Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Čelešnik

Dopolnjevanje podatkov play-by-play z analizo premikov igralcev

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Igor Kononenko

Kanal, avgust 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Opis Raziskave na področju športne analitike in modeliranja razvoja športnih dogodkov so se intenzivirale zaradi široke dostopnosti t. i. podatkov play-by-play, ki predstavljajo kronološki zapis ključnih dogodkov v poteku športnih tekem. Ekstrakcija vzorcev iz podatkov play-by-play omogoča izdelavo modelov za razvoj športne tekme. Podatke play-by-play večinoma zbirajo ročno in so v obliki stavkov v naravnem jeziku, kar pomeni, da so možna nekonsistentnost oblike zapisov, tipkarske napake in nelogična zaporedja dogodkov. V zadnjih nekaj letih pa so pri košarki na voljo tudi kronološki podatki o položajih igralcev in žoge (koordinate) na igrišču, dobljeni iz videoposnetkov tekme. Cilj diplomske naloge je izdelava sistema, ki bo omogočil popravljanje in dopolnjevanje podatkov play-by-play z analizo premikanja igralcev na igrišču med tekmo. Pri tem naj se podatki play-by-play dopolnijo z akcijami podaje, z imeni igralcev, ki so dano akcijo izvedli, ter za mete na koš tudi podatek o tem, ali je bil met (ne)oviran.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Miha Čelešnik**,
z vpisno številko **63040200**,

sem avtor diplomskega dela z naslovom:

Dopolnjevanje podatkov play-by-play z analizo premikov igralcev

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Igorja Kononenka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Kanalu, 25. 8. 2016 Podpis avtorja/-ice:

Zahvaljujem se mentorju prof. dr. Igorju Kononenku in njegovemu asistentu mag. Petru Vračarju za strokovno in hitro odzivanje ter svoji družini za (takšno in drugačno) podporo. Posebna zahvala gre moji ženi za spodbudo, dobre živce in potrpežljivost.

Svoji dragi ženi.

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

kratica	angleško	slovensko
JSON	JavaScript Object Notation	JavaScript objektna notacija
NBA	National Basketball Association	Nacionalna košarkarska zveza
JFC	Java Foundation Classes	Temeljni javanski razredi
API	Application Programming Interface	Aplikacijski programski vmesnik
JVM	Java Virtual Machine	Navidezni javanski stroj
IDE	Integrated Development Environment	Integrirano razvojno okolje
HTML	Hyper Text Markup Language	Jezik za označevanje nadbesedila
GD	Google Drive	Googlova shramba v oblaku
GUI	Graphic User Interface	Grafični uporabniški vmesnik
AWT	Abstract Window Toolkit	Orodje za abstraktna okna

Povzetek

Naslov: Dopolnjevanje podatkov play-by-play z analizo premikov igralcev

V diplomski nalogi je predstavljen razvoj aplikacije za avtomatsko obdelavo podatkov SportVU-ja s košarkarskih tekem NBA. Po kratkem uvodu v drugem poglavju kot vir podatkov predstavimo snemalni sistem SportVU ter tehnologije, knjižnice in orodja, ki so bili uporabljeni pri razvoju. Tretje poglavje prinaša natančen opis vhodnih podatkov. V nadaljevanju je opisan javanski podatkovni model, način pristopa k reševanju problema in glavni algoritem, ki je jedro aplikacije. V tem poglavju predstavimo tudi testiranje in rezultate pravilnosti algoritma ter opišemo nekatere funkcionalnosti, s katerimi smo eksperimentirali, a niso nujno vključene v aplikacijo. Peto poglavje vsebuje opis rezultatov kot izhod algoritma ter navaja težave, s katerimi smo se soočali. Zaključek predstavi ugotovitve in možnosti nadaljnje uporabe naših ugotovitev.

Ključne besede: NBA, SportVU, Play-by-play, Avtomatska obdelava podatkov.

Abstract

Title: Filling in Play-by-play data with player movement analysis

This thesis presents the development of an application for automatic processing of SportVU data from basketball games. A short introduction is followed by a chapter, in which the recording system SportVU as the data source is presented next to the technologies, libraries and tools used in the development. The third chapter gives a detailed description of the input data. In continuation, a Java data model, means of approach to the salvation of the problem and the main algorithm, which is the core of application, are presented. This chapter also shows testing and results of algorithm regularity as well as describes some functionalities, which were part of the experiments, but are not necessarily included into the application. The fifth chapter contains the description of results as an output of the algorithm and lists the problems that arose during the experiments. The concluding part of the thesis presents the findings and possibilities for further use of the latter.

Keywords: NBA, SportVU, Play-by-play, Automatic data processing.

Poglavje 1

Uvod

Spremljanje športnih tekem je zelo priljubljeno. Med moštvenimi športi izstopa košarka, še posebej severnoameriška profesionalna liga NBA, zato je tam veliko zanimanje za statistične podatke in analizo igre. Do nedavnega je bilo zajemanje in obdelovanje teh podatkov ročno, leta 2012 pa uvedejo sistem kamer STAT SportVU, ki je podrobneje opisan v nadaljevanju. Cilj pričujoče diplomske naloge je izdelava aplikacije za avtomatsko obdelavo podatkov o položaju igralcev in žoge, tako da bo prepoznala športni prvini: podajo in met na koš. Aplikacijo smo razvijali z uporabo vhodnih podatkov (glej poglavje 3) tekme med moštvoma Miami Heat in Washington Wizards, ki je potekala 3. 1. 2016 v Washingtonu. V diplomski nalogi smo predstavili snemalni sistem, obliko podatkov, na kratko smo opisali uporabljene tehnologije, knjižnice in orodja. V nadaljevanju smo orisali način pristopa, proces razvoja aplikacije, težave ter samo aplikacijo. Našo rešitev smo opisali s stališča točnosti, možnosti izboljšave in nadgradnje.

Namen diplomske naloge je prispevati k avtomatizaciji obdelave podatkov SportVU. Aplikacija naj bi zmogla iz obdelanih podatkov izluščiti vzorce, ki predstavljajo podaje in mete na koš, ter te podatke dodati k obstoječim play-by-play podatkom. Izhodni podatki so vir za nadaljnjo analizo.

Poglavje 2

Pregled tehnologij in orodij

2.1 Snemalni sistem

Podjetje STAT SportVU so leta 2005 ustanovili izraelski znanstveniki z namenom slediti raketnim izstrelkom preko napredne optične prepoznave. V nevojaške namene so ga sprva uporabili za sledenje dogodkov na nogometnih tekmah v Izraelu, sedaj pa se sistem uporablja v NBA. SportVU je sistem kamer, pritrjenih na ostrešje dvorane, ki zajema podatke 25-krat v sekundi in sledi žogi in vsem igralcem na igrišču. Daje podrobne realnočasovne podatke o položaju igralcev in žoge ter nudi izhodišče za nadaljnjo analizo. Postavitev igralcev je predstavljena s koordinatami x in y, žoge pa s koordinatami x, y in z. S podatki STAT lahko oblikuje performančno metriko za igralce in moštva [7]. Evidenco igralcev sistem vodi s pomočjo optične prepoznave številk na dresih [9].

