UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

SEMINARSKI RAD

Alati i metode veštačke inteligencije i softverskog inženjerstva

RAZVOJ INTERAKTIVNE IGRE “GUESSING GAME” korišćenjem funkcionalne paradigme u jeziku Clojure

Mentor:
Prof. dr Dragan Đurić

Autor:
Lazar Cvetković 2025/3833

Beograd, 2026.

APSTRAKT

U ovom seminarskom radu opisan je proces projektovanja i implementacije interaktivne konzolne igre pod nazivom „GUESSING GAME“. Rad se fokusira na primenu funkcionalne paradigme programiranja korišćenjem programskog jezika Clojure na Java virtuelnoj mašini (JVM). Centralni deo rada prikazuje razvoj logike za prepoznavanje korisničkog unosa uz toleranciju grešaka u kucanju, kao i kreiranje sistema za čuvanje podataka i rezultata. Kroz praktičan primer demonstrirano je kako se koncepti poput nemutabilnosti podataka i upravljanja stanjem koriste za izgradnju stabilne i funkcionalne aplikacije.

Ključne reči: Clojure, funkcionalno programiranje, ASCII art, Levenshtein algoritam, softversko inženjerstvo.

SADRŽAJ

1. UVOD

U svetu modernog softverskog inženjerstva, funkcionalno programiranje doživljava renesansu kao odgovor na sve veće zahteve za konkurentnošću, skalabilnošću i održivošću softverskih sistema. Za razliku od imperativnog pristupa, koji se oslanja na promenu stanja i mutabilne podatke, funkcionalna paradigma naglašava korišćenje čistih funkcija, nemutabilnost podataka i deklarativni stil pisanja koda.

Cilj ovog seminarskog rada je prikaz praktične primene programskog jezika Clojure, modernog dijalekta Lispa koji se izvršava na Java virtuelnoj mašini (JVM). Kao studija slučaja, razvijena je interaktivna konzolna igra pod nazivom „guessing-game“.

Ova aplikacija predstavlja igru u kojoj korisnik pogađa skrivene pojmove na osnovu vizuelnih prikaza kreiranih od ASCII karaktera. Cilj igre je tačno identifikovati šta se nalazi na ASCII slici kako bi se osvojili poeni, pri čemu sistem dodatno nagrađuje brzinu i tačnost odgovora, a kažnjava greške oduzimanjem života.

2. TEHNOLOŠKI OKVIR I ALATI

Za realizaciju softverskog rešenja „guessing-game“ korišćeni su alati koji omogućavaju stabilan rad aplikacije i lako pokretanje. Osnovu sistema čini Java virtuelna mašina (JVM), koja služi kao platforma za izvršavanje koda. Na ovaj način osigurana je visoka efikasnost izvršavanja koda, što je ključno za fluidan rad interaktivne konzolne aplikacije.

2.1 PROGRAMSKI JEZIK CLOJURE

Clojure je funkcionalni programski jezik koji se izvršava na JVM-u. Osnovne odlike ovog jezika su da on omogućava pisanje čistog i preglednog koda. Glavna razlika u odnosu na standardne jezike poput Jave i C# je u tome što su podaci u Clojure-u podrazumevano nepromenljivi (nemutabilni). To znači da kada jednom definišemo podatak, on se ne menja, već se za svaku promenu pravi nova verzija tog podatka, što znatno smanjuje mogućnost grešaka u radu programa.

2.2 KORIŠĆENE KOMANDE I FUNKCIJE

Da bi igra funkcionisala, korišćene su specifične komande jezika Clojure. U nastavku su objašnjene najvažnije funkcije koje čine strukturu ovog projekta:

- **def** i **defn** su osnovne komande, **def** se koristi kada želimo da definišemo neku globalnu vrednost, dok se **defn** koristi za pravljenje funkcija. Svaka radnja u igrici (npr. provera slova, crtanje ekrana) je definisana pomoću **defn**.
- **let** služi za definisanje lokalnih promenljivih unutar jedne funkcije, a te promenljive važe samo dok se ta funkcija izvršava i ne vide se u ostatku programa.
- **atom, swap!** i **reset!**: iako su podaci u Clojure-u nepromenljivi, igra mora da pamti trenutno stanje (broj poena, preostale živote). Za to je korišćen **atom**. Atom je poseban tip podataka koji dozvoljava promenu vrednosti. Komanda **swap!** služi

da se trenutna vrednost ažurira (npr. dodaj 1000 poena na trenutni rezultat), dok **reset!** služi da se vrednost postavi na nulu ili neki drugi broj (pomoglo je kod resetovanja igre).

