho[i] <= 0.05) {
            g.setColor(Color.CYAN);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.05 && rho[i] <= 0.1) {
            g.setColor(Color.GRAY);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.1 && rho[i] <= 0.2) {
```

```
            g.setColor(Color.LIGHT_GRAY);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.2 && rho[i] <= 0.3) {
            g.setColor(Color.GREEN);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.3 && rho[i] <= 0.4) {
            g.setColor(Color.YELLOW);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.4 && rho[i] <= 0.5) {
            g.setColor(Color.ORANGE);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.5 && rho[i] <= 0.6) {
            g.setColor(Color.PINK);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.6 && rho[i] <= 0.7) {
            g.setColor(Color.MAGENTA);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.7 && rho[i] <= 0.8) {
            g.setColor(Color.RED);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        } else if (rho[i] > 0.8) {
            g.setColor(Color.RED);
            g.fillRect(i * sirina, t * 2 * sirina, sirina, sirina);
        }
        g.setColor(Color.BLACK);
        g.drawRect(i * sirina, t * 2 * sirina, sirina, sirina);
    }

    //Stampanje duzine x u konzoli
    for (i = 0; i <= nx; i++) {
        System.out.print(x[i]);
        System.out.println();
    }
    System.out.println();
    System.out.println();
    System.out.println();

    //Stampanje temperature na rastojanju u konzoli
    for (i = 0; i <= nx; i++) {
        System.out.print(rho[i]);
        System.out.println();
    }
    System.out.println();
    System.out.println();
    System.out.println();
}
    //Stampanje temperature u vremenu u konzoli
    for (t = 0; t <= nt; t++) {
        System.out.print(T[t]);
        System.out.println();
    }
    System.out.println();
    System.out.println();
    System.out.println();

//Definisanje broja decimalnih mesta
DecimalFormat df = new DecimalFormat("0.00000000");

//Cuvanje rezultata u tekstualnom dokumentu
String Temperature = "Temperature_vreme_rastojanje.txt";
try {
    PrintWriter Rezultat = new PrintWriter(Temperature);
    Rezultat.println("Rastojanja: ");
    for (i = 0; i <= nx; i++) {
        Rezultat.print(df.format(x[i]) + "     ");
    }
    Rezultat.println("");
    Rezultat.println("");

    Rezultat.println("Temperatura i rastojanje: ");
```

```
            for (i = 0; i <= nx; i++) {
                Rezultat.print(df.format(rho[i]) + "    ");
            }
            Rezultat.println("");
            Rezultat.println("");