2.2 Uporabljene tehnologije

2.2.1 Java

Java je splošnonamenski, visoko nivojski, objektno usmerjen programski jezik, ki je v današnjem času zelo priljubljen. Srečamo ga na vsakem koraku

– nahaja se na mobilnih telefonih (operacijski sistem Android), samostojnih namiznih aplikacijah (JavaSE), spletnih aplikacijah (JavaEE) ipd. Velik del sintakse jezika izhaja iz C/C++, vendar ima manj nizkonivojskih zmožnosti. Prvo verzijo je razvil James Gosling in je bila objavljena leta 1995 v podjetju Sun Microsystems. Velika prednost Jave je prenosljivost kode. Izvorna koda se namreč ne prevede neposredno v binarno kodo, temveč v vmesno kodo, ki se izvaja na javanskem navideznem stroju (JVM). V ta namen je potrebno razviti JVM za vsak operacijski sistem posebej. Dobra lastnost tega je, da so aplikacije napisane le enkrat in se lahko nato v veliki večini primerov izvajajo na katerem koli JVM-ju. Prav potreba po JVM-ju in slaba optimizacija sta botrovali slabemu delovanju zgodnjih verzij. Java 8 je trenutno najsodobnejša verzija Jave, uradni lastnik licence pa je podjetje Oracle Corporation [4].

2.2.2 Octave

Octave je program, ki uporablja visokonivojski programski jezik, in je primarno namenjen numeričnim izračunom. Napisan je v jeziku C/C++. V pomoč je pri numeričnem reševanju linearnih in nelienarnih problemov. Zagotavlja obsežne grafične zmogljivosti za vizualizacijo in obdelavo podatkov. Običajno se uporablja preko interaktivnega vmesnika, lahko pa se uporablja za pisanje neinteraktivnih programov. Uporablja interpreter za izvajanje skriptnega jezika Octave, ki je precej podoben Matlabu, in je zato večina programov lahko prenosljivih. Za izdelavo naloge smo uporabljali grafične zmožnosti spletne različice (Octave Online) za izris koordinat [6].

2.2.3 JSON

JSON (JavaScript Object Notation) je podatkovni format odprtega standarda, ki v lažje berljivi obliki podaja informacije, sestavljene iz parov atribut–vrednost. Je najbolj pogost podatkovni format, ki ga uporabljajo za asinhrono komunikacijo brskalnik–server (AJAJ) in v veliki meri zamenjuje XML

(AJAX). JSON je neodvisen podatkovni format. Izhaja iz JavaScripta, a je koda za generiranje in razčlenjevanje podatkov JSON-formata danes na voljo v mnogo programskih jezikih. Datotečna končnica formata, v katerem so zapisani tudi vhodni podatki za našo aplikacijo, se glasi .json.

2.3 Uporabljene knjižnice

2.3.1 Apache Maven

Apache Maven [1] je orodje, ki poskrbi za avtomatizacijo izdelave projekta. Uporablja se za prevod javanske kode in za avtomatski zagon testov programskih enot ter meddrugim skrbi, da ne prihaja do cikličnih odvisnosti med moduli aplikacije. Z njim lahko izdelamo različico aplikacije in namestimo aplikacijo v razvojno okolje. Zgrajen je na arhitekturi, ki temelji na uporabi vtičnikov, kar ga naredi izredno razširljivega. Teoretično to pomeni, da lahko vsakdo napiše svoj vtičnik, ki ga potem uporabi v življenjskem ciklu razvoja programske opreme.

Maven poskrbi tudi za avtomatski prenos potrebnih knjižnic iz za to namenjenih strežnikov. Razvijalcem se tako ni potrebno ukvarjati z dodajanjem knjižnic na projekt, da bi se koda lahko ustrezno prevedla. Ta funkcija je bila uporabljena tudi pri razvoju naše aplikacije.

2.3.2 Java Swing

Java Swing je zbirka orodij za gradnike uporabniškega vmesnika. Je del JFC-ja API-ja za zagotavljanje grafičnega uporabniškega vmesnika za Java programe. Swing je bil razvit za zagotavljanje naprednejšega nabora komponent grafičnega vmesnika, kot ga ima predhodni AWT. Swing zagotavlja naraven izgled in občutek (angl. look and feel), ki posnema izgled in občutek več platform, npr. Mac, Windows ... Podpira tudi vstavljiv izgled in občutek, ki ni povezan z osnovno platformo. Ima zmogljivejše in bolj prilagodljive dele kot AWT, saj so v celoti napisani v Javi in so zato platformsko neodvisni.

Swing vse bolj zamenjuje JavaFx.

2.3.3 GSON

GSON, poznan tudi kot Google Gson, je odprtokodna javanska knjižnica za seriliziranje in deseriliziranje javanskih objektov v in iz JSON-formata. Prvotno je bila knjižnica razvita za notranje potrebe Googla, kasneje pa je bila objavljena pod pogoji licence Apache 2.0. Zadnja različica, GSON 2.7, je bila objavljena 14. junija 2016. V diplomski nalogi smo jo uporabili za branje in mapiranje vhodnih podatkov.

2.4 Razvojna orodja

2.4.1 Intellij IDEA

Intelij IDEA je Java integrirano razvojno orodje za razvoj programske opreme. Razvito je bilo s strani JetBrainsa (prej znanega kot IntelliJ) in je na voljo pod licenco Apache 2 Licensed community edition in kot lastniške komercialne izdaje. Obe verziji je mogoče uporabiti za komercialni razvoj. Med drugim ima vgrajen lastni sistem za vodenje verzij. V diplomski nalogi je bilo uporabljeno za razvoj aplikacije.

2.4.2 Google Drive

Google Drive - prej znan kot Google Docs - je Googlov servis za hranjenje in sinhronizacijo datotek v oblaku. Uporabnikom omogoča ustvarjanje, shranjevanje, urejanje in delitev datotek, dokumentov, preglednic, predstavitev ... GD zajema Googlove Dokumente, Preglednice, Predstavitve, Obrazce, Risbe idr. in omogoča sodelovalno urejanje vsebine.

Poglavje 3

Struktura in opis podatkov

Datoteka JSON predstavlja strukturo, ki opisuje tekmo. Sestavljena je iz identifikacijske številke tekme, datuma tekme in iz seznama dogodkov (angl. event). Vsak dogodek ima svojo identifikacijsko številko, seznam podatkov o vseh igralcih (ta podatek je po našem mnenju odvečen, saj bi bilo dovolj, če bi vseboval le seznam igralcev na igrišču) ter seznam trenutkov (angl. moment). En trenutek vsebuje naslednje podatke: informacijo, kateri četrtini pripada, čas v milisekundah, igralni čas, čas napada, ter seznam položajev (angl. position) žoge in igralcev. Zajem trenutkov se vrši vsakih 40 ms, torej se čas v milisekundah pri dveh sosednjih trenutkih razlikuje za 40 enot.

