UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

RAČUNALNIŠKA IGRA "POŠASTI" Z UPORABNIŠKO INTERAKCIJO TEMELJEČO NA RAČUNALNIŠKEM VIDU

MATEJ ŠTOLFA

2014
ZAKLJUČNA NALOGA
ŠTOLFA

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Računalniška igra »Pošasti« z uporabniško interakcijo temelječo na računalniškem vidu

("Monsters", a computer game with user interaction based on computer vision)

Ime in priimek: Matej Štolfa
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, avgust 2014

Ključna dokumentacijska informacija

Ime in PRIIMEK: Matej ŠTOLFA

Naslov zaključne naloge: Računalniška igra »Pošasti« z uporabniško interakcijo temelječo na računalniškem vidu

Kraj: Koper

Leto: 2014

Število listov: 45 Število slik: 13

Število prilog: 1 Število strani prilog: 1 Število referenc: 11

Mentor: doc. dr. Peter Rogelj

Ključne besede: programsko inženirstvo, Java, računalniški vid, OpenCV, računalniška igra

Izvleček:
V zaključni nalogi je predstavljen potek programskega procesa na praktičnem primeru in sicer preko načrtovanja in implementacije računalniške igre »Pošasti«. Glavni namen naloge je prikazati premišljen proces, ki je ključen za razvoj kvalitetnega programskega produkta. Arkadna računalniška igra je implementirana v objektnem programskem jeziku Java in s pomočjo knjižnjice za računalniški vid OpenCV. Namen igre je ljudem, ki dolgotrajno delajo za računalnikom, nuditi velikokrat potrebno razgibavanje ob sproščanju in zabavi. Predstavljena je večina faz procesa. Fazi predaja produkta ter obratovanje in vzdrževanje sta izpuščeni, saj naročnik igre ne obstaja. Faze definicija problema, študija izvedljivosti, analiza in definiranje zahtev, načrtovanje aplikacije, izvedba ter testiranje so predstavljene vsaka v svojem poglavju. V zadnjem poglavju so podane zaključne misli in ideje za nadaljnje delo.

Key words documentation

Name and SURNAME: Matej ŠTOLFA

Title of final project paper: "Monsters", a computer game with user interaction based on computer vision

Place: Koper

Year: 2014

Number of pages: 45 Number of figures: 13

Number of appendices: 1 Number of appendix pages: 1 Number of references: 11

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: software engineering, Java, computer vision, OpenCV, computer game

Abstract:
This final project paper presents the course of the software development process with the design and implementation of the game "Monsters". The main purpose of the final project is to show that a well planned process is key to developing a quality software product. The arcade computer game is implemented with the object-oriented programming language Java and with the help of the computer vision library OpenCV. The purpose of the game is to offer often needed limbering up and relaxation simultaneously to people who work long hours behind a computer. Almost all phases of the development process are presented except the deployment, operation and maintenance phase which are skipped because there is no customer for the product. The problem definition, feasibility study, analysis and requirements definition, software design, implementation and testing phase are each described in its own chapter. The last chapter consists of final thoughts and future ideas for further work.

Zahvala

Najprej bi se rad zahvalil mentorju za koristne smernice in vso pomoč pri izdelavi zaključne naloge. Zahvalil bi se tudi vsem najbližjim, ki so me podpirali pri študiju.

Kazalo vsebine

1 Uvod **1**

2 Definicija problema **2**

3 Študija izvedljivosti **3**

4 Analiza in definiranje zahtev **5**
4.1 Namen aplikacije . . . 5
4.2 Primeri uporabe . . . 6
4.3 Igranje igre . . . 6
4.3.1 Opis igre . . . 6
4.3.2 Diagram aktivnosti . . . 8
4.4 Zaznava obraza . . . 10
4.4.1 Diagram aktivnosti zaznave obraza . . . 11
4.5 Lestvica . . . 12
4.6 Nastavitve . . . 12
4.7 Ostale zahteve . . . 13
4.8 Omejitve . . . 13

5 Načrtovanje aplikacije **14**
5.1 Aplikacija . . . 14
5.2 Lestvica . . . 16
5.3 Nastavitve . . . 16
5.4 Grafični vmesnik . . . 17
5.5 Programska oprema . . . 18
5.5.1 Java . . . 18
5.5.2 OpenCV . . . 18

6 Izvedba **19**
6.1 Potek izvedbe in uporabljena orodja . . . 19
6.1.1 Potek . . . 19
6.1.2 Eclipse . . . 20

6.1.3 JavaCV 20
6.1.4 GIMP 20
6.2 Ključni praktični deli implementacije 21
6.2.1 Grafični vmesnik 21
6.2.2 Upodabljanje 23
6.2.3 Zaznava obraza 24

7 Testiranje **27**
7.1 Načrt testiranja 27
7.2 Testiranje glavnega menija 28
7.3 Testiranje lestvice 28
7.4 Testiranje nastavitev 29
7.5 Testiranje zaznave obraza 29
7.6 Testiranje igre 30
7.7 Validacija 30

8 Zaključek **32**

9 Literatura in viri **33**

Kazalo slik

1 Diagram primera uporabe aplikacije 6
2 Skica grafičnega vmesnika 7
3 Diagram aktivnosti 8
4 Diagram aktivnosti zaznave obraza 11
5 Diagram primera uporabe aplikacije 12
6 Komponentni diagram 14
7 Razredni diagram 15
8 Struktura datoteke rezultati.txt 16
9 Skica grafičnega vmesnika glavnega menija 17
10 Glavni meni 21
11 Primer grafičnega vmesnika igre 23
12 Grafični vmesnik nastavitev igre 29
13 Primer trka bombe s pošastjo 31

Kazalo prilog

A Izvorna koda 34

Slovarček

Java: objektno usmerjen programski jezik

JVM (*angl. Java virtual machine*)**:** Javanska navidezna naprava je programje, ki izvaja prevedeno Javansko kodo

JRE (*angl. Java Runtime Environment*)**:** Javanska distribucija, ki vsebuje izvajalno okolje JVM in osnovne knjižnjice za poganjanje Javanskih aplikacij

JDK (*angl. Java Development Kit*)**:** Javanska distribucija, ki poleg vsega kar vsebuje JRE, vsebuje še obsežen nabor programskih knjižnjic, ki omogočajo hiter razvoj Javanskih aplikacij.

AWT (*angl. Abstract Window Toolkit*)**:** Javanska programska knjižnjica, ki omogoča izdelavo grafičnih vmesnikov

Swing: Javanska programska knjižnjica, ki omogoča izdelavo grafičnih vmesnikov

OpenCV: odprtokodna knjižnjica za računalniški vid napisana v C in C++

JavaCV: vmesnik za uporabo funkcij knjižnjice OpenCV s programskim jezikom Java

720p: oznaka za ločljivost velikosti 1280x720 pik

1 Uvod

Igranje video iger je vse bolj pogosta oblika preživljanja prostega časa med odraslimi in seveda mladostniki. Posledica trenda je obilen svetovni trg video iger, ki je v letu 2013 znašal 54 milijard eurov. Napoveduje se 11% letno rast trga in tako naj bi v letu 2017 trg znašal 82 milijard eurov. Trg predstavlja velike finančne možnosti za razvijalce programske opreme [10].

Zgoraj omenjeni trend je eden izmed glavnih razlogov za debelost med mladostniki. Igranje iger naj bi povečevalo njihov apetit. V povprečju mladostnik pri igranju iger porabi le 21 kalorij na uro več kot, če bi počival, pri tem pa naj bi pojedel 80 kalorij več kot sicer. Tako vsaka ura zabave pomeni odvečnih 60 kalorij. Prav zaradi tega predsednik Nacionalnega Foruma za Debelost, David Haslam, meni da bi igralne konzole kot so Wii, ki spodbujajo fizično aktivnost, lahko pomagale pri preprečevanju debelosti [7]. Med Ameriškimi mladostniki, starimi med 12 in 19 let, jih je imelo v letu 2012 z debelostjo probleme kar 20.5% [6].

Iz navedenih razlogov sem se odločil za zaključno nalogo razviti enostavno arkadno računalniško igro z uporabniško interakcijo, temelječo na računalniškem vidu, ki bo spodbujala blago fizično aktivnost. Pri razvijanju igre sem se bolje spoznal s programskim jezikom Java in z zelo razširjeno programsko knjižnjico OpenCV.

Namen naloge je predstaviti vidike programskega inženirstva preko programskega procesa računalniške igre »Pošasti« in sicer definicijo problema, študijo izvedljivosti, analizo in definiranje zahtev, načrtovanje programa, izvedbo ter testiranje. Fazi prevzem produkta ter obratovanje in vzdrževanje sta izpuščeni, saj ni končnega kupca.

Naloga je sestavljena iz sedmih poglavij. V vsakem poglavju je predstavljena ena faza programskega procesa. V zadnjem poglavju so podane zaključne misli in ideje za nadaljnje delo.

2 Definicija problema

Video igre so nezanemarljiv del industrije programske opreme, saj zajemajo kar velik del trga. Vedno več ljudi vseh starosti in obeh spolov igra video igre iz različnih razlogov. Gotovo je zabava, kompeticija, razvedritev in sproščanje od vsakodnevnega stresa del teh razlogov. Igranje iger lahko zelo hitro postane slaba navada ali celo zasvojenost. Slaba drža in zmanjšana fizična aktivnost sta le eni od mnogih posledic zasvojenosti z video igrami, ki negativno vplivata na naše zdravje.

Nemogoče je razviti igro, ki bi bila privlačna za vse uporabnike, saj je privlačnost in ovrednotenje igre subjektivno in varira od posameznika do posameznika. Vseeno pa obstaja nekaj objektivnih smernic za razvoj kvalitetne in privlačne igre. Originalnost igre, kvaliteta in kreativnost grafike so pomembne lastnosti, saj je od njih odvisno, če bo uporabnik igro sploh preizkusil. Neobičajen način uporabniške interakcije, ki ni omejen na standardne periferne naprave (tipkovnica, miška, igralna palica), zagotovo pripomore k originalnosti igre. Enostavna in hitro razumljiva pravila so zaželjena, saj omogočajo hiter začetek igranja. Kljub enostavnim pravilom mora biti igra dovolj zahtevna in izzivalna. Morebitna lestvica še pripomore k izzivalnosti. Napetost in zahtevnost igre se morata postopoma večati in tako nuditi uporabniku dovolj časa, da se privadi na igro. Pogostost uporabnikove želje po igranju igre je dober faktor za oceno kvalitete igre [4].