- **loop i recur** su zamena za klasične **for** i **while** petlje. Pošto igra mora da se vrti u krug dok korisnik ne izađe, korišćen je **loop** da se označi početak petlje i **recur** da se igra vrati na početak sa novim podacima. Ovo omogućava da igra radi beskonačno dugo bez opterećenja memorije, odnosno sprečava ***Stack Overflow***.
- **Threading Macros (-> i ->>)** se koriste da kod bude čitljiviji. Umesto da se piše funkcija u funkciji, ovi makroi dozvoljavaju da se pišu koraci jedan ispod drugog, gde rezultat prve funkcije automatski postaje ulaz za sledeću.
- **map, filter i reduce** se koriste za rad sa listama podataka. Korišćene su da se prođe kroz listu rezultata, filtriranje tačnih odgovora i sabiranje ukupnih poena, bez potrebe za ručnim pisanjem petlji.
- **try** i **catch** služe za hvatanje grešaka. Ako program pokuša da učita fajl koji ne postoji, **try** i **catch** blok sprečava da se igrica sruši i umesto toga ispisuje poruku o grešci.

2.3 JAVA INTEROPERABILNOST (JAVA INTEROP)

Jedna od prednosti programskog jezika Clojure je mogućnost direktnog korišćenja klasa i biblioteka koje pripadaju Java sistemu. U okviru ovog projekta, ova funkcionalnost je iskorišćena za implementaciju audio sistema, bez potrebe za uključivanjem dodatnih eksternih zavisnosti. Umesto traženja specifičnih Clojure biblioteka za zvuk, iskorišćen je standardni Java paket **javax.sound.sampled**. Na ovaj način omogućeno je učitavanje i reprodukcija zvučnih efekata (.wav fajlova) direktnim povezivanjem Java metoda iz Clojure koda, a time je postignuta visoka efikasnost i jednostavnost implementacije što spada u željeni kriterijum za izradu ovog projekta.

2.4 EDN FORMAT (EXTENSIBLE DATA NOTATION)

Za potrebe perzistencije (trajnog čuvanja) podataka, kao što su baza pojmova za pogađanje i rezultati igrača, odabran je **EDN** format. **EDN** je format za serijalizaciju podataka koji je prirodan za Clojure. Iako je sličan popularnom **JSON** formatu, **EDN** je odabran jer podržava specifične strukture podataka koje se koriste u kodu (poput ključnih reči i skupova). Time je omogućeno da se podaci čitaju iz fajlova i direktno pretvaraju u upotrebljive strukture u memoriji, bez potrebe za kompleksnim procesom parsiranja ili transformacije podataka.