            Rezultat.println("Temperatura i vreme: ");
            for (t = 1; t <= nt; t++) {
                Rezultat.print(df.format(T[t]) + "    ");
            }
            Rezultat.close();
        } catch (FileNotFoundException ex) {
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Kreiranje prozora i graficki prikaz rezultata
        JFrame window = new JFrame();
        window.setTitle("Prenos toplote");
        window.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        window.setBounds(10, 10, 300, 600);
        window.getContentPane().add(new NestacionarniPrenosToplote());
        window.setVisible(true);
    }
}
```

Прилог 8 – *GUI.java*

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package nestacionarniprenostoplote;

/**
 *
 * @author Hp
 */
public class GUI extends javax.swing.JFrame {

    /**
     * Creates new form GUI
     */
    public GUI() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        jTextField2 = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        jTextField3 = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
```

```
jTextField4 = new javax.swing.JTextField();
jLabel5 = new javax.swing.JLabel();
jTextField5 = new javax.swing.JTextField();
jLabel7 = new javax.swing.JLabel();
jTextField6 = new javax.swing.JTextField();
jLabel8 = new javax.swing.JLabel();
jTextField7 = new javax.swing.JTextField();
jLabel9 = new javax.swing.JLabel();
jTextField8 = new javax.swing.JTextField();
jButton1 = new javax.swing.JButton();
jLabel6 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jPanel1.setBackground(new java.awt.Color(0, 102, 51));

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel1.setForeground(new java.awt.Color(255, 255, 255));
jLabel1.setText("Broj čvornih tačaka duž pravca x:");

jTextField1.setText("1");

jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel2.setForeground(new java.awt.Color(255, 255, 255));
jLabel2.setText("Broj vremenskih koraka: ");

jTextField2.setText("1");

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel3.setForeground(new java.awt.Color(255, 255, 255));
jLabel3.setText("Vrednost fluksa:");

jTextField3.setText("1.0");

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel4.setForeground(new java.awt.Color(255, 255, 255));
jLabel4.setText("Ukupno rastojanje koje se posmatra:");

jTextField4.setText("1.0");
jTextField4.setToolTipText("");

jLabel5.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel5.setForeground(new java.awt.Color(255, 255, 255));
jLabel5.setText("Posmatrani vremenski trenutak:");

jTextField5.setText("0.1");

jLabel7.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel7.setForeground(new java.awt.Color(255, 255, 255));
jLabel7.setText("Toplotna provodnost materijala:");

jTextField6.setText("1.0");

jLabel8.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel8.setForeground(new java.awt.Color(255, 255, 255));
jLabel8.setText("Gustina materijala:");

jTextField7.setText("1.0");

jLabel9.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel9.setForeground(new java.awt.Color(255, 255, 255));
jLabel9.setText("Specifična toplota materijala:");

jTextField8.setText("1.0");

jButton1.setBackground(new java.awt.Color(255, 255, 255));
jButton1.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jButton1.setText("Izračunaj");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            jButton1ActionPerformed(evt);
        }
    });

    jLabel6.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
    jLabel6.setForeground(new java.awt.Color(255, 255, 255));
    jLabel6.setText("NESTACIONARNI PRENOST TOPLOTE");

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addContainerGap()
                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                                    .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                    .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                    .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                    .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                                .addComponent(jLabel5))
                            .addGap(30, 30, 30)
                            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                                .addComponent(jTextField1)
                                .addComponent(jTextField2)
                                .addComponent(jTextField3, javax.swing.GroupLayout.DEFAULT_SIZE, 200, Short.MAX_VALUE)
                                .addComponent(jTextField4)
                                .addComponent(jTextField5))
                            .addGap(0, 0, Short.MAX_VALUE))
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addComponent(jLabel7)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jTextField6, javax.swing.GroupLayout.PREFERRED_SIZE, 200,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addComponent(jLabel8)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jTextField7, javax.swing.GroupLayout.PREFERRED_SIZE, 200,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addComponent(jLabel9)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jTextField8, javax.swing.GroupLayout.PREFERRED_SIZE, 200,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(167, 167, 167)
                    .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(0, 0, Short.MAX_VALUE)))
            .addContainerGap())
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addGap(111, 111, 111)
            .addComponent(jLabel6)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    jPanel1Layout.setVerticalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
            .addContainerGap(29, Short.MAX_VALUE)
```

```
            .addComponent(jLabel6)
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel1)
                .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel2)
                .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel3)
                .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel4)
                .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel5)
                .addComponent(jTextField5, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel7)
                .addComponent(jTextField6, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel8)
                .addComponent(jTextField7, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel9)
                .addComponent(jTextField8, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(35, 35, 35)
            .addComponent(jButton1,                javax.swing.GroupLayout.PREFERRED_SIZE,                50,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(29, 29, 29))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1,        javax.swing.GroupLayout.DEFAULT_SIZE,        javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1,        javax.swing.GroupLayout.DEFAULT_SIZE,        javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        );

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        int broj_cvornih_tacaka = Integer.parseInt(jTextField1.getText());
        int broj_vremenskih_koraka = Integer.parseInt(jTextField2.getText());
        double fluks = Double.parseDouble(jTextField3.getText());
        double ukupno_posmatrano_rastojanje = Double.parseDouble(jTextField4.getText());
        double posmatrani_vremenski_trenutak = Double.parseDouble(jTextField5.getText());
        double toplotna_provodnost = Double.parseDouble(jTextField6.getText());
```

```
        double gustina = Double.parseDouble(jTextField7.getText());
        double specificna_toplota = Double.parseDouble(jTextField8.getText());

        nestacionarniprenostoplote.NestacionarniPrenosToplote p = new NestacionarniPrenosToplote();
        p.main(null);
        p.setNx(broj_cvornih_tacaka);
        p.setNt(broj_vremenskih_koraka);
        p.setq(fluks);
        p.setux(ukupno_posmatrano_rastojanje);
        p.setvt(posmatrani_vremenski_trenutak);
        p.setk(toplotna_provodnost);
        p.setro(gustina);
        p.setCp(specificna_toplota);
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(GUI.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new GUI().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    private javax.swing.JTextField jTextField7;
    private javax.swing.JTextField jTextField8;
    // End of variables declaration
}
```