Z računalniško igro »Pošasti« želimo nuditi ljudem sprostitev in razgibavanje, ki jo potrebujejo zaradi dolgotrajnega dela in sedenja za računalnikom. Igra je namenjena vsem, ki si želijo nekaj minut dnevne zabave in razbremenitve stresa brez negativnih učinkov na zdravje.

Cilj projekta je razviti zgoraj omenjeno igro in pri tem upoštevati smernike za razvoj kvalitetne igre. Stroški projekta bodo ničelni, saj gre za akademski projekt in bodo za razvoj uporabljena samo odprtokodna orodja.

3 Študija izvedljivosti

Predlagana rešitev je arkadna igra z nazivom »Pošasti«. Glavni akter bo pošast, ki si jo bo uporabnik izbral iz nabora simpatičnih pošasti, s katero bo nato poskušal uloviti čim več padajočih dragocenih kamnov in specialnih predmetov. Pri tem se bo moral izogibati nevarnemu predmetu, ki bo pomenil konec igre. Pošast bo kontroliral preko spletne kamere tako, da bo premikal obraz levo in desno vzporedno kameri. Za igro se pričakuje, da bo zabavna in ne bo negativno vplivala na uporabnikovo držo.

Pri študiji izvedljivosti projekta sta prisotni dve glavni oviri. Prva ovira so stroški projekta, ki niso relevantni v našem primeru, saj gre za zaključno nalogo *oz*. univerzitetni projekt. Uporabljala se bodo le odprtokodna orodja in sistemi, tako bodo stroški ničelni. Druga ovira pa je rok izdelave. Projekt mora biti pravočasno zaključen in pri tem se ne sme ustvariti dodatnih stroškov poleg lastnega dela. Zaradi neobstoječih stroškov se lahko projekt izvede brez finančnih sredstev.

V navadi je razdeliti razvoj večjih aplikacij v manjše neodvisne podkomponente. Razdelitev bistveno pohitri razvoj produkta v primeru razvojnih skupin, saj omogoča vzporeden razvoj. V našem primeru to ni relevantno, saj skupino sestavlja samo en član. Ne glede na velikost razvijalske skupine, so manjše neodvisne komponente boljše za obvladovanje razvoja in za testiranje.

Predvideni rok za zaključek projekta je konec avgusta 2014, kar pomeni da smo časovno omejeni na približno osem tednov. Razvoj računalniške igre »Pošasti« bo razdeljen na štiri skoraj neodvisne komponente in sicer na igro, zaznavo obraza, nastavitve in lestvico. Igri z grafičnim vmesnikom bosta namenjena dva tedna, nastavitvam in lestvici en teden, zaznavanju obraza pa kar tri tedne, saj se pričakuje, da bo ta komponenta tehnično najzahtevnejša. Predzadnji teden bo namenjen testiranju, zadnji teden pa popravkom in izdelavi dokumentacije.

Za razvoj bo uporabljen razširjen programski jezik Java zaradi prenosljivosti med operacijskimi sistemi, mnogih brezplačnih knjižic ter same tehnične podkovanosti. Pri implementaciji zaznavanja obraza bo uporabljena odprtokodna knjižnica za računalniški vid OpenCV.

Projekt je operativno izvedljiv, saj ne gre za zlonamenski programski produkt in ne bo kršil nobenih zakonov in etičnih norm in pri tem se ne bo ustvarilo protipravno

premoženje.

Ker z izpeljanim projektom dosežemo zastavljeni cilj, je izpeljava projekta upravičena. Cilj je opraviti zaključno nalogo in s tem dodiplomski študij.

4 Analiza in definiranje zahtev

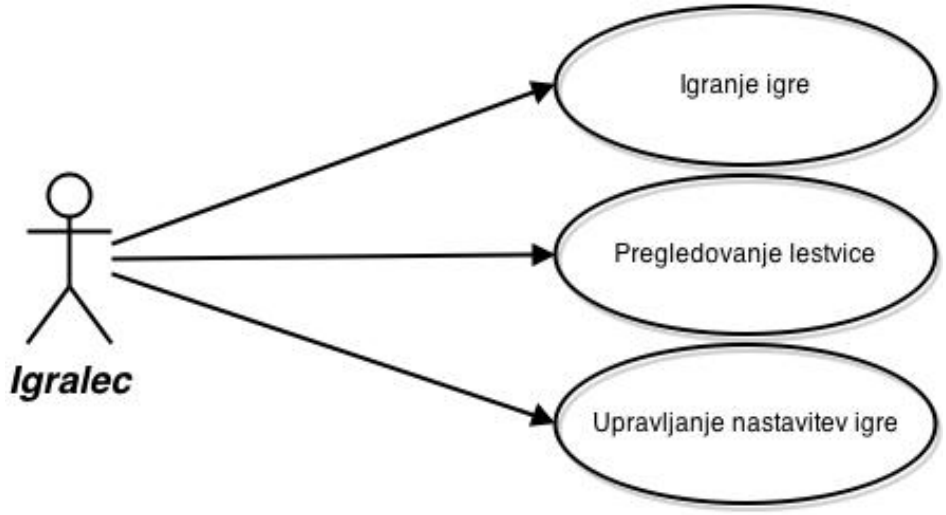
V tem poglavju je predstavljena tretja faza linearnega programskega procesa *tj.* analiza in definiranje zahtev. Običajno jo izvaja sistemski analitik, ki pri tem skuša definirati zahteve programskega produkta na čim bolj popoln in ekspliciten način. Informacije dobi preko intervjujev, vprašalnikov in z analizo obstoječih produktov. Končni rezultat faze je dokument imenovan specifikacija zahtev, ki nudi celovit vpogled v interakcijo uporabnika s programskim produktom in v njegovo delovanje. Iz specifikacije zahtev je razvidna pravilnost razumevanja problema s strani organizacije, ki bo za naročnika *oz.* končne uporabnike razvila programski produkt. Faza se konča z recenzijo dokumenta, s katero se preveri veljavnost, konsistentnost, popolnost in realnost vseh zahtev [9]. V našem primeru gre za zaključno nalogo, zato bom sam prevzel vlogo končnega uporabnika, naročnika in sistemskega analitika.

4.1 Namen aplikacije

Računalniška igra "Pošasti" se razvija za ljudi, potrebne razgibavanja povezanega z dolgotrajnim sedenjem za računalnikom. Igra ne bo negativno vplivala na igralca, nasprotno ob sprostitvi mu bo nudila še koristno razgibavanje. Zahteve so naslednje:

- originalna grafika ter zvočni efekti
- neobičajen način uporabniške interakcije
- enostavna in hitro razumljiva pravila igre
- lokalna lestvica
- osnovno upravljanje nastavitev
- postopno večanje zahtevnosti igre

4.2 Primeri uporabe



Slika 1: Diagram primera uporabe aplikacije

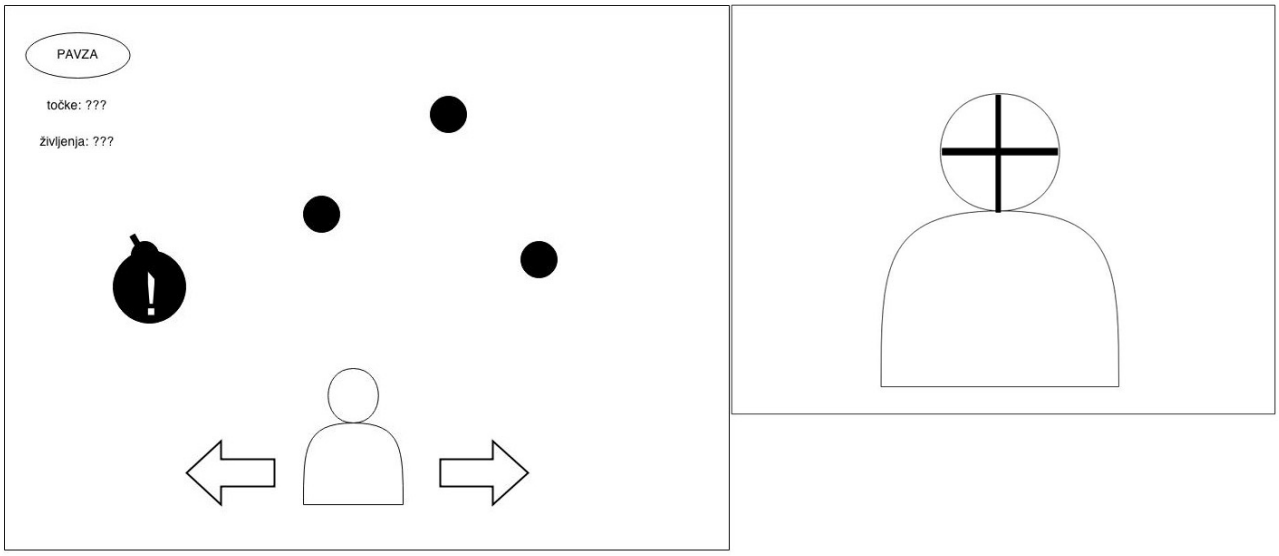
Diagram prikazuje primer uporabe glavnih funkcij aplikacije. Aplikacija se zažene z dvoklikom na *.jar* datoteko ali preko ukazne vrstice z ukazom $java -jar\ Posasti.jar$. Igralcu se prikaže glavni meni, iz katerega lahko zažene igro, pregleda lokalno lestvico ali vstopi v nastavitve aplikacije.