Seznam položajev vsebuje podatke o koordinatah žoge in igralcev za dani trenutek. En položaj vsebuje id-moštva, id-igralca, koordinate x, y in z. Ker je uporabljena ista struktura za žogo in igralce, sta polji id-moštva in id-igralca, če gre za žogo, prazni. Če pa gre za igralca, je prazno polje koordinate z (višina). Dogodki se lahko kronološko prekrivajo, na primer: zadnjih nekaj trenutkov v dogodku skoka za žogo se ponovi na začetku dogodka napada. Enota koordinatnega sistema je en čevelj – večina igre se tako odvija znotraj okvirja dimenzij 94 x 50 enot, kar so dimenzije igrišča v čevljih po NBA pravilih. Koordinatni sistem igrišča ima izhodišče v levem zgornjem kotu, obroč levega koša ima koordinate (5.5, 25), obroč desnega pa (89.5, 25).

Podrobna struktura podatkov:

- gameid (šifra tekme)
- gamedate (datum tekme)
- events (dogodki)
  - eventid (šifra dogodka)
  - home (podatki o domači ekipi)
    - players (seznam igralcev)
      - playerid (šifra igralca)
      - firstname (ime)
      - lastname (priimek)
      - jersey (številka dresa)
      - position (igralno mesto)
    - teamid (šifra moštva)
    - name (ime moštva)
    - abbreviation (okrajšava imena ekipe)
  - visitor (podatki o gostujoči ekipi) – se ponovi kot pri prejšnji alineji
  - moments (seznam trenutkov)
    - quarter (četrtina)
    - time (čas v milisekundah)
    - gametime (igralni čas četrtine)
    - shottime (čas napada)
    - positions (seznam pozicij žoge in igralcev (1+10 elementov seznama))
      - teamid (šifra moštva)
      - playerid (šifra igralca)

- x
- y
- z (višina žoge)

Poglavje 4

Razvoj algoritma

4.1 Priprava podatkovnega modela

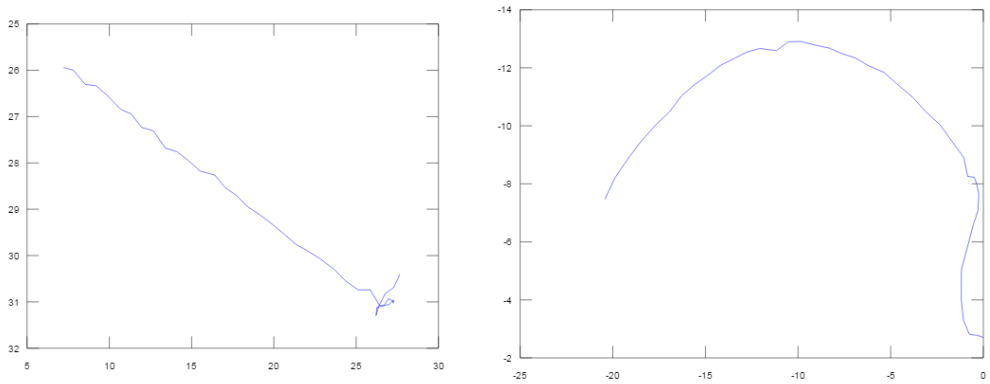
Mapiranje podatkov v javanske razrede smo izvedli na skoraj enak način, kakor so bili predstavljeni v datoteki JSON, z dvema izjemama: prva je povezava nazaj iz položaja na trenutek, ker je tako mogoč dostop do vseh položajev znotraj trenutka, druga pa je sprememba pri nalaganju podatkov o igralcih. Prvotno so se namreč vsi podatki o igralcih brez sprememb nalagali za vsak dogodek, to pomeni tudi več kot štiristokrat, po naši spremembi pa se naložijo le enkrat v razpršeno tabelo. Ključ je id-igralca, ostali podatki pa so vrednost.

Ob zagonu aplikacija podatkovno strukturo prebrano iz datoteke JSON s pomožnimi funkcijami preoblikuje v zaporeden seznam trenutkov. S prehodom čez vse trenutke v vseh dogodkih izloči podvojene zapise glede na časovni žig. Tak seznam predstavlja vhodne podatke za glavni algoritem.

4.2 Vizualizacija

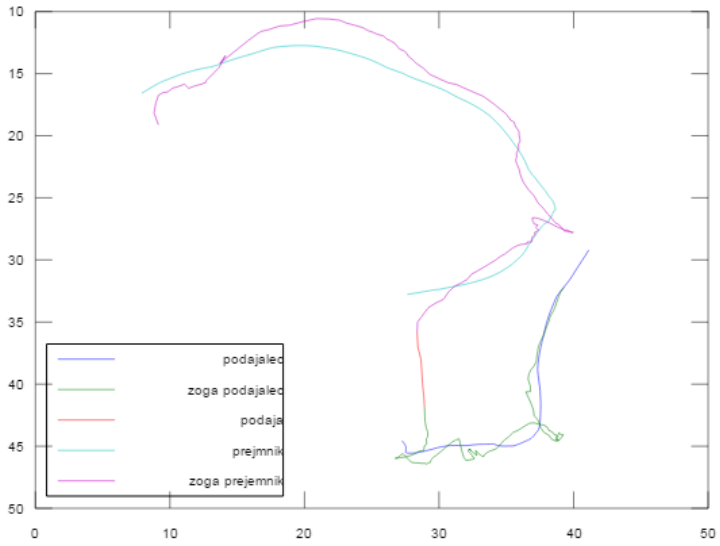
Po končanem mapiranju smo pripravili vizualizacijo. Najprej smo s pomočjo knjižnice Java Swing naredili animirano vizualizacijo, ki pa za naše delo ni imela večje dodane vrednosti. Zato smo se odločili za izrisovanje koordinat

objektov v Octavu, kjer smo lahko z veliko večjo natančnostjo spremljali pravilnost delovanja algoritma.



(a) Primer meta na koš – gledano z vrha.(b) Primer meta na koš – gledano s strani.

Slika 4.1: Primer vizualizacije meta na koš.



Slika 4.2: Primer, kjer algoritem zazna podajo (gledano s tlorisa).

