Šolski center Novo mesto  
Srednja elektro šola in Tehniška gimnazija  
Šegova ulica 112  
8000 Novo mesto

Maturitetna seminarska naloga pri predmetu računalništvo

**IZDELAVA IGER – 2D PLATFORMSKA IGRA**

Predmet: Računalništvo  
Avtor: Jure Zupančič, T4C  
Šolsko leto: 2020/21 Mentor: dr. Albert Zorko

Novo mesto, april 2021

POVZETEK

V svoji seminarski nalogi bom predstavil izdelavo 2D platformske igre v programskem jeziku Java. Pri izdelavi programa in seminarske naloge sem si pomagal z raziskovanjem strokovne literature, spletnimi viri in pripomočki in forumi ter z internimi gradivi. Kot programersko okolje sem uporabil program Eclipse, pri izdelavi igre pa sem uporabljal še programe Excel, Audacity in Windows 3D Slikar. V prvem delu so predstavljene vse knjižnice razredi in metode, ki sem jih uporabil pri programiranju. Za ustvarjanje okna sem uporabil razreda JFrame in JPanel, za ustvarjanje niti, na kateri teče program pa razred Thread in vmesnik Runnable. Za risanje elementov igre na zaslon sem uporabil razreda BufferedImage in Graphics. Uporabnik igro nadzoruje z tipkovnico in miško. Zaznavanje pritiskov gumbov in premikov miške omogočajo KeyListener, MouseListener in MouseMotionListener. Za predvajanje zvokov igre sem v programu uporabil razreda Clip in AudioInputStream, za branje in pisanje tekstovnih datotek s podatki pa razreda BufferedReader in BufferedWriter. V drugem delu naloge sem predstavil potek načrtovanja igre. Igralec mora priti od začetka do konca nivoja, sestavljenega iz opek in špic, pri čemer pa se ne sme dotakniti špic ali pasti v prepad. Igra vsebuje več takih nivojev. Uporabnik ima tudi možnost spreminjanja nastavitev jezika in glasnosti igre, na voljo pa mu je tudi okno s pomočjo. Do različnih delov aplikacije bo uporabnik lahko dostopal preko glavnega menija. V zadnjem delu naloge sem predstavil program. Program je razdeljen na več razredov, med katerimi vsak opravlja svojo nalogo, na primer ustvarjanje okna, nalaganje in hranjenje slik, zvokov, nivojev in nastavitev. Program je razdeljen na več stanj: glavni meni, nastavitve, pomoč, izbira nivoja in nivo, ki je glavni del igre. Vsako izmed teh stanj deluje kot neke vrste podprogram in skrbi za inicializacijo, posodabljanje in risanje svojih elementov. Vse razrede povezuje glavni razred, v katerem se nahaja glavna zanka programa.

Izvorno kodo programa lahko najdete na spletni strani GitHub, če sledite povezavi:  
[*https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-*](https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-)

KLJUČNE BESEDE: Java, programiranje iger, 2D igra, razred, metoda, stanje programa, grafični vmesnik, predvajanje zvoka, risanje

ABSTRACT

In my paper, I am going to present the creation of a 2D platform game in the programming language Java. As help, when making this game, I used technical literature, internet sources, tools, forums and study material. As a programming environment I am going to use Eclipse. When making my game, I am also going to use the programs Excel, Audacity and Windows 3D Painter. In the first part of my paper I am going to present the libraries, classes and methods, which I will use when programming. For the creation of the game window, I am going to use JFrame and JPanel. For the creation of the thread on which the program will run, I am going to use Thread and Runnable. To render the elements of the game on the screen I will use classes BufferedImage and Graphics. The user can control the game using the keyboard and the mouse. Keyboard and mouse events are detected using KeyListener, MouseListener and MouseMotionListener. The classes Clip and AudioInputStream are used for loading and playing sounds. BufferedReader and BufferedWriter are used to read and write text files, which my program will use. In the second part of my paper I present the design process of the game. To win the player must reach the end of the level, which is made up of bricks and spikes. The player has to avoid the spikes and gaps in the level. The game includes multiple levels. The user can also change the sound volume and language settings. The user also has access to a help window. Different segments of the application can be accessed through the main menu. In the last part of the paper I present the code behind the game. The program is made up of multiple classes, of which each is responsible for one or more tasks. These tasks include: the creation of the window, loading and storage of the images, sounds, levels and settings. The game is divided into different states: main menu, settings, help, level selection and level, which is the main state of the game. Each of these states works like a subprogram, which is in charge of initializing, updating and rendering its elements. All classes are organised by the main class, that holds the main loop of the game.

You can find the source code of my program on the website GitHub, if you follow the link: <https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga->

KEY WORDS: Java, game programming, 2D game, class, method, program state, graphic interface, audio playing, rendering

KAZALO VSEBINE

[1 UVOD 9](#_Toc69232325)

[1.1 JFRAME in JPANEL 9](#_Toc69232326)

[1.2 RUNNABLE 10](#_Toc69232327)

[1.3 BUFFEREDIMAGE IN GRAPHICS 11](#_Toc69232328)

[1.4 ZAZNAVANJE TIPKOVNICE IN MIŠKE 12](#_Toc69232329)

[1.5 BUFFEREDREADER IN BUFFEREDWRITER 14](#_Toc69232330)

[1.6 ZVOK V JAVI 15](#_Toc69232331)

[2 NAČRTOVANJE PROGRAMA 16](#_Toc69232332)

[2.1 SHEMA PROGRAMA 17](#_Toc69232333)

[2.2 USTVARJANJE NIVOJEV 18](#_Toc69232334)

[2.3 USTVARJANJE SLIK 18](#_Toc69232335)

[2.4 ZVOKI 20](#_Toc69232336)

[3 REALIZACIJA PROGRAMA 21](#_Toc69232337)

[3.1 RAZRED IGRA 21](#_Toc69232338)

[3.2 RAZRED OKNO 23](#_Toc69232339)

[3.3 RAZRED TIPKE 23](#_Toc69232340)

[3.4 RAZRED SLIKE 24](#_Toc69232341)

[3.5 RAZRED ZVOK 25](#_Toc69232342)

[3.6 RAZRED NIVOJI 26](#_Toc69232343)

[3.7 RAZRED JEZIK 27](#_Toc69232344)

[3.8 RAZRED NASTAVITVE 28](#_Toc69232345)

[3.9 ELEMENTI MENIJEV 29](#_Toc69232346)

[3.9.1 BESEDILO 29](#_Toc69232347)

[3.9.2 GUMB 29](#_Toc69232348)

[3.9.3 DRSNIK 31](#_Toc69232349)

[3.9.4 SPUSTNI MENI 33](#_Toc69232350)

[3.10 STANJA IGRE 35](#_Toc69232351)

[3.10.1 STANJE MENI 35](#_Toc69232352)

[3.10.2 STANJE NASTAVITVE 36](#_Toc69232353)

[3.10.3 STANJE POMOČ 38](#_Toc69232354)

[3.10.4 STANJE IZBIRA NIVOJA 38](#_Toc69232355)

[3.10.5 STANJE NIVO 40](#_Toc69232356)

[3.11 KOCKE 45](#_Toc69232357)

[3.12 SVET 49](#_Toc69232358)

[3.13 IGRALEC 51](#_Toc69232359)

[3.14 DELCI 59](#_Toc69232360)

[3.15 KAMERA 63](#_Toc69232361)

[3.16 OZADJE 63](#_Toc69232362)

[4 ZAKLJUČEK 65](#_Toc69232363)

[5 ZAHVALA 66](#_Toc69232364)

[6 BIBLIOGRAFIJA 67](#_Toc69232365)

[7 PRILOGA 69](#_Toc69232366)

KAZALO SLIK

[Slika 1: shema programa 17](#_Toc69228350)

[Slika 3: gumbi 18](#_Toc69228351)

[Slika 4: kocke 19](#_Toc69228352)

[Slika 5: Igralec 19](#_Toc69228353)

[Slika 6: Drsnik 32](#_Toc69228354)

[Slika 7: Spustni meni 34](#_Toc69228355)

[Slika 8: Glavni meni 36](#_Toc69228356)

[Slika 9: Nastavitve 37](#_Toc69228357)

[Slika 10: Pomoč 38](#_Toc69228358)

[Slika 11: Izbira nivoja 39](#_Toc69228359)

[Slika 12: Nivo 44](#_Toc69228360)

[Slika 13: Nivo - ustavljen 44](#_Toc69228361)

[Slika 14: Preverjanje trka 1 47](file:///C:\Users\Jure%20Zupančič\Desktop\maturitetna%20seminarska.docx#_Toc69228362)

[Slika 15: Preverjanje trka 2 48](#_Toc69228363)

[Slika 16: Skupina delcev 1 61](#_Toc69228364)

[Slika 17: Skupina delcev 2 62](#_Toc69228365)

KAZALO KODE

[Koda 1: ustvarjanje okna: 10](#_Toc69228366)

[Koda 2: primer Runnable 10](#_Toc69228367)

[Koda 3: metode razreda Graphics 11](#_Toc69228368)

[Koda 4: nalaganje slike 11](#_Toc69228369)

[Koda 5: primer risanja Besedila na zaslon 12](#_Toc69228370)

[Koda 6: pridobivanje fokusa 12](#_Toc69228371)

[Koda 7: primer KeyListener 12](#_Toc69228372)

[Koda 8: Primer MouseListener 13](#_Toc69228373)

[Koda 9: Primer MouseMotionListener 13](#_Toc69228374)

[Koda 10: branje datoteke 14](#_Toc69228375)

[Koda 11: pisanje na datoteko 14](#_Toc69228376)

[Koda 12: nalaganje in predvajanje posnetka 15](#_Toc69228377)

[Koda 13: spreminjanje glasnosti posnetka 15](#_Toc69228378)

[Koda 14: metoda main 21](#_Toc69228379)

[Koda 15: metoda run 22](#_Toc69228380)

[Koda 16: metoda ustvariOkno 23](#_Toc69228381)

[Koda 17: dodajanje KeyListener igri 23](#_Toc69228382)

[Koda 18: metoda tipkaPritisnjena 24](#_Toc69228383)

[Koda 19: miskaPremaknjena 24](#_Toc69228384)

[Koda 20: metoda za pridobivanje stanja tipke 24](#_Toc69228385)

[Koda 21: metoda setGlasnost 25](#_Toc69228386)

[Koda 22: konstruktor razreda Nivoji 27](#_Toc69228387)

[Koda 23: metodi shraniGlasnost in shraniJezik 28](#_Toc69228388)

[Koda 24: metoda shraniDatoteko 28](#_Toc69228389)

[Koda 25: metoda narisi razreda Besedilo 29](#_Toc69228390)

[Koda 26: metoda posodobiStanje razreda Gumb 30](#_Toc69228391)

[Koda 27: metoda narisi razreda Gumb 30](#_Toc69228392)

[Koda 28: metoda posodobiStanje razreda Drsnik 32](#_Toc69228393)

[Koda 29: metoda narisi razreda Drsnik 32](#_Toc69228394)

[Koda 30: metoda posodobiStanje razreda SpustniMeni 33](#_Toc69228395)

[Koda 31: metoda narisi razreda SpustniMeni 34](#_Toc69228396)

[Koda 32: inicializacija gumbov glavnega menija 35](#_Toc69228397)

[Koda 33: preverjanje stanja gumbov menija 35](#_Toc69228398)

[Koda 34: inicializacija elementov nastavitev 36](#_Toc69228399)

[Koda 35: metoda shraniNastavitve 36](#_Toc69228400)

[Koda 36: inicializacija gumbov izbire nivoja 38](#_Toc69228401)

[Koda 37: preverjanje stanja gumbov 39](#_Toc69228402)

[Koda 38: konstruktor razreda Stanje\_Nivo 40](#_Toc69228403)

[Koda 39: metoda posodobiStanje razreda Stanje\_Nivo 42](#_Toc69228404)

[Koda 40: metoda prekiniIgro 42](#_Toc69228405)

[Koda 41: metoda zmaga 43](#_Toc69228406)

[Koda 42: metoda smrt 43](#_Toc69228407)

[Koda 43: metoda narisi razreda Kocka 45](#_Toc69228408)

[Koda 44: določanje orientacije špice 46](#_Toc69228409)

[Koda 45: metoda ploscina 46](#_Toc69228410)

[Koda 46: določanje točk špice 46](#_Toc69228411)

[Koda 47: prvi korak preverjanja trka 48](#_Toc69228412)

[Koda 48: drugi korak preverjanja trka 49](#_Toc69228413)

[Koda 49: računanje začetnih in končnih indeksov 50](#_Toc69228414)

[Koda 50: risanje kock 50](#_Toc69228415)

[Koda 51: metoda getKocka 50](#_Toc69228416)

[Koda 52: metoda posodobiStanje razreda Igralec 51](#_Toc69228417)

[Koda 53: metoda izracunajXHitrost 52](#_Toc69228418)

[Koda 54: metoda izracunajYHitrost 53](#_Toc69228419)

[Koda 55: metoda premakniX 54](#_Toc69228420)

[Koda 56: metoda preveriTrk 54](#_Toc69228421)

[Koda 57: metoda orientacija 54](#_Toc69228422)

[Koda 58: metoda pogojZaSmrt 55](#_Toc69228423)

[Koda 59: metoda pogojZaZmago 55](#_Toc69228424)

[Koda 60: metoda dotikTal 56](#_Toc69228425)

[Koda 61: metoda narisi razreda Igralec 56](#_Toc69228426)

[Koda 62: metoda trenutnaSlika 57](#_Toc69228427)

[Koda 63: metoda posodobiStanje razreda AnimacijaIgralca 58](#_Toc69228428)

[Koda 64: metoda getTrenutnaSlika 58](#_Toc69228429)

[Koda 65: inicializacija objektov animacije 58](#_Toc69228430)

[Koda 66: generiranje novih delcev skupine 1 60](#_Toc69228431)

[Koda 67: premikanje delcev skupine 1 60](#_Toc69228432)

[Koda 68: odstranjevanje delcev 60](#_Toc69228433)

[Koda 69: generiranje novih delcev skupine 2 61](#_Toc69228434)

[Koda 70: premikanje delcev skupine 2 62](#_Toc69228435)

[Koda 71: metoda narisi razreda SkupinaDelcev 62](#_Toc69228436)

[Koda 72: računanje odmika kamere 63](#_Toc69228437)

[Koda 73: metoda posodobiStanje razreda Ozadje 63](#_Toc69228438)

[Koda 74: metoda narisi razreda Ozadje 64](#_Toc69228439)

# UVOD

V svoji maturitetni seminarski nalogi bom naredil 2D platformsko igro za operacijski sistem Windows. Igra bo napisana v programskem jeziku Java. Pri izdelavi igre bom uporabljal urejevalnik kode Eclipse. Vse slike, ki jih bom potreboval za igro bom narisal v Windows 3D slikarju, za zvočne efekte pa bom uporabil program Audacity. Pri programiranju bom uporabil tudi več knjižnic. V igri bo moral igralec priti od začetka do konca nivoja, ne da bi se pri tem dotaknil špic, ali padel v brezno. Igra bo vsebovala več nivojev, v igro pa bodo vključeni tudi zvočni efekti. Poleg tega bodo uporabniku na voljo tudi nastavitve, kjer bo mogoče spreminjati glasnost ter jezik. Uporabniku bo na voljo tudi okno z navodili za igranje igre. Do različnih delov igre bo igralec lahko dostopal preko glavnega menija.

Igra bo tekla na svoji niti, za kar bom uporabil vmesnik Runnable. Za prikaz okna bom uporabil razreda jFrame in jPanel. Vsebina igre bo z objektom razreda Graphics narisana na BufferedImage. Da bo z igro mogoče nadzorovati z tipkami in miško, bom uporabil vmesnike KeyListener, MouseListener in mouseMotionListener ter razrede KeyAdapter, KeyEvent, MouseAdapter in MouseEvent. Za delo z datotekami bom uporabil razreda BufferedReader in FileReader ter BufferedWriter in FileWriter. za delo z zvokom pa razrede AudioInputStream, Clip in FloatControl V več delih programa bom uporabil še razreda ArrayList in Math.

## JFRAME in JPANEL

JFrame je razred, ki omogoča izdelavo okna na ekranu računalnika. JFrame deduje razred Frame in dodaja nekaj novih funkcionalnosti. JFrame ima okvir, lahko mu spreminjamo velikost in ga premikamo po zaslonu. Vse druge komponente morajo biti znotraj objekta JFrame. (1) Objekt razreda lahko ustvarimo z konstruktorjem JFrame(String naslov).Ustvarimo ga lahko tudi brez naslova. Velikost okna nastavimo z metodami setPrefferedSize(Dimension d), setMinimumSize (Dimension d), in setMaximumSize(Dimension d), ki kot argument vzamejo objekt Dimension. (2) Kot alternativo lahko uporabimo metodo pack(), ki velikost okna prilagodi objektom v oknu. Komponente lahko oknu dodajamo z metodo add(Objekt o). Z metodo setDeafultCloseOperation(int vrednost) povemo, kaj naj se zgodi, ko zapremo okno. V večini primerov želimo da okno izgine, zato vrednost nastavimo na jFrame.EXIT\_ON\_CLOSE. Druga pomembna metoda je še setVisible(), saj bo okno, ki ga ustvarimo vidno šele po klicu te metode. (1) Preostale metode ki jih bom uporabil so še setResizable(Boolean b), s katero določimo, če je oknu mogoče spreminjati velikost in setLocationRelativeTo  
(Objekt o), s katero določimo pozicijo glede na nek objekt. (2)

JPanel je razred, ki deduje JComponent in ga je mogoče dodati objektu JFrame z metodo add(JPanel panel). JPanel je splošen vsebnik (angl. container), v katerega lahko dodajamo druge komponente. (3) Objektu razreda lahko spremenimo velikost z enakimi metodami, kot so nam na voljo za JFrame. Prav tako mu lahko dodajamo druge objekte razredov, ki dedujejo razred JComponent.

V Kodi 1 lahko vidimo primer ustvarjanja objekta JFrame, ki mu dodamo JPanel.

JFrame frame = **new** JFrame();

JPanel panel = **new** JPanel();

panel.setPreferredSize(**new** Dimension(dolzina,visina));

panel.setMaximumSize(**new** Dimension(dolzina,visina));

panel.setMinimumSize(**new** Dimension(dolzina,visina));

frame.add(igra);

frame.pack();

frame.setResizable(**false**);

frame.setLocationRelativeTo(**null**);

frame.setDefaultCloseOperation(JFrame.**EXIT\_ON\_CLOSE**);

frame.setVisible(**true**);

Koda : ustvarjanje okna:

## RUNNABLE

Runnable je vmesnik, ki ga implementiramo v nek razred, za katerega želimo da teče na svoji niti. Tak razred mora vsebovati metodo run(). (4) V tem razredu nato ustvarimo objekt Razreda Thread, ki mu podamo primerek našega razreda z ključno besedo this. Ko z objektom Thread pokličemo metodo start(), se začne izvajati nova nit. Na niti se izvaja koda znotraj metode run(). V metodi run() se bo nahajala glavna zanka programa. Ko želimo da se program neha izvajati, nit ustavimo z metodo join().

Primer razreda, ki implementira Runnable:

**public** **class** Razred **implements** Runnable{

**private** Thread nit;

**public** Razred(){

**if**(nit == **null**){

nit = **new** Thread(**this**);

nit.start();

}

}

**public** **void** run(){

//koda programa

**try** {

nit.join();

}

**catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

Koda : primer Runnable

## BUFFEREDIMAGE IN GRAPHICS

V svoji igri bom za risanje vsebine uporabljal razreda Graphics in BufferedImage.

BufferedImage je podrazred razreda Image in se uporablja za upravljanje in manipuliranje podatkov neke slike. (5) Objekt lahko ustvarimo z več konstruktorji. V svojem programu bom uporabljal BufferedImage(int dolzina, int visina, int tip). Dolžina in višina slike se merita v pikslih. Tip slike je konstanta razreda BufferedImage. Možna tipa slike sta TYPE\_BYTE\_BINARY in TYPE\_BYTE\_INDEXED. (5) Same po sebi slike ne moremo spreminjati. Za to potrebujemo objekt Graphics.