3. ARHITEKTURA SISTEMA

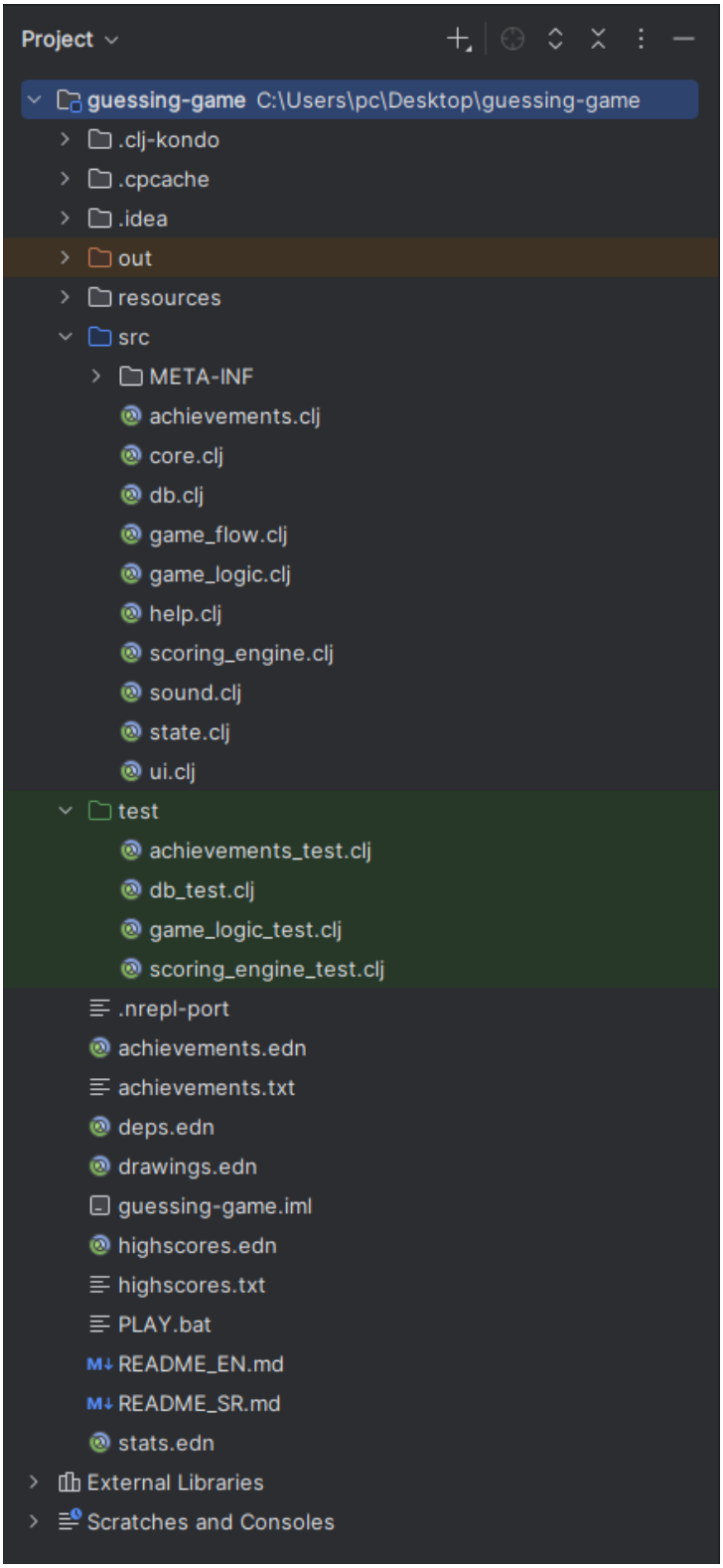
Igra „guessing-game“ projektovana je modularno, poštujući princip razdvajanja odgovornosti (Separation of Concerns). Ovakva arhitektura omogućava lakše održavanje koda, testiranje pojedinačnih komponenti i buduća proširenja bez narušavanja osnovne logike igre. Celu igru čini skup međusobno povezanih modula, gde svaki modul ima svoju jasno definisanu ulogu, počevši od upravljanja sa bazom podataka, preko logike igre, do interakcije sa korisnikom.

3.1 ORGANIZACIJA PROJEKTA I DIREKTORIJUMA

Projekat prati standardnu strukturu Clojure aplikacija, što olakšava navigaciju i razumevanje koda drugim programerima. U korenu projekta nalaze se dva ključna direktorijuma:

- **src (Source)**: Sadrži sav izvorni kod aplikacije organizovan po imenskim prostorima.
- **resources**: Sadrži statičke resurse neophodne za rad igre, kao što su audio datoteke (.wav format).
- **root direktorijum**: U njemu se nalaze konfiguracioni fajlovi i .edn fajlovi sa bazom podataka u kojoj su smešteni ASCII crteži. Takođe se ovde nalazi i „PLAY.bat“ skripta za pokretanje igre preko Windows Terminala.