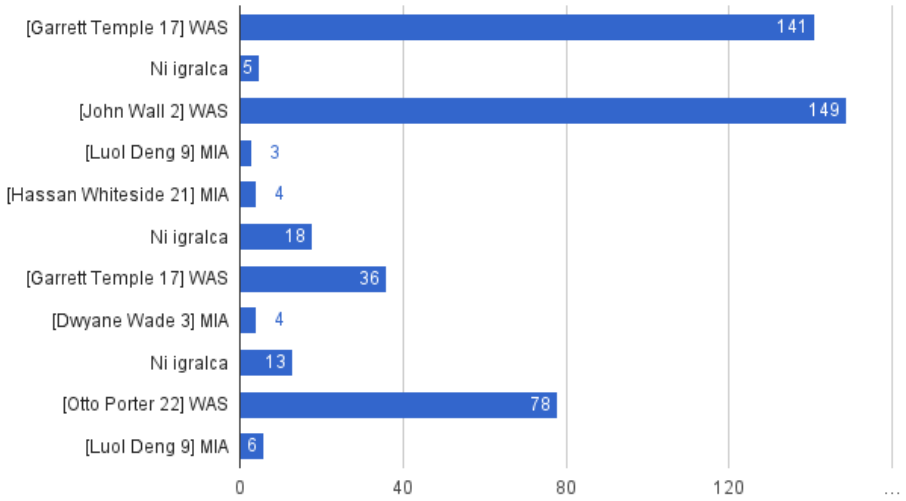
4.3 Opis glavnega algoritma

Delovanje glavnega algoritma v splošnem razdelimo na tri dele: v prvem delu naredi kronološko urejen seznam možnih krajišč podaj (krajišči podaj

sta točki, kjer žoga zapusti okolico podajalca in kjer žoga pride v okolico prejemnika), drugi del ugotovi, katera so dejanska krajišča, tretji del pa preveri, če so med njimi povezave. Algoritem za obdelavo podatkov ene tekme v povprečju potrebuje 32,5 sekunde.

4.3.1 Prvi del

Seznam možnih krajišč podaj naredimo tako, da ob prehodu skozi seznam trenutkov naredimo seznam seznamov pozicij igralcev, ki so najbližji žogi. Vsi elementi enega podseznama imajo id-igralca isti ali pa so brez igralca. Dva sosednja podseznama imata vedno različen id-igralca. S tako združenimi pozicijami igralcev dobimo podsezname, kjer konec enega in začetek naslednjega podseznama predstavljata možni krajišči podaje. Na sliki spodaj (4.3) je grafičen prikaz rezultata prvega dela algoritma, kjer so na navpični osi od zgoraj navzdol zaporedno nanizani igralci, na vodoravni pa število trenutkov, ko je njim žoga najbližje. Če trenutke pomnožimo z vrednostjo 40, dobimo čas trajanja v milisekundah.



Slika 4.3: Primer grupiranja, ko posest žoge še ni označena.

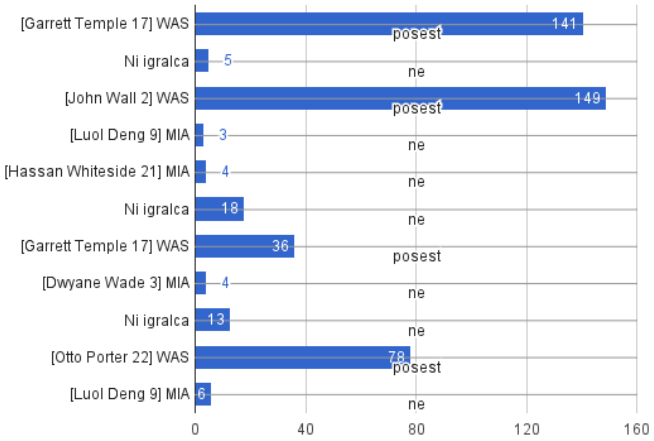
Psevdo koda:

```
    seznam momentov sm
    pripravi prazen seznam seznamov ss
    pripravi prazen podseznam ps
    najbližjaPozicija <- funkcija najbližjiIgralec(sm[0])
    na začetek ps dodamo najbližjaPozicija
    na začetek ss dodamo ps

    zanka skozi seznam momentov i=0...dolzina sm
        zadnjaPozicija iz ss
        najbližjaPozicija <- funkcijapoiščiŽogi
NajbližjegaIgralca(sm[i])
        če zadnjaPozicija.idIgralca != najbližja
Pozicija.idIgralca
            naredimo nov ps
            ps dodamo na konec ss
        iz ss poiščemo zadnji podseznam na konec dodamo
najbližjaPozicija
    konec zanke
    vrne ss
```

4.3.2 Drugi del

Na vhod drugega dela algoritma dobimo seznam seznamov. Zdaj je potrebno ugotoviti, v katerem podseznamu ima igralec žogo v posesti in v katerem ne. Tako lahko sestavimo seznam, ki zdaj vsebuje samo podsezname, v katerih je bila ugotovljena posest žoge. Na sliki 4.4 so prikazani rezultati prvega in drugega dela, ko je algoritem zaznal posest žoge.



Slika 4.4: Primer grupiranja, kjer je označena posest žoge.

Posest ugotavljamo tako, da v zanki preverimo vsak podseznam položajev, ali vsebuje naslednje lastnosti:

Pot žoge v koordinatah x-y Izhajamo iz predpostavke, da je žoga v posesti določenega igralca takrat, ko je dalj časa blizu njega. To pomeni, da je slika poti žoge v primeru posesti kompleksna krivulja. Če pa je krivulja ravna, lahko sklepamo, da posesti ni, vendar je potrebno še preveriti koordinate čas–višina. Preverjanje, ali je pot ravna, naredimo z aproksimacijo po metodi linearne regresije. Dopustili smo 5-odstotno odstopanje. V večini primerov algoritem pravilno razloči dano krivuljo.

Parabolično pot žoge v koordinatah čas–z (višina) Podobno kakor za koordinate x–y tudi za koordinate čas–višina dejanske vrednosti primerjamo aproksimirano funkcijo. Razlika je v tem, da je tu zaradi fizikalnih zakonitosti aproksimirana funkcija drugega reda dobljena z metodo polinomske regresije. Dopustili smo 5,5-odstotno odstopanje.

Naknadno smo ugotovili, da bi bilo potrebno upoštevati poseben primer, ko pri podaji pride do talnega odboja.

Pot žoge gre čez igralca Preverimo, ali je zadostno število elementov podseznama nad določeno vrednostjo koordinate z (višina). Ker podatek

o višini igralcev ne obstaja, preverjamo zato pozicije z vnaprej izbrano višino 8,5 čevlja.

Žoga v bližini koša med zaznavanjem posesti V okolici koša je potrebno posebno preverjanje zaradi odbojev, ki močno vplivajo na zanesljivost algoritma. Zato je potrebno vpeljati preverjanje žoge v bližini koša, kot je opisano v razdelku 4.5.1.

Psevdo koda:

```
funkcija imaVpliv podseznamPozicij
    aFunk - izračunajAproksimacijo podseznamPozicij
    aNapaka - izračunajNapako podseznamPozicij aFunk

    če aNapaka večja Tolerance
        če je v bližini koša
            če je min višina žoge na intervalu < 9
                vrni true
            drugače
                vrni false
        drugače
            vrni true
    drugače
        vrni false
konec funkcije

zanka skozi glavniSeznam
    če trenutniPodseznamPozicij nima igralca
        preskočimo celoten blok
    če trenutniPodseznamPozicij imaVpliv na žogo
        dodamo v seznam posesti
konec zanke
```

4.3.3 Tretji del

V naslednjem, tretjem delu obdelave poiščemo podaje in mete na koš. Ker se podaja razlikuje od meta na koš, gre algoritem skozi seznam dvakrat, enkrat za vse mete in enkrat za vse podaje.