Graphics je abstrakten razred, ki dovoli aplikaciji, da riše na različne komponente, pa tudi na slike, ki jih ni na zaslonu. (6) Rišemo lahko na primer na objekt JPanel, ali pa na objekt BufferedImage. Graphics uporabljamo tako, da ga najprej dodamo na komponento ali sliko, na katero želimo risati. Nekemu objektu dodamo Graphics objekt tako:

Graphics g = objekt.getGraphics();

Ko nekemu objektu dodamo objekt Graphics lahko z različnimi metodami rišemo na ta objekt. Nekatere izmed teh metod so:

drawLine(int x1, int y1, int x2, int y2)  
drawOval(int x, int y, int dolzina, int visina)  
drawPolygon(int[] xTocke, int[] yTocke, int nTock)  
drawRect(int x, int y, int dolzina, int visina)  
drawString(String str, int x, int y)  
fillRect(int x, int y, int dolzina, int visina)  
fillOval(int x, int y, int dolzina, int visina)  
drawImage(Image img, int x, int y, int dolzina, int visina, ImageObserver observer)

Koda : metode razreda Graphics

Poleg tega lahko spreminjamo barvo, v kateri objekt riše z metodo setColor(Color barva). Color je objekt, ki hrani vrednost neke barve. To vrednost mu lahko podamo v obliki RGB, RGBA ali HSB. (7) Če želimo spremeniti font pisave v kateri piše objekt lahko to storimo z metodo setFont (Font font). Razred Font predstavlja različne fonte pisave, ki se jih uporablja za risanje teksta. (8) Ko aplikacija konča z risanjem, je dobra praksa, da se objekta Graphics znebimo. To storimo z metodo dispose().

V svojem programu bom moral nekatere slike naložiti iz datotek. To je mogoče narediti s pomočjo razreda ImageIO. Image IO je razred, ki omogoča branje ali pisanje slikovnih datotek z ali na lokalni disk oziroma URL. Sliko preberemo z metodo read(), pot do slike podamo v obliki niza. (9) Primer branja slike lahko vidite spodaj.

String pot;

BufferedImage slika = ImageIO.read(**new** File(pot));

Koda : nalaganje slike

ATTRIBUTED STRING

Ko z metodo drawString narišemo nek niz imamo opcijo, da namesto navadnega niza narišemo oblikovano besedilo AttributedString. AttributedString shranjuje vsebino niza in vse informacije o lastnostih. (10) Oblikovanemu besedilu lahko spreminjamo pisavo, velikost pisave in druge lastnosti. Za spreminjanje lastnosti bom uporabil tudi objekt Font. V kodi ustvarimo oblikovano besedilo in ga narišemo kot:

Font font = **new** Font("Arial",Font.**BOLD**, velikost);

AttributedString besedilo =   
**new** AttributedString(vsebina);

besedilo.addAttribute(TextAttribute.**FONT**, font);

g.setColor(barva);

g.drawString(besedilo.getIterator(), x, y);

Koda : primer risanja Besedila na zaslon

## ZAZNAVANJE TIPKOVNICE IN MIŠKE

Vmesniki KeyListener, MouseListener in MouseMotionListener delujejo na podobnih principih in dovoljujejo aplikaciji, da prejema dogodke z tipkovnice in miške. Da aplikacija prejema dogodke mora imeti fokus. Moja aplikacija bo kot glavni okvir uporabljala objekt JPanel, zato fokus dobimo tako, da s tem objektom pokličemo metodo requestFocus():

**this**.setFocusable(**true**);  
**this**.requestFocus();

Koda : pridobivanje fokusa

Java KeyListener je obveščen vsakič, ko se spremeni stanje tipke na tipkovnici. Komponenta, ki ji dodamo vmesnik KeyListener vsebuje metode keyPressed (KeyEvent e), keyReleased(KeyEvent e) , keyTyped(KeyEvent e). Metode dobijo objekt tipa KeyEvent, ki vsebuje podatke o dogodku. (11) To nam omogoča, da izvemo, kateri gumb je uporabnik pritisnil. Neki komponenti lahko vmesnik KeyListener dodamo z metodo addKeyListener:

**this**.addKeyListener(**new** KeyAdapter() {

**public void** keyPressed(KeyEvent e) {

//naredi nekaj

}

**public void** keyReleased(KeyEvent e) {

//naredi nekaj

}

});

Koda : primer KeyListener

Java MouseListener je obveščen vsakič, ko spremeni stanje miške. Komponenta, ki ji dodamo vmesnik MouseListener, lahko vsebuje metode mousePressed(MouseEvent e), mouseReleased(MouseEvent e), mouseClicked(MouseEvent e), mouseEntered(MouseEvent e) in mouseExited(MouseEvent e). Metode prejmejo objekt tipa MouseEvent, ki vsebuje podatke o dogodku, ki je sprožil klic metode. (12) Preko tega objekta izvemo kateri gumb na miški je bil pritisnjen. Da komponenta, ki ji dodamo MouseListener prejme obvestilo o dogodku, mora imeti fokus. Neki komponenti vmesnik MouseListener dodamo z metodo addMouseListener:

**this**.addMouseListener(**new** MouseAdapter() {

**public void** mousePressed(MouseEvent e) {

//naredi nekaj

}

**public void** mouseReleased(MouseEvent e) {

//naredi nekaj

}

});

Koda : Primer MouseListener

MouseMotionListener je obveščen vsakič, ko je miška premaknjena. Metodi, ki sta nam na voljo sta mouseMoved(MouseEvent e) in mouseDragged(MouseEvent e). Prva je poklicana, ko uporabnik premakne miško, druga pa ko uporabnik premakne miško, medtem ko drži gumb. Metodi prejmeta objekt MouseEvent, ki vsebuje podatke o dogodku. (13) Iz dogodka lahko izvemo na katerih koordinatah se je ta zgodil in tako sledimo miški. Vmesnik neki komponenti dodamo z metodo addMouseMotionListener:

**this**.addMouseMotionListener(**new** MouseAdapter() {

**public void** mouseMoved(MouseEvent e) {

//naredi nekaj

}

**public void** mouseDragged(MouseEvent e) {

//naredi nekaj

}

});

Koda : Primer MouseMotionListener

## BUFFEREDREADER IN BUFFEREDWRITER

Razred BufferedReader omogoča branje tekstovnih datotek. V konstruktorju moramo podati objekt FileReader, ki z diska prebere neko datoteko. Pot do te datoteke podamo v obliki niza. Pri branju datoteke lahko pride do napake, če datoteka ni najdena, zato moramo ta del kode obdati z try in catch, ali pa metodi dodati ključno besedo throws. Besedilo z datoteke beremo z metodo readLine(), ki vrne spremenljivko tipa String. Ko pridemo do konca datoteke metoda vrne vrednost null. Dobra praksa je, da ko aplikacija konča z branjem datoteke, BufferedReader zapremo. To storimo z metodo close(). (4) Primer branja datoteke lahko vidite v Kodi 10.

String pot;

**try**{

BufferedReader br = **new** BufferedReader(**new** FileReader(pot));

String niz = br.readLine();

}**catch**(Exception e){

e.getStackTrace;

}

br.close();

Koda : branje datoteke

Razred BufferedWriter omogoča shranjevanje podatkov na tekstovno datoteko. V konstrukrorju objekta podamo objekt FileWriter. Pot do datoteke objektu podamo v obliki niza. Podatke zapišemo na datoteko v obliki niza z metodo write(String niz), v novo vrstico pa skočimo z ukazom newLine(). Ko končamo z pisanjem podatkov na datoteko, jo shranimo tako, da pokličemo metodo close(). (14) V nasprotnem primeru datoteka ne bo shranjena. V Kodi 11 lahko vidite primer pisanja datoteke.

String pot;

String podatki1, podatki2;

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(pot));

br.write(podatki1);

br.newLine();

br.write(podatki2);

br.close();

Koda : pisanje na datoteko

## ZVOK V JAVI

V programskem jeziku Java obstaja več različnih načinov za upravljanje in predvajanje zvočnih datotek. V svojem programu bom uporabil razreda Clip in AudioInputStream, za upravljanje glasnosti zvoka pa bom uporabil še razred FloatControl. Razred Clip omogoča predvajanje zvočnih datotek v programu. Vsebuje metode open(), ki odpre zvočno datoteko, start(), ki začne predvajati posnetek, stop(), ki ustavi predvajanje, loop(), ki ponavlja posnetek in setFramePosition(int frame), ki nastavi posnetek na neko pozicijo. Poleg teh vsebuje tudi druge uporabne metode. (15) Razred AudioInputStream omogoča nalaganje zvočnih datotek v program. Za inicializacijo objekta potrebujemo pot do datoteke. Poleg tega pa moramo uporabiti še razred File.

Najprej ustvarimo objekt AudioInputSystem. Nato od AudioSystem pridobimo objekt Clip. Zvočni posnetek odpremo z metodo open(). Nastavimo potrebne lastnosti posnetka, kot je trenutna pozicija. Nazadnje zaženemo posnetek z start(). (16) Primer nalaganja in predvajanja zvočnega posnetka lahko vidite v kodi 12.

String pot;

AudioInputStream input = AudioSystem.getAudioInputStream(**new** File(pot).getAbsoluteFile());

Clip posnetek = AudioSystem.getClip();

posnetek.open(input);

posnetek.setFramePosition(0);

posnetek.start();

Koda : nalaganje in predvajanje posnetka

Glasnost predvajanega zvoka lahko uravnavamo s pomočjo razreda FloatControl. Razred se uporablja za upravljanje z lastnosti tipa Float različnih objektov, med drugim tudi upravljanje glasnosti objekta Clip. Objekt FloatControl dobimo z metodo getControl(). Nato lahko z njim pokličemo metodo setValue(Float vrednost). (17) Razred Clip kot enoto za glasnost uporablja decibele, mi pa hočemo programu podati glasnost med 0 in 1, zato moramo to vrednost nekako pretvoriti. Za to bom uporabil formulo s spletnega vira. (18) Formula poskrbi tudi, da se bo glasnost spreminjala linearno in ne logaritemsko, kot je to značilno za decibele. Spodaj lahko vidite formulo. Gain je v formuli vrednost med 0 in 1, dB pa je končni rezultat v decibelih. Za vrednosti med 0 in 1, formula vrača vrednosti med -80 in 0.

float dB =(float)(Math.log(gain== 0.0 ? 0.0001 :gain) / Math.log(10.0) \*20.0);

V kodi glasnost nekega zvoka spremenimo kot:

**public** **void** setGlasnost(**double** gain) {

**float** dB =(**float**)(Math.log(gain == 0.0 ? 0.0001 :gain)/Math.log(10.0) \* 20.0);

FloatControl glasnost = (FloatControl)zvok.getControl(FloatControl.Type.**MASTER\_GAIN**);

glasnost.setValue(dB);

}

Koda : spreminjanje glasnosti posnetka

# NAČRTOVANJE PROGRAMA

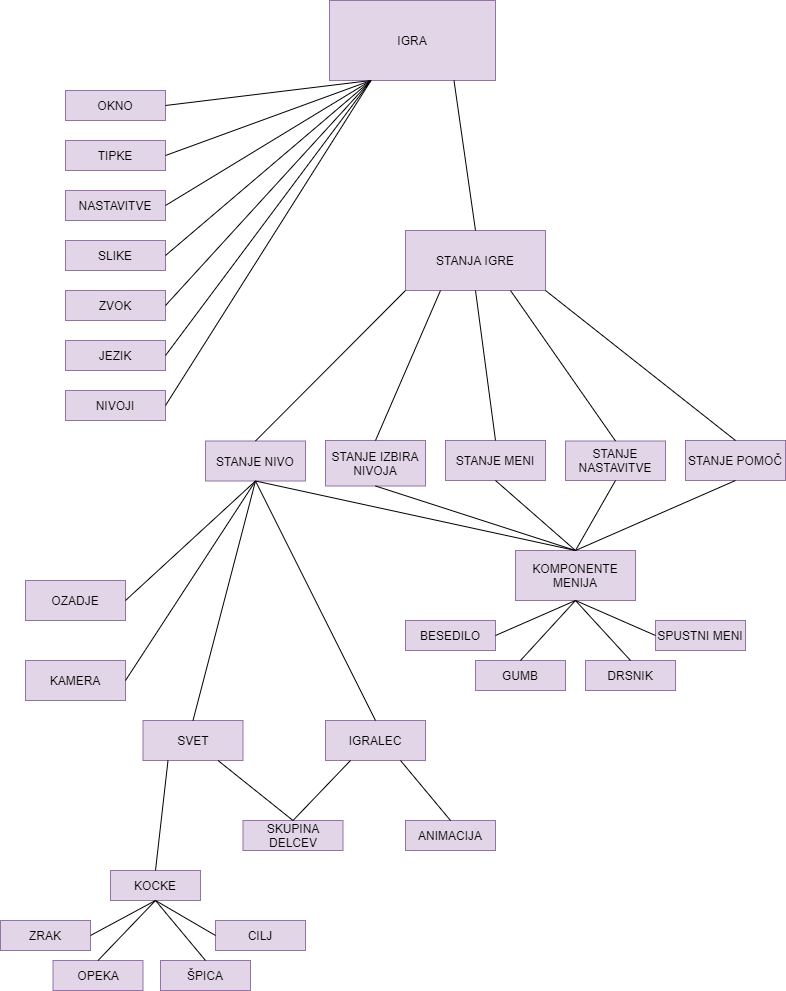
V igri bo moral igralec priti od točke A do točke B, pri tem pa ne bo smel pasti ali se dotakniti špic. V 2D prostoru se bo lahko premikal levo in desno, poleg tega pa bo lahko tudi skakal. Da bo igralec zaključil nivo, bo moral priti do cilja. V igro bo vključena tudi animacija igralca, tako se bo slika igralca med premikanjem spreminjala. Igralec bo imel animacijo tudi za zmago in smrt. Navidezna kamera bo morala ves čas slediti igralcu, ko se bo premikal po nivoju. V nasprotnem primeru bi igralec preprosto odkorakal z zaslona. Igra bo vsebovala več nivojev in vsak bo bil sestavljen iz kvadratnih opek in špic, skozi katere igralec ne bo mogel prehajati. Podatki o razporeditvi le teh bodo shranjeni v dvodimenzionalni tabeli. V igro bodo vključeni tudi zvočni efekti.

Da bo uporabnik lažje dostopal do različnih nivojev in nastavitev, bo igra potrebovala menije. Program bo moral biti zmožen preklapljanja med različnimi stanji igre. Tako bo igra imela stanje za nastavitve, stanje za glavni meni, stanje za izbiro nivoja, stanje za nastavitve in stanje za pomoč, kjer bo uporabnik lahko našel navodila za igranje igre. Vsako stanje si lahko predstavljamo kot svoj podprogram, med katerimi lahko prosto preklapljamo preko gumbov v programu. Vsa stanja bo nadzoroval glavni program, aktivno pa bo lahko le eno stane naenkrat. Za preklapljanje med stanji in spreminjanje nastavitev bo igra potrebovala komponente grafičnega vmesnika. V program bodo vključeni: besedilo, gumb, drsnik in spustni meni. Vsaka izmed teh komponent se bo morala odzivati na klike miške in izvesti neko nalogo.

Igra bo potrebovala tudi razrede, ki bodo odgovorni za nalaganje slik, zvokov in podatkov o nivojih. Vsak izmed elementov igre bo imel svojo sliko, ki jo bo program narisal na zaslon. Ob določenih dogodkih v igri (npr. klik gumba) bo program izvedel zvočni efekt. Glasnost zvokov bo mogoče spreminjati v nastavitvah. Poleg glasnosti bo mogoče spreminjati tudi jezik aplikacije. Vsa besedila in gumbi bodo pri tem morali prilagoditi svojo vsebino. Nazadnje bo igra potrebovala tudi razred, ki bo shranjeval trenutno stanje gumbov tipkovnice in miške, preko katerega bomo lahko zaznavali vnos podatkov s strani uporabnika.

## SHEMA PROGRAMA

Na spodnji sliki lahko vidimo shemo programa. Igra teče v glavnem razredu, v katerem so shranjeni primerki vseh drugih razredov. Z glavnim razredom so direktno povezani razredi za ustvarjanje okna ter upravljanje z različnimi viri. Preostali deli programa se delijo na različna stanja. Vsa stanja uporabljajo komponente menija, stanje Nivo, ki predstavlja glavni del igre pa vsebuje tudi razrede Igralec, Svet, Kamera in Ozadje.



Slika : shema programa

## USTVARJANJE NIVOJEV

Podatki o nivojih bodo shranjeni v datotekah. Posamezen nivo bo predstavljen kot 2D tabela celih števil. Vsako število bo predstavljalo enega izmed možnih kock, ki sestavljajo nivo. Tako bo 0 označevala zrak, 1 bo označevala opeko, številke od 2 do 5 bodo označevale različno orientirane špice, številke od 6 do 9 pa različno orientirane polovične špice. Z številko 10 bo označen cilj.

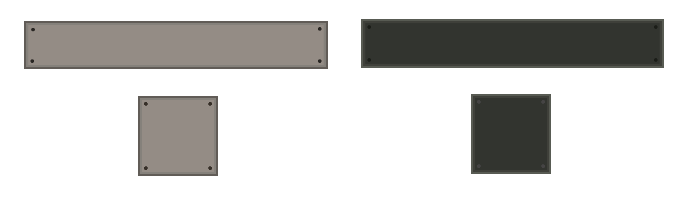
Datoteka bo shranjena kot datoteka CSV. CSV (angl. comma-seperated values) je tekstovna datoteka, v kateri so vrednosti razdeljene z podpičjem. Vsaka vrstica v datoteki predstavlja eno vrstico tabele. Vsaka vrstica tabele mora vsebovati enako število podatkov. (19) Datoteke se bodo nahajale v posebni mapi. Za lažje izdelovanje tabel uporabil program Microsoft Excel. Spodaj lahko vidite primer tabele s podatki o nivoju.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **2** | **2** | **0** | **0** | **6** | **6** | **6** | **0** | **0** | **0** | **10** | **0** | **0** | **0** |
| **0** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **0** |
| **0** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |

## USTVARJANJE SLIK

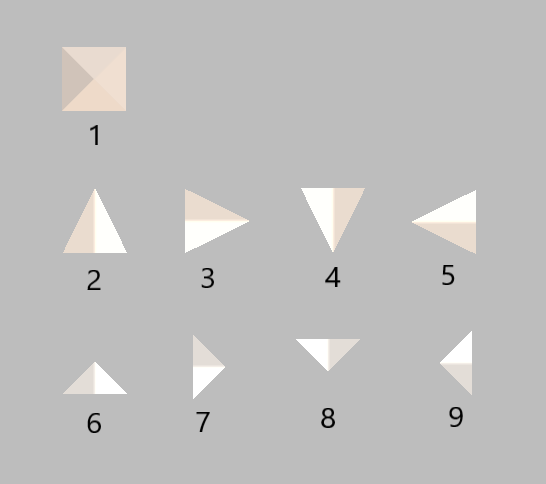
Vsak element, ki ga mora program narisati na zaslon bo potreboval svojo sliko. To vključuje igralca, opeke, špice in vse elemente grafičnega vmesnika, navsezadnje pa tudi ozadja. Za izdelavo slik bom uporabil program Windows 3D slikar. Slike bodo tipa PNG, saj ta tip slike omogoča transparentnost. To je pomembno pri vseh elementih igre, ki nimajo kvadratne ali pravokotne oblike.

Spodaj je mogoče videti primer slike, ki jo bom v programu uporabil za gumb. Vsak gumb ima poleg navadnega stanja, tudi stanje, ko je nad njim kurzor.

****

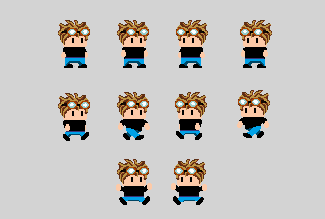
Slika : gumbi

Spodnje slike predstavljajo opeko in špice.

****

Slika : kocke

Za igralca bom potreboval več slik, saj bo igralec animiran. Vidimo lahko, da bo igralec imel različne slike za hojo v levo in desno, potrebujemo pa tudi sliko, ko bo igralec miroval in sliko, ko bo igralec v zraku.