4.3 Igranje igre

4.3.1 Opis igre

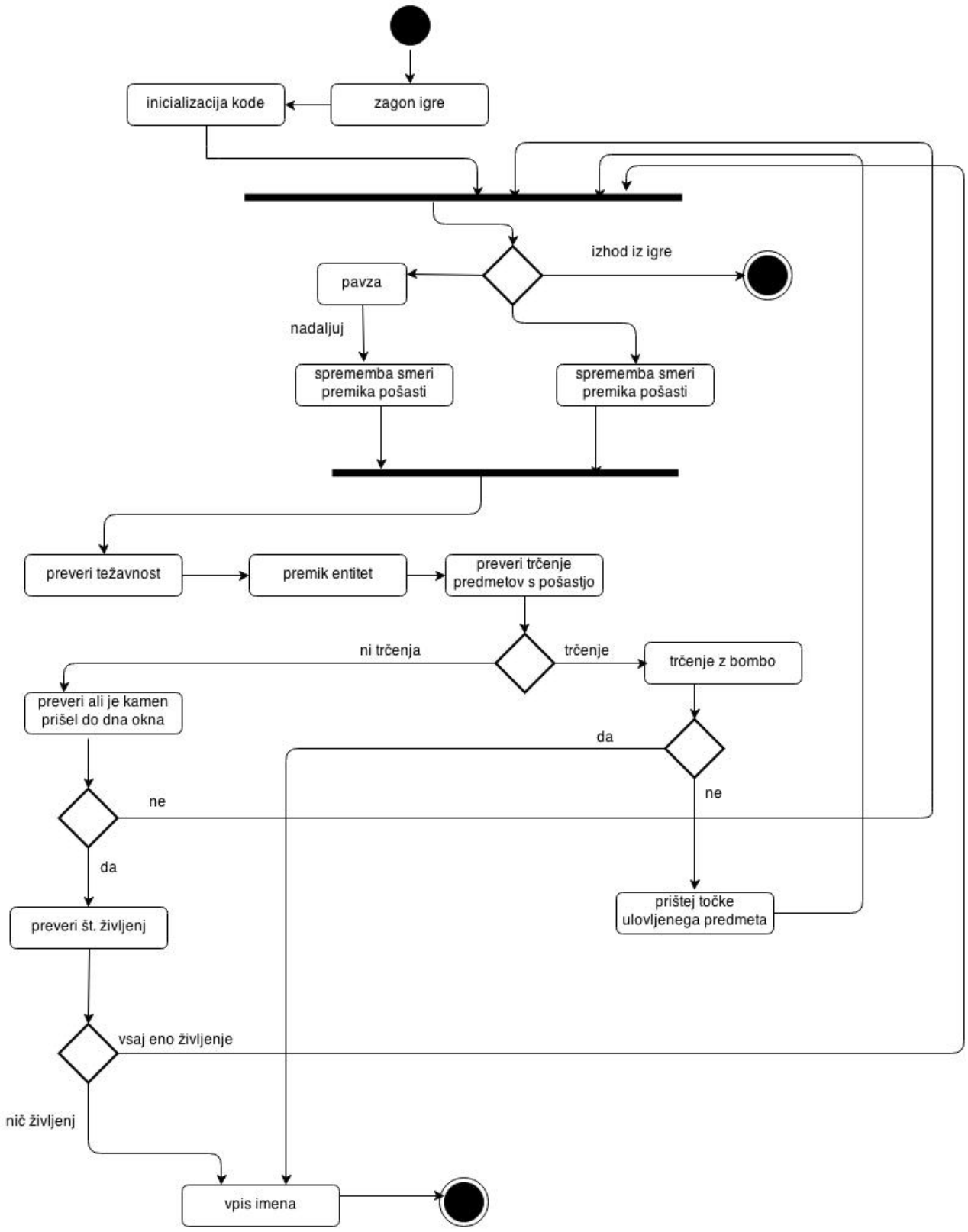
Igra se bo začela v prazni sobi, kjer se bo nahajal glavni akter igre *oz.* pošast, katero bo vodil igralec. Vodil jo bo tako, da bo premikal svoj obraz vzporedno spletni kameri v levo ali desno smer. Pošast se bo premikala v izbrani smeri, dokler igralec ne bo premaknil obraza v drugo smer. Z vrha ekrana se bodo z naključno hitrostjo in v naključni smeri prikazovali *oz.* padali dragoceni kamni. Sčasoma jih bo vedno več in tako bo igra vedno težja. Različna barva dragocenega kamna bo prinašala različno število točk. Poleg kamnov se bo občasno pojavil napis *Famnit*, ki bo prinašal največ točk med padajočimi predmeti. Cilj igre bo zbrati čim več točk in se tako uvrstiti v sam vrh lokalne lestvice. Igra se bo zaključila, ko bo igralcu zmanjkalo vseh pet začetnih življenj. Igralec bo izgubil eno življenje za vsak neulovljen dragocen kamen. Če se igralec ne bo uspel izogniti še tretji vrsti padajočega predmeta - bombi, se bo igra zaključila. Igro bo lahko igralec kadarkoli začasno prekinil (premor) s klikom na tipko

pavza, ki se bo nahajala v zgornjem levem kotu igralnega okna. Pod gumbom bosta dva napisa. Prvi gumb bo prikazoval število doseženih točk, drugi pa preostalo število življenj. Poleg okna, v katerem se bo izvajala igra, bo še okno spletne kamere. Igralcu bo v tem oknu posredovana informacija, ali je aplikacija zaznala sredino njegovega obraza ali ne, tako da se bo tam izrisal strelski merek. V primeru, da aplikacija ne bo zaznala pravilno obraza, se priporoča igralcu, da ponovno postavi svoj obraz vzporedno s spletno kamero.



Slika 2: Skica grafičnega vmesnika

4.3.2 Diagram aktivnosti



Slika 3: Diagram aktivnosti

Potek diagrama aktivnosti:

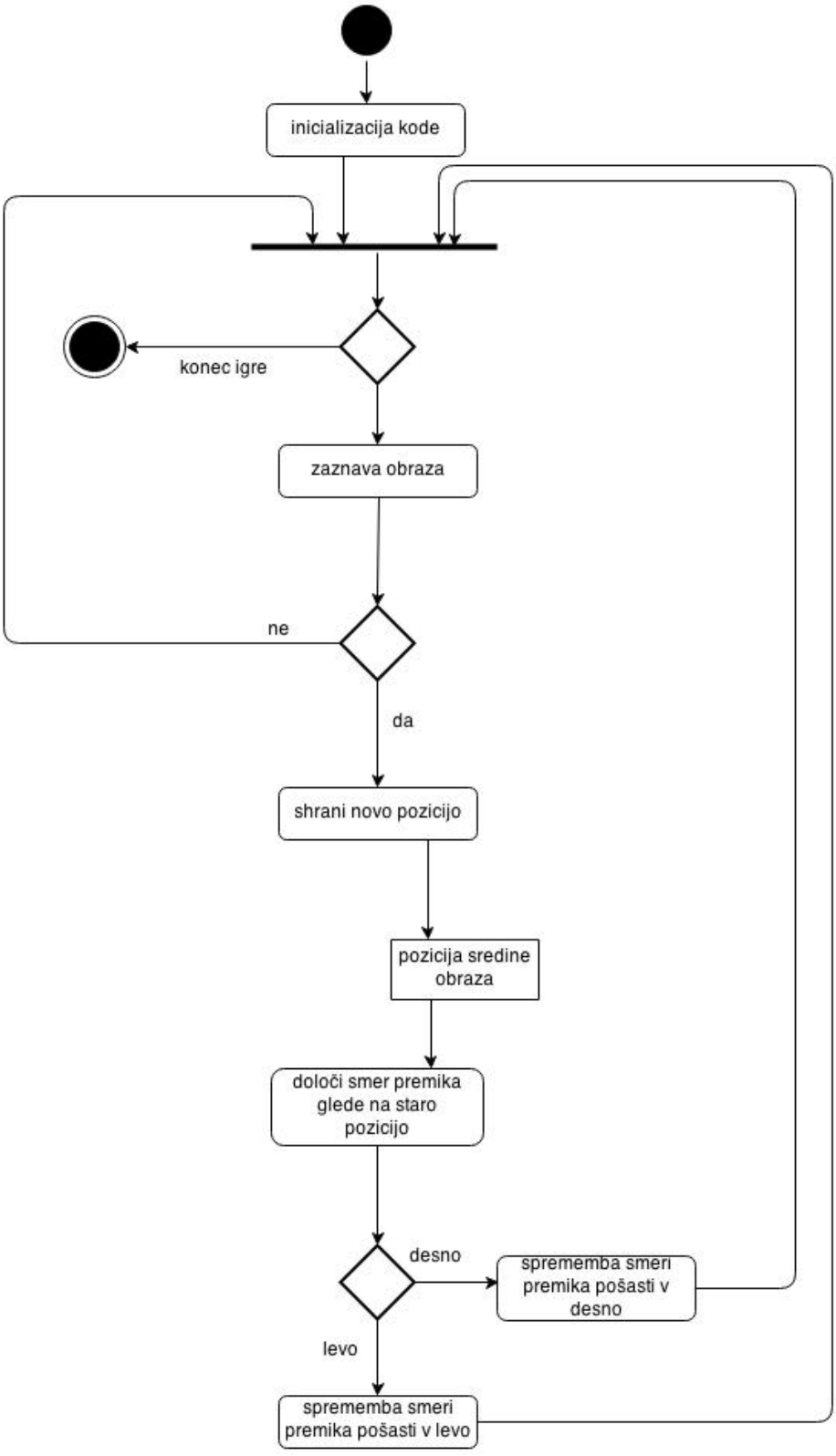
1. Igralec zažene igro
2. Inicializirata se programska koda in spletna kamera
   - 2.1. Alternativa: Igralec zapusti igro
   - 2.2. Alternativa: Igralec postavi igro v premor
     - 2.2.1. Igralec nadaljuje igro
3. Kamera zazna premik obraza in nastavi smer premika pošasti
4. Preveri se trajanje igre in nastavi primerno težavnost
5. Premaknejo se padajoči predmeti ter pošast (entitete)
6. Preveri se morebitno trčenje entitet
7. Ker ni bilo trčenja, se preveri, ali je kamen prišel do dna igralnega okna
   - 7.1. Alternativa: Prišlo je do trčenja
     - 7.1.1. Ni bilo trčenja z bombo
       - 7.1.1.1. Prišteje se vrednost ulovljenega predmeta k skupnim točkam
       - 7.1.1.2. Začne se nov korak poteka diagrama od prvega stika naprej
     - 7.1.2. Alternativa: Prišlo je do trčenja z bombo
       - 7.1.2.1. Igralec vpiše svoje ime
       - 7.1.2.2. Igra se zaključi
8. Kamen je prišel do dna, igralcu se odvzame eno življenje
   - 8.1. Alternativa: Kamen ni prišel do dna okna
     - 8.1.1. Začne se nov korak poteka diagrama od prvega stika naprej
9. Preveri se, ali igralcu preostane še kakšno življenje
10. Igralec ima še življenj, začne se nov korak poteka diagrama od prvega stika naprej
    - 10.1. Alternativa: Igralec nima več življenj
      - 10.1.1. Igralec vpiše svoje ime
      - 10.1.2. Igra se zaključi

4.4 Zaznava obraza

Igralec bo z igro interaktiral izključno preko računalniškega vida *oz*. spletne kamere. Računalniško miško bo potreboval le za postavitev igre v premor in za navigacijo po aplikaciji, tipkovnico pa le pri vpisovanju imena ob zaključku igre.