Najprej se izvede prehod za mete, in sicer zato ker ob zapisu rezultata v kronološko urejen seznam avtomatsko dobimo razmejitev med možno spremembo posesti žoge (npr. dosežen koš). Ko algoritem najde trenutek, ko je žoga v okolici koša, vzvratno pregleda seznam, dokler ne najde najbližjega igralca, ki se ujema z igralcem iz seznama posesti. Tako dobi trenutek, ga opremi z informacijo o metu in ga vnese v seznam, urejen po časovnem žigu. Slabost takega pristopa je, da lahko v gneči zgreši, lahko se tudi zgodi, da prejšnji del obdelave naredi napako, ki se tako prenese naprej.

Naslednji prehod paroma pregleda sezname posesti tako, da preveri pot žoge od zadnjega elementa prvega seznama do prvega elementa drugega seznama. Najdene pare trenutkov opremi z informacijo o podaji in vnese v urejen seznam.

Psevdo koda:

```
nov seznamPodajMetov
nov seznamMet // pozicije od koša do igralca
zanka skozi seznamVsehPozicij od i=0 naprej
    p <- seznamVsehPozicij[i]
    če p blizu koša
        zanka skozi seznamVsehPozicij od i nazaj
            če p ne vsebuje igralca s posestvijo
                dodamo p na začetek v seznamMet
            konec če
        konec zanke
    konec če
konec zanke

seznamMet dodaj urejeno v seznamPodajMetov kot met
```

```
zanka skozi seznamPosesti od i=0 naprej
    sezPodajalec <- seznamPosesti[i]
    sezPrejemnik <- seznamPosesti[i+1]
    če ((sezPodajalec[zadnji] in sezPrejemnik[prvi])==daljica)||
    (sezPodajalec[zadnji] in sezPrejemnik[prvi])==minRazdalja
        nov sezPodaja(sezPodajalec[zadnji], sezPrejemnik[prvi])
        sezPodaja dodaj urejeno v seznamPodajMetov kot podaja
    konec če
konec zanke
```

4.4 Opis in rezultati testiranja

Testiranje smo naredili s primerjavo rezultatov algoritma s posnetki štirih tekem: Miami Heat proti Washington Wizards (3. 1. 2016), Cleveland Cavaliers proti Washington Wizards (6. 1. 2016), Miami Heat proti Phoenix Suns (8. 1. 2016) in Los Angeles Lakers proti Golden State Warriors (14. 1. 2016). Za potrebe testiranja smo priredili izpis rezultatov.

Kot smo že zgoraj omenili, smo si na začetku pomagali z vizualizacijo podatkov s pomočjo javanske knjižnice Swing. Rezultati so bili na trenutke zelo nenavadni – najprej smo menili, da je tako zaradi naše napake. V ta namen smo pripravili izpis vrednosti objektov v Javi za izris v Octavu. Na ta način smo po dolgem raziskovanju ugotovili, da so za težavo, ki se občasno pojavlja, krivi podatki (glej razdelek Težave 5.3).

Ob ogledu smo bili pozorni na podaje, mete na koš in odvzeme žoge, ki so definirani in izpisani na naslednji način:

Definicija podaje je: žoga kontrolirano preide od igralca do soigralca, torej je nekaj časa pri podajalcu, nato nekaj časa pri prejemniku. Če je žoga pri igralcu le za kratek čas, ali če igralec dobi odbito žogo, to ni podaja. V rezultatih je izpisana kot par igralcev, med katerima je oznaka "podaja"in čas podaje v trenutku, ko je prejemnik žogi najbližji

(čas sprejema žoge).

Met na koš prav tako predstavlja par igralcev, metalec in igralec, ki je pod košem žogi najbližji; med njima je oznaka "met na koš". Vsak tak par se zapiše v svojo vrstico, vmes je vrstica prazna.

Odvzem žoge pomeni , da igralec nasprotnega moštva za dalj časa dobi žogo v bližino (kar se lahko ugotovi na danem intervalu). Ta funkcija algoritma deluje nezanesljivo.

Tako zapisane rezultate algoritma smo zatem primerjali s posnetki in na koncu vsake vrstice ročno dodali oznako "pravilna"ali "napačna". V primeru napačne je sledil še kratek opis napake. Za primer, da bi se v danem času morala zabeležiti podaja ali met na koš in je algoritem ni zaznal, smo v prazen prostor med vrsticami zabeležili napako.

Poseben primer so prekinitve, takrat namreč žoga ne sledi pravilom, ki so med igro: igralci si lahko žogo podajajo oziroma predajajo naključno, žogo lahko prevzamejo sodniki, katerih položaja ne poznamo. Najprej smo nameravali izpuščati intervale s prekinitvami, vendar se je izkazalo, da igralni čas ni vedno ustavljen točno, medtem pa se akcija lahko že zaključi in tako pride do napak pri zaznavanju. Namesto tega smo intervale s prekinitvami vseeno dodajali v seznam in jih izpuščali pri testiranju (glej tabele: 4.1, 4.2, 4.3 in 4.4).

Četrtine	Pravilno	Vsi	Odstotek
1	178	204	87
2	202	218	92
3	185	212	87
4	213	236	90
Skupaj	778	870	89

Tabela 4.1: Testiranje algoritma na tekmi Miami Heat proti Washington Wizards.

Četrtine	Pravilno	Vsi	Odstotek
1	195	228	85
2	176	210	83
3	169	209	80
4	141	165	85
Skupaj	681	812	83

Tabela 4.2: Testiranje algoritma na tekmi Cleveland Cavaliers proti Washington Wizards.

Četrtine	Pravilno	Vsi	Odstotek
1	157	190	82
2	144	144	84
3	165	190	86
4	164	217	75
Skupaj	630	768	82

Tabela 4.3: Testiranje algoritma na tekmi Miami Heat proti Phoenix Suns.

Četrtine	Pravilno	Vsi	Odstotek
1	178	190	82
2	144	144	84
3	165	190	86
4	164	217	75
Skupaj	651	793	82

Tabela 4.4: Testiranje algoritma na tekmi Los Angeles Lakers proti Golden State Warriors.

Skupen odstotek točnosti za vse štiri testirane primere je 84,48.

4.4.1 Napake pri testiranju

Napake pri testiranju lahko v grobem razdelimo v dve skupini: prva skupina so napake zaradi napačnega delovanja algoritma, druga skupina pa so napake zaradi napačnih podatkov.

Prva skupina napak se običajno ponavlja v določenih primerih, na primer: v gneči pod košem se lahko zgodi, da zazna napačno zaporedje podaj ali napačnega metalca na koš.

Druga skupina napak pa predstavlja napake v podatkih, zaradi katerih se prav tako prikažejo napačna zaporedja podaj, vendar niso ponovljiva in tudi sama po sebi niso napačna.

4.5 Eksperimentiranje

V tem podpoglavju so opisani pristopi, s katerimi smo eksperimentirali, vendar niso nujno vključeni v aplikacijo.