Slika : Igralec

## ZVOKI

Zvoki, ki jih bo uporabljala igra bodo shranjeni v svoji mapi. Za format datoteke ki ga bom uporabljal sem si izbral WAV. WAV ali WAVE je format zvočne datoteke, ki sta ga razvila IBM in Microsoft. Je glavni format za shranjevanje nekompresiranega zvoka na sistemih Windows. (20) Ta vrsta datoteke je primerna predvsem za kratke zvočne efekte, kot jih bom potreboval v svoji igri. Nekaj zvokov bom posnel sam, druge pa bom pridobil s spletnega mesta freesound.org [[1]](#footnote-1). Kot programsko orodje za urejanje zvočnih datotek bom uporabil program Audacity. Zvočni efekti, ki jih bom uporabil so:

- klik.wav  
Se bo sprožil, ko bo uporabnik pritisnil katerikoli gumb na zaslonu  
- hover.wav  
Se bo sprožil, ko bo kurzor nad gumbom  
- skok.wav  
Se bo sprožil, ko bo igralec skočil  
- padec.wav  
Se bo sprožil, ko se bo igralec dotaknil tal  
- smrt.wav  
Se bo sprožil ob smrti igralca  
- zmaga.wav  
Se bo sprožil ob zmagi

# REALIZACIJA PROGRAMA

Program je razdeljen na več razredov. Vsak izmed njih bo opravljal svojo nalogo. Igra bo imela glavni razred Igra, v katerem bo glavna zanka igre. Program se začne z inicializacijo novega objekta igre v metodi main. Konstruktor kot argumente vzame dolžino in višino okna. V Kodi 14 lahko vidite metodo main.

**public** **static** **void** main(String[] args) **throws** IOException, UnsupportedAudioFileException, LineUnavailableException {

Igra igra = **new** Igra(1200,600);

}

Koda : metoda main

## RAZRED IGRA

Razred Igra je glavni razred programa. V tem razredu teče glavna zanka. Razred tudi hrani primerke vseh stanj programa (str. 35) in vseh drugih razredov, kot so: Okno (str. 23), Slike (str. 24), Zvok (str. 25), Nastavitve (str. 28), Jezik (str. 27), Nivoji (str. 26) in Tipke (str. 23). Druge lastnosti razreda so še objekt Thread (slo. nit), ki nam omogoča, da program teče na svoji niti, dolžina in višina okna v pikslih, ter boolean spremenljivka igra\_poteka, ki je pogoj glavne zanke. Razred Igra deduje razred JPanel. Tako lahko Igro dodamo v JFrame in nanjo rišemo trenutno sliko. Razred implementira vmesnik Runnable.

Konstruktor igre kot argumente prejme dolžino in višino okna. Najprej inicializira te spremenljivke. Nato pokliče metodo inicializiraj(), ki ustvari vse potrebne primerke drugih razredov. Nato ustvari Okno v katerega se doda igra. Nazadnje pokliče še metodo zacniIgro(), ki ustvari novo nit in začne izvajanje glavne zanke.

Metoda run() se pokliče, ko ustvarimo novo nit. Predstavljamo si jo lahko kot glavno metodo programa. Metoda najprej spremeni stanje spremenljivke igra\_poteka na true, nato pa začne izvajanje glavne zanke. Zanka kliče metode posodobiStanje(), narisi() in pokaziSliko(). posodobiStanje() posodobi stanje vseh elementov igre, metoda narisi() nariše vse elemente na sliko, metoda pokaziSliko() pa sliko doda na okno. Nato metoda izračuna koliko časa mora počakati do naslednje izvedbe zanke. To naredi zato, da igra teče s stalno hitrostjo, ki ji igralec lahko sledi. Zanko, ki sem jo uporabil v igri sem dobil v knjigi Killer Game Programming. (21) V spodnji kodi lahko vidite metodo run() z glavno zanko.

public void run()

igra\_poteka = true;

int fps = 80;

long perioda = 1000000000/fps;

long casPrej,casPotem,razlikaCasa,casSpanja;

long prespanCas = 0l;

casPrej = System.nanoTime();

while(igra\_poteka) {

try {

posodobiStanje();

} catch (IOException e) {

e.printStackTrace();

}

narisi();

pokaziSliko();

casPotem = System.nanoTime();

razlikaCasa = casPotem - casPrej;

casSpanja = (perioda - razlikaCasa) - prespanCas;

if(casSpanja > 0) {

try {

Thread.sleep(casSpanja/1000000l);

}catch(InterruptedException e){

prespanCas = (System.nanoTime()-casPotem)-casSpanja;

}

}

else {

prespanCas = 0l;

}

casPrej = System.nanoTime();

}

System.exit(0);

}

Koda : metoda run

Perioda je čas v katerem se zanka enkrat izvede. To je 12,5 ms ali 80 ponovitev na sekundo. casSpanja se izračuna kot perioda – razlikaCasa. razlikaCasa je čas porabljen za izvajanje programa. Spremenljivka prespanCas označuje čas porabljen za klicanje metode sleep() in se upošteva pri računanju časa spanja v naslednjem krogu. Če se stanje spremenljivke igra\_poteka spremeni na false, se izvajanje zanke preneha, in pokliče se metoda System.exit(0), ki ustavi program. (21)

Razred vsebuje tudi metodo ustaviIgro(), ki spremeni stanje spremenljivke igra\_poteka na false, in tako konča izvajanje zanke.

Za spreminjanje stanja igre ima razred metode spremeniStanjeNaNivo(int nivo), spremeniStanjeNaMeni(), spremeniStanjeNaNastavitve(), spremeniStanjeNa Pomoc() in spremeniStanjeNaIzbiraNivoja() ter spremeniStanjeNa Predhodno(). Vsaka izmed teh metod pokliče metodo setStanje(Stanje stanje) razreda Stanje (str. 35). Za spremembo stanja na nek nivo potrebujemo tudi številko nivoja.

Razred vsebuje tudi get metode za vse primerke drugih razredov. Tako lahko vsi elementi igre dostopajo do potrebnih podatkov kot: igra.getObjekt(). getPodatek().

## RAZRED OKNO

Razred Okno ustvari okno na zaslonu in vanj doda igro. Konstruktor razreda kot argument prejme objekt Igra. Konstruktor shrani objekt, prav tako pa shrani tudi dolžino in višino igre. Nato izvede metodo ustvariOkno(), ki jo lahko vidimo spodaj.

**private** JFrame frame;

**public** **void** ustvariOkno() {

frame = **new** JFrame();

igra.setPreferredSize(**new** Dimension(dolzina,visina));

igra.setMaximumSize(**new** Dimension(dolzina,visina));

igra.setMinimumSize(**new** Dimension(dolzina,visina));

frame.add(igra);

frame.pack();

frame.setResizable(**false**);

frame.setLocationRelativeTo(**null**);

frame.setDefaultCloseOperation(JFrame.**EXIT\_ON\_CLOSE**);

frame.setIconImage(Slike.opeka);

frame.setVisible(**true**);

igra.setFocusable(**true**);

igra.requestFocus();

}

Koda : metoda ustvariOkno

## RAZRED TIPKE

Razred Tipke bo odgovoren za zaznavanje pritiskov tipk in premikov miške. Stanje vsake tipke bo shranjeval kot boolean vrednost. Shranjeval bo vrednosti levo, desno, gor, esc ter miska1. Poleg tega bo shranjeval tudi x in y pozicijo miške. Konstruktor bo kot argument prejel objekt Igra in mu dodal KeyListener, MouseListener in MouseMotionListener. Kot primer se KeyListener doda z metodo addKeyListener. Ta način dodajanja KeyListener-ja sem našel na spletni strani StackOverflow. (22) Dodajanje KeyListener lahko vidite v Kodi 17.

igra.addKeyListener(**new** KeyAdapter() {

**public** **void** keyPressed(KeyEvent e) {

tipkaPritisnjena(e);

}

**public** **void** keyReleased(KeyEvent e) {

tipkaSpuscena(e);

}

});

Koda : dodajanje KeyListener igri

Metodi tipkaPritisnjena(KeyEvent e) in tipkaSpuscena(KeyEvent e) Posodobita stanje spremenljivke, ki pripada nekemu gumbu. Kateri gumb je bil pritisnjen ali spuščen ugotovimo z e.getKeyCode. Spodaj lahko vidimo metodo tipkaPritisnjena. Metoda tipkaSpuscena deluje enako, le da spremenljivko posodobi na false. Metodo lahko vidite v Kodi 18.

**public** **void** tipkaPritisnjena(KeyEvent e) {

**int** tipka = e.getKeyCode();

**if**(tipka==KeyEvent.**VK\_W**)

gor = **true**;

**if**(tipka==KeyEvent.**VK\_A**)

levo = **true**;

**if**(tipka==KeyEvent.**VK\_D**)

desno = **true**;

**if**(tipka==KeyEvent.**VK\_ESCAPE**)

esc = **true**;

}

Koda : metoda tipkaPritisnjena

Metoda, ki posodablja x in y pozicijo miške je miskaPremaknjena(MouseEvent e). Pozicijo dobimo z e.getX() in e.getY(). Metodo lahko vidimo spodaj.

**public** **void** miskaPremaknjena(MouseEvent e) {

x = e.getX();

y = e.getY();

}

Koda : miskaPremaknjena

Razred vsebuje tudi get metode za vse spremenljivke, preko katerih lahko drugi elementi igre dostopajo do njih. Primer take get metode lahko vidite v Kodi 20.

**public** **boolean** getDesno() {

**return** desno;

}

Koda : metoda za pridobivanje stanja tipke

## RAZRED SLIKE

Razred Slike je zadolžen za nalaganje in shranjevanje slik. Shranjuje slike vseh ozadij, slike vseh različnih kock in špic, slike vseh elementov menija ter vse slike igralca. Slike igralca shranjuje kot tabele, za animacijo mirovanja, hoje in skoka, za levo in desno stran. Vse slike so objekta BufferedImage in imajo stopnjo zaščite public, tako da razred ne potrebuje get metod. Postopek nalaganja in shranjevanja slik lahko najdete na strani 11.

## RAZRED ZVOK

Razred Zvok je odgovoren za nalaganje, shranjevanje in predvajanje zvočnih efektov. Postopek nalaganja in shranjevanja zvokov lahko najdete na strani15. Razred hrani zvočne posnetke kot objekte Clip, naloži pa jih z objektom AudioInputStream. Poleg predvajanja zvokov pa razred skrbi tudi za uravnavanje glasnosti zvokov. Glasnost zvokov je mogoče spreminjati z metodo setGlasnost(double d), ki kot argument prejme realno število med 0 in 1. Nato izračuna spremenljivko float db, po formuli s spletnega vira. (18) (formula je podrobneje razložena na strani15.

dB = (float) (Math.log(d) / Math.log(10.0) \* 20.0);

Metoda prilagodi glasnost zvokov s pomočjo razreda FloatControl. Objektu priredimo vrednost zvok.getControl(FloatControl.Type.MASTER\_GAIN), nato pa pokličemo metodo setValue(dB). Metodo lahko vidite v kodi 21.

**public** **void** setGlasnost(**double** d) {

**float** dB = (**float**) (Math.log(d) / Math.log(10.0) \* 20.0);

FloatControl kontrola = (FloatControl) klik.getControl(FloatControl.Type.**MASTER\_GAIN**);

kontrola.setValue(dB);

kontrola = (FloatControl) hover.getControl(FloatControl.Type.**MASTER\_GAIN**);

kontrola.setValue(dB);

kontrola = (FloatControl) jump.getControl(FloatControl.Type.**MASTER\_GAIN**);

kontrola.setValue(dB);

kontrola = (FloatControl) death.getControl(FloatControl.Type.**MASTER\_GAIN**);

kontrola.setValue(dB);

kontrola = (FloatControl) fall.getControl(FloatControl.Type.**MASTER\_GAIN**);

kontrola.setValue(dB);

}

Koda : metoda setGlasnost

## RAZRED NIVOJI

Razred Nivoji je odgovoren za nalaganje in shranjevanje podatkov o nivojih. Vsak nivo je predstavljen kot dvodimenzionalna tabela celih števil med 0 in 10. Več informacij o nivojih lahko dobite v poglavju Načrtovanje programa. Poleg tega vsebuje datoteka tudi startno pozicijo igralca. Poti do datotek so shranjene v tabeli String [] pot. Dvodimenzionalne tabele so shranjene na seznamih (angl. ArrayList). Prav tako so na seznamih shranjene začetne pozicije igralca. Istoležni zapisi na teh seznamih pripadajo enemu nivoju.

V konstruktorju razreda najprej preberemo prvo vrstico datoteke, ki vsebuje startne koordinate igralca. Nato v dvodimenzionalni zanki preberemo vse vrstice tabele in jih shranimo v obliki Dvodimenzionalnega seznama. Ker je dvodimenzionalni seznam po branju obrnjen (vrstice so stolpci in stolpci vrstice), ga moramo transponirati. Hkrati ga tudi pretvorimo v dvodimenzionalno tabelo, za lažji dostop in shranjevanje. Konstruktor razreda je mogoče videti v kodi 22.

**public** Nivoji() **throws** NumberFormatException, IOException {

**for**(**int** z=0; z<pot.length; z++) {

ArrayList<ArrayList<Integer>> datoteka = **new** ArrayList<ArrayList<Integer>>();

String niz = **null**;

BufferedReader br = **new** BufferedReader(**new** FileReader(pot[z]));

//v prvi vrstici sta zapisani startni koordinati

niz = br.readLine();

String [] vrstica0 = niz.split(";");

igralecX.add(Integer.parseInt(vrstica0[0]));

igralecY.add(Integer.parseInt(vrstica0[1]));

**while**((niz = br.readLine())!=**null**) {

String [] vrstica\_String = niz.split(";");

ArrayList<Integer> vrstica = **new** ArrayList<Integer>();

**for**(**int** i=0; i<vrstica\_String.length; i++) {

vrstica.add(Integer.parseInt(vrstica\_String[i]));

}

datoteka.add(vrstica);

}

br.close();

//izračunamo število vrstic in stolpcev

**int** yVelikost = datoteka.size();

**int** xVelikost = 0;

**for**(**int** i=0; i<datoteka.size(); i++) {

**if**(datoteka.get(i).size()>xVelikost)

xVelikost = datoteka.get(i).size();

}

Integer [][] nivo = **new** Integer[xVelikost][yVelikost];

**for**(**int** x=0; x<xVelikost; x++) {

**for**(**int** y=0; y<yVelikost; y++) {

nivo[x][y] = datoteka.get(y).get(x);

}

}

nivoji.add(nivo);

}

}

Koda : konstruktor razreda Nivoji

Razred vsebuje tudi metode getNivo(int index), getIgralecX(int index) in getIgralecY(int index), ki vrnejo podatke o nivoju, kjer je index številka nivoja. Vsebuje tudi metodo getStNivojev(), ki vrne skupno število vseh nivojev.

## RAZRED JEZIK

Razred Jezik je zadolžen za shranjevanje trenutnega izbranega jezika. Razred ima za vsako besedo, ki se bo morala spreminjati glede na izbran jezik, tabelo nizov z vsemi prevodi. Istoležni nizi v tabelah pripadajo istemu jeziku. Razred hrani celo število trenutniJezik, ki je eden izmed možnih indeksov v tabelah. V moji igri bodo možni trije različni jeziki: Slovenščina, Angleščina in Nemščina, lahko pa bi se jih dodalo tudi več. Slovenščina ima indeks 0, Angleščina 1 in Nemščina 2. Razred ima metodi setJezik(int x), s katero nastavimo številko trenutnega jezika in getTrenutni(), ki vrne številko trenutnega jezika.

## RAZRED NASTAVITVE

Razred Nastavitve je odgovoren za beleženje in posredovanje nastavitev igre. Nastavitve, ki bodo shranjene so glasnost in jezik. Nastavitve bodo shranjene na tekstovni datoteki. Vsaka vrednost neke nastavitve bo zapisana v svoji vrstici. V konstruktorju se z datoteke nastavitve preberejo in shranijo v spremenljivke s pomočjo razreda BufferedReader. Razred ima za spreminjanje nastavitev metodi shraniGlasnost(double glasnost)in shraniJezik(int jezik). Ko spremenimo glasnost, metoda spremeni glasnost tudi v razredu Zvok. Enako tudi metoda shraniJezik pokliče metodo setJezik. Metodi lahko vidite v spodnji kodi.

**public** **void** shraniGlasnost(**double** glasnost) **throws** IOException {

**this**.glasnost = glasnost;

**this**.igra.getZvok().setGlasnost(glasnost);

shraniDatoteko();

}

**public** **void** shraniJezik(**int** jezik) **throws** IOException {

**this**.jezik = jezik;

**this**.igra.getJezik().setJezik(jezik);

shraniDatoteko();

}

Koda : metodi shraniGlasnost in shraniJezik

Vsakič, ko spremenimo nastavitve se morajo te zabeležiti tudi na datoteko, zato se pokliče metoda shraniDatoteko(). Metoda s pomočjo BufferedWriter na datoteko zapiše nastavitve. Metodo lahko vidite v Kodi 24.

**public** **void** shraniDatoteko() **throws** IOException {

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(pot));

bw.write("Glasnost:"+glasnost);

bw.newLine();

bw.write("Jezik:"+jezik);

bw.close();

}

Koda : metoda shraniDatoteko

## ELEMENTI MENIJEV

Elementi menijev so vsi elementi igre preko katerih je mogoča interakcija z igro. Delujejo kakor elementi opisnega jezika HTML, ali podrazredi JComponent v jeziku Java. V mojem programu bom naredil Besedilo, Gumb, Drsnik in SpustniMeni. Vsi bodo podrazredi abstraktnega razreda KomponentaMenija. Konstruktor razreda bo kot argumente vzel x in y pozicijo ter objekt Igra. Razred ima tudi abstraktni metodi posodobiStanje() in narisi(Graphics g).

### BESEDILO

Besedilo je najenostavnejši element. Poleg objekta Igra in x ter y pozicije konstruktor vzame še velikost besedila, barvo besedila in tabelo nizov vsebina. Velikost je celo število barva pa je objekt razreda Color. Tabela nizov je ena izmed tabel iz razreda Jezik (str. 27). Razred ima tudi objekta Font in AttributedString s pomočjo katerih lahko oblikujemo besedilo. Font inicializiramo kot: new Font("Arial",Font.BOLD, velikost).

**public** **void** narisi(Graphics g) {

text = **new** AttributedString(vsebina[igra.getJezik().getTrenutni()]);

text.addAttribute(TextAttribute.**FONT**, font);

g.setColor(barva);

g.drawString(text.getIterator(), x, y);

}

Koda : metoda narisi razreda Besedilo

V metodi narisi ustvarimo AttributedString z vsebino v pravilnem jeziku in mu z metodo addAttribute spremenimo font. Nato z metodo drawString narišemo tekst na pravilne koordinate. Besedilu se stanje ne spreminja, zato je metoda posodobiStanje() prazna. Metodo lahko vidite v Kodi 25.

### GUMB

Razred Gumb ima v konstruktorju argumente: String [] vsebina, int x, int y, int dolzina, int visina, int velikost, BufferedImage []slika, Igra igra. Vsebina je tako kot pri besedilu podana z tabelo, saj igra podpira več jezikov. Velikost označuje velikost pisave. Argument konstruktorja je tudi tabela dveh slik. Ena je normalno stanje gumba, druga pa je stanje, ko je nad gumbom kurzor.

V metodi posodobiStanje() program določi, ali je nad gumbom kurzor in ali je bil gumb pritisnjen. Razred ima boolean spremenljivko hover, ki jo posodobi na true, če sta x in y pozicija miške v območju gumba. Razred ima tudi spremenljivko miskaPrej, ki hrani vrednost gumba miške iz prejšnjega kroga. Če je bila miška v prejšnjem krogu pritisnjena, v tem pa ni več in če se miška nahaja v območju gumba, to pomeni, da je bil gumb pritisnjen. Če program ugotovi, da je bil gumb pritisnjen posodobi vrednost spremenljivke jePritisnjen na true. Metoda tudi pokliče metodo zvok(), ki zaigra zvok hover, ko kurzor vstopi v območje gumba in zvok klik, ko je gumb pritisnjen. Metodo posodobiStanje() lahko vidite v spodnji kodi.

**public** **void** posodobiStanje() {

**if**(igra.getTipke().getX() > x && igra.getTipke().getX() < x+dolzina && igra.getTipke().getY() > y && igra.getTipke().getY() < y+visina) {

hover = **true**;

}

**else** {

hover = **false**;

}

**if**(hover)

**if**(miskaPrej == **true**) {

**if**(igra.getTipke().getMiska1() == **false**) {

pritisnjen = **true**;

}

**else**

pritisnjen = **false**;

}

**else**

pritisnjen = **false**;

**else**

pritisnjen = **false**;

miskaPrej = igra.getTipke().getMiska1();

zvok();

}

Koda : metoda posodobiStanje razreda Gumb

Metoda narisi(Graphics g) nariše gumb na sliko. Pri tem upošteva, če je kurzor nad gumbom in nariše primerno sliko. Metoda tudi oblikuje besedilo z objektoma AttributedString in Font (Za razlago postopka oblikovanja besedila glej podnaslov 4.9.1). Besedilo mora biti narisano na sredino gumba zato program izračuna x in y pozicijo besedila s pomočjo razreda FontMetrics, ki lahko določi dolžino in višino nekega besedila v pikslih, glede na vsebino in font. Vsak niz ima višino in dolžino, ki sta odvisni od fonta in vsebine niza. Višina je enaka višini najvišjega znaka, dolžina pa je enaka vsoti dolžin vseh znakov. (23) Metodo narisi lahko vidite v Kodi 27.