Pošast, ki jo bo igralec kontroliral, bo ob začetku igre stala na mestu do trenutka, ko bo aplikacija zaznala igralčev obraz. Ob prepoznavi obraza se bo pošast začela premikati v levo, če bo igralčev obraz na levi polovici *oz*. v desno, če bo igralčev obraz na desni polovici zornega kota spletne kamere. Pošast bo spremenila smer premika v trenutku, ko bo igralec premaknil obraz v drugo smer. Od igralca ne bodo zahtevani dolgi premiki, ampak le tolikšni, da bo kamera zaznala premik.

4.4.1 Diagram aktivnosti zaznave obraza



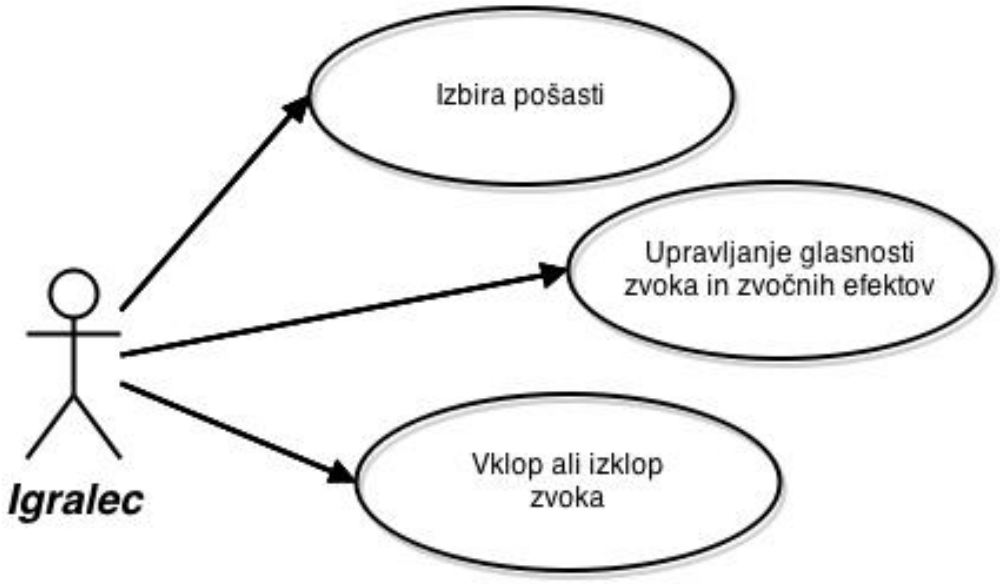
Slika 4: Diagram aktivnosti zaznave obraza

4.5 Lestvica

Lestvica bo igri prinesla dodatno kompetitivnost med igralci, ki bodo igrali na istem lokalnem računalniku. Prikazovala bo pet najvišjih rezultatov, imena tistih, ki so rezulate dosegli ter čas in datum doseženega rezultata. Zaradi majhnega števila vnosov, ki jih bo hranila lestvica, ne bo uporabljena podatkovna baza za implementacijo lestvice, ampak le običajna tekstovna datoteka.

4.6 Nastavitve

Igralec bo imel možnost izbire pošasti, s katero bo lovil dragocene kamne. Izbiral bo med petimi pošasti, ki se bodo razlikovale v barvi in obraznih potezah. Izbira bo predvsem namenjena mlajšim igralcem, ki jim bo funkcionalnost prinašala dodatno zabavo. Poleg tega bo še možnost nastavljanja glasnosti glasbe in efektov v igri. Po želji se bo lahko zvok tudi izključil.



Slika 5: Diagram primera uporabe aplikacije

4.7 Ostale zahteve

Uporabnik bo moral imeti za delovanje aplikacije inštalirano Javo. Bolj specifično, JDK - Java Development Kit *oz.* vsaj JRE - Java Runtime Environment. Poleg Jave bo moral imeti še OpenCV 2.4.6 knjižnico. Oboje je prosto dostopno na internetu. Kar se tiče strojne opreme, bo uporabnik za igranje igre potreboval spletno kamero. Priporoča se HD *oz.* 720p spletna kamera, saj boljša kot bo kamera, boljše bo aplikacija zaznavala obraz igralca.

4.8 Omejitve

Za implementacijo prepoznavanja obraza se bo uporabljal predhodno usposobljen Haar klasifikator[1], ki je del knjižnice OpenCV. Namenjen je prepoznavanju obrazov le iz frontalne perspektive. Bolj specifično, za dobro detekcijo, bo moral biti obraz skoraj v celoti viden in čim manj pod kotom *oz.* paralelen spletni kameri. Ob neupoštevanju slabosti klasifikatorja, bo prepoznava obraza med igro neoptimalna.

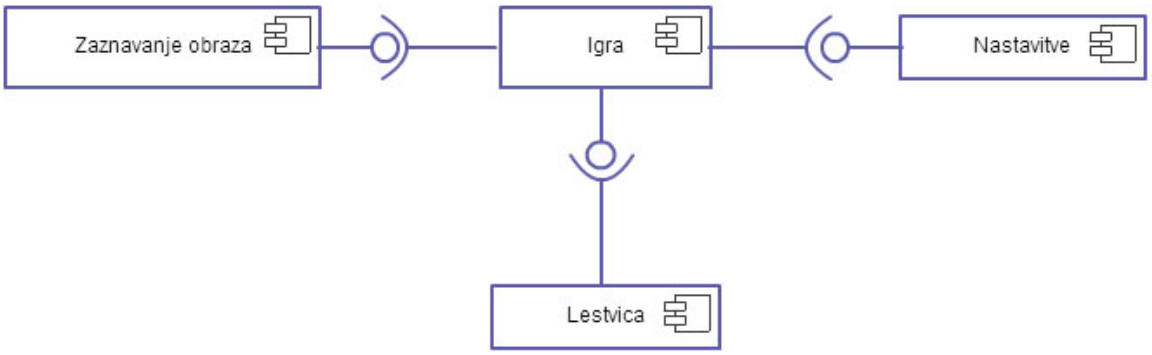
[1] `https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml`

5 Načrtovanje aplikacije

Pri načrtovanju sistema *oz.* aplikacije je potrebno aplikacijo razdeliti na komponente, ki bodo zagotavljale stabilno arhitekturo in čim manjšo medsebojno odvisnost in s tem zagotavljale enostavno vzdrževanje in dodajanje funkcionalnosti [8]. Razdelitev opišemo s preslikavo modela specifikacije zahtev v načrtovalski model aplikacije. Model prikazuje komponente, njihove vmesnike in medsebojno odvisnost. V našem primeru je v fazi načrtovanja aplikacije združeno celotno načrtovanje, *tj.* sistema in komponent.

5.1 Aplikacija

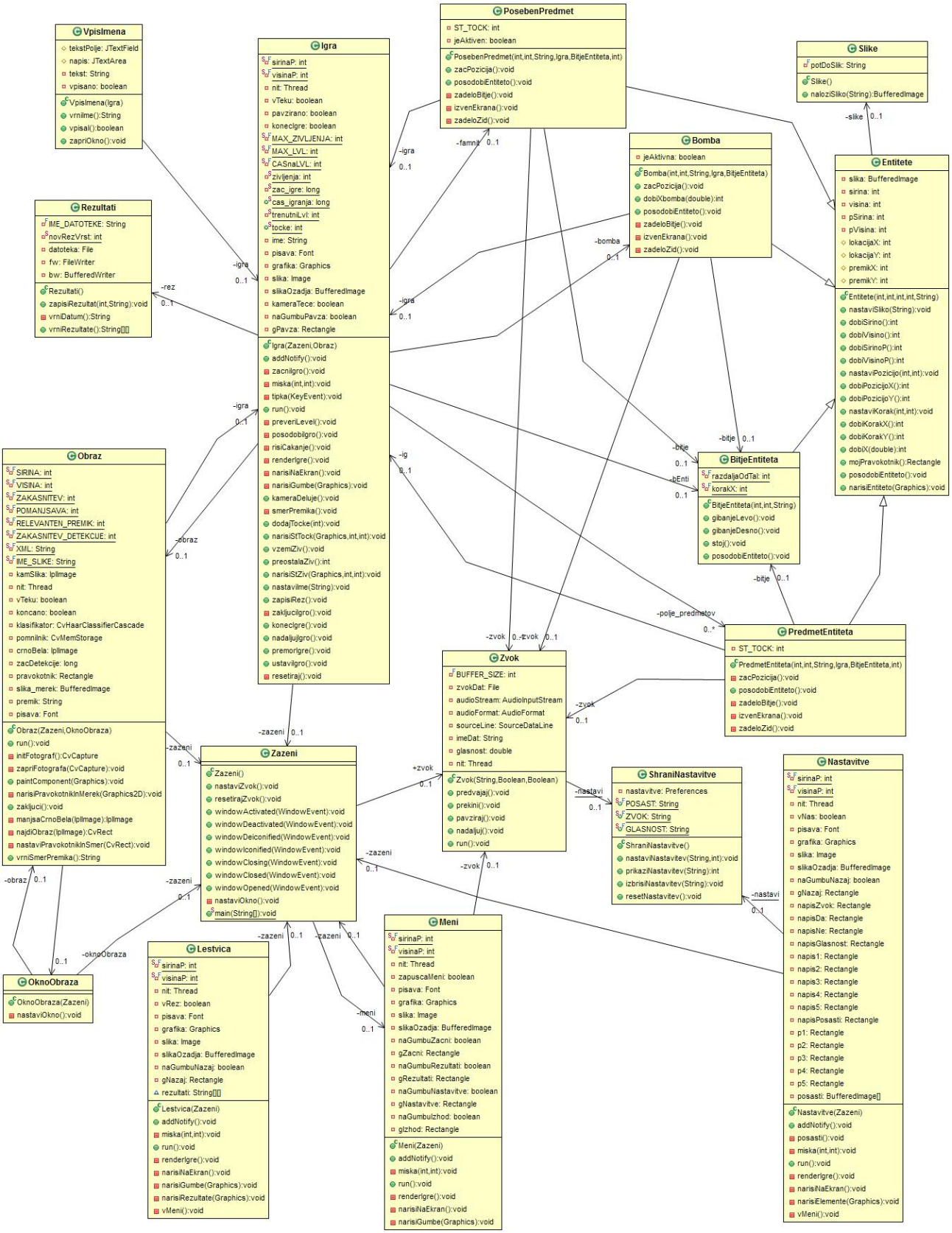
V našem primeru je primerna razdelitev, ki jo prikazuje naslednji komponentni diagram.