Na slici ispod prikazana je struktura root foldera u razvojnom okruženju:



Slika 1. Prikaz strukture projekta u IntelliJ IDEA okruženju

3.2 MODULI (NAMESPACES)

Na osnovu strukture unutar **src** foldera, logika aplikacije je podeljena na sledeće module:

- **core.clj**: Ulazna tačka aplikacije. Sadrži **main** funkciju koja pokreće igru i inicijalizuje sve ostale komponente.
- **game_logic.clj i game_flow.clj**: Ovi moduli sadrže "mozak" igre. Definišu pravila, tok jedne partije i proveru uslova za pobedu ili poraz.
- **scoring_engine.clj**: Ovo je specijalizovan modul za računanje poena, koji uzima u obzir brzinu odgovora i tačnost unosa.
- **db.clj**: Zadužen za komunikaciju sa fajlovima. Njegova uloga je da učita podatke iz **drawings.edn** i **highscores.edn** fajlova i pretvori ih u format koji igra može da koristi.
- **ui.clj**: Upravlja korisničkim interfejsom u konzoli, iscrtava ASCII crteže i prikazuje poruke igraču.
- **state.clj**: Centralno mesto za čuvanje stanja igre (broj života, trenutni poeni) korišćenjem Clojure atoma.
- **sound.clj**: Modul koji povezuje igru sa zvučnim fajlovima iz resources foldera i reprodukuje zvukove u odgovarajućim trenucima.

4. IMPLEMENTACIJA LOGIKE

Srž igre „guessing-game“ leži u njenoj sposobnosti da inteligentno obrađuje korisnički unos i nagrađuje igrača na osnovu performansi. Ovaj deo sistema implementiran je kroz čiste funkcije u modulima **game_logic.clj** i **scoring_engine.clj**, dok se upravljanje stanjem vrši kroz **atom** u modulu **state.clj**.

4.1 ALGORITAM ZA TOLERANCIJU GREŠAKA (LEVENSHTEIN)

Igra je uz pomoć ovog algoritma tolerantnija na slučajne slovne greške prilikom unošenja odgovora, a to je bitno zato što su korisnici koji igraju ovu igru pod pritiskom vremena, a to dovodi do grešaka u kucanju tačnog odgovora. Ovim se veoma poboljšava užitak igranja ove igrice zato što se dozvoljava manja greška u kucanju i ukoliko se ona desi igrač ipak dobije poene za svoj dati odgovor. Da bi se sve ovo postiglo i sprovelo sa plana u delo, zaslužan je algoritam Levenshtein-ove distance. Ovaj algoritam meri razliku između dva stringa računajući minimalan broj operacija (brisanja, umetanja ili zamene karaktera) potrebnih da se jedan string transformiše u drugi.

U funkciji **levenshtein** (slika 2), korišćen je pristup dinamičkog programiranja optimizovan za Clojure. Umesto klasične matrice, koristi se **vektor (v0)** koji se ažurira kroz **loop/recur** petlju, čime se u velikoj meri smanjuje potrošnja memorije.

Funkcija judge-answer koristi ovaj izračunati rezultat za klasifikaciju odgovora:

1. **:perfect** – Ako je distanca 0 (potpuno tačan unos igrača).
2. **:close** – Ako postoji mala greška. Prag tolerancije je drugačiji:
    - Za reči duže od 5 slova, dozvoljene su 2 greške.
    - Za reči od 5 slova ili kraće, dozvoljena je 1 greška.
3. **:wrong** – U svim ostalim slučajevima.

```clojure
(ns game-logic  11 usages
  (:require [clojure.string :as str])
  (:import [java.util Random Collections]))

(defn levenshtein [s1 s2]  8 usages
  (let [s1 (str/lower-case s1)
        s2 (str/lower-case s2)
        v0 (vec (range (inc (count s2))))]
    (loop [s1 s1 v0 v0]
      (if (empty? s1)
        (peek v0)
        (let [prev-v0 v0
              c1 (first s1)]
          (recur (rest s1)
                 (reduce (fn [v i]
                           (let [c2 (nth s2 (dec i))
                                 cost (if (= c1 c2) 0 1)
                                 del (inc (nth prev-v0 i))
                                 ins (inc (peek v))
                                 sub (+ (nth prev-v0 (dec i)) cost)]
                             (conj v (min del ins sub))))
                         [(inc (first prev-v0))]
                         (range 1 (inc (count s2)))))))))))

(defn judge-answer [user-input solution]  7 usages
  (let [dist (levenshtein (if (nil? user-input) "" (str/trim user-input))
                          solution)
        len (count solution)]
    (cond
      (= dist 0) :perfect
      (<= dist (if (> len 5) 2 1)) :close
      :else :wrong)))
```