**public** **void** narisi(Graphics g) {

**if**(!hover) {

g.drawImage(slika[0],x,y,dolzina,visina,**null**);

g.setColor(barvaBesedila);

}

**else** {

g.drawImage(slika[1],x,y,dolzina,visina,**null**);

g.setColor(barvaBesedilaAlt);

}

text = **new** AttributedString(vsebina[igra.getJezik().getTrenutni()]);

font = **new** Font("Arial",Font.**BOLD**, velikost);

text.addAttribute(TextAttribute.**FONT**, font);

//Izračunamo pozicijo teksta

FontMetrics m = g.getFontMetrics(font);

**int** xText = x + (dolzina-m.stringWidth(vsebina[igra.getJezik().getTrenutni()]))/2;

**int** yText = y + (visina - m.getHeight())/2 + m.getAscent();

g.drawString(text.getIterator(), xText, yText);

}

Koda : metoda narisi razreda Gumb

Razred Gumb ima tudi metodo jePritisnjen(), ki vrne vrednost spremenljivke pritisnjen.

### DRSNIK

Konstruktor razreda Drsnik ima argumente int x, int y, int dolzina, int visina, double vrednost, Igra igra. Vrednost je realno število med 0 in 1 in označuje trenutno pozicijo drsnika. Če je vrednost enaka 0 je gumb drsnika v skrajni levi legi, in v skrajni desni legi, če je vrednost enaka 1. Iz vrednosti konstruktor izračuna tudi dejansko pozicijo gumba drsnika relativno glede na x pozicijo drsnika. Konstruktor določi tudi dolžino in višino gumba.

V metodi posodobiStanje() najprej preverimo, če se kurzor nahaja nad gumbom drsnika in ustrezno posodobimo spremenljivko hover. Nato preverimo če je uporabnik pritisnil gumb. Če miška v prejšnjem krogu še ni bila pritisnjena, shranimo njen položaj v spremenljivko miskaXPrejsnji. Na true posodobimo tudi spremenljivki pritisnjen in slediMiski. Če miška ni pritisnjena spremenimo slediMiski na false. Nato, če je miška pritisnjena izračunamo razliko med trenutno in prejšnjo pozicijo miške in za toliko premaknemo tudi gumb drsnika. Nazadnje preverimo še, če se je gumb morda premaknil izven obsega drsnika in izračunamo novo vrednost drsnika. Metodo lahko vidite v Kodi 28.

**boolean** hover = **false**;

**boolean** pritisnjen = **false**;

**boolean** slediMiski = **false**;

**int** miskaXPrejsnji;

**public** **void** posodobiStanje() {

**if**(igra.getTipke().getX() > x+pozicija && igra.getTipke().getX() < x+pozicija+dolzinaGumb && igra.getTipke().getY() > y-(visinaGumb-visina)/2 && igra.getTipke().getY() < y-(visinaGumb-visina)/2+visinaGumb) {

hover = **true**;

}

**else** {

hover = **false**;

}

**if**(hover && igra.getTipke().getMiska1()) {

//če miška v prejšnjem krogu še ni bila pritisnjena zabeležimo njen položaj

**if**(!pritisnjen) {

miskaXPrejsnji = igra.getTipke().getX();

slediMiski = **true**;

pritisnjen = **true**;

}

}

**else**

pritisnjen = **false**;

**if**(!igra.getTipke().getMiska1())

slediMiski = **false**;

**if**(slediMiski) {

pozicija += igra.getTipke().getX() - miskaXPrejsnji;

miskaXPrejsnji = igra.getTipke().getX();

}

**if**(pozicija<0)

pozicija = 0;

**if**(pozicija>dolzina-dolzinaGumb)

pozicija = dolzina-dolzinaGumb;

vrednost = pozicija / (dolzina-dolzinaGumb);

}

Koda : metoda posodobiStanje razreda Drsnik

V metodi narisi(Graphics g) ni posebnosti. Narišemo dva pravokotnika. Eden predstavlja drsnik, drugi pa gumb drsnika. Metodo lahko vidite v spodnji kodi.

**public** **void** narisi(Graphics g) {

g.setColor(Color.**black**);

g.fillRect(x, y, dolzina, visina);

g.setColor(**new** Color(173,166,158));

g.fillRect((**int**)(x+pozicija), y-(visinaGumb-visina)/2, dolzinaGumb,visinaGumb);

}

Koda : metoda narisi razreda Drsnik

V igri drsnik izgleda tako:



Slika : Drsnik

Razred ima tudi metodo getVrednost(), ki vrne trenutno vrednost drsnika. Preko te metode ostali elementi igre dostopajo do vrednosti nekega drsnika.

### SPUSTNI MENI

Konstruktor razreda ima argumente: int x, int y, int dolzina, int visina, int velikost, String [] izbira, int vrednost, Igra igra. Dolzina in visina sta velikost ene celice spustnega menija, velikost pa je velikost pisave. Izbira je tabela nizov, izmed katerih lahko uporabnik izbira, vrednost pa je indeks izbranega niza. V konstruktorju tudi oblikujemo besedila z objektoma AttributedString in Font (glej stran 29).

V metodi posodobiStanje() posodobimo stanje spustnega menija. Najprej preverimo, če je kurzor nad spustnim menijem in posodobimo spremenljivko hover. Nato določimo, če je uporabnik odprl spustni meni. Spremenljivko odprt spremenimo na true, če je kurzor nad spustnim menijem in je bil gumb miške v prejšnjem krogu pritisnjen, v tem pa ni. V tem primeru tudi zaigramo zvok hover. Če je spustni meni že odprt, in je uporabnik kliknil gumb miške v območju odprtega menija, potem program izračuna, na katero izbiro je kliknil uporabnik kot: vrednost = (igra.getTipke().getY()-y-visina)/visina in to vrednost shrani. Program v tem primeru tudi zaigra zvok klik. Nazadnje, če je spustni meni odprt in je uporabnik kliknil gumb miške izven območja spustnega menija, se spustni meni zapre in program zaigra zvok hover. Metodo lahko vidite v Kodi 30.

**private boolean** miskaPrej = **false**;

**private boolean** hover = **false**;

**private boolean** odprt = **false**;

**public** **void** posodobiStanje() {

**if**(igra.getTipke().getX() > x && igra.getTipke().getX() < x+dolzina && igra.getTipke().getY() > y && igra.getTipke().getY() < y+visina) {

hover = **true**;

}

**else** {

hover = **false**;

}

**if**(hover && miskaPrej)

**if**(igra.getTipke().getMiska1() == **false**) {

odprt = **true**;

igra.getZvok().zvok\_Hover();

}

**if**(odprt && miskaPrej) {

**if**(igra.getTipke().getMiska1() == **false**)

**if**(igra.getTipke().getX() > x && igra.getTipke().getX() < x+dolzina && igra.getTipke().getY() > y+visina && igra.getTipke().getY() < y+(izbira.length+1)\*visina){

vrednost = (igra.getTipke().getY()-y-visina)/visina;

igra.getZvok().zvok\_Klik();

}

}

**if**(!hover && odprt)

**if**(miskaPrej == **true**)

**if**(igra.getTipke().getMiska1() == **false**) {

odprt = **false**;

igra.getZvok().zvok\_Hover();

}

miskaPrej = igra.getTipke().getMiska1();

}

Koda : metoda posodobiStanje razreda SpustniMeni

Metoda narisi(Graphics g) nariše spustni meni različno, glede na to ali je zaprt ali odprt. Če je spustni meni zaprt, metoda nariše le sliko zaprtega menija ter niz, ki je trenutno izbran. Niz tudi poravna z sliko na y osi. Višino besedila dobimo s pomočjo FontMetrics. Če je spustni meni odprt, metoda najprej nariše sliko odprtega menija. Nato v zanki nariše vsako celico z izbiro. Vsako celico nariše eno višino nižje kot prejšnjo. Tudi v teh celicah je besedilo poravnano. Metodo lahko vidite v spodnji kodi.

**public** **void** narisi(Graphics g) {

**if**(!odprt) {

g.drawImage(Slike.spustni\_meni[0],x,y,dolzina,visina,**null**);

FontMetrics m = g.getFontMetrics(font);

**int** xText = x + 10;

**int** yText = y + (visina - m.getHeight())/2 + m.getAscent();

g.setColor(barva);

g.drawString(text[vrednost].getIterator(), xText, yText);

}

**else** {

g.drawImage(Slike.spustni\_meni[1],x,y,dolzina,visina,**null**);

**for**(**int** i=0; i<izbira.length; i++) {

g.setColor(ozadje);

g.fillRect(x, y+(i+1)\*visina, dolzina, visina);

FontMetrics m = g.getFontMetrics(font);

**int** xText = x + 10;

**int** yText = y+(visina-m.getHeight())/2+m.getAscent()+ (i+1)\*visina;

g.setColor(barva);

g.drawString(text[i].getIterator(), xText, yText);

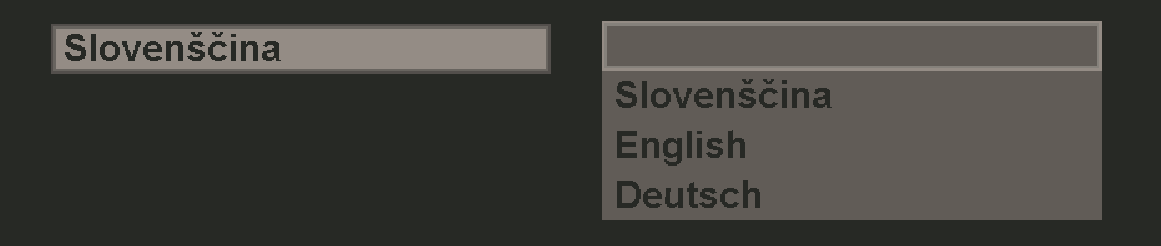
}

}

}

Koda : metoda narisi razreda SpustniMeni

V igri spustni meni izgleda tako:



Slika : Spustni meni

Razred vsebuje tudi metodo getVrednost(). Ki vrne indeks trenutne izbire.

## STANJA IGRE

Razred Stanje je abstrakten razred, ki ga dedujejo vsi drugi razredi stanj. Razred ima abstraktni metodi posodobiStanje() in narisi(Graphics g). Poleg tega bo razred hranil tudi trenutnoStanje in predhodnoStanje. Druge metode razreda so še setStanje(Stanje stanje), s pomočjo katere spremenimo trenutno stanje igre ter getStanje() in getPredhodnoStanje(), ki vrneta trenutno stanje in predhodno stanje igre.

### STANJE MENI

Razred Stanje\_Meni je odgovoren za hranjenje, posodabljanje in risanje elementov glavnega menija. Glavni meni bo vseboval gumbe igraj, nastavitve, pomoc in izhod. Poleg tega bo razred imel tudi BufferedImage ozadje. Inicializacijo ozadja in gumbov lahko vidite v spodnji kodi.

ozadje = Slike.ozadje\_meni;

igraj = **new** Gumb(Jezik.**igraj**,390,240,420,70,50,Slike.gumb,igra);

nastavitve = **new** Gumb(Jezik.**nastavitve**,390,350,420,70,50,Slike.gumb,igra);

izhod = **new** Gumb(Jezik.**izhod**,390,460,420,70,50,Slike.gumb,igra);

pomoc = **new** Gumb(Jezik.**vprasaj**,1100,20,80,80,60,Slike.gumb\_kvadrat,igra);

Koda : inicializacija gumbov glavnega menija

V metodi posodobiStanje() pokličemo istoimensko metodo vsakega gumba. Poleg tega v tej metodi preverimo stanje vsakega gumba. Če je kateri gumb pritisnjen izvedemo ukaz, ki pripada temu gumbu. Če uporabnik pritisne gumb igraj, zamenjamo stanje na IzbiraNivoja. Če pritisne nastavitve, spremenimo stanje na Nastavitve in če pritisne pomoc na Pomoč. Če uporabnik pritisne gumb izhod, pokličemo metodo ustaviIgro(). Pogojne stavke lahko vidite v Kodi 33. Enako kot v metodi posodobiStanje(), tudi v metodi narisi pokličemo metode narisi vseh gumbov, pri čemer jim kot argument podamo objekt Graphics.

**if**(igraj.jePritisnjen())

igra.spremeniStanjeNaIzbiraNivoja();

**if**(nastavitve.jePritisnjen())

igra.spremeniStanjeNaNastavitve();

**if**(pomoc.jePritisnjen())

igra.spremeniStanjeNaPomoc();

**if**(izhod.jePritisnjen())

igra.ustaviIgro();

Koda : preverjanje stanja gumbov menija

Izgled stanja Meni lahko vidite na sliki 8.



Slika : Glavni meni

### STANJE NASTAVITVE

Razred Stanje\_Nastavitve je odgovoren za hranjenje, posodabljanje in risanje elementov nastavitev. Nastavitve bodo vsebovale gumba shrani in nazaj, drsnik za spreminjanje glasnosti, spustni meni za spremembo jezika in besedila, ki označujejo različne nastavitve. Nastavitve imajo tudi svoje ozadje. Inicializacijo elementov lahko vidite v spodnji kodi.

nazaj = **new** Gumb(Jezik.**nazaj**, 10, 10, 180, 50, 30,Slike.gumb,igra);

besedilo1 = **new** Besedilo(Jezik.**glasnost**,20,150,40,**new** Color(173,166,158),igra);

glasnost = **new** Drsnik(20,180,400,20,igra.getNastavitve().getGlasnost(),igra);

besedilo2 = **new** Besedilo(Jezik.**jezik**,20,260,40,**new** Color(173,166,158),igra);

jezik = **new** SpustniMeni

(20,300,400,40,30,Jezik.**jeziki**,igra.getNastavitve().getJezik(),igra);

shrani = **new** Gumb(Jezik.**shrani**,20,500,200,60,40,Slike.gumb,igra);

Koda : inicializacija elementov nastavitev

V metodi posodobiStanje() posodobimo stanja vseh elementov nastavitev. Nato preverimo, če je kateri izmed gumbov pritisnjen. Če uporabnik pritisne gumb nazaj se stanje igre spremeni na Stanje\_Meni. Prav tako se program vrne na glavni meni, če uporabnik pritisne gumb esc na tipkovnici. Če uporabnik pritisne gumb shrani, program pokliče metodo shraniNastavitve(), ki shrani nastavitve glasnosti in jezika. Metodo lahko vidite v spodnji kodi.

**public** **void** shraniNastavitve() **throws** IOException {

igra.getNastavitve().shraniGlasnost(glasnost.getVrednost());

igra.getNastavitve().shraniJezik(jezik.getVrednost());

}

Koda : metoda shraniNastavitve

V metodi narisi(Graphics g) program nariše ozadje, nato pa pokliče metodo narisi vsakega elementa nastavitev, pri čemer jim naprej poda objekt Graphics.

Izgled stanja Nastavitve v igri lahko vidite na sliki 9.



Slika : Nastavitve

### STANJE POMOČ

Razred Stanje\_Pomoc je manjši od drugih razredov stanj, saj vsebuje le tabelo slik in gumb nazaj. V tabeli slik je za vsak jezik ena verzija slike. Stanje prikaže le sliko z enakim indeksom, kot je indeks trenutnega jezika. V metodi posodobiStanje() program posodobi stanje gumba nazaj in preveri, če je morda pritisnjen. V metodi narisi(Graphics g) program nariše ustrezno sliko, glede na indeks ki ga vrne igra.getJezik().getTrenutni() in gumb nazaj. Gumb inicializiramo kot:

nazaj = **new** Gumb(Jezik.**nazaj**, 20, 20, 200, 60, 50,Slike.gumb,igra);

Izgled stanja Pomoc lahko vidite na sliki 10.



Slika : Pomoč

### STANJE IZBIRA NIVOJA

Razred je odgovoren za hranjenje, posodabljanje in risanje elementov stanja izbire nivoja. Stanje vsebuje gumb nazaj ter 10 oštevilčenih gumbov, enega za vsak nivo. Če uporabnik pritisne gumb nazaj, se stanje spremeni na glavni meni. Če uporabnik pritisne katerega od desetih gumbov, program odpre ustrezni nivo igre. V Kodi 36 lahko vidite inicializacijo gumbov.

nazaj = **new** Gumb(Jezik.**nazaj**, 10, 10, 200, 60, 50,Slike.gumb, igra);

nivo0 = **new** Gumb(Jezik.**\_1**, 145, 130, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo1 = **new** Gumb(Jezik.**\_2**, 335, 130, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo2 = **new** Gumb(Jezik.**\_3**, 525, 130, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo3 = **new** Gumb(Jezik.**\_4**, 715, 130, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo4 = **new** Gumb(Jezik.**\_5**, 905, 130, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo5 = **new** Gumb(Jezik.**\_6**, 145, 320, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo6 = **new** Gumb(Jezik.**\_7**, 335, 320, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo7 = **new** Gumb(Jezik.**\_8**, 525, 320, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo8 = **new** Gumb(Jezik.**\_9**, 715, 320, 150, 150, 100,Slike.gumb\_kvadrat, igra);

nivo9 = **new** Gumb(Jezik.**\_10**, 905, 320, 150, 150, 100,Slike.gumb\_kvadrat, igra);

Koda : inicializacija gumbov izbire nivoja

V metodi posodobiStanje() program posodobi stanja vseh gumbov. Program nato preveri, če je kateri izmed gumbov pritisnjen. Spodaj lahko vidite preverjanje za gumb nazaj in za nek gumb nivoja N.

**if**(nazaj.jePritisnjen()) {

igra.spremeniStanjeNaMeni();

}

**if**(nivoN.jePritisnjen()) {

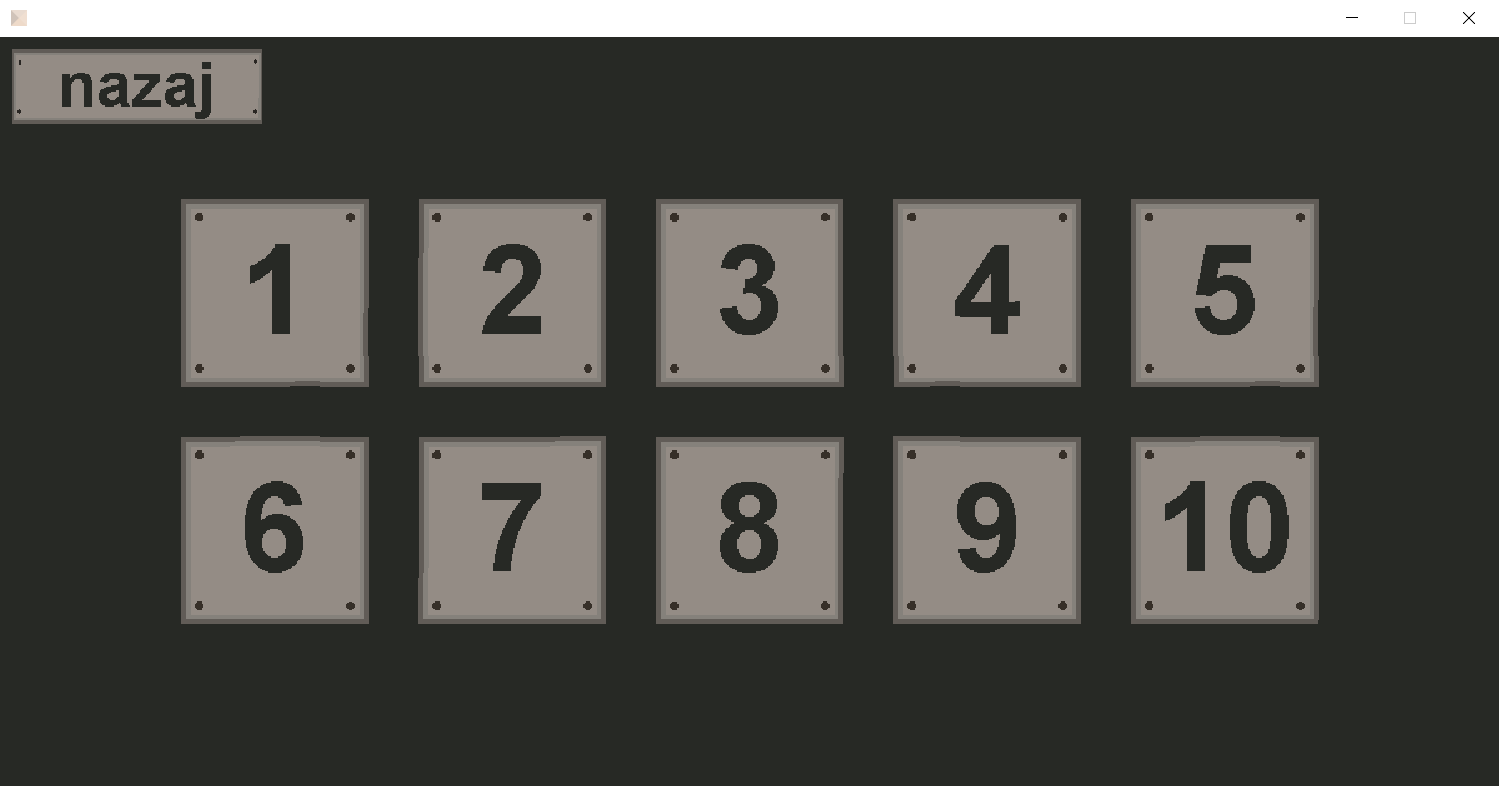
igra.spremeniStanjeNaNivo(N);

}

Koda : preverjanje stanja gumbov

V metodi narisi(Graphics g) program nariše ozadje, ki je v tem stanju enobarvno. Nato program pokliče metode narisi vseh gumbov, pri čem er jim poda objekt Graphics.