Slika 6: Komponentni diagram

Komponetni diagram nam prikazuje strukturo aplikacije v višjem abstraktnem nivoju kot razredni diagram, ki mu sledi. Iz slike je razvidno, da je igra glavna komponenta in potrebuje dve komponenti za delovanje. Zaznavanje obraza posreduje podatek pozicije igralčevega obraza in s tem kontrolo igri. Nastavitve nudijo igri dodatno prilagajanje. Igra zapiše rezultate v lestvico in s tem zagotovi njeno uporabnost.

Sledeči razredni diagram na sliki 7 prikazuje razdelitev aplikacije bolj podrobno in sicer v razrede. Poleg razredov *oz.* strukture aplikacije nam diagram prikazuje še



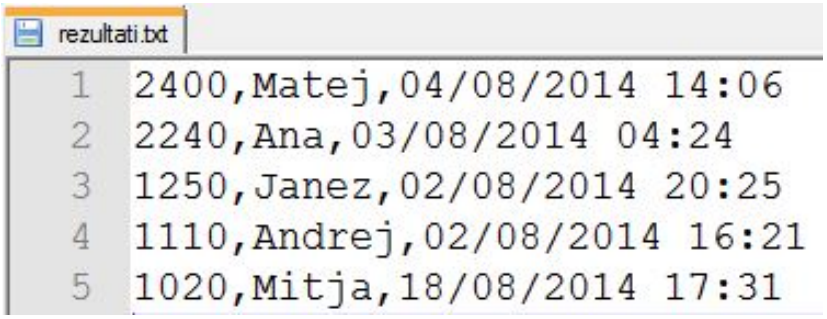
Slika 7: Razredni diagram

atribute, metode in relacije med njimi. Med relacije spadajo morebitne asociacije, agregacije, kompozicije in posploševanja [2].

Komponenta "Zaznavanje obraza" bo implementirana z razredoma "OknoObraza" in "Obraz", komponenta "Nastavitve" z razredom "Nastavitve", komponenta "Lestvica" z razredi "VpisImena", "Rezultati" in "PrikazRez". Osrednja komponenta "Igra" pa s preostalimi razredi, ki so prikazani na razrednem diagramu in sicer "Zazeni", "Meni", "Igra", "Slike", "Zvok", "Entitete", "BitjeEntiteta", "PredmetEntiteta", "PosebenPredmet" in z razredom "Bomba".

5.2 Lestvica

Iz zahtev definiranih v prejšnji fazi ni potrebe po podatkovni bazi za implementacijo funkcionalnosti "Lestvica", ki bo hranila le pet vnosov najboljših rezulatov. Za vsak vnos bo hranila še ime igralca, ki je dosegel rezultat ter datum in čas dosega rezultata. Podatki se bodo shranjevali v navadni tekstovni datoteki (*.txt*) in bodo med seboj ločeni z vejico. Podatki bodo oblike "rrrr,ime,dd/mm/yyyy hh:mm", kjer "r" predstavlja posamezno cifro doseženih točk, "ime" vpisani naziv igralca, in "dd/mm/yyyy hh:mm" obliko datuma doseženega rezultata. Primer take datoteke vidimo na naslednji sliki.



Slika 8: Struktura datoteke rezultati.txt

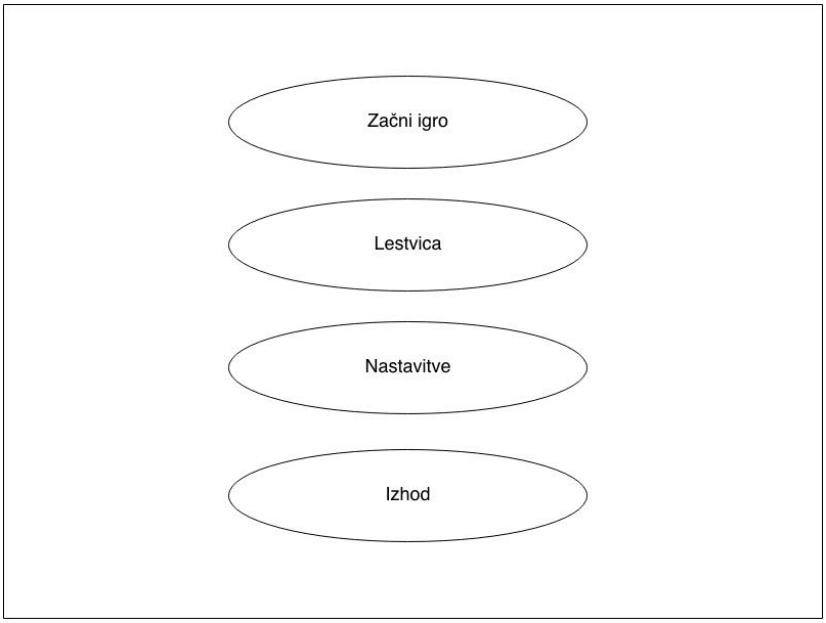
5.3 Nastavitve

Tudi nastavitve igre bi bile lahko shranjene v navadni tekstovni datoteki, saj bo nastavitev razmeroma malo. Predvidene so tri vrste nastavitev. Prva nastavitev je možnost vključitve in izključitve zvoka ter zvočnih efektov. Druga nastavitev je izbira glasnosti zvoka z naborom vrednosti od ena do pet. Zadnja predvidena nastavitev je izbira izgleda pošasti z naborom petih različnih likov. Zaradi možne kasnejše

razširitve pa se bo lahko pojavila potreba po večjem številu različnih nastavitev. S tem namenom se bo uporabil programski vmesnik $Preferences\ API$, ki je del programskega jezika Java. Vmesnik omogoča shranjevanje nastavitev na sistematičen in eleganten način. Nastavitve se shranjujejo v paru oblike ključ / vrednost v posebne datoteke - vozlišča. Dejansko shranjevanje podatkov se razlikuje od operacijskega sistema. V primeru Windows 7, se nastavitve shranijo v registre pod lokacijo "$HKEY\_CURRENT\_USER\backslash Software\backslash JavaSoft\backslash Prefs$".

5.4 Grafični vmesnik

Kot definirano v fazi analize in definiranja zahtev bo igra "Pošasti" enostavna arkadna igra s kreativnim in enostavnim grafičnim vmesnikom. Zatorej morajo biti tudi grafični vmesnik glavnega menija, nastavitev in lestvice skladno enostavni in primerni arkadni igri. Na sliki 9 je prikazana skica glavnega menija.



Slika 9: Skica grafičnega vmesnika glavnega menija

Kot je razvidno is slike, bo glavni meni vseboval štiri zmerno velike in čitljive gumbe za hiter dostop do omenjenih funkcionalnosti. Ozadje bo sestavljeno v enakem slogu ozadja, ki bo prikazano med igro. S tem bo imela igra konsistenten izgled čez celotno aplikacijo. K poživljajočemu občutku grafičnega vmesnika bodo pripomogle svetle barve napisov in drugih grafičnih elementov. Ostali meniji bodo implementirani v istem slogu.

Grafični vmesnik bo implementiran s pomočjo Javanskih programskih knjižnjic za izdelavo grafičnih programov Java AWT in Java Swing. Knjižnjica AWT bo uporabljena za izdelavo uporabniških oken v slogu operacijskega sistema, na katerem bo tekla aplikacija. Za izdelavo preostalih delov grafičnega vmesnika bo uporabljena knjižnjica Swing. S tem bo zagotovljen isti izgled aplikacije na vseh operacijskih sistemih, hkrati s tem pa tudi prenosljivost.

5.5 Programska oprema

Izbira programske opreme je pomemben dejavnik za uspešnost naslednjih faz programskega procesa *oz*. dobrega razvoja aplikacije. Pri izbiri igrajo ključno vlogo predhodne izkušnje skupine oseb in razvijalcev, ki izbirajo opremo. V naslednjih dveh podpoglavjih sta na kratko predstavljena programski jezik Java in odprtokodna knjižnjica OpenCV kot predlagana programska oprema.

5.5.1 Java

Programski jezik Java je popularna izbira razvijalcev zaradi dobrih performans, dokumentacije in podpore. V našem primeru je dobra izbira zaradi same tehnične podkovanosti z jezikom, odprtokodnosti in prenosljivosti med operacijskimi sistemi. Delovanje naše aplikacije bo enako in ne bo pod vplivom specifičnosti posameznega operacijskega sistema, ker Javanske aplikacije tečejo v svojem virtualnem okolju JVM.

5.5.2 OpenCV

Knjižnjica za računalniški vid OpenCV je odlična izbira za našo aplikacijo, saj je zelo razširjena, odprtokodna, s številnimi funkcijami, ki zajemajo različna področja računalniškega vida in tudi strojnega učenja. Je enostavna za uporabo in omogoča hiter razvoj kompleksnih aplikacij, ki temeljijo na zaznavi in prepoznavi različnih objektov. Vse njene funkcionalnosti so obširno dokumentirane [1]. Ne glede na to, da je OpenCV napisana v programskem jeziku C in C++, obstajajo vmesniki za številne druge programske jezike, med katere spada tudi Java.