4.5.1 Zaznavanje bližine koša

Preverjanje, ali se žoga nahaja v bližini levega ali desnega koša, smo naredili s primerjavo položaja košev in trenutnega položaja žoge. Položaj košev je na koordinatah, ki se z manjšim odstopanjem ujemajo s pravilnikom NBA. Za koordinati x in y preverjamo območje z določeno toleranco, koordinato z preverjamo, ali je večja od višine obroča koša. Toleranci po osi x smo določili vrednost 1.8, po osi y pa 2.5, da se kompenzirajo "luknje"v osveževanju za primer odboja žoge od zunanjega roba obroča. Slaba stran tega pristopa je, da se skupaj z gnečo pod košem pogosto zgodi, da se izgubi podaja v koš ali popravek meta na koš.

4.5.2 Razčlenitev dela parabole žoge

Pri tej funkciji gre za razčlenjevanje dela parabole leta žoge na koš. Vzet je časovni interval posesti žoge žogi najbližjega igralca, ko je žoga v okolici koša. Razčleniti pomeni razdeliti interval na del pred okolico koša, na okolico koša in del po okolici koša. Okolico koša predstavlja območje znotraj prej določene tolerance po oseh x, y in z. Tak interval lahko ali samo raste ali

samo pada, če odmislimo šum v podatkih. Kot rezultat dobimo podatek, ali je igralec kandidat za metalca ali lovilca žoge.

4.5.3 Aproksimiranje poti žoge

Uporabili smo linearno aproksimacijo po metodi linearne regresije (glej poglavje [5]) za koordinati x–y v kombinaciji s polinomsko aproksimacijo reda dva za koordinati čas–z za zaznavanje hitrih podaj, kjer je žoga v bližini igralca manj kot sekundo. S poskušanjem smo ugotovili, da če je odstopanje do 6-odstotno, lahko dovolj zanesljivo rečemo, da je igralec imel žogo v posesti.

4.5.4 Ugotavljanje najmanjše višine žoge

Na časovnem intervalu posesti žoge smo poiskali najnižjo vrednost koordinate z in jo primerjali z določeno mejno vrednostjo. Zaradi pomanjkljivih podatkov o višini igralcev smo mejo postavili na 8,5 čevlja. Namen funkcije je ugotoviti, ali igralec žogo lahko doseže ali pa leti previsoko čezenj. Funkcija je v aplikacijo vključena, a ni zanesljiva, saj igralec lahko skoči in z iztegnjeno roko doseže precej več kot 8,5 čevlja.

4.5.5 Zaznavanje vodenja

Za vsak element intervala smo preverjali, kolikokrat je žoga na tleh (koordinata z v okolici ničle). Velikokrat smo naleteli na primer, ko je bilo zelo težko razbrati, ali je igralec žogo vodil, ali jo je samo vihtel v bližini tal v izogib odvzetju. Zaradi takih primerov smo to funkcijo opustili.

4.5.6 Zaznavanje hitrosti in pospeška žoge

Želeli smo ugotoviti, ali ima sprememba hitrosti vpliv na zaznavanje podaj. Ker smo ničkolikokrat dobili nenavadne rezultate, na primer: žoga je med prostim letom spreminjala smer (po domače povedano: letela je "cik-cak--

glej razdelek Težave 5.3, posebej sliko 5.2), in tako hitrosti ter pospeškov ni bilo mogoče izračunati, smo to opustili.

Poglavje 5

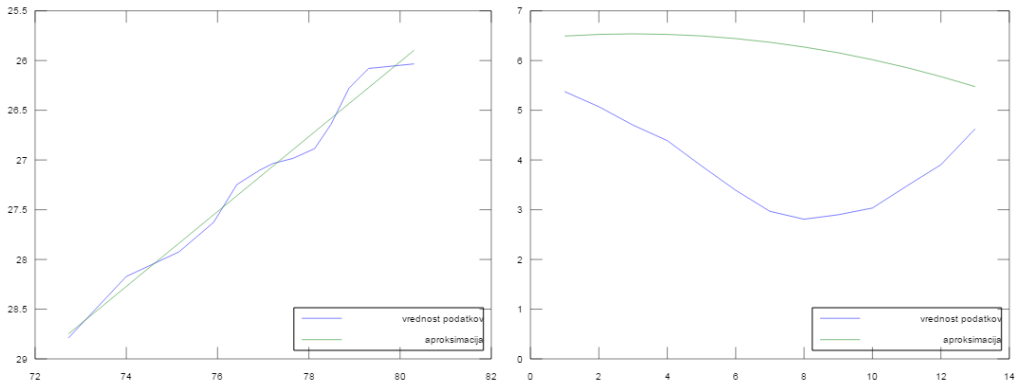
Rezultati

5.1 Ocene uspešnosti

Aplikacija je uspešna, če algoritem pravilno prepozna podaje in mete na koš.

Pri podajah je algoritem uspešen, če zazna podajo, pri kateri gre za odstopanje znotraj dane tolerance. Povedano drugače: algoritem je uspešnejši, če prepozna podajo, pri kateri je čas, ko je žoga blizu podajalca, čim krajši, sprememba smeri od sprejema do predaje pa je čim manjša. V primeru protinapada igralec na primer dobi žogo in jo v manj kot sekundi poda naprej, pri tem ji spremeni smer za manj kot 10 stopinj. Na sliki 5.1 je primer hitre podaje, kjer igralec v protinapadu žogo v približno 500 milisekundah ujame in poda v konico napada. Algoritem pri preverjanju z vrha ni bil uspešen (slika 5.1a), saj je bila napaka aproksimacije v tem primeru majhna, pri preverjanju s strani pa je prišlo do večje napake in s tem do pravilne odločitve algoritma, da ima igralec vpliv na žogo (slika 5.1b).

Pri metih na koš je algoritem uspešen, ko prepozna metalca na koš.



(a) Primer hitre podaje z vrha. (b) Primer hitre podaje s strani.

Slika 5.1: Primer hitre podaje.

5.2 Zapis rezultatov

Rezultat algoritma je seznam javanskih objektov. Tak objekt predstavlja en met ali eno podajo in vsebuje opis dogodka ter par pozicij (glej razdelek 4.3.3). V primeru podaje je to par podajalec–prejemnik, v primeru meta pa metalec in najbližji igralec pod košem. Na ta način dostopamo do nadrejenega objekta trenutek, kjer se nahajajo podatki o trenutnem času. Dobimo natančen čas metov in podaj, lahko tudi na primer izmerimo razdaljo med dvema igralcema, preverjamo pokritost igralcev, ocenimo težavnost meta itd.

Tak seznam, zapisan v tekstovno datoteko, predstavlja izhodišče za nadaljnjo obdelavo.

5.3 Težave

V nadaljevanju so opisane težave, ki so najbolj ovirale razvoj aplikacije:

Ni podatkov o višini igralcev V podatkih je polje o višini igralcev prazno, zaradi tega lahko samo domnevamo, kaj točno počne (stoji na tleh, je med skokom v zraku ...). Zlasti med igro pod košem je zelo pomembno razločiti, v kakšnem položaju je igralec.