V igri stanje IzbiraNivoja izgleda tako:



Slika : Izbira nivoja

### STANJE NIVO

Stanje Nivo je najkompleksnejše izmed stanj in predstavlja glavno stanje igre. V tem stanju uporabnik dejansko igra igro. Stanje ima tri lastna podstanja, ki pa nimajo svojih razredov. Prvo izmed teh stanj je navadno stanje igre, ko uporabnik premika igralca. Drugo stanje je meni, ki se odpre, če uporabnik pritisne tipko esc na tipkovnici. Preko tega menija lahko igralec igro ponastavi, se vrne na izbiro nivoja, se vrne na glavni meni ali odpre pomoč. Zadnje podstanje se odpre, ko igralec zmaga. Igralec se lahko nato odloči, ali bo nadaljeval z naslednjim nivojem, ali pa se vrnil na glavni meni.

Konstruktor razreda kot argumente prejme objekt Igra in številko nivoja stNivoja. Nato konstruktor inicializira vse elemente igre. Za navadno podstanje inicializira objekte Svet (str. 49), Igralec (str. 51), Kamera (str. 63), Ozadje (str. 63) in Podatki. Objekt Podatki je zadolžen za hranjene podatkov o drugih objektih igre. Te mu podamo z metodami setSvet, setIgralec in setKamera. Za podsatnje menija, konstruktor inicializira gumbe ponastavitev, izbiraNivoja, glavniMeni in pomoc. Nazadnje konstruktor zabeleži, ali je trenutni nivo zadnji. Če nivo ni zadnji, se inicializira še gumb naslednjiNivo. Konstruktor razreda lahko vidite v kodi 38.

**public** Stanje\_Nivo(Igra igra, **int** stNivoja) **throws** IOException {

**this**.igra = igra;

**this**.stNivoja = stNivoja;

//inicalizacija elementov igre

**this**.podatki = igra.getPodatki();

svet = **new** Svet(igra, stNivoja);

podatki.setSvet(svet);

igralec1 = **new** Igralec(igra.getNivoji().getIgralecX(stNivoja), igra.getNivoji().getIgralecY(stNivoja), igra);

podatki.setIgralec(igralec1);

kamera = **new** Kamera(igra);

podatki.setKamera(kamera);

**this**.ozadje = **new** Ozadje(Slike.ozadje,igra);

//inicializacija gumbov

ponastavitev = **new** Gumb(Jezik.**zacni\_znova**,390,120,420,70,50,Slike.gumb,igra);

izbiraNivoja = **new** Gumb(Jezik.**izbira\_nivoja**,390,230,420,70,50,Slike.gumb,igra);

glavniMeni = **new** Gumb(Jezik.**glavni\_meni**,390,340,420,70,50,Slike.gumb,igra);

pomoc = **new** Gumb(Jezik.**vprasaj**,1100,20,80,80,60,Slike.gumb\_kvadrat,igra);

//če nivo ni zadnji se inicializira gumb naslednjiNivo

**if**(igra.getNivoji().getStNivojev()-1 == stNivoja) {

zadnjiNivo = **true**;

}

**else** {

naslednjiNivo = **new** Gumb(Jezik.**naslednji\_nivo**, 390, 190, 420, 70, 50,Slike.gumb, igra);

zadnjiNivo = **false**;

}

}

Koda : konstruktor razreda Stanje\_Nivo

V metodi posodobiStanje() program posodobi elemente tistega podstanja, ki je trenutno aktivno. Razred ima v ta namen spremenljivki meniZmaga, ki se spremeni na true, ko igralec zmaga in pavza, ki se spremeni na true, ko igralec ustavi igro. Metoda najprej preveri, ali je igralec pritisnil tipko esc. Če jo je pritisnil spremeni vrednost pavza na obratno vrednost in pokliče metodo prekiniIgro(), ki pripravi trenutno sliko igre. Ta bo služila kot ozadje za meni. Metoda je natančneje razložena na strani 42. Nato program nadaljuje z preverjanjem spremenljivk meniZmaga in pavza. Če je spremenljivka meniZmaga enaka true, program posodobi stanji gumbov naslednjiNivo in glavniMeni. Če je nivo zadnji je gumb naslednjiNivo enak null, zato se tudi njegovo stanje ne posodobi. Program tudi preveri, če je kateri od gumbov pritisnjen, in izvede ustrezen ukaz. Če spremenljivka meniZmaga ni enaka true, metoda preveri vrednost spremenljivke pavza. Če je ta enaka false posodobi stanja objektov Igralec, Svet in Kamera. Pokliče tudi metodi zmaga() in smrt(), ki preverita ali je igralec bodisi zmagal ali umrl. Metodi sta natančneje razloženi na strani 43. Nazadnje, če je vrednost spremenljivke pavza enaka true, metoda posodobi stanja gumbov ponastavitev, izbiraNivoja, glavniMeni in pomoc. Metoda preveri, ali je kateri izmed gumbov morda pritisnjen in izvede ustrezni ukaz. Metodo posodobiStanje() lahko vidite v kodi 39.

**private boolean** pavza = **false**;

**private boolean** dovoliSprememboPavze = **false**;

**private boolean** meniZmaga = **false**;

**public** **void** posodobiStanje() **throws** IOException {

//preverimo ali je igralec ustavil igro

**if**(igra.getTipke().getEsc()) {

**if**(dovoliSprememboPavze) {

pavza = !pavza;

dovoliSprememboPavze = **false**;

prekiniIgro();

}

}

**else** {

dovoliSprememboPavze = **true**;

}

**if**(meniZmaga) {// če je igralec zmagal posodobi stanje menija ob zmagi

**if**(!zadnjiNivo) {

naslednjiNivo.posodobiStanje();

**if**(naslednjiNivo.jePritisnjen())

igra.spremeniStanjeNaNivo(stNivoja+1);

}

glavniMeni.posodobiStanje();

**if**(glavniMeni.jePritisnjen())

igra.spremeniStanjeNaMeni();

}

**else** **if**(!pavza) {// če igra ni ustavljena posodobi vsebino igre

igralec1.posodobiStanje();

svet.posodobiStanje();

kamera.posodobiStanje();

ozadje.posodobiStanje();

zmaga();

smrt();

}

**else** {// če je igra ustavljena posodobi gumbe menija v igri

ponastavitev.posodobiStanje();

izbiraNivoja.posodobiStanje();

glavniMeni.posodobiStanje();

pomoc.posodobiStanje();

**if**(glavniMeni.jePritisnjen())

igra.spremeniStanjeNaMeni();

**if**(izbiraNivoja.jePritisnjen())

igra.spremeniStanjeNaIzbiraNivoja();

**if**(ponastavitev.jePritisnjen()) {

igralec1.ponastavi();

pavza = **false**;

}

**if**(pomoc.jePritisnjen())

igra.spremeniStanjeNaPomoc();

}

}

Koda : metoda posodobiStanje razreda Stanje\_Nivo

Metoda narisi(Graphics g) ima v tem razredu enako logično sestavo kot metoda posodobiStanje(), le da kliče metode narisi(Graphics g) ustreznih objektov. Če je spremenljivka meniZmaga enaka true, metoda nariše gumba naslednjiNivo in glavniMeni. Če je spremenljivka pavza enaka false, metoda nariše objekte Ozadje, Igralec in Svet. Drugače metoda nariše gumbe ponastavitev, izbiraNivoja, glavniMeni in pomoc. Kot ozadje v tem primeru metoda nariše sliko ustavljenaIgra, ki jo ob ustavitvi igre ustvari metoda prekiniIgro().

Metoda prekiniIgro() je poklicana, ko uporabnik prekine igro. Metoda ustvari sliko ustavljenaIgra, ki bo služila kot ozadje menija v igri. Slika je enaka sliki igre v trenutku, ko pokličemo metodo. Poleg tega program celotno sliko prebarva z prosojno barvo, kar da uporabniku še dodatno vedeti, da je igra ustavljena. Če metodo kličemo prvič in je vrednost slike enaka null sliko najprej inicializiramo. Ker se slika v času, ko je igra ustavljena ne spreminja, program deluje veliko hitreje. Metodo lahko vidite spodaj.

**private** BufferedImage ustavljenaIgra;

**public** **void** prekiniIgro() {

**if**(ustavljenaIgra == **null**)

ustavljenaIgra = **new** BufferedImage(igra.getDolzina(),igra.getVisina(), BufferedImage.**TYPE\_INT\_RGB**);

tempG = ustavljenaIgra.getGraphics();

tempG.setColor(Color.**black**);

tempG.fillRect(0,0, igra.getDolzina(),igra.getVisina());

ozadje.narisi(tempG);

igralec1.narisi(tempG);

svet.narisi(tempG);

tempG.setColor(**new** Color(200,200,200,150));

tempG.fillRect(0,0, igra.getDolzina(),igra.getVisina());

tempG.dispose();

}

Koda : metoda prekiniIgro

Metoda zmaga() se pokliče v vsakem krogu zanke in preverja, če je igralec morda zmagal. Metoda igralec.getZmaga() vrne vrednost true, če je igralec zmagal in false, če ni. Če je bila vrednost v prejšnjem krogu enaka false, v tem pa je enaka true, metoda zaigra zvok Zmaga in začne z odštevanjem. Po 120 krogih metoda spremeni vrednost meniZmaga na true, kar odpre meni ob zmagi. Metodo lahko vidite spodaj.

**private boolean** zmaga = **false**;

**public** **void** zmaga() {

**if**(igralec1.getZmaga() && !zmaga) {

odštevanje = 120;

igra.getZvok().zvok\_Zmaga();

}

**if**(zmaga) {

**if**(odštevanje < 0) {

meniZmaga = **true**;

}

odštevanje--;

}

zmaga = igralec1.getZmaga();

}

Koda : metoda zmaga

Metoda smrt() se pokliče vsak krog zanke in preveri, ali je igralec morda umrl. Metoda igralec.getSmrt() vrne vrednost true, če je igralec mrtev in false če ni. Če je bila vrednost v prejšnjem krogu enaka false, v tem pa je enaka true, metoda zaigra zvok Smrt in začne z odštevanjem. Po 120 krogih metoda pokliče ukaz igra.spremeniStanjeNaNivo(stNivoja), kar ponastavi trenutni nivo. Metodo lahko vidite spodaj.

**private boolean** smrt = **false**;

**public** **void** smrt() **throws** IOException {

**if**(igralec1.getSmrt() && !smrt) {

igra.getZvok().zvok\_Smrt();

odštevanje = 120;

}

**if**(smrt) {

**if**(odštevanje < 0) {

igra.spremeniStanjeNaNivo(stNivoja);

}

odštevanje --;

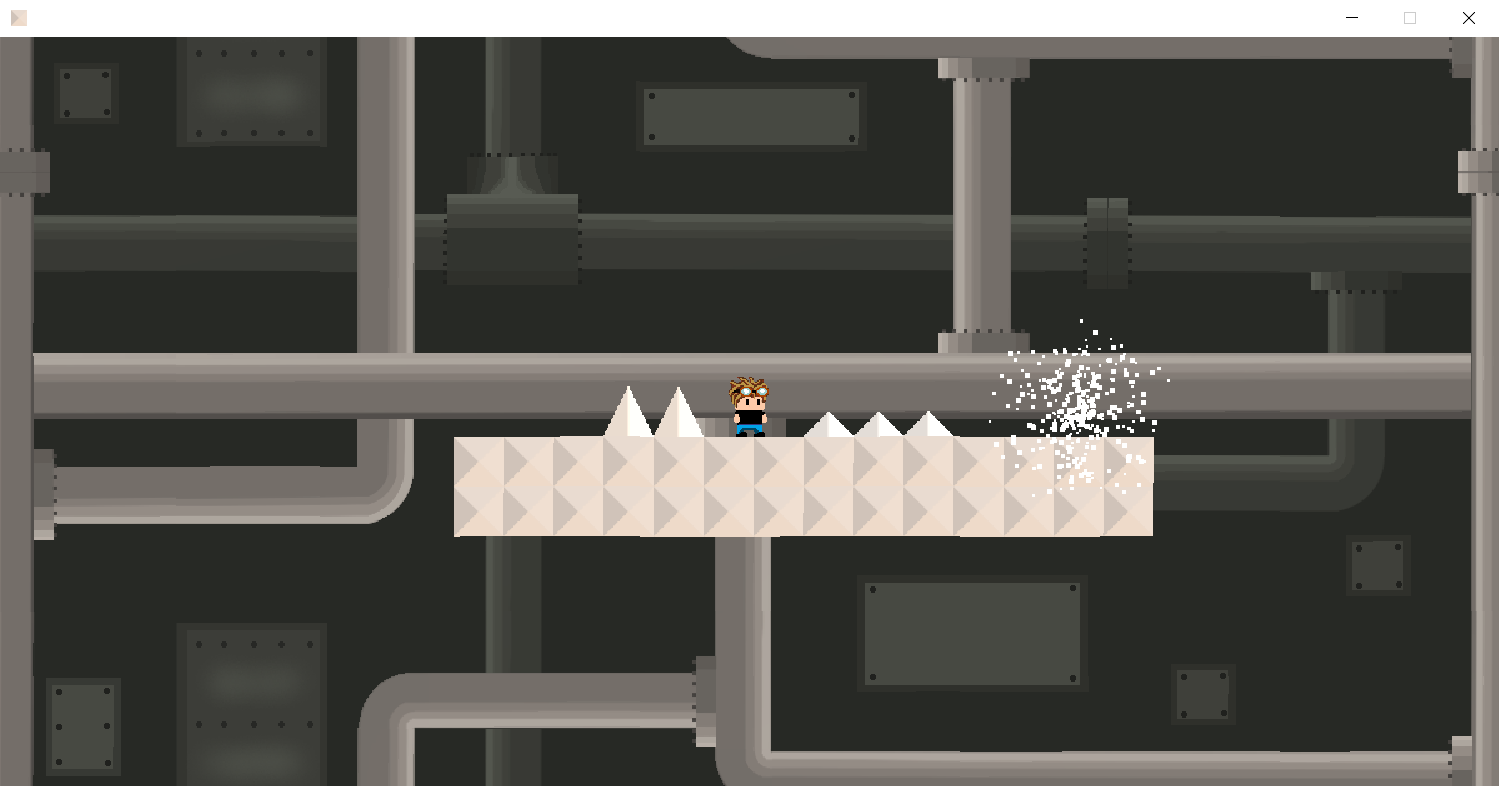
}

smrt = igralec1.getSmrt();

}

Koda : metoda smrt

Izgled stanja Nivo lahko vidite na sliki 12.



Slika : Nivo

Izgled stanja Nivo, ko je igra prekinjena, lahko vidite na sliki 13.



Slika : Nivo - ustavljen

RAZRED PODATKI

Razred Podatki je zadolžen zgolj za hranjenje objektov Igralec, Svet in Kamera. Ti trije objekti bodo med seboj velikokrat menjali podatke. Brez tega razreda pa je dostop do podatkov v tem primeru zelo težaven. Razred Podatki nam zato precej olajša delo. Razred ima metode setIgralec, setSvet, setKamera ter getIgralec, getSvet in getKamera.

## KOCKE

Kocke so glavni sestavni deli nivojev v moji igri. V programu obstajajo 4 različne vrste: Zrak, Opeka, Spica in Cilj. Vsaka vrsta kocke ima svoj razred, ki deduje razred Kocka. Vsaka kocka ima svojo unikatno številko id in svojo sliko. Vsaka kocka ima tudi objekta Igra in Podatki. V konstruktorju razreda inicializiramo te lastnosti. Razred ima tudi metodo narisi(Graphics g), ki nariše sliko kocke na neki poziciji (X, Y). X in Y nista dejanski koordinati na zaslonu, ampak označujeta mesto, ki ga kocka zaseda v nivoju. Metoda ju zato pomnoži z dolžino in višino kocke, da dobi prave koordinate. Metoda sliko nariše z odmikom, ki ga določi razred Kamera (str. 63) Metodo lahko vidite spodaj.

**public** **void** narisi(Graphics g, **int** x, **int** y) {

g.drawImage(**this**.slika,(x\***dolzinaK**)-podatki.getKamera().getXOdmik(),y\***visinaK**-podatki.getKamera().getYOdmik(),**dolzinaK**,**visinaK**,**null**);

}

Koda : metoda narisi razreda Kocka

Razred ima poleg tega metodi getId(), ki vrne id kocke. Razred ima tudi metode, jeTrdno(), jeNevarno(), jeCilj(), in preveriTrk, ki bodo za različne kocke vračale različne logične vrednosti. To nam omogoča koncept polimorfizma. V razredu Kocka vse metode vračajo vrednost false, v dedovanih razredih, pa je to vrednost mogoče spremeniti.

ZRAK

Razred Zrak, je najenostavnejši izmed razredov kock. V njegovem konstruktorju zgolj podamo podatke konstruktorju nadrazreda s ključno besedo super. V tem razredu ne spreminjamo nobenih metod nadrazreda, niti ne dodajamo novih.

OPEKA

Enako kot pri razredu Zrak, tudi v razredu Opeka konstruktor poda podatke konstruktorju nadrazreda. Za razliko od prejšnjega razreda pa v tem spremenimo metodo jeTrdno(). Metoda določa, ali se bo igralec lahko premikal skozi kocko. V tem primeru tega nočemo, zato bo metoda vrnila vrednost true. Drugih metod ne spreminjamo.

ŠPICA

Razred Spica je bolj kompleksen od ostalih razredov kock iz dveh razlogov. Prvi razlog je, da je v igri osem vrst špic, ki zasedajo številke id od 2 do 9. Vsaka špica nima svojega razreda, zato bodo vse združene v enega. V konstruktorju razreda določimo sliko glede na id. To naredimo z stavkom switch:

**switch**(id) {

**case** 2: **this**.slika = Slike.spica1;**break**;

**case** 3: **this**.slika = Slike.spica2;**break**;

**case** 4: **this**.slika = Slike.spica3;**break**;

**case** 5: **this**.slika = Slike.spica4;**break**;

**case** 6: **this**.slika = Slike.spica5;**break**;

**case** 7: **this**.slika = Slike.spica6;**break**;

**case** 8: **this**.slika = Slike.spica7;**break**;

**case** 9: **this**.slika = Slike.spica8;**break**;

}

Koda : določanje orientacije špice

Drugi razlog za kompleksnost je to, da špice za razliko od drugih kock niso kvadratne oblike. Za lažje računanje trkov z igralcem, bo špica zato imela tudi objekt Trikotnik. Ta razred bo hranil koordinate treh točk, ki jih bomo podali v konstruktorju. Razred bo imel tudi metodo za računanje ploščine trikotnika, ki bo kot argumente prejela koordinate treh točk. Metodo lahko vidite v kodi 45.