6 Izvedba

Izvedba je najbolj kritična faza programskega procesa, saj je od nje odvisno, ali bo projekt uspešen. V nekaterih primerih celo vpliva na uspešnost celotnega podjetja, ki razvija programski produkt. Posledice slabo definiranih zahtev iz prejšnjih faz se izkažejo ravno v izvedbi. Vzrokov za neuspešnost je veliko, na primer premalo dodeljenega časa za implementacijo ali nerealne zahteve in pričakovanja. Posledice pa so neskladnosti funkcionalnosti produkta z željami ter zahtevami naročnika, ne dovolj prepusten in kvaliteten produkt, *itd.*.

V nadaljevanju poglavja je predstavljen postopek izvedbe, uporabljena orodja in ključni praktični deli implementacije.

6.1 Potek izvedbe in uporabljena orodja

6.1.1 Potek

Implementacije produkta smo se lotili z inštalacijo vseh potrebnih programskih orodij. Namestili smo naslednje:

- Java SDK 7[1]
- Eclipse IDE Juno[2]
- OpenCV 2.4.6[3]
- JavaCV 0.6[4]
- GIMP[5]

[1] `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

[2] `http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junosr2`

[3] `http://opencv.org/downloads.html`

[4] `https://code.google.com/p/javacv/downloads/detail?name=javacv-0.6-src.zip&can=4&q=`

[5] `http://www.gimp.org/downloads/`

Po namestitvi smo začeli z implementacijo osnovnih gradnikov, kot so okno igre, glavni meni, entitete (pošast, dragoceni kamni, napis "Famnit" in bomba). Zatem smo s pomočjo urejevalnika slik GIMP ustvarili ozadja igre in napis "Famnit". Gumbe smo ustvarili kar z osnovnimi funkcijami Java Swinga. Odprtokodne slike za entiteto pošast, kamne in bombo smo pridobili na spletni strani OpenGameArt[6]. Poleg slik pa tudi vse zvoke potrebne za igro, ki so seveda odprtokodni. Nato smo implementirali glavne komponente igre kot so igralni pogon, premik entitet, kolizija entitet. V nadaljevanju pa še manj zahtevne dele igre. Igra je v tem obdobju že delovala vendar z uporabniško interakcijo temelječo na smernih tipkah na tipkovnici. Dodali smo še lestvico in nastavitve. Nazadnje je bila implementirana še kontrola igre temelječa na zaznavi igralčevega obraza. S tem se je končala faza izvedbe.

6.1.2 Eclipse

Eclipse je integrirano razvojno okolje implementirano v Javi, ki podpira mnoge programske jezike. Okolje prinaša mnogo prednosti razvijalcem ne glede na kompleksnost projekta, na katerem delajo. Med glavne funkcionalnosti razvojnega okolja spada sprotno opozarjanje na sintaktične napake, orodja za razhroščevanje, pregleden vmesnik za upravljanje z razredi, lahko dostopna dokumentacija v pojavnih oknih in še marsikaj. Poleg privzetih funkcionalnosti omogoča dodajanje vtičnikov, ki dodajo okolju še več funkcionalnosti kot na primer risanje različnih UML diagramov.

6.1.3 JavaCV

JavaCV je ovojnica oziroma vmesnik za dostop do funkcij knjižnjice OpenCV napisanih v C in C++. Je potreben pripomoček, če želimo uporabljati knjižnjico OpenCV s programskim jezikom Java. Za vsako različico knjižnjice potrebujemo primerno ovojnico za dostop do nje. Tako v našem primeru za uporabo OpenCV 2.4.6 potrebujemo JavaCV 0.6. Če bi naprimer želeli uporabljati OpenCV 2.4.2, bi tako potrebovali JavaCV 0.2 *itd.*. Potrebno je omeniti, da ovojnica ni namenjena ovijanju izključno OpenCV-ja ampak hkrati tudi ostalih popularnih knjižnjic kot so FFmpeg, OpenKinect in drugih.

6.1.4 GIMP

GIMP (GNU Image Manipulation Program) je odprtokodno programsko orodje za obdelavo slik. Deluje na operacijskih sistemih Linux, Windows in tudi Mac OSX. Orodje je odlična alternativa ostalim komercialnim programom, saj zna obdelovati večino grafičnih formatov in nudi številen nabor funkcionalnosti, ki ga je mogoče še

[6] `http://opengameart.org/`

razširiti s pomočjo dodajanja vtičnikov. Ponuja tudi vmesnik za poganjanje naprednih ukaznih datotek (skript), ki omogočajo avtomatizacijo od najbolj osnovnih do najbolj zahtevnih množic procedur.

6.2 Ključni praktični deli implementacije

Sledijo kratki opisi delovanja posameznih delov aplikacije in pomembne vrstice programske kode.

6.2.1 Grafični vmesnik

Za izdelavo enostavnega grafičnega vmesnika, tako kot je bil opredeljen v fazi analize in definiranja zahtev, smo uporabljali v večini le osnovne komponente Java Swing-a. Gumbe smo izdelali s pomočjo razreda Rectangle in ne s pomočjo razreda JButton. Tako izdelani gumbi dajejo vmesniku bolj enostaven in kreativen videz kot sicer. Na sliki 10 je prikazan glavni meni igre, ki vsebuje štiri gumbe.

Slika 10: Glavni meni

Gumbe smo implementirali na sledeči način:

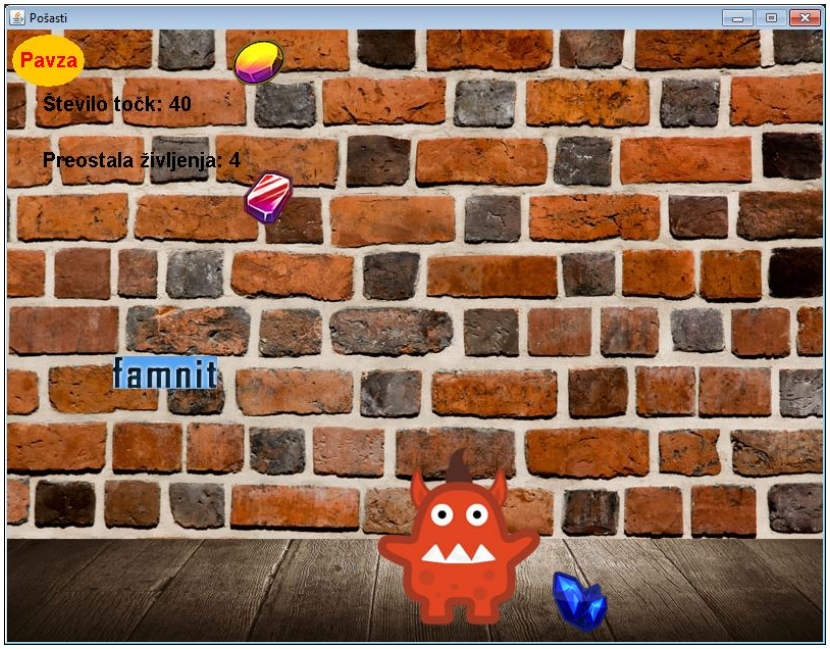
```
gumb = new Rectangle(x, y, sirina, visina);

narisiGumbe(Graphics g){
        g.setColor(Color.ORANGE);
        g.fillOval( gumb.x, gumb.y, gumb.width, gumb.height);
        g.setFont(pisava); g.setColor(Color.RED);
        g.drawString("Nastavitve", gumb.x, gumb.y);
}

/*S klikom miske se sprozi metoda mousePressed v MouseListener-ju */
mousePressed(MouseEvent e){
        miska(e.getX(), e.getY());
}

/* Metoda miska upravlja s kliki */
miska(x,y){
        /* Ali se je zgodil klik nad gumbom? */
        naGumbu = gumb.contains(x,y) ? true : false;
        if(naGumbu){/* akcija, ki jo sprozi gumb */}
        ...
}
```

Za izdelavo ozadja igre »Pošasti«, kot ga vidimo na sliki 11, nam je orodje za perspektivo omogočilo ustvariti 3D efekt sobe. Sloj oziroma sliko z leseno podlago smo izkrivili in postavili na sliko s kamnito podlago. Nato smo z omenjenim orodjem spremenili perspektivo sloja z leseno podlago in ga namestili na primerno mesto na dno kamnitega sloja. Novonastala slika daje prostorski občutek.



Slika 11: Primer grafičnega vmesnika igre

6.2.2 Upodabljanje

Upodabljanje je le ena izmed mnogih funkcionalnosti, ki sodijo med igralni pogon. Ostale funkcionalnosti pogona so fizika, zvok, animacija, umetna inteligenca, omrežje, uporabniški vmesnik in druge.

V našem primeru, ker gre za računalniško igro, je pričakovano, da bo potrebna visoka gostota prikazovanja slik na zaslon. Zato se ne moremo zanesti na pasivno upodabljanje, ki je primerno za večino programskih produktov, ki niso igre.

Pri pasivnem upodabljanju se vso programsko kodo namenjeno grafiki napiše v povoženo metodo *paint*() ali *paintComponent*() razreda JPanel. Prikaz na zaslon se sproži z ukazom repaint(), vendar pri tem nimamo kontrole, kdaj se bo dejansko grafika prikazala. Prikaz se izvrši, ko pride dogodek na konec čakalne vrste JVM. Zato se običajno za računalniške igre uporablja aktivno upodabljanje.

Z aktivnim upodabljanjem pa imamo kontrolo nad tem, kdaj se bo grafika prikazala na zaslonu. Ustvariti moramo metodo, ki zgradi sliko ne da bi jo pri tem izrisovala na zaslon. Ko je slika pripravljena za izris, pokličemo svojo metodo za izris na zaslon [11]. Pri takem pristopu prikazovanja je potrebno paziti, da se slike menjujejo v določenem časovnem intervalu. S tem zagotovimo gladko igralno izkušnjo. Programska koda s kratkimi komentarji opisanega postopka je podana v nadaljevanju.

```
void run(){
        while (true) {
                if (((System.currentTimeMillis() - cas_posodobitve) >
                    ZAKASNITEV)){ /*ZAKASNITEV je casovni interval med
                    posodobitvijo zaslona*/
                        posodobiIgro(); /* posodobi stanje igre in pozicije
                            entitet*/
                        sestaviPrikaz(); /* sestavi sliko za prikaz na zaslon*/
                        narisi(); /*narisi grafiko na zaslon*/
                        cas_posodobitve = System.currentTimeMillis();
                }
                trajanje = System.currentTimeMillis() - cas_posodobitve;
                if (trajanje < ZAKASNITEV){
                        /* V primeru, da se ni minil casovni interval, pocakaj
                            preostali cas*/
                        try { Thread.sleep(ZAKASNITEV-trajanje); }
                        catch (Exception e) {}
                }
        }
}
```

6.2.3 Zaznava obraza

Za prikazovanje slik povzetih iz spletne kamere ni pomembno, da se slike prikazujejo točno v časovnih intervalih, zato v tem primeru uporabljamo pasivno upodabljanje.