Slika 2. Implementacija Levenshtein algoritma i logike za proveru odgovora (game_logic.clj)

4.2 MATEMATIČKI MODEL BODOVANJA

Sistem bodovanja je dizajniran da motiviše igrača na brže odgovore i održavanje niza pogodaka **(streak)**. Logika je izolovana u modulu **scoring-engine** kroz funkciju **calculate-points**.

Formula za računanje poena sastoji se od tri komponente:

1. Osnovna vrednost (base-value):

- 1000 poena za tačan odgovor **(:perfect)**.
- 500 poena za približan odgovor **(:close)**.

2. Vremenski bonus (time-bonus): Dodeljuje se samo za perfektne odgovore.

Računa se kao procenat preostalog vremena u odnosu na ukupno vreme, pomnožen sa 1000.

- Primer: Ako je ostalo 50% vremena, igrač dobija dodatnih 500 poena.

3. Množilac niza (multiplier): Nagrada za uzastopne pogotke. Svakih 10 uzastopnih pogodaka povećava množilac za 1.

- Formula: 1 + (current-streak / 10).

Konačna formula glasi:

$$\textit{Poeni} = (\textit{Base} + \textit{TimeBonus}) \times \textit{Multiplier}$$

```
(ns scoring-engine)  4 usages

(defn calculate-points [result time-left total-time current-streak]  5 usages
  (let [
        base-value (if (= result :perfect) 1000 500)
        time-bonus (if (= result :perfect)
                     (int (* 1000 (/ time-left total-time)))
                     0)
        multiplier (+ 1 (/ current-streak 10))]
    (int (* (+ base-value time-bonus) multiplier))))

(defn update-streak [current-streak result]  5 usages
  (if (= result :perfect)
    (inc current-streak)
    0))
```

Slika 3. Funkcija za kalkulaciju poena (scoring_engine.clj)

4.3 UPRAVLJANJE STANJEM I JAVA INTEROPERABILNOST

Iako Clojure favorizuje nemutabilnost, interaktivna priroda igre zahteva praćenje promena (skor, životi, trenutni nivo). Ovo je rešeno u modulu state korišćenjem **atoma (game-state)**. Atom sadrži mapu sa svim ključnim parametrima igre **(:score, :lives, :streak, :difficulty-mode)**. Funkcije poput **reset-game-stats!** koriste **swap!** mehanizam za atomičnu promenu stanja, čime se osigurava konzistentnost podataka čak i u slučaju asinhronih događaja (primer je isticanje tajmera).

Poseban segment implementacije posvećen je takmičarskom duhu i ponovljivosti igre, implementacijom **Seed sistema (Seme za generisanje)**. Inspiracija za ovaj mehanizam potekla je iz jedne od najpopularnijih igara na svetu – **Minecraft-a**. Baš kao što u Minecraft-u isti „seed“ uvek generiše identičan svet, tako i u igri „guessing-game“ isti unos seed-a (npr. seed „FON2026“) garantuje da će redosled zagonetki i ASCII crteža biti potpuno identičan.

Ovaj sistem u potpunosti rešava problem fer-pleja: kako omogućiti ravnopravno takmičenje igračima na različitim računarima? Zahvaljujući ovom sistemu, dva igrača mogu uneti isti seed i takmičiti se ko će sakupiti više poena na **istom nizu zagonetki**, čime se eleminiše faktor sreće, a u prvi plan dolaze znanje, brzina i snalažljivost.