**public** **static** **double** ploscina(**double** tx1,**double** ty1,**double** tx2,**double** ty2,**double** tx3,**double** ty3) {

**return** 0.5\*Math.abs((tx2-tx1)\*(ty3-ty1)-(tx3-tx1)\*(ty2-ty1));

}

Koda : metoda ploscina

Točke trikotnika bomo prav tako določili v konstruktorju, glede na id. Koordinate točk bodo relativne, glede na pozicijo špice. Zgornji levi kot navidezne kocke, ki obdaja špico, bo tako imel koordinate (0, 0). Spodaj lahko vidite switch stavek, ki določi trikotnik špice.

**switch**(id) {

**case** 2: trikotnik = **new** Trikotnik(0,Kocka.**visinaK**,Kocka.**dolzinaK**,Kocka.**visinaK**,(**double**)Kocka.**dolzinaK**/2,0);**break**;

**case** 3: trikotnik = **new** Trikotnik(0,0,0,Kocka.**visinaK**,Kocka.**dolzinaK**,(**double**)Kocka.**visinaK**/2);**break**;

**case** 4: trikotnik = **new** Trikotnik(0,0,(**double**)Kocka.**dolzinaK**/2,Kocka.**visinaK**,Kocka.**dolzinaK**,0);**break**;

**case** 5: trikotnik = **new** Trikotnik(0,(**double**)Kocka.**visinaK**/2,Kocka.**dolzinaK**,Kocka.**visinaK**,Kocka.**dolzinaK**,0);**break**;

**case** 6: trikotnik = **new** Trikotnik(0,Kocka.**visinaK**,Kocka.**dolzinaK**,Kocka.**visinaK**,(**double**)Kocka.**dolzinaK**/2, (**double**)Kocka.**visinaK**/2);**break**;

**case** 7: trikotnik = **new** Trikotnik(0,0,0,Kocka.**visinaK**,(**double**)Kocka.**dolzinaK**/2,(**double**)Kocka.**visinaK**/2);**break**;

**case** 8: trikotnik = **new** Trikotnik(0,0,(**double**)Kocka.**dolzinaK**/2,(**double**)Kocka.**visinaK**/2,Kocka.**dolzinaK**,0); **break**;

**case** 9: trikotnik = **new** Trikotnik((**double**)Kocka.**dolzinaK**/2,(**double**)Kocka.**visinaK**/2,Kocka.**dolzinaK**,Kocka.**visinaK**, Kocka.**dolzinaK**,0);

}

Koda : določanje točk špice

Metoda jeNevarno() bo v tem razredu vrnila vrednost true. Druga metoda, ki je v tem razredu spremenjena je metoda preveriTrk. Metoda bo izračunala, ali je prišlo do trka med igralcem in špico, ter vrnila ustrezno logično vrednost. Metoda bo vrnila true v dveh primerih:

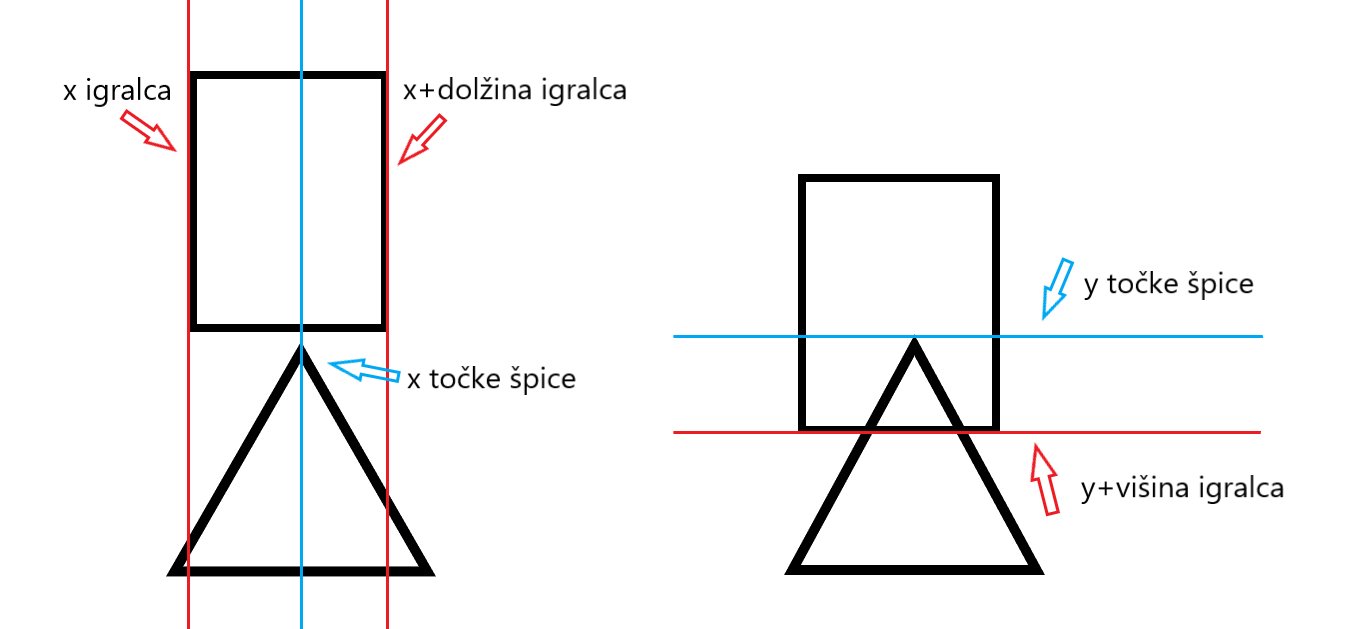
- če se skrajna točka špice nahaja v igralcu  
- če se skrajna točka igralca nahaja v špici

Metoda dobi argumente (Igralec igralec, double ix, double iy), kjer sta ix in iy koordinati skrajne točke igralca, za katero računamo trk. Najprej izračunamo koordinati zgornje desne točke kvadrata, v katerem se nahaja trikotnik. To naredimo tako, vrednosti ix in iy celoštevilsko delimo, ter nato spet pomnožimo z dolžino ali širino ene kocke:

**double** x=(**int**)((ix)/Kocka.**dolzina**)\*Kocka.**dolzina**;

**double** y=(**int**)((iy)/Kocka.**visina**)\*Kocka.**visina**;

Metoda nato najprej preveri, ali se skrajna točka špice nahaja v igralcu. Ker so špice lahko različno orientirane, se izračun za vsako rahlo razlikuje, osnovna ideja pa je enaka. Kot primer bom predstavil način zaznavanja trka za navzgor orientirano špico. Metoda najprej preveri, ali je igralec glede na x os poravnan nad špico. Če je x koordinata skrajne točke špice večja od x koordinate igralca in manjša od vsote x koordinate igralca in njegove dolžine, vemo da je igralec poravnan s konico špice. Metoda nato preveri ali je vsota y koordinate igralca in njegove višine manjša od y koordinate skrajne točke špice. To lahko vidite na spodnji sliki.



Slika 14: Preverjanje trka 1

Spodaj lahko vidite switch stavek, ki določi, če se točka špice nahaja znotraj igralca.

**switch**(id){

**case** 2: **case** 6:

**if**(igralec.getXPozicija()<trikotnik.x3+x && igralec.getXPozicija()+igralec.getDolzina()>trikotnik.x3+x) {

**if**(igralec.getYPozicija()+igralec.getVisina()>trikotnik.y3+y)

**return** **true**;

}**break**;

**case** 3: **case** 7:

**if**(igralec.getYPozicija()<trikotnik.y3+y && igralec.getYPozicija()+igralec.getVisina()>trikotnik.y3+y) {

**if**(igralec.getXPozicija()<trikotnik.x3+x)

**return** **true**;

}**break**;

**case** 4: **case** 8:

**if**(igralec.getXPozicija()<trikotnik.x2+x && igralec.getXPozicija()+igralec.getDolzina()>trikotnik.x2+x) {

**if**(igralec.getYPozicija()<trikotnik.y2+y)

**return** **true**;

}**break**;

**case** 5: **case** 9:

**if**(igralec.getYPozicija()<trikotnik.y1+y && igralec.getYPozicija()+igralec.getVisina()>trikotnik.y1+y) { **if**(igralec.getXPozicija()+igralec.getDolzina()>trikotnik.x1+x)

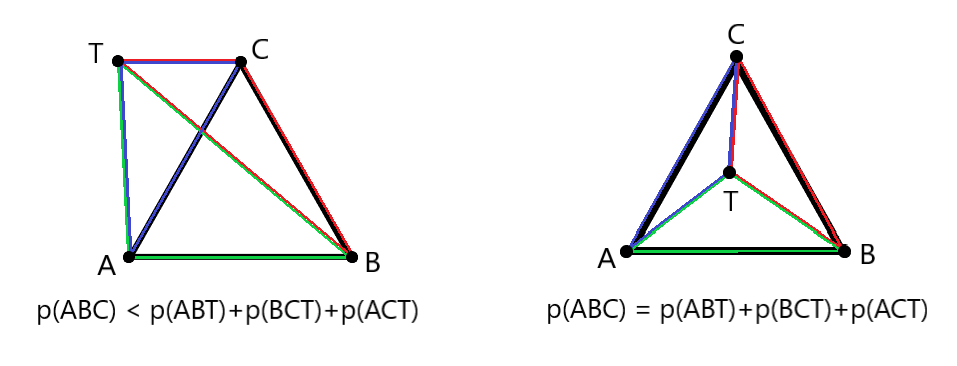
**return** **true**;

}**break**;

}

Koda : prvi korak preverjanja trka

V drugem koraku metoda preveri, če se točka igralca nahaja znotraj špice. Ta korak je neodvisen od orientacije špice, zato je za vse enak. Algoritem določi, če se točka nahaja znotraj trikotnika s pomočjo ploščin. Metoda izračuna ploščine trikotnikov ABC, ki predstavlja špico ter ploščine trikotnikov ABT, BCT in ACT. Te trikotnike določata po dve točki trikotnika špice ter točka igralca. Če je vsota ploščin teh trikotnikov večja od ploščine trikotnika špice, se točka nahaja izven trikotnika. Če je vsota ploščin enaka ploščini trikotnika ABC se točka nahaja v trikotniku. (24) Na spodnji sliki lahko vidite grafični prikaz delovanja algoritma.



Slika : Preverjanje trka 2

Kodo, ki preveri, ali se točka nahaja znotraj trikotnika, lahko vidite spodaj. (24)

**double** P = Trikotnik.*ploscina*(trikotnik.x1+x,trikotnik.y1+y,trikotnik.x2+x,trikotnik.y2+y,trikotnik.x3+x,trikotnik.y3+y);

**double** P1 = Trikotnik.*ploscina*(ix,iy,trikotnik.x2+x,trikotnik.y2+y,trikotnik.x3+x,trikotnik.y3+y);

**double** P2 = Trikotnik.*ploscina*(trikotnik.x1+x,trikotnik.y1+y,ix,iy,trikotnik.x3+x,trikotnik.y3+y);

**double** P3 = Trikotnik.*ploscina*(trikotnik.x1+x,trikotnik.y1+y,trikotnik.x2+x,trikotnik.y2+y,ix,iy);

**if**(P1+P2+P3 <= P)

**return** **true**;

Koda : drugi korak preverjanja trka

CILJ

Vsak nivo mora imeti cilj, ki ga mora igralec doseči za zmago. Praviloma ima vsak nivo le en cilj. Konstruktor razreda Cilj kot argumente vzame objekt Igra in številko id. Id cilja bo bil v igri enak 10. Cilj ne bo imel lastne slike, temveč ga bo označeval vizualni efekt sestavljen iz skupine delcev (str. 59), zato konstruktorju nadrazreda namesto slike podamo vrednost null. V tem razredu tudi spremenimo metodo jeCilj(), da vrne vrednost true. Drugih metod v tem razredu ne spreminjamo.

## SVET

Razred Svet je zadolžen za hranjenje podatkov vseh različnih kock. Razred hrani položaje kock v dvodimenzionalni tabeli. Poleg tega je razred zadolžen tudi za risanje vseh kock na zaslon. Konstruktor razreda prejme objekt Igra in številko nivoja. Nato inicializira objekt Podatki, preko katerega bo lahko razred kasneje dostopal do podatkov o igralcu in kameri. Konstruktor nato pridobi dvodimenzionalno tabelo kock nivoja od razreda Nivoji (str. 26) z metodo igra.getNivoji().getNivo (stNivoja). Konstruktor tudi shrani dolžno in višino nivoja v spremenljivki xVelikost in yVelikost. Konstruktor nato inicializira en primerek objekta vsake kocke in jih shrani v tabelo kocke, kjer je id kocke enak njenemu indeksu v tabeli. Konstruktor nazadnje poišče položaj cilja in na tistem mestu ustvari nov objekt SkupinaDelcev (str. 59), ki bo vizualno označeval cilj.

V metodi posodobiStanje() razred posodobi stanje skupine delcev, ki označuje cilj. V metodi narisi(Graphics g) razred nariše slike vseh kock, ki so trenutno na zaslonu. Metoda mora najprej izračunati začetne in končne indekse v tabeli nivoja. Metoda bo narisala le slike kock med temi indeksi. Indeksi so izračunani glede na pozicijo igralca. Narisane bodo le kocke, ki so od igralca oddaljene za pol dolžine oziroma pol višine okna. Če so izračunani indeksi izven meja tabele, tj. manjši od nič ali večji od velikosti nivoja, jih metoda spremeni na nič oz. velikost nivoja. Računanje začetnih in končnih indeksov lahko vidite v kodi 49.

**int** xZacetni = (**int**) (podatki.getIgralec().getXPozicija() - (igra.getDolzina()/2)) / Kocka.**dolzina**;

**int** yZacetni = (**int**) (podatki.getIgralec().getYPozicija() - (igra.getVisina()/2)) / Kocka.**visina**;

**int** xKoncni = (**int**) (podatki.getIgralec().getXPozicija() + (igra.getDolzina()/2)) / Kocka.**dolzina**;

**int** yKoncni = (**int**) (podatki.getIgralec().getYPozicija() + (igra.getVisina()/2)) / Kocka.**visina**;

Koda : računanje začetnih in končnih indeksov

Metoda nato nariše slike vseh kock, tako da v dvojni zanki potuje skozi tabelo nivoja in na vsakem koraku pokliče metodo narisi(Graphics g, int x, int y) ustrezne kocke:

**for**(**int** i=xZacetni; i<xVelikost; i++) {

**for**(**int** e=yZacetni; e<yVelikost; e++) {

kocke[svet[i][e]].narisi(g,i,e);

}

}

Koda : risanje kock

Metoda pokliče tudi metodo narisi objekta SkupinaDelcev.

Razred vsebuje tudi metodo getKocka. Metoda vrne objekt Kocka na nekih podanih koordinatah. Metoda najprej izračuna indeks kocke v tabeli nivoja. Če je indeks izven območja tabele, metoda vrne 0. Drugače metoda vrne kocko iz tabele kocke z id enakim tistemu v dvodimenzionalni tabeli nivoja. Metodo lahko vidite v spodnji kodi.

**public** Kocka getKocka(**double** xPiksli, **double** yPiksli) {

**int** x=(**int**)((xPiksli)/Kocka.**dolzina**);

**int** y=(**int**)((yPiksli)/Kocka.**visina**);

**if**(x<0 ||y<0||x>=xVelikost|| y>=yVelikost)

**return** kocke[0];

**return** kocke[svet[x][y]];

}

Koda : metoda getKocka

Razred ima tudi metodi getXVelikost() in getYVelikost(), ki vrneta dolžino in višino sveta v kockah.

## IGRALEC

Razred Igralec vsebuje metode za premikanje igralca, zaznavanje trkov z drugimi objekti v igri, zaznavanje smrti in zmage, animacijo in risanje igralca na zaslon. Konstruktor razreda prejme x ter y, ki določata startno pozicijo igralca. Ker sta x in y merjena v kockah, ju moramo pomnožiti z dolžino in višino kocke, da dobimo dejansko pozicijo. Konstruktor tudi shrani objekta Igra in Podatki, preko katerih bo razred kasneje lahko dostopal do podatkov drugih elementov igre. V konstruktorju prav tako inicializiramo objekte za animacijo igralca, o katerih bom več povedal na strani 58. V razredu Igralec je veliko metod, ki pa se jih da v grobem razdeliti na metode odgovorne za premikanje in posodabljanje stanja igralca ter metode odgovorne za animacijo in risanje igralca.

Koda metode posodobiStanje() je zaradi večje preglednosti razdeljena na več manjših metod. Vsaka od teh metod opravi svoj del naloge. Metodo lahko vidite spodaj.

**private boolean** zmaga = **false**;

**private boolean** smrt = **false**;

**public** **void** posodobiStanje() {

**if**(!smrt && !zmaga) {

izracunajHitrostX();

izracunajHitrostY();

premakniX();

premakniY();

orientacija();

}

**if**(pogojZaSmrt()) {

**this**.smrt = **true**;

**if**(delci == **null**) {

delci = **new** SkupinaDelcev2((**int**)x+**dolzina**/2, (**int**)y+**visina**/2, 200,800,podatki);

}

}

**if**(delci != **null**) {

delci.posodobiStanje();

}

**if**(pogojZaZmago())

**this**.zmaga = **true**;

dotikTal();

//posodobitev objektov animacije

mirovanjeD.posodobiStanje();

hojaD.posodobiStanje();

mirovanjeL.posodobiStanje();

hojaL.posodobiStanje();

skokD.posodobiStanje();

skokL.posodobiStanje();

}

Koda : metoda posodobiStanje razreda Igralec

Če igralec ni mrtev ali če še ni zmagal, program najprej pokliče metodi izracinajXHitrost() in izracunajYHitrost(). Ti metodi izračunata hitrost igralca na obeh oseh in posodobita vrednosti hitrostX in hitrostY.

Metoda izracunajXHitrost() izračuna hitrost glede na to, katera tipka je trenutno pritisnjena. Če je hitrost pozitivna se igralec premika desno, drugače pa levo. Če je pritisnjena tipka desno, metoda spremenljivki hitrostX doda pospesek. Pospesek je neka konstantna vrednost, ki je skozi izvajanje programa vedno enaka. Če pa tipka desno ni pritisnjena, se od hitrosti odšteje vrednost pojemek, ki je prav tako konstanta. Če je po izračunu absolutna vrednost hitrosti manjša od pojemka, se njena vrednost zaokroži na 0. Nato metoda enak po enakem postopku izračuna če hitrost, če je pritisnjena tipka levo, le da se v tem primeru pospešek odšteva in pojemek prišteva. Na koncu metoda preveri še, če je hitrost večja od največje dovoljene hitrosti, in jo po potrebi zmanjša. Tako igralec ne more pospeševati v nedogled. Metodo lahko vidite v kodi 53.

**final private static double** pospesek = 0.1;

**final private static double** pojemek = 0.1;

**final private static double** maxHitrostX = 2.5;

**final private static double** maxHitrostY = 14;

**private double** hitrostX = 0;

**private double** hitrostY = 0;

**public** **void** izracunajHitrostX() {

**if**(igra.getTipke().getDesno())

hitrostX += **pospesek**;

**else** {

**if**(hitrostX > 0) {

hitrostX -= **pojemek**;

**if**((Math.abs(hitrostX)) < **pojemek**)

hitrostX = 0;

}

}

**if**(igra.getTipke().getLevo())

hitrostX -= **pospesek**;

**else** {

**if**(hitrostX < 0) {

hitrostX += **pojemek**;

**if**((Math.abs(hitrostX)) < **pojemek**)

hitrostX = 0;

}

}

**if**(hitrostX>**maxHitrostX**)

hitrostX = **maxHitrostX**;

**if**(hitrostX < (-1\***maxHitrostX**))

hitrostX = -1\***maxHitrostX**;

}

Koda : metoda izracunajXHitrost

V metodi izracunajYHitrost() program izračuna hitrost na osi y. Če je vrednost hitrosti pozitivna se igralec premika navzdol, drugače pa navzgor. Metoda se razlikuje od prejšnje, saj deluje na igralca stalen pospešek navzdol. Metoda zato hitrosti najprej doda vrednost konstante gravitacija. Nato metoda preveri ali so izpolnjeni pogoji za skok igralca. Igralec se mora najprej dotikati tal, pritisnjena pa mora biti tudi tipka gor. Poskrbeti moramo tudi, da igralec skoči le enkrat na en pritisk tipke. Zato preverimo tudi vrednost spremenljivke ustaviSkok, ki se spremeni na true, ko igralec skoči in na false, ko je izpuščena tipka gor. Če so izpolnjeni vsi pogoji za skok, se hitrosti odšteje konstanta hitrostSkoka, program pa zaigra tudi zvok Skok. Na koncu metode preverimo še, da hitrost ne presega maksimalne hitrosti. Metodo lahko vidite v spodnji kodi.