Najprej smo implementirali zajem slike iz spletne kamere. Uporabili smo razred CvCapture, ki smo ga najprej inicializirali in nastavili višino in širino slike, ki jo bomo zajemali.

```
CvCapture fotograf = opencv_highgui.cvCreateCameraCapture(0);

try {
opencv_highgui.cvSetCaptureProperty(fotograf,
    opencv_highgui.CV_CAP_PROP_FRAME_HEIGHT, VISINA);
opencv_highgui.cvSetCaptureProperty(fotograf,
    opencv_highgui.CV_CAP_PROP_FRAME_WIDTH, SIRINA);
}
catch(Exception e) { }
```

Zdaj smo lahko že zajemali slike. To smo storili s pomočjo metode cvQueryFrame.

```
slika = opencv_highgui.cvQueryFrame(fotograf);
```

Nato smo morali zrcaliti zajeto sliko, saj želimo igralcu ustvariti občutek, kot da se gleda v ogledalo. Uporabili smo metodo cvFlip, ki kot prvi parameter potrebuje izvorno sliko, drugi parameter je ime novonastale slike, zadnji parameter pa je način preslikave. Obstajajo trije načini preslikave in sicer preko x-osi, y-osi in preko obeh hkrati. Uporabili smo preslikavo preko y-osi tako, da smo zadnjemu parametru dodelili vrednost večjo od 0.

```
cvFlip(slika, slika, 2);
```

Začeli smo z implentacijo zaznave obraza. Za hitro in kvalitetno zaznavo je bilo potrebno najprej predpripraviti sliko. Potrebno jo je bilo zmanjšati in tako izboljšati hitrost zaznave ter pretvoriti barvno sliko v črno belo in tako izboljšati kvaliteto zaznave. Za še boljšo kvaliteto zaznave sliki normaliziramo svetlost in povečamo kontrast.

```
/*Ustvarimo crno belo sliko*/
IplImage crnoBela = cvCreateImage(cvGetSize(slika), IPL_DEPTH_8U, 1);
/*Pretvorimo barvno sliko v crno belo in jo shranimo v crnoBela*/
cvCvtColor(slika, crnoBela, CV_BGR2GRAY);
/*Ustvarimo manjso sliko*/
IplImage majhna = IplImage.create(crnoBela.width()/POMANJSAVA,
    crnoBela.height()/POMANJSAVA, IPL_DEPTH_8U, 1);
/*Zmanjsamo crnoBela in jo shranimo v majhna*/
cvResize(crnoBela, majhna, CV_INTER_LINEAR);
/*Sliki majhna normaliziramo svetlost in izboljsamo kontrast*/
cvEqualizeHist(majhna, majhna);
```

Sedaj je slika pripravljena, da se na njej izvede zaznava obraza. Uporabili bomo metodo *cvHaarDetectObjects*, ki za uporabo potrebuje inicializiran klasifikator. Metoda vrne polje zaznanih objekov, vendar nas bo zanimal le največji, saj bo le-ta v večini primerov igralčev obraz. Z zastavico $CV\_HAAR\_DO\_ROUGH\_SEARCH$ bomo pohitrili detekcijo, saj nam je pomembnejša hitrost od kvalitete, ker bomo metodo poganjali velikokrat.

```
CvSeq obrazi = cvHaarDetectObjects(majhna, klasifikator, pomnilnik, 1.1, 1,
    CV_HAAR_DO_ROUGH_SEARCH | CV_HAAR_FIND_BIGGEST_OBJECT);
```

Po zaključku izvajanja metode *cvHaarDetectObjects* lahko ustvarimo nov CvRect objekt nad zaznanim obrazom. Razred CvRect je zelo podoben Javanskemu razredu

Rectangle. Sedaj lahko na igralčev obraz izrišemo strelski merek in okoli obraza nastavimo pravokotnik. Pri dodeljevanju koordinat pravokotnika in strelskega merka moramo upoštevati, da smo predhodno pomanjšali sliko. Pravilne koordinate dodelimo tako, da pomnožimo trenutne s faktorjem pomanjšave.

Sprememba smeri pošasti je implementirana ravno na podlagi koordinat pravokotnika. V vsakem koraku primerjamo nove koordinate pravokotnika s starimi in glede na spremembe na osi x lahko določimo spremembo smeri premika igralčevega obraza.

7 Testiranje

Faza testiranja je zadnja faza pred predajo razvitega programskega produkta končnemu uporabniku. V tej fazi ugotavljamo pravilnost delovanja produkta in izvajamo validacijo. Validacija je proces preverjanja skladnosti razvitega produkta z definiranimi funkcijskimi zahtevami. Morebitne odkrite napake in neskladnosti je seveda potrebno odpraviti.

Testiranje izvajajo preizkuševalci, ki so do produkta kritični in destruktivni, saj taka vrsta pristopa prinaša največjo možnost odkritja napak. Preizkuševalci so lahko razvijalci, ki so razvijali produkt, neodvisne osebe ali tudi končni uporabniki. Izkaže se, da neodvisne osebe in končni uporabniki odkrijejo več napak, saj ne poznajo produkta in ga tako ne testirajo na pričakovan način [5]. V našem primeru bo aplikacijo testiral njen razvijalec.

7.1 Načrt testiranja

Aplikacijo bomo testirali na način črne škatle, pri katerem nas zanima le njeno delovanje in funkcionalnost. Pri takem tipu testiranja nas ne zanima notranjost *oz.* struktura aplikacije [3]. Taka oblika testiranja je nepogrešljiva in tudi najpogostejša.

Testiranje se bo začelo z samim zagonom aplikacije. Ob uspešnem zagonu se bo prikazal glavni meni. Testirali bomo vsako komponentno aplikacije posebej, če bo to le mogoče. Začeli bomo s prvo in najenostavnejšo in sicer z glavnim menijem. Najprej bomo preverili, ali je ozadje menija pravilno prikazano. Nato bomo pregledali, ali so gumbi menija enakomerno in primerno razporejeni, pregledali bomo pravilnost njihovega izrisa in ujemanja pisave. Poleg grafičnih lastnosti gumbov moramo preveriti, ali delujejo pravilno njihove povezave do ostalih komponent aplikacije. Naslednja komponenta, ki jo bomo testirali, je lestvica. Ponovno moramo preveriti prikaz ozadja. Dodatno pozornost moramo posvetiti kontrastu, ki ga ustvarja ozadje v povezavi s tekstom, saj je ključnega pomena za dobro vidljivost prikaza rezultatov. Preveriti moramo še pravilnost izpisanih rezultatov in delovanja gumba »Nazaj«, ki nas povrne v glavni meni. Tretja komponenta, ki jo bomo testirali istočasno z delovanjem zvoka, so nastavitve. Prav tako kot pri ostalih komponentah, je potrebno preveriti izris, postavitev ter vidljivost grafičnih elementov in delovanje gumbov. Nato bomo začeli s testiranjem

nastavljanja glasnosti zvočnih efektov. Na grafičnem vmesniku mora biti označena le ena številka, ki je skladna z jakostjo zvoka. Označena mora biti z modrozeleno barvo. Ob vsakem kliku *oz.* izboru vrednosti mora biti slišna sprememba glasnosti zvoka, ki se predvaja v ozadju ampak le, če je zvok vključen. Testirati moramo tudi persistentnost nastavitve med različnimi zagoni aplikacije. Naslednja funkcionalnost nastavitev, ki bo testirana, je možnost vključitve in izključitve zvoka. Testirana bo na enak način kot prejšnja funkcionalnost. Dodatno moramo biti pozorni le na to, da bo pri ponovni vključitvi zvoka jakost ista kot pred zadnjo izključitvijo zvoka. Lastnosti grafičnega vmesnika zadnje funkcionalnosti nastavitev, ki omogoča izbor pošasti, se bodo testirale po enakem postopku kot predhodni dve. Dejansko funkcionalnost pa bo treba testirati s pomočjo grafičnega vmesnika same igre in tako preveriti, ali je glavni akter igre res izbrana pošast iz nastavitev. Predzadnja komponenta, ki jo bomo testirali, je zaznava obraza, ki je edini način krmiljenja pošasti. Najprej bomo preverili grafični vmesnik komponente. Preveriti moramo, ali se pojavi okno, v katerem se prikazujejo zajete slike, njihovo postavitev glede na okno, izris pravokotnika okoli in izris strelskega merka na sredini igralčevega obraza. Slike morajo biti zrcaljene tako, da dajejo občutek gledanja v ogledalu. Glede funkcionalnosti komponente moramo preveriti kvaliteto zaznave *oz.* pogostost neuspešne zaznave in samo hitrost. Občutljivost spremembe smeri premikanja pošasti glede na pozicijo igralčevega obraza bomo preverili skupaj s testiranjem same igre. Ponovno bomo začeli s testiranjem grafičnega vmesnika. Preverili bomo pravilnost izrisa, pozicijo in primernost slik za posamezne entitete igre. Prikazovanje grafičnega vmesnika mora biti tekoče. Igra mora pravilno obravnavati medsebojne trke entitet z določenim točkovanjem, odvzemom življenj, zaključkom igre in s predvidenimi zvočnimi efekti. Naključnost, hitrost in smeri padanja entitet morajo biti skladne z zahtevami. Ob zaključku igre se mora prikazati okence, v katero igralec vpiše svoje ime. Ob dovolj visokem rezultatu se mora ta zapisati na lestvico. Po vseh uspešno opravljenih testih in končni validaciji bo produkt pripravljen za predajo.