Tehnička realizacija ovog sistema oslanja se na Java Interoperabilnost. U funkciji **seeded-shuffle** (modul **game_logic.clj**), korišćene su Java klase **java.util.Random** i **java.util.Collections**. Clojure ovde koristi Javin generator pseudo-slučajnih brojeva inicijalizovan specifičnim seed-om, što prikazuje sinergiju između funkcionalnog jezika i bogatog Java sistema.

```
(defn seeded-shuffle [coll seed-str] 1 usage

  (let [al (java.util.ArrayList. coll)
        rng (if (str/blank? seed-str)
              (Random.)
              (Random. (.hashCode seed-str)))]
    (Collections/shuffle al rng)
    (vec al)))

(defn time-up? [state] 3 usages
  (let [limit (:time-limit state)
        start (:start-time state)
        now (System/currentTimeMillis)]
    (> (- now start) limit)))
```

Slika 4. Implementacija seed sistema i Java interoperabilnost (game_logic.clj)

4.4 KORISNIČKI INTERFEJS I VIZUELIZACIJA

S obzirom na to da je reč o konzolnoj aplikaciji, vizuelna reprezentacija je realizovana isključivo putem teksta i ASCII karaktera. Modul **ui.clj** služi kao prezentacioni sloj aplikacije, zadužen za komunikaciju sa korisnikom.

Funkcija **render-art** je centralna komponenta za prikaz zagonetki. Ona prihvata niz linija teksta (ASCII crteža) i ispisuje ih u formatiranom bloku, jasno odvajajući grafiku od ostatka interfejsa. Pored toga, ovaj modul implementira i navigacione menije **(select-category, select-difficulty)** koji koriste **loop/recur** mehanizam kako bi validirali korisnički unos i sprečili nastavak igre dok se ne izabere validna opcija. I za kraj uz funkciju **print-full-leaderboard** formatiraju se podaci u pregledne tabele koristeći **clojure.core/format** funkciju za poravnanje kolona, čime se postiže pregledan izgled tabele rezultata u tekstualnom režimu.

```
(defn print-high-scores [scores] 1 usage
  (println "\n========================================")
  (println "       🏆 HALL OF FAME (TOP 5) 🏆       ")
  (println "========================================")
  (println (format "%-5s %-15s %-10s" "RANK" "PLAYER" "SCORE"))
  (println "----------------------------------------")
  (doseq [[idx entry] (map-indexed vector scores)]
    (println (format "%-5d %-15s %-10d"
                     (inc idx)
                     (:name entry)
                     (:score entry))))
  (println "========================================\n"))

(defn print-full-leaderboard [all-scores] 1 usage
  (println "\n========================================")
  (println "           🏆 HALL OF FAME 🏆           ")
  (println "========================================")

  (doseq [diff [:easy :medium :hard :hardcore]]
    (let [scores (get all-scores diff [])]
      (println (str "\n      HIGH SCORES (" (str/upper-case (name diff))")"))
      (println "")
      (println (format "%-5s %-15s %-10s" "RANK" "PLAYER" "SCORE"))
      (println "----------------------------------------")
      (if (empty? scores)
        (println "  (No scores yet)")
        (doseq [[idx entry] (map-indexed vector scores)]
          (println (format "%-5d %-15s %-10d"
                           (inc idx)
                           (:name entry)
                           (:score entry)))))))
  (println "\n========================================"))
```

Slika 5. Implementacija funkcija za prikaz tabele rezultata (ui.clj)

4.5 RAD SA PODACIMA MODUL DB.CLJ

Perzistencija podataka je ključna za dugoročnu igrivost. Modul **db.clj** implementira kompletan sloj za pristup podacima **(Data Access Layer)**, izolujući logiku čitanja i pisanja fajlova od ostatka aplikacije.

Sistem koristi tri glavna fajla za skladištenje podataka u EDN formatu:

1. **drawings.edn:** Sadrži bazu svih pojmova i ASCII crteža.
2. **highscores.edn:** Čuva liste najboljih rezultata po težinama.
3. **stats.edn:** Prati detaljnu statistiku igrača (broj odigranih partija, ukupni poeni).
4. **achievements.edn:** Čuva otključana dostignuća svakog igrača ponaosob.