Nenatančni podatki o višini žoge pri vodenju Med vodenjem žoge v podatkih ni pravilno zabeležen najnižji položaj žoge. Običajno to ni težava, vendar se lahko zgodi, da igralec žogo v izogib odvzetju premika v bližini tal, kar lahko privede do napačne zaznave vodenja žoge. Na primer: igralec vodi žogo, nato se ustavi in se izogiba odvzetju na prej omenjen način. Graf poti žoge je na mestu, kjer je bila žoga samo v bližini tal, nižji, kakor na mestu, kjer se je od tal odbila.

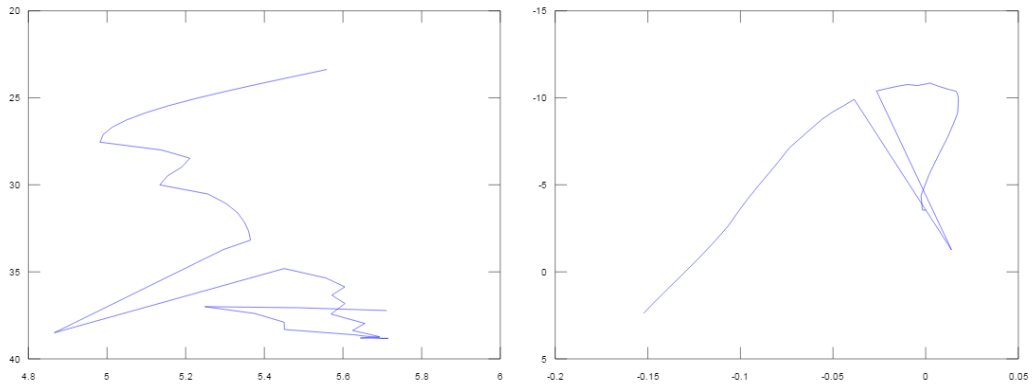
Nezanesljivi podatki o času napada V določenih trenutkih je čas napada v podatkih napačen. Tega ne bi niti opazili, če ne bi na tekmi Miami proti Washingtonu 5 sekund po začetku igre preostali čas za napad kazal 16 sekund namesto 19.

Možno neujemanje podatkov kateri igralci so trenutno na igrišču Do te napake pride zelo redko. Običajno se to zgodi po menjavi, lahko pa tudi sredi igre.

Nenatančni podatki o smeri žoge pri metu Kmalu po začetku smo s pomočjo grafične vizualizacije, ki smo jo naredili s pomočjo javanske knjižnice Swing, opazili, da pot žoge v nekaterih primerih poteka nenavadno. Z vrha gledano je bila pot žoge v letu namesto ravne linije bolj ali manj nazobčana. Sprva smo odstopanja pripisovali neprimerni uporabi Swinga in smo poskusili z Octavom, kjer pa se stanje ni popravilo (glej sliko 5.2).

Po dolgotrajnem preučevanju nam je uspelo ugotoviti vzorec pojavitve odstopanj: odstopanja so se pojavljala pri metih s strani; bolj kot je bila pot žoge vzporedna s tablo, večje je bilo odstopanje od ravne črte. To spoznanje je botrovalo opustitvi uporabe linearne aproksimacije pri obdelavi metov.

Težave pri zaznavanju igralca, ki žogo izbije Ker tak dogodek traja zelo kratek čas, lahko pride do tega, da igralec, ki žogo izbija, ni zabeležen v bližnji okolici žoge. V primeru, da je igralcev več, pa iz podatkov ni



(a) Met na koš – gledano z vrha. (b) Met na koš – gledano s strani.

Slika 5.2: Primer vizualizacije meta na koš s popačenimi podatki.

jasno, kdo je žogo zbil.

Ta problem se da do neke mere zaobiti tako, da se v primeru, da je naslednji igralec s posestjo iz drugega moštva, zabeleži anonimno izgubo žoge. Če pa je iz istega moštva, se preveri pot žoge za podajo.

Manjakajoči podatki Včasih se lahko zgodi, da manjka celoten začetni interval tekme (tudi do pol minute).

Poglavje 6

Zaključek

V okviru diplomske naloge smo relativno (glej poglavje 5) uspešno rešili osnovno zadano nalogo. Uspelo nam je ugotoviti skladnost podatkov s fizikalnimi zakonitostmi (da je pot žoge pri metu na koš, gledano z vrha, daljica in parabola, gledano s strani), vendar teh ugotovitev nismo mogli upoštevati pri vseh primerih. Razlog za to je stanje vhodnih podatkov. V določenih primerih je namreč prisoten šum, ki močno zniža zanesljivost algoritma, da bo prepoznal lastnosti (glej razdelek 5.3). Sprva smo menili, da gre za našo napako, ko pa smo podatke vnesli in izrisovali grafe gibanja žoge v Octavu, smo ugotovili, da je razlog drugje.

Stanje vhodnih podatkov je tudi glavni razlog za omejeno rešitev, saj smo zelo veliko časa porabili za ugotavljanje vzorca ponovitve napak in prilagajanje algoritma. Prilagoditev ni uspela v celoti.

Zanimiv je tudi podatek, da od konca januarja 2016 surovi podatki za javnost niso več dostopni "zaradi tehničnih težav" [13].

Na podlagi opravljenega dela lahko izluščimo naslednje ugotovitve:

- na osnovi žogi najbližjega igralca je mogoče zanesljivo slediti preprostim podajam, ko je žoga zelo kratek čas blizu nasprotnikovim igralcem,
- s preprostim preverjanjem je mogoče precej zanesljivo prepoznati zapletene situacije, na primer preigravanje, kjer pride do izmenjujočih

se intervalov žogi najbližjega igralca med napadalcem in obrambnim igralcem nasprotnega moštva,

- obstajajo napake (šum v podatkih), ki minimizirajo točnost pristopa z aproksimacijo,
- zaradi teh napak se je potrebno zateči k sklepanju namesto k izračunavanju,
- točnost aplikacije je na dane podatke relativno dobra – uspešna je v povprečno 84 odstotkih primerov (glej razdelek Opis in rezultati testiranja 4.4),
- zdi se, da je težava v optični prepoznavi položaja žoge, ko je ta med kamero in gledalci, oziroma je prepoznava slabša, bolj ko je žoga blizu roba igrišča (gledalci so ozadje slike), vendar je to bolj domneva oziroma sklepanje, saj točnih specifikacij sistema nismo našli,
- kjer je mogoče dokaj dobro predvideti pravilne vrednosti – primer za to je slika 5.2, iz katere je očitno, da gre za parabolično pot žoge – bi lahko do neke mere izboljšali pravilnost algoritma.

Med procesom razvoja aplikacije in pisanjem diplomske naloge smo nadgradili programerske veščine in pridobili nova znanja. Rezultati predstavljajo izhodišče za nadaljnje delo na področju statistične analize in modeliranja. Zasnova aplikacije omogoča nadgradnjo v smeri večje zanesljivosti in v smeri večjega nabora prepoznanih prvin košarke.

Literatura

[1] Apache Maven[Online] Dosegljivo:
https://en.wikipedia.org/wiki/Apache_Maven [Dostopano: 6. 7. 2016].