7.2 Testiranje glavnega menija

Testiranje glavnega menija se je izšlo skoraj v celoti uspešno, kar smo tudi pričakovali zaradi same kompleksnosti komponente. Popraviti smo morali le napis gumba, s katerim dostopamo do lestvice. Napis smo spremenili iz "Rezultati" v "Lestvica".

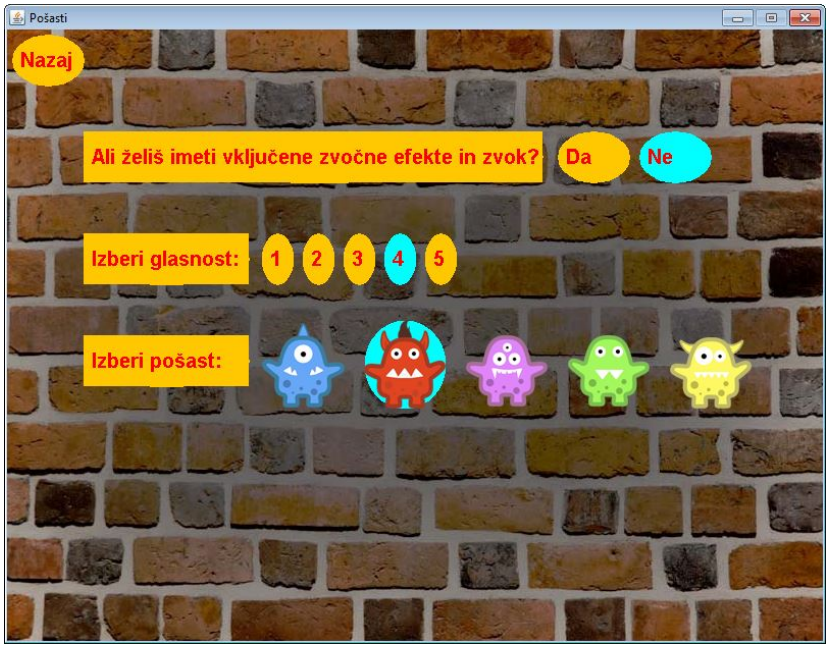
7.3 Testiranje lestvice

Tudi pri lestvici nismo imeli več kot eno napako. Pri testiranju smo opazili slabo čitljivost prve vrstice izpisa zaradi slabega kontrasta med pisavo rdeče barve in ozadjem

komponente. Rdečo barvo pisave smo zamenjali z modrozeleno barvo in tako dosegli veliko boljšo čitljivost.

7.4 Testiranje nastavitev

Pri testiranju nastavitev se je izkazalo že nekaj več napak kot pri prejšnjih komponentah. Že pri testiranju grafičnih elementov nas je zmotila postavitev in izris pošasti. Slike pošasti so bile prevelike in neenakomerno porazdeljene po horizontalni liniji. Prikazne slike smo proporcionalno zmanjšali in uredili njihovo postavitev. Nov izgled nastavitev lahko vidimo na sliki 12.



Slika 12: Grafični vmesnik nastavitev igre

Druga napako smo odkrili pri testiranju nastavitev glasnosti. Opazili smo, da je razlika v glasnosti med glasbo ozadja in zvočnimi efekti premajhna. Kot posledica premajhne razlike so bili zvočni efekti gumbov in efekti ostalih dogodkov v igri skoraj neslišni. Z ureditvijo razmerja smo uspešno končali testiranje nastavitev.

7.5 Testiranje zaznave obraza

Zaradi kompleksnosti same komponente je pričakovano, da smo pri njenem testiranju odkrili največ napak. Najprej smo se lotili popravljanja grafičnega vmesnika, saj smo opazili slabo postavitev prikaza zajetih slik in odvečne obrobe okoli le-teh. S

pomanjšavo JPanela in uporabo omejitev pri uporabi funkcije *add* razreda *Container* smo odpravili napako. Nato smo pri testiranju funkcionalnosti opazili, da je zaznava delovala prepočasi. Težavo smo odpravili z optimizacijo priprave slike nad katero poganjamo algoritem zaznave obraza in sicer tako, da smo še povečali faktor pomanjšave. Še zadnja težava, ki smo jo odkrili, je bila previsoka nesenzibilnost premika igralčevega obraza za spremembo smeri premika pošasti. Težavo smo rešili zelo hitro s spremembo vrednosti parametra. Testiranje komponente smo uspešno zaključili s pomočjo igre, s katero smo preverili krmilnik igre *oz.* povezanost teh dveh komponent.

7.6 Testiranje igre

Testiranje igre se je začelo že na koncu testiranja zaznave obraza, ko smo testirali krmilnik igre. Nato smo ob zagonu igre testirali sinhronizacijo inicializacije zaznave obraza z začetkom igre. Opazili smo, da se je igra začela pred inicializacijo, kar pomeni da igralec ni mogel premikati pošasti prvih nekaj sekund poteka igre. Opazili smo, da smo pozabili nastaviti čakanje na klic metode *kameraDeluje*, ki javi igri, da lahko začne igro. Nadaljevali smo s testiranjem grafičnega vmesnika in pri tem nismo odkrili napak. Pri preverjanju delovanja trkov entitet in dogodkov, ki sledijo trkom, smo ugotovili le formalno in nemotečo napako. Ob trku bombe s pošastjo so se igralcu odvzela vsa življenja in tako zaključila igra. V specifikaciji zahtev pa je bil dogodek trka definiran tako, da se igra enostavno zaključi, ne da bi se pri tem igralcu odvzela vsa življenja. Napaka je nerelevantna, vendar smo jo vseeno odpravili, ker nam je to čas dopuščal. Na sliki 13 lahko vidimo primer trka. Poskušali smo tudi čim bolj objektivno testirati kvaliteto igranja in hitrost večanja težavnosti igre. Nazadnje smo preverili še prikaz in delovanje vpisnega okna, kjer igralec vpiše svoje ime za vstop na lestvico. Tudi zapis rezultatov v tekstovno datoteko je deloval brezhibno.

7.7 Validacija

Pred predajo produkta je potrebno preveriti, ali smo zadovoljili vse funkcijske zahteve definirane v fazi analize, katere je potrdil tudi naročnik pri pregledu funkcijske specifikacije. Implementirali smo arkadno računalniško igro "Pošasti" z uporabniško interakcijo temelječo na računalniškem vidu, ki ne vpliva negativno na držo igralca. Pri razvoju smo se držali definiranih smernic. Trudili smo se implementirati čim bolj enostavno igro s hitro razumljivimi pravili. Pri tem smo pazili, da je igra še vedno dovolj izzivalna in privlačna. Glavni namen igre poleg razvedrilne vrednosti je nudenje potrebnega razgibavanja ljudem, ki dolgotrajno sedijo za računalnikom. Menimo, da neobičajen in tekoč krmilnik ter zmerno kreativen grafični vmesnik veliko pripomoreta

Slika 13: Primer trka bombe s pošastjo

k privlačnosti same igre. Igra ponuja zadovoljivo število možnosti prilagajanja igre. Lokalna lestvica je morda še dodaten razlog za igranje. Po uspešnem pregledu skladnosti zahtev s funkcionalnostimi aplikacije menimo, da je aplikacija pripravljena za predajo v uporabo.

8 Zaključek

V zaključni nalogi smo z načrtovanjem in implementacijo aplikacije želeli predstaviti potek programskega procesa. Med samim načrtovanjem in izvedbo aplikacije smo se prepričali, kako pomembna je faza analize in definiranja zahtev. Obširna in podrobna specifikacija zahtev je ključnega pomena za tekoč in uspešen potek nadaljnih faz. Prav zaradi tega lahko sklepamo, da je analiza zahtev najpomembnejša faza programskega procesa, pri kateri igra pomembno vlogo naročnik in končni uporabnik. Le-ta najbolj pozna področje problema in potrebe, ki naj bi jih končni produkt reševal. Pomembno pa je, da pri projektu sodeluje strokovnjak, ki usklajuje naročnikove zahteve z realizacijo projekta in pri tem ločuje med potrebami in željami.

Končni produkt je arkadna računalniška igra z neobičajno igralčevo interakcijo, ki pušča odprte možnosti nadgradnje. Ena od možnih nadgradenj je ta, da zaznavo, temelječo na računalniškem vidu, ki trenutno zaznava le igralčev obraz, razširimo na zaznavo različnih delov telesa in s tem omogočimo več različnih načinov fizičnega razgibavanja ob igranju igre. Druga možna nadgradnja, ki pomeni zahtevnejši in obsežnejši projekt, je nadgradnja igre v celozaslonsko 3D igro, ki bi bila kompleksnejša in primernejša za vstop na tržišče.

Projekt zaključujemo z željo, da bi bil produkt uspešen in bi kot tak služil naročniku v koristno zabavo in sprostitev.

9 Literatura in viri

[1] Bradski G. in Kaehler A., *Learning OpenCV Computer Vision with the OpenCV Library*, 2008.

[2] Fowler M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition*, 2003.

[3] Kaner C. in Falk J. in Nguyen H., *Testing Computer Software, 2nd Edition*, 1999.

[4] Kramer Wolfgang, What Makes a Game Good?, Julij 2000, http://www.thegamesjournal.com/articles/WhatMakesaGame.shtml

[5] McConnell S., *Code Complete: A Practical Handbook of Software Construction, Second Edition*, 2004.

[6] Ogden CL in Carroll MD in Kit BK in Flegal KM, Prevalence of childhood and adult obesity in the united states, 2011-2012. V *JAMA*, 2014, 806-814.

[7] Playing computer games increases obesity risk in teens by making them hungry, Mail Online 2011, http://www.dailymail.co.uk/health/article-1389096/Playing-games-encourages-obesity-teens-making-hungry.html

[8] Pressman Roger S., *Software Engineering: A Practitioner's Approach, Sixth Edition*, 2005.

[9] Wiegers K. in Beatty J., *Software Requirements (3rd Edition) (Developer Best Practices)*, 2013.

[10] World Video Game Market: Eight key trends to watch in 2014, GameSummit, December 2013, http://gamesummit.pro/world-video-game-market-eight-key-trends-to-watch-in-2014/

[11] Wright Tim, Java Games: Active Rendering, September 2007, http://www.gamedev.net/page/resources/_/technical/general-programming/java-games-active-rendering-r2418

Priloge

Izvorna koda

Priložena je zgoščenka z vso izvorno kodo aplikacije.