**final private static double** gravitacija = 0.3;

**final private static double** hitrostSkoka = 9;

**final private static double** maxHitrostY = 14;

**private double** hitrostY = 0;

**private boolean** ustaviSkok = **false**;

**public** **void** izracunajHitrostY() {

hitrostY += **gravitacija**;

**if**(seDotikaTal) {

**if**(igra.getTipke().getGor()) {

**if**(!ustaviSkok) {

hitrostY = -1\***hitrostSkoka**;

igra.getZvok().zvok\_Skok();

ustaviSkok = **true**;

}

}

**else** {

ustaviSkok = **false**;

}

}

**if**(hitrostY>**maxHitrostY**)

hitrostY = **maxHitrostY**;

**if**(hitrostY < (-1\***maxHitrostY**))

hitrostY = -1\***maxHitrostY**;

}

Koda : metoda izracunajYHitrost

Po tem ko program izračuna hitrost igralca po obeh oseh, program pokliče metodi premakniX() in premakniY(). Metodi delujeta popolnoma enako. Razlikujeta se samo v imenih spremenljivk, zato bom predstavil le premakniX(). Metoda mora pri premikanju igralca upoštevati, da se igralec ne more premikati skozi opeke. Metoda zato pred premikom preveri ali se nova pozicija igralca nahaja znotraj opeke. Pri premiku v desno tako preveri novo pozicijo dveh desnih skrajnih točk igralca. To naredi s pomočjo metode preveriTrk, ki jo bom predstavil v naslednjem odstavku. Če ne pride do trka, se poziciji x doda hitrostX, drugače se hitrostX spremeni na 0, igralec pa se premakne v skrajno pozicijo, kjer se še ne nahaja znotraj opeke. Metodo lahko vidite spodaj.

**public** **void** premakniX() {

**if**(hitrostX < 0)

**if**(!preveriTrk(x+hitrostX,y) && !preveriTrk(x+hitrostX,y+**visina**-1))

x = (x + hitrostX);

**else** {

x = (**int**)(((x)/Kocka.**dolzina**))\*Kocka.**dolzina**;

hitrostX = 0;

}

**if**(hitrostX > 0) {

**if**(!preveriTrk(x+**dolzina**+hitrostX,y) && !preveriTrk(x+**dolzina**+hitrostX,y+**visina**-1))

x = (x + hitrostX);

**else** {

x = (**int**)((x+**dolzina**+hitrostX)/Kocka.**dolzina**)\*Kocka.**dolzina** -**dolzina**;

hitrostX = 0;

}

}

}

Koda : metoda premakniX

Metoda preveriTrk(double novaXPozicija, double novaYPozicija) preveri ali se neka podana točka nahaja znotraj opeke. To naredi tako, da pokliče metodo getKocka objekta Svet (str. 49). Ta vrne kocko, ki se nahaja na danih koordinatah. Nato pokliče metodo jeTrdno(), ki vrne true, če je igralec ne more prehajati skozi kocko. Metoda preveriTrk nato vrne to vrednost. Metodo lahko vidite v spodnji kodi.

**public** **boolean** preveriTrk(**double** novaXPozicija, **double** novaYPozicija) {

**return** podatki.getSvet().getKocka(novaXPozicija, novaYPozicija).jeTrdno();

}

Koda : metoda preveriTrk

Po tem, ko program premakne igralca, pokliče še metodo orientacija() ta zabeleži v katero smer je igralec obrnjen. Če je hitrost večja od 0 spremeni vrednost jeObrnjenDesno na true, če pa je manjša spremeni vrednost na false. To vrednost bo program uporabil pri risanju igralca. Metodo lahko vidite spodaj.

**private boolean** jeObrnjenDesno = **true**;

**public** **void** orientacija() {

**if**(hitrostX>0)

jeObrnjenDesno = **true**;

**else** **if**(hitrostX<0)

jeObrnjenDesno = **false**;

}

Koda : metoda orientacija

Program nato preveri, ali je igralec morda umrl. To preveri metoda pogojZaSmrt(). Metoda najprej preveri, ali je igralec padel z nivoja. Če je y pozicija igralca večja od višine sveta, metoda vrne vrednost true. Če ta pogoj ni izpolnjen, metoda preveri ali se igralec dotika špice. Metoda za vsako skrajno točko igralca preveri, če se nahaja v kvadratu, v katerem je špica, tako da pokliče metodo getKocka, da dobi kocko na danih koordinatah. Nato pokliče metodo jeNevarno(), ki vrne true če je kocka na tem mestu Spica. Nato pokliče metodo preveriTrk kocke, da ugotovi, če se igralec dotika špice. Metodo lahko vidite v kodi 58.

Če program ugotovi, da je izpolnjen pogoj za smrt, spremeni stanje spremenljivke smrt na true. Na mestu, kjer je igralec program ustvari nov objekt SkupinaDelcev2 (str. 59) **.**

**public** **boolean** pogojZaSmrt() {

//igralec je padel z nivoja

**if**(y > podatki.getSvet().getYVelikost()\*Kocka.**visina**)

**return** **true**;

//igralec se je dotaknil spice

**if**(podatki.getSvet().getKocka(x,y).jeNevarno())

**return** (podatki.getSvet().getKocka(x,y).preveriTrk(**this**,x,y));

**if**(podatki.getSvet().getKocka(x+**dolzina**-1,y).jeNevarno())

**return** (podatki.getSvet().getKocka(x+**dolzina**-1,y).preveriTrk(**this**,x+**dolzina**-1,y));

**if**(podatki.getSvet().getKocka(x+**dolzina**-1,y+**visina**-1).jeNevarno())

**return** (podatki.getSvet().getKocka(x+**dolzina**-1,y+**visina**-1).preveriTrk(**this**,x+**dolzina**-1,y+**visina**-1));

**if**(podatki.getSvet().getKocka(x,y+**visina**-1).jeNevarno())

**return** (podatki.getSvet().getKocka(x,y+**visina**-1).preveriTrk(**this**,x,y+**visina**-1));

**return** **false**;

}

Koda : metoda pogojZaSmrt

Program nato preveri če je igralec morda zmagal. Program pokliče metodo pogojZaZmago(). Metoda za vsako skrajno točko igralca preveri, ali se nahaja znotraj kocke Cilj. Najprej pokliče metodo getKocka, ki vrne kocko na tistih koordinatah. Nato pokliče metodo jeCilj(), ki vrne vrednost true, če je kocka Cilj. Če se katerakoli točka nahaja v kocki Cilj metoda vrne true, v nasprotnem primeru pa false. Metodo lahko vidite v kodi 59.

Če je pogoj za zmago izpolnjen metoda spremeni vrednost zmaga na true.

**public** **boolean** pogojZaZmago() {

**if**(podatki.getSvet().getKocka(x,y).jeCilj())

**return** **true**;

**if**(podatki.getSvet().getKocka(x+**dolzina**-1,y).jeCilj())

**return** **true**;

**if**(podatki.getSvet().getKocka(x+**dolzina**-1,y+**visina**-1).jeCilj())

**return** **true**;

**if**(podatki.getSvet().getKocka(x,y+**visina**-1).jeCilj())

**return** **true**;

**return** **false**;

}

Koda : metoda pogojZaZmago

Program nato pokliče metodo dotikTal(). Metoda posodobi vrednost spremenljivke seDotikaTal. Najprej preveri, če je kocka eno točko pod igralcem trdna, tako da pokliče metodo preveriTrk za spodnji skrajni točki igralca. Če metoda ugotovi, da se igralec dotika tal, vrednost seDotikaTal pa še ni enaka true, posodobi vrednost in zaigra zvok Padec. Če se igralec ne dotika tal spremeni vrednost seDotikaTal na false. Metodo lahko vidite spodaj.

**private** **boolean** seDotikaTal = **false**;

**public** **void** dotikTal() {

**if**(preveriTrk(x,y+**visina**+1)||preveriTrk(x+**dolzina**-1,y+**visina**+1)) {

**if**(!seDotikaTal) {

seDotikaTal = **true**;

igra.getZvok().zvok\_Padec();

}

}

**else**

seDotikaTal = **false**;

}

Koda : metoda dotikTal

Nazadnje program v metodi posodobiStanje() posodobi še stanja objektov za animacijo igralca (str. 58).

Razred Igralec vsebuje tudi metodo narisi(Graphics g), ki je odgovorna za risanje igralca na zaslon. Če igralec še ni umrl ali zmagal. Metoda pridobi trenutno sliko igralca, tako da pokliče metodo trenutnaSlika(). Nato metoda nariše sliko igralca. Ker velikost slike ni enaka velikosti igralca, moramo sliko zamakniti za odmikSlikeX in odmikSlikeY. Če je igralec zmagal, metoda ne nariše njegove slike, če pa je umrl namesto njega nariše sliko skupine delcev (str. ).Metodo narisi lahko vidite spodaj.

**private final int** odmikSlikeX = 6;

**private final int** odmikSlikeY = 8;

**public** **void** narisi(Graphics grafika) {

**if**(!smrt && !zmaga) {

BufferedImage slika = trenutnaSlika();

grafika.drawImage(slika,(igra.getDolzina()/2-**dolzina**/2-odmikSlikeX), (igra.getVisina()/2-**visina**/2-odmikSlikeY), slika.getWidth(), slika.getHeight(),**null**);

}

**if**(delci != **null**) {

delci.narisi(grafika);

}

}

Koda : metoda narisi razreda Igralec

Metoda trenutnaSlika() od objektov za animacijo igralca pridobi ustrezno sliko za trenutno stanje igralca. Metoda najprej preveri če je igralec v zraku in če je, v katero smer je obrnjen. Nato vrne trenutno sliko iz animacije skokD ali skokL. Če igralec ni v zraku metoda preveri, če se igralec morda premika v levo ali desno in vrne trenutno sliko iz animacije hojaD ali hojaL. Če se igralec ne premika metoda vrne sliko iz animacije mirovanjeD ali mirovanjeL, glede na to, v katero smer je igralec obrnjen. Metodo lahko vidite v kodi 62.

**public** BufferedImage trenutnaSlika() {

**if**(!seDotikaTal) {

**if**(jeObrnjenDesno)

**return** skokD.getTrenutnaSlika();

**else**

**return** skokL.getTrenutnaSlika();

}

**else** **if**(hitrostX>0)

**return** hojaD.getTrenutnaSlika();

**else** **if**(hitrostX<0)

**return** hojaL.getTrenutnaSlika();

**else** **if**(jeObrnjenDesno)

**return** mirovanjeD.getTrenutnaSlika();

**else**

**return** mirovanjeL.getTrenutnaSlika();

}

Koda : metoda trenutnaSlika

Poleg teh metod ima razred Igralec tudi metodo ponastavi(), ki stanje igralca spremeni na enako stanje, v katerem je bil na začetku igre. Razred ima tudi get metode za spremenljivke zmaga, smrt, x, y ter dolzina in visina.

ANIMACIJA IGRALCA

AnimacijaIgralca je manjši razred, ki skrbi za animacijo igralca. Konstruktor razreda prejme realno število interval, ki določa na koliko sekund se bo menjala slika v animaciji. Število se pomnoži z 1000000000, da dobimo interval v nanosekundah. Konstruktor prejme tudi tabelo slik slike, v kateri so podane slike igralca v takem vrstnem redu, kot se bodo spreminjale v animaciji. Razred hrani tudi spremenljivki index, ki hrani indeks trenutne slike in stevec, ki hrani čas, ki je pretekel od zadnje menjave slike (v nanosekundah). Ti spremenljivki v konstruktorju inicializiramo na 0.

V metodi posodobiStanje() program posodobi stanje animacije. Metoda najprej števcu doda čas, ki je pretekel od zadnjega klica (v nanosekundah). Nato metoda preveri, če je vrednost števca večja od intervala in poveča vrednost index za ena ter spremeni vrednost števca na 0. Če je index preseže velikost tabele, se spremeni nazaj na 0. Metodo lahko vidite v spodnji kodi.

**public** **void** posodobiStanje() {

stevec += System.nanoTime()-prejsnjiCas;

prejsnjiCas = System.nanoTime();

**if**(stevec>interval) {

index ++;

stevec = 0;

**if**(index>=slike.length) {

index = 0;

}

}

}

Koda : metoda posodobiStanje razreda AnimacijaIgralca

Razred ima tudi metodo getTrenutnaSlika(), ki vrne trenutno sliko iz animacije. Metodo lahko vidite spodaj.

**public** BufferedImage getTrenutnaSlika() {

**return** slike[index];

}

Koda : metoda getTrenutnaSlika

V razredu Igralec inicializiramo šest objektov animacije, enega za vsako stanje premikanja, v katerem je lahko igralec. Inicializacijo objektov lahko vidite v kodi 65.

mirovanjeD = **new** AnimacijaIgralca(1,Slike.mirovanjeD);

hojaD = **new** AnimacijaIgralca(0.15,Slike.hojaD);

mirovanjeL = **new** AnimacijaIgralca(1,Slike.mirovanjeL);

hojaL = **new** AnimacijaIgralca(0.15,Slike.hojaL);

skokD = **new** AnimacijaIgralca(1,Slike.skokD);

skokL = **new** AnimacijaIgralca(1,Slike.skokL);

Koda : inicializacija objektov animacije

## DELCI

V svoji igri bom kot dodatne vizualne efekte uporabljal delce (angl. particles). Vsak delec bo imel x in y pozicijo, velikost in barvo. Imel bo tudi dostop do objekta Podatki. Konstruktor razreda prejme in inicializira te podatke. Razred nima lastne metode posodobiStanje(), kot jo imajo drugi elementi igre, namesto tega ima metodo premakni(double premikX, double premikY). Metoda prejme realni števili premikX in premikY ter ju doda poziciji delca. Razred ima tudi metodo narisi(Graphics g), ki na zaslon nariše na točki (x, y) kvadrat v velikosti delca. Pri tem upošteva tudi odmik kamere (str. 63). Razred ima poleg teh metod še metodi getX() in getY(), ki vrneta x in y delca.

Ker delci sami po sebi ne delajo veliko, potrebujemo razred, ki bo določal njihovo obnašanje v igri. Naredil sem dva taka razreda: SkupinaDelcev1 in SkupinaDelcev2. SkupinaDelcev1 je vizualni efekt, ki v igri označuje cilj nivoja. SkupinaDelcev2 je efekt, ki se sproži, ko igralec umre.

SKUPINA DELCEV 1

SkupinaDelcev1 ima obliko kroga. Delci se pojavljajo na naključnih koordinatah znotraj tega kroga. Velikost delcev je naključno število med minVelikost in maxVelikost. Nato se v vsakem krogu zanke premaknejo za vrednost konstante premik proti središču kroga. Ko dosežejo središče delci izginejo. Konstruktor razreda prejme x in y središča, radij kroga, največje število delcev maxStevilo ter objekt Podatki. Razred objekte delcev shranjuje v seznamu (angl. ArrayList) delci.

V metodi posodobiStanje(), program posodobi stanje skupine delcev. Metoda najprej generira nove delce. Število delcev ki jih mora generirati izračuna kot maxStevilo – delci.size(). Nato metoda v for zanki generira potrebno število novih delcev. Metoda za generiranje naključne lokacije delca uporabi spodnjo formulo. (25)

**double** a = random() \* 2 \* PI

**double** r = R \* sqrt(random())

**double** x = r \* cos(a)

**double** y = r \* sin(a)

Ko metoda izračuna lokacijo delca, mu določi še naključno velikost po formuli: velikost = (int) (Math.random() \* (maxVelikost -minVelikost)) + minVelikost. Metoda nato ustvari nov delec bele barve in ga doda seznamu delci. Kodo ki generira nove delce lahko vidite spodaj.

**int** novi = maxStevilo - delci.size();

**for**(**int** i=0; i<novi; i++) {

**double** a = Math.random() \* 2 \* Math.**PI**;

**double** r = radij \* Math.sqrt(Math.random());

**double** xDelca = r \* Math.cos(a) + x;

**double** yDelca = r \* Math.sin(a) + y;

**int** velikost =(**int**) (Math.random()\*(maxVelikost-minVelikost))+minVelikost;

Delec d = **new** Delec(xDelca,yDelca,velikost,Color.**white**,podatki);

delci.add(d);

}

Koda : generiranje novih delcev skupine 1

Metoda nato premakne vse delce proti središču. Za to mora iz konstante premik za vsak delec izračunati premik v x in y smeri. Metoda najprej izračuna razdaljo delca do središča v x in y smeri. Metoda shrani tudi predznak razdalje. Nato metoda izračuna razmerje med razdaljaX in razdaljaY. PremikX in premikY se nato izračunata po formulah:

premikX = Math.sqrt(Math.pow(premik, 2)/(1+Math.pow(razmerje, -2)))  
premikY = Math.sqrt(Math.pow(premik,2)/(1+Math.pow(razmerje, 2)))

Nato pokličemo metodo premakni delca, pri čemer upoštevamo tudi predznak premika. Kodo, ki izračuna premik delca lahko vidite spodaj.

**for**(**int** i=0; i<delci.size(); i++) {

**double** razdaljaX = x - delci.get(i).getX();

**double** razdaljaY = y - delci.get(i).getY();

**double** predznakX = razdaljaX/Math.abs(razdaljaX);

**double** predznakY = razdaljaY/Math.abs(razdaljaY);

**double** razmerje = razdaljaX / razdaljaY;

**double** premikX = Math.sqrt(Math.pow(premik, 2)/(1+Math.pow(razmerje, -2)));

**double** premikY = Math.sqrt(Math.pow(premik,2)/(1+Math.pow(razmerje, 2)));

delci.get(i).premakni(premikX\*predznakX, premikY\*predznakY);

}

Koda : premikanje delcev skupine 1

V zadnjem koraku metoda posodobiStanje() odstrani delce, ki so dosegli središče kroga. Kot to, da je delec dosegel središče štejemo, da je njegova oddaljenost od središča manj kot dve enoti v x in y smeri. Če je to za nek delec res, ga odstranimo z seznama delcev, v naslednjem krogu pa bo metoda generirala novega. Kodo, ki odstranjuje delce, lahko vidite spodaj.

**for**(**int** i=0; i<delci.size(); i++) {

**if**(delci.get(i).getX()>x-2 && delci.get(i).getX()<x+2)

**if**(delci.get(i).getY()>y-2 && delci.get(i).getY()<y+2)

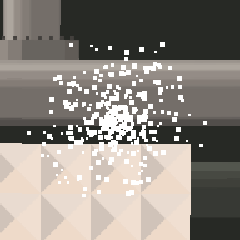
delci.remove(i);

}

Koda : odstranjevanje delcev

Razred SkupinaDelcev1 vsebuje tudi metodo narisi(Graphics g). Metoda preprosto pokliče metode narisi vseh delcev na seznamu delci.

Vizualni efekt SkupinaDelcev1 lahko vidite na sliki 16. Delci se premikajo proti središču.



Slika : Skupina delcev 1

SKUPINA DELCEV 2

SkupinaDelcev2 deluje kot vizualni efekt ob smrti igralca. Koda je v zelo podobna kodi v razredu SkupinaDelcev1, z nekaj manjšimi razlikami. V tej skupini delcev delce generiramo le enkrat, in jih kasneje tudi ne brišemo z seznama. Delci imajo tudi naključno izbrano barvo. Delci se v tem razredu premikajo stran od središča kroga, namesto proti njemu. Vse druge lastnosti razreda so enake.

Generiranje delcev deluje povsem enako, kot v prejšnjem razredu, z dodatkom generacije naključne barve. Za podrobnejšo razlago določanja naključnih koordinat glejte prejšnji podnaslov. Koda za izbiro barve najprej generira naključno celo število med 1 in 6. Nato V switch stavku poveže število z eno izmed šestih možnih barv. Kodo, ki generira delce lahko vidite spodaj.