Funkcija **save-high-score** demonstrira manipulaciju kolekcijama. Kada igrač završi partiju, ova funkcija učitava postojeće rezultate, dodaje novi rezultat, sortira listu opadajuće po broju poena, zadržava samo top 5 rezultata i zatim atomično upisuje novo stanje nazad u fajl. Pored **EDN formata**, ovaj modul automatski generiše i čitljive **.txt** izveštaje **(highscores.txt i achievements.txt)**, što omogućava adminu lak pregled rezultata van igre.

```
(defn save-high-score [nickname score difficulty]  2 usages
  (try
    (let [all-scores (load-high-scores)
          current-list (get all-scores difficulty [])
          new-list (take 5 (reverse (sort-by :score (conj current-list {:name nickname :score score}))))
          updated-all-scores (assoc all-scores difficulty new-list)]
      (spit "highscores.edn" (pr-str updated-all-scores))
      (let [txt-output (str (format-scores-for-txt updated-all-scores :easy)
                            (format-scores-for-txt updated-all-scores :medium)
                            (format-scores-for-txt updated-all-scores :hard)
                            (format-scores-for-txt updated-all-scores :hardcore))]
        (spit "highscores.txt" txt-output))
      new-list)
    (catch Exception e (println "Error saving highscore:" (.getMessage e)) [])))
```

```
(defn save-new-achievements [nickname new-unlocked-set]  1 usage
  (try
    (let [all-data (load-achievements)
          existing-set (get all-data nickname #{})
          updated-set (set/union existing-set new-unlocked-set)
          truly-new (set/difference new-unlocked-set existing-set)
          final-data (assoc all-data nickname updated-set)]
      (spit "achievements.edn" (pr-str final-data))
      (spit "achievements.txt" (format-achievements-for-txt final-data))
      truly-new)
    (catch Exception e #{})))
```

Slika 6. Implementacija funkcija za perzistenciju rezultata i dostignuća (db.clj)

4.6 GLAVNA PETLJA I ARHITEKTURA APLIKACIJE

Sve opisane komponente u prethodnim poglavljima povezuju se u modulu **core.clj**, koji predstavlja ulaznu tačku aplikacije. Funkcija **-main** inicijalizuje globalno stanje učitavanjem baze pojmova i postavlja korisničko ime. Srce navigacije je funkcija main-menu-loop. Ona implementira beskonačnu petlju koja prikazuje glavni meni i delegira izvršavanje odgovarajućim modulima na osnovu korisničkog izbora. Ovakva struktura osigurava da se aplikacija nikada ne ugasi sama od sebe, već uvek vraća korisnika na početni meni nakon završene akcije, sve dok eksplicitno ne izabere opciju za izlaz iz nje.

4.7 AUDIO SISTEM I DOSTIGNUĆA IGRAČA

Kako bi korisničko iskustvo bilo potpunije, aplikacija implementira audio povratne informacije i sistem naprednih dostignuća kako bi se povisio nivo takmičarskog duha u igračima. Ovi moduli su ključni za motivaciju igrača i osećaj ispunjenosti i želje za ponovnim igranjem ove igre.

Audio sistem **(sound.clj)** realizovan je korišćenjem Java Interoperabilnosti (paket javax.sound.sampled). Funkcija **play** prikazuje korišćenje konkurentnog programiranja, a to je reprodukcija zvuka koja se izvršava unutar **future** bloka. Ovo je bitno jer omogućava da se zvuk izvršava na zasebnoj niti **(thread)**, sprečavajući da glavni tok igre zamrzne dok traje zvučni efekat.

Zvučni efekti u igrici su odabrani i inspirisani legendarnom igrom **Super Mario**. Korišćenje prepoznatljivih zvukova budi u igračima nostalgičan retro osećaj i pruža jasnu povratnu informaciju o uspehu ili neuspehu, a to psihološki doprinosi većem zadovoljstvu tokom igranja.