[2] Json [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/JSON [Dostopano: 19. 7. 2016].

[3] Jsoup[Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Jsoup [Dostopano: 19. 7. 2016].

[4] Programski jezik Java[Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Java_(programming_language) [Dostopano: 19. 7. 2016].

[5] Numerična metode[Online]. Dosegljivo:
https://http://mathforcollege.com/nm/ [Dostopano: 24. 7. 2016].

[6] Octave online[Online]. Dosegljivo:
http://octave-online.net/ [Dostopano: 19. 7. 2016].

[7] SportVU[Online]. Dosegljivo:
https://en.wikipedia.org/wiki/SportVU [Dostopano: 19. 7. 2016].

[8] www.physicsclassroom.com Dosegljivo:
http://www.physicsclassroom.com/class/vectors/Lesson-2/Horizontal-and-Vertical-Components-of-Velocity [Dostopano: 19. 7. 2016].

[9] The future of NBA statistics[Online]. Dosegljivo: https://www.youtube.com/watch?v=EpVmtEpa9lE [Dostopano: 19. 7. 2016].

[10] The Math Behind Basketball's Wildest Moves Rajiv Maheswaran TED Talks. Dosegljivo: https://www.youtube.com/watch?v=66ko_cWSHBU [Dostopano: 24. 7. 2016].

[11] Orel, B. Osnove numerične matematike Ljubljana: Fakulteta za računalništvo in informatiko.

[12] 2016 NBA Raw SportVU Game Logs. Dosegljivo: https://github.com/neilmj/BasketballData/tree/master/2016.NBA.Raw.SportVU.Game.Logs [Dostopano: 24. 7. 2016].

[13] What's New Dosegljivo: http://stats.nba. com/help/whatsnew/ [Dostopano: 24. 7. 2016].

Dodatki

Dodatek A

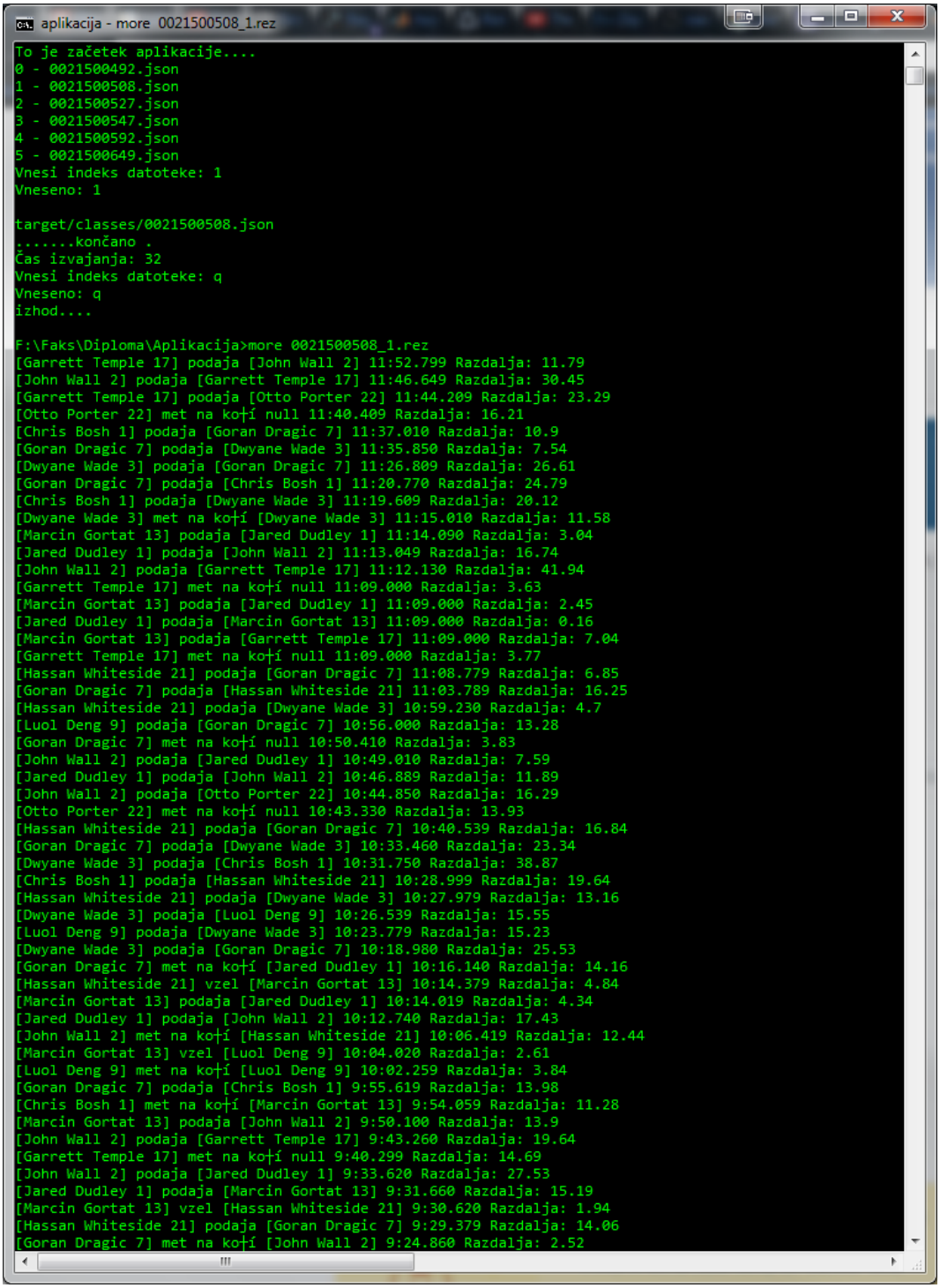
Uporabniška navodila

Aplikacijo se uporablja preko ukazne vrstice. Ob zagonu v delovnem direktoriju aplikacija prebere vsebino in izpiše datoteke .json, na katere je možen sklic po indeksu prikaza ali vnos absolutne poti do datoteke. Ekranska slika A.1 prikazuje primer že izvedene aplikacije, ko je uporabnik izbral zapis številka ena (drugi po vrsti). Pri tem mu je uporabniški vmesnik prikazal pot do izbrane datoteke.

Vrstica s pikami predstavlja prikazovalnik napredka (angl. progress bar), beseda "končano"pomeni konec obdelave trenutne datoteke. Sledi izpis trajanja obdelave v sekundah. Uporabniški vmesnik nato čaka na uporabnikovo izbiro – na voljo ima šest različnih datotek. V primeru s slike je uporabnik izbral izhod iz aplikacije s črko "q"in je nato v ukazni vrstici z ukazom "more"[1] izpisal del rezultata (spodnji del slike).

Aplikacija zatem obdela podatke o tekmi in zapiše v datoteko z enakim imenom in končnico .rez.

[1]Aplikacija je razvita in testirana samo za sistem Windows®(7).



Slika A.1: Ekranska slika vhoda v aplikacijo in izpisa dela rezultata