**double** a = Math.random() \* 2 \* Math.**PI**;

**double** r = radij \* Math.sqrt(Math.random());

**double** xDelca = r \* Math.cos(a) + x;

**double** yDelca = r \* Math.sin(a) + y;

**int** velikost =(**int**) (Math.random() \* (maxVelikost - minVelikost)) + minVelikost;

**int** steviloBarve = (**int**) (Math.random() \* (6)) + 1;

Color barva = Color.**white**;

**switch**(steviloBarve) {

**case** 1:barva = Color.**black**;**break**;

**case** 2:barva = **new** Color(0,162,232);**break**;

**case** 3:barva = **new** Color(253,202,168);**break**;

**case** 4:barva = **new** Color(189,149,69);**break**;

**case** 5:barva = **new** Color(88,46,19);**break**;

**case** 6:barva = Color.**red**;**break**;

}

Delec d = **new** Delec(xDelca,yDelca,velikost,barva,podatki);

delci.add(d);

Koda : generiranje novih delcev skupine 2

V metodi posodobiStanje() se posodobi stanje skupine delcev. Metoda vse delce premakne za vrednost konstante premik stran od središča kroga. Koda je povsem enaka tisti v razredu SkupinaDelcev1, le da spremenimo predznak premika delca. Za podrobnejšo razlago kode glejte podnaslov SKUPINA DELCEV 1. Spodaj lahko vidite metodo posodobiStanje().

**public** **void** posodobiStanje() {

**for**(**int** i=0; i<delci.size(); i++) {

**double** razdaljaX = x - delci.get(i).getX();

**double** razdaljaY = y - delci.get(i).getY();

**double** predznakX = -razdaljaX/Math.abs(razdaljaX);

**double** predznakY = -razdaljaY/Math.abs(razdaljaY);

**double** razmerje = razdaljaX / razdaljaY;

**double** premikX = Math.sqrt(Math.pow(premik, 2)/(1+Math.pow(razmerje, -2)));

**double** premikY = Math.sqrt(Math.pow(premik,2)/(1+Math.pow(razmerje, 2)));

delci.get(i).premakni(premikX\*predznakX, premikY\*predznakY);

}

}

Koda : premikanje delcev skupine 2

Metoda narisi(Graphics g) nariše vse delce iz skupine delcev na zaslon. Metoda zgolj pokliče metodo narisi vsakega delca na seznamu delci.

**public** **void** narisi(Graphics g) {

**for**(**int** i=0; i<delci.size(); i++) {

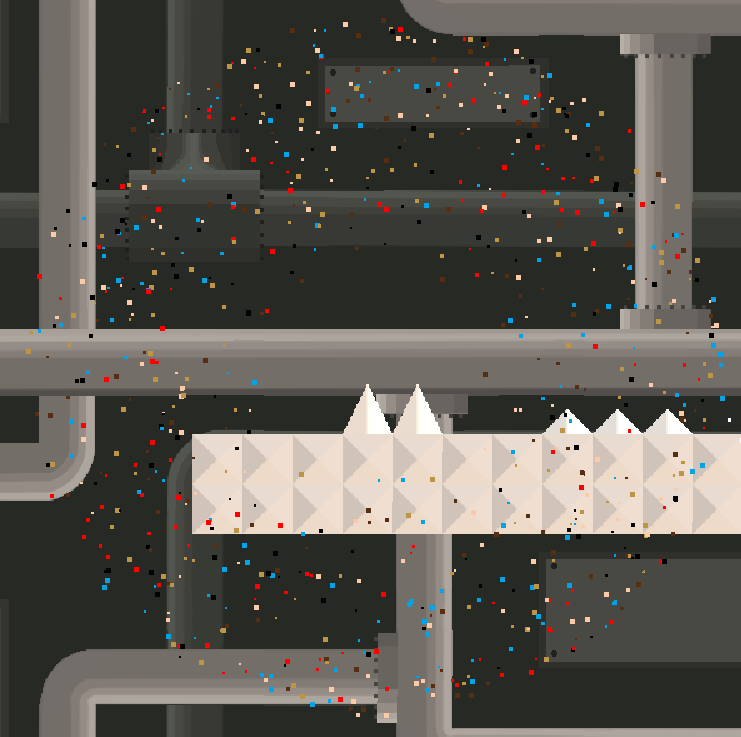
delci.get(i).narisi(g);

}

}

Koda : metoda narisi razreda SkupinaDelcev

Vizualni efekt SkupinaDelcev2 lahko vidite na sliki 17. Delci se premikajo stran od središča.



Slika : Skupina delcev 2

## KAMERA

Razred Kamera je zadolžen na računanje odmika navidezne kamere v igri. Ko se igralec premika, se mora naš pogled premikati skupaj z njim tako, da ostane na sredini zaslona. Kamera mora zato izračunati koliko moramo pri risanju zamakniti vse druge entitete v igri, kot so opeke, špice, cilj in ozadje.

Konstruktor razreda prejme objekt Igra, preko katerega bo lahko razred dostopal do podatkov o igri. Konstruktor shrani tudi objekt Igralec. V metodi posodobiStanje() se izračunata xOdmik in yOdmik kamere. Izračunamo ju tako, da poziciji igralca odštejemo polovico dolžine oz. višine igre, ter prištejemo polovico dolžine oz. višine igralca. Kodo, ki sem jo uporabil za Računanje odmika kamere sem prilagodil s spletnega vira StackOverflow. (26)

xOdmik = (**int**) (igralec.getXPozicija()-(igra.getDolzina()/2)+ igralec.getDolzina()/2);

yOdmik = (**int**) (igralec.getYPozicija()-(igra.getVisina()/2)+ igralec.getVisina()/2);

Koda : računanje odmika kamere

Razred ima tudi metodi getXOdmik() in getYOdmik(), ki vrneta xOdmik in yOdmik kamere. Preko teh metod bodo drugi elementi igre lahko dostopali do odmika kamere.

## OZADJE

Razred Ozadje je zadolžen za posodabljanje in risanje ozadja igre na zaslon. Da bo igra izgledala naravno se mora ozadje premikati glede na pozicijo kamere. Razred bo imel sliko ozadja, x in y pozicijo ter faktor, za katerega se bo ozadje premikalo. Razred bo potreboval tudi dostop do objekta Kamera.

V metodi posodobiStanje() izračunamo pozicijo ozadja glede na pozicijo kamere. Ozadje se bo premikalo v obratno smer od kamere, zato poziciji dodamo negativen predznak. Nato odmik kamere pomnožimo z faktorjem premika. Nazadnje vrednost, ki jo dobimo celoštevilsko delimo z dolžino oz. višino igre in shranimo ostanek kot novo pozicijo kamere. Na ta način ozadje ne more zapustiti zaslona. Spodaj lahko vidite metodo posodobiStanje().

**public** **void** posodobiStanje() {

**this**.x = (podatki.getKamera().getXOdmik()\*faktor\*-1)%igra.getDolzina();

**this**.y = (podatki.getKamera().getYOdmik()\*faktor\*-1)%igra.getVisina();

}

Koda : metoda posodobiStanje razreda Ozadje

V metodi narisi(Graphics g) narišemo sliko ozadja na zaslon. Ker je velikost ozadja manjša od velikosti igre, moramo ozadje narisati večkrat, tako da zapolnimo celotno velikost igre. Da se izognemo nepotrebni izgubi časa, narišemo slike ozadja le na tistih mestih kjer je to potrebno. V vsakem primeru narišemo sliko ozadja najprej v njeni osnovni poziciji. Nato preverimo x in y pozicijo ozadja. Če je x pozicija manjša od 0, moramo narisati dodatne slike ozadja še na desni strani, če pa je večja od 0, narišemo dodatne slike ozadja na levi strani. Enako storimo tudi za y. Če je ta večji od 0 narišemo dodatne slike na zgornji strani osnovne slike, drugače pa na spodnji strani. Tako v vsakem primeru narišemo 4 slike ozadja. Metodo narisi lahko vidite spodaj.

**public** **void** narisi(Graphics g) {

g.drawImage(slika, (**int**)x, (**int**)y, slika.getWidth(), slika.getHeight(), **null**);

**if**(x<0) {

g.drawImage(slika, (**int**)x+slika.getWidth(), (**int**)y, slika.getWidth(), slika.getHeight(), **null**);

**if**(y<0) {

g.drawImage(slika, (**int**)x, (**int**)y+slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

g.drawImage(slika, (**int**)x+slika.getWidth(), (**int**)y+slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

}

**else** {

g.drawImage(slika, (**int**)x, (**int**)y-slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

g.drawImage(slika, (**int**)x+slika.getWidth(), (**int**)y-slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

}

}

**else** {

g.drawImage(slika, (**int**)x-slika.getWidth(), (**int**)y, slika.getWidth(), slika.getHeight(), **null**);

**if**(y<0) {

g.drawImage(slika, (**int**)x, (**int**)y+slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

g.drawImage(slika, (**int**)x-slika.getWidth(), (**int**)y+slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

}

**else** {

g.drawImage(slika, (**int**)x, (**int**)y-slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

g.drawImage(slika, (**int**)x-slika.getWidth(), (**int**)y-slika.getHeight(), slika.getWidth(), slika.getHeight(), **null**);

}

}

}

Koda : metoda narisi razreda Ozadje

# ZAKLJUČEK

V svoji seminarski nalogi sem predstavil načrtovanje in delovanje svoje 2D igre. V igro mi je uspelo vključiti vse, kar sem si zastavil na začetku naloge. Igra vsebuje več nivojev, do katerih igralec dostopa preko menijev. Igralec lahko spreminja nastavitve jezika in glasnosti, na voljo pa mu je tudi pomoč. Testiranje programa sem izvajal med programiranjem vsakič, ko sem v igro dodajal novo funkcionalnost, da sem poskrbel da ta deluje pravilno in da igra poteka tekoče. Končna verzija igre deluje tekoče, delitev na več datotek pa omogoča večjo preglednost in lažje popravljanje napak. Igro bi se dalo še naprej nadaljnjo nadgrajevati. Obstaja več različnih načinov in algoritmov za posodabljanje in risanje igre. S testiranjem bi bilo mogoče izbrati bolj učinkovite algoritme, kar bi še dodatno pospešilo igro. V igro bi lahko dodal tudi nove elemente, kot predvajanje glasbe v ozadju. Lahko bi dodal tudi več novih kock z drugačnimi lastnostmi, ki bi omogočale izdelavo še več različnih nivojev. V igro bi lahko dodal tudi sovražnike, ki bi uporabljali primitivno umetno inteligenco. Obstaja več možnosti, kako bi igro lahko še izboljšal.

# ZAHVALA

Na koncu svoje seminarske naloge bi se rad zahvalil vsem, ki so mi kakorkoli pomagali pri izdelavi te seminarske naloge. Rad bi se zahvalil mentorju, ki je podal nešteto uporabnih nasvetov, tako pri pisanju kode, kot tudi pri oblikovanju seminarske naloge. Zahvalil bi se rad tudi sošolcem, ki so bili vedno pripravljeni pomagati, ko sem potreboval pomoč. Nazadnje bi se rad zahvalil tudi svoji družini, ki mi se stala ob strani in me prenašala, potem ko sem po celem dnevu programiranja zaspan prikorakal iz sobe. Še enkrat. Hvala.

Jure Zupančič

# BIBLIOGRAFIJA

1. **Niemeyer, Patrick in Knudsen, Jonathan.** *Learning Java.* s.l. : O'Reilly Media, Inc., 2005. ISBN 0596008732, 9780596008734.

2. **Oracle.** Class JFrame. *docs.oracle.com.* [Elektronski] [Navedeno: 2. april 2021.] https://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html.

3. —. Class JPanel. *docs.oracle.com.* [Elektronski] [Navedeno: 2. april 2021.] https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html.

4. **Bruce, Eckel.** *Thinking in Java (4th edition).* s.l. : Prentice Hall, 2006 .

5. **Tutorialspoint.** Java BufferedImage Class. *Tutorialspoint.* [Elektronski] [Navedeno: 2. april 2021.] https://www.tutorialspoint.com/java\_dip/java\_buffered\_image.htm.

6. **Oracle.** Class Graphics. *docs.oracle.com.* [Elektronski] [Navedeno: 2. april 2021.] https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html.

7. **GeeksForGeeks.** Java AWT | Color Class. *GeeksForGeeks.* [Elektronski] 31. oktober 2019. [Navedeno: 2. april 2021.] https://www.geeksforgeeks.org/java-awt-color-class/.

8. **Oracle.** Class Font. *dosc.oracle.com.* [Elektronski] [Navedeno: 3. april 2021.] https://docs.oracle.com/javase/7/docs/api/java/awt/Font.html.

9. **pramodbablad.** How To Read And Write Images In Java? – ImageIO Example. *Java concept of the day.* [Elektronski] 24. november 2016. [Navedeno: 4. april 2021.] https://javaconceptoftheday.com/read-and-write-images-in-java/.

10. **Oracle.** Class AttributedString. *docs.oracle.com.* [Elektronski] [Navedeno: 3. april 2021.] https://docs.oracle.com/javase/7/docs/api/java/text/AttributedString.html#:~:text=An%20AttributedString%20holds%20text%20and%20related%20attribute%20information.,wants%20to%20access%20attributed%20text%20through%20the%20AttributedCharacterIterator.

11. **Java t point.** Java KeyListener Interface. *Java t point.* [Elektronski] [Navedeno: 3. april 2021.] https://www.javatpoint.com/java-keylistener.

12. —. Java MouseListener Interface. *Java t point.* [Elektronski] [Navedeno: 3. april 2021.] https://www.javatpoint.com/java-mouselistener.

13. —. Java MouseMotionListener Interface. *Java t point.* [Elektronski] [Navedeno: 3. april 2021.] https://www.javatpoint.com/java-mousemotionlistener#:~:text=The%20Java%20MouseMotionListener%20is%20notified%20whenever%20you%20move,found%20in%20java.awt.event%20package.%20It%20has%20two%20methods..

14. **Tutorialspoint.** Java Examples - Write to a file. *Tutorialspoint.* [Elektronski] [Navedeno: 4. april 2021.] https://www.tutorialspoint.com/javaexamples/file\_write.htm.

15. **Oracle.** Interface Clip. *docs.oracle.com.* [Elektronski] [Navedeno: 4. april 2021.] https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/Clip.html.

16. **GeeksForGeeks.** How to play an Audio file using Java. *GeeksForGeeks.* [Elektronski] 24. marec 2018. [Navedeno: 4. april 2021.] https://www.geeksforgeeks.org/play-audio-file-using-java/.

17. **Oracle.** Class FloatControl. *docs.oracle.com.* [Elektronski] [Navedeno: 4. april 2021.] https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/FloatControl.html.

18. **Valentino, John.** Obtaining the Frequency Content of a Sampled Sound Using Java and the Discrete Fourier Transform. *jvalentino2.tripod.com.* [Elektronski] 18. oktober 2005. [Navedeno: 4. april 2021.] https://jvalentino2.tripod.com/dft/index.html.

19. **Wikipedia.** Comma-seperated values. *Wikipedia.* [Elektronski] 23. marec 2021. [Navedeno: 4. april 2021.] https://en.wikipedia.org/wiki/Comma-separated\_values.

20. —. WAV. *Wikipedia.* [Elektronski] 3. april 2021. [Navedeno: 5. april 2021.] https://en.wikipedia.org/wiki/WAV.

21. **Davison, Andrew.** *Killer game programing in Java.* s.l. : O'Reilly, 2005.

22. **Bowersox, Kevin.** How can i add a keylistener to my Gui in java. *StackOverflow.* [Elektronski] 6. december 2013. [Navedeno: 5. april 2021.] https://stackoverflow.com/questions/20413910/how-can-i-add-a-keylistener-to-my-gui-in-java.

23. **Oracle.** Class FontMetrics. *docs.orale.com.* [Elektronski] [Navedeno: 4. april 2021.] https://docs.oracle.com/javase/7/docs/api/java/awt/FontMetrics.html.

24. **GeeksForGeeks.** Check whether a given point lies inside a triangle or not. *GeeksForGeeks.* [Elektronski] 23. marec 2021. [Navedeno: 6. april 2021.] https://www.geeksforgeeks.org/check-whether-a-given-point-lies-inside-a-triangle-or-not/.

25. **Programming guide.** Generating a random point within a circle. *programming.guide.* [Elektronski] 2016. [Navedeno: 5. april 2021.] https://programming.guide/random-point-within-circle.html.

26. **StackOverflow.** Having The “Camera” positioning On The Player in Java. *StackOverflow.* [Elektronski] 20. december 2015. [Navedeno: 5. april 2021.] https://stackoverflow.com/questions/34385383/having-the-camera-positioning-on-the-player-in-java.

27. **Burd, Barry.** *Java for Dummies, 7th Edition.* s.l. : John Wiely & Sons, Inc., 2017.

28. **Holder, Wayne in Doug, Bell.** *Java Game Programming For Dummies.* s.l. : For Dummies, 1998.

29. **Klawonn, Frank.** *Introduction to Computer Graphics: Using Java 2D and 3D.* 2008 : Springer.

30. **Mesojedec, Uroš in Fabjan, Borut.** *Java 2: temelji programiranja.* s.l. : Založba Pasadena, 2004.

# PRILOGA

V prilogi lahko najdete spletne povezave do vseh datotek razredov.

Celoten projekt lahko najdete na naslednji povezavi:  
<https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga->

Razredi Main, Igra, Jezik, Nastavitve, Okno, Podatki in Tipke : <https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-/tree/master/platformska_igra/src/igra>

Razredi Stanje, Stanje\_Meni, Stanje\_IzbiraNivoja, Stanje\_Nivo, Stanje\_Nastavitve in Stanje\_Pomoc :  
<https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-/tree/master/platformska_igra/src/stanja>

Razredi KomponentaMenija, Besedilo, Drsnik, Gumb in SpustniMeni :  
<https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-/tree/master/platformska_igra/src/meni>

Razredi Svet, Kamera, Ozadje, Igralec, AnimacijaIgralca, Delec, SkupinaDelcev1 in SkupinaDelcev2 :  
<https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-/tree/master/platformska_igra/src/elementi_igre>

Razredi Kocka, Opeka, Zrak, Cilj, Spica in Trikotnik:  
<https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-/tree/master/platformska_igra/src/kocke>

Razredi Slike, Zvok in Nivoji:  
<https://github.com/Jure-Z/2D-platformska-igra-maturitetna-seminarska-naloga-/tree/master/platformska_igra/src/viri>

STVARNO KAZALO

Animacija, 16, 57, 58

AttributedString, 12, 29, 30, 33

AudioInputStream, 15, 25

Besedilo, 16, 29, 36

BufferedImage, 11, 24, 35

BufferedReader, 14, 28

BufferedWriter, 14

Cilj, 16, 45, 49, 55

Clip, 15, 25

Color, 11, 29, 61

Drsnik, 16, 31, 36

FloatControl, 15, 25

Font, 11, 29, 30, 33

FontMetrics, 30, 34

Graphics, 11

Gumb, 16, 29, 35, 36, 38, 40

Igra, 21, 23, 40, 49, 51, 63

Igralec, 24, 40, 43, 47, 51, 58, 63

Izbira nivoja, 16

Jezik, 16, 21, 27, 28

JFrame, 9, 10, 21

JPanel, 10, 12, 21

Kamera, 16, 40, 45, 63

KeyListener, 12, 23

Kocka, 18, 24, 45, 49, 54

meni, 16

MouseListener, 12, 13, 23

MouseMotionListener, 12, 13, 23

narisi, 21, 29, 30, 32, 34, 35, 37, 38, 39, 42, 45, 49, 56, 60, 62, 64

Nastavitve, 16, 21, 28

nivo, 16, 18, 21, 26, 49

Okno, 21, 23

Opeka, 16, 45, 53

Ozadje, 24, 40, 63

Podatki, 45, 51, 59

Pomoč, 16

posodobiStanje, 21, 29, 31, 33, 35, 36, 38, 39, 41, 49, 51, 58, 59, 62, 63

Runnable, 10, 21

Skupina delcev, 49, 55, 56, 59, 61

slika, 38, 58, 63

Slike, 18, 19, 21, 24

Spustni meni, 16, 36

Stanje Izbira nivoja, 38

Stanje Meni, 35

Stanje Nastavitve, 36

Stanje Nivo, 40

Stanje Pomoc, 38

Svet, 40, 49, 54

Špica, 16, 45, 46, 55

Tipke, 21, 23

Trikotnik, 46, 48

Zrak, 45

Zvok, 20, 21, 25, 28, 29, 33

1. URL naslovi na katerih sem dobil zvočne posnetke:  
   - //https://freesound.org/people/Lefty\_Studios/sounds/369515/  
   -//https://freesound.org/people/likeclockwork/sounds/168313/  
   -//https://freesound.org/people/xtrgamr/sounds/277441/ [↑](#footnote-ref-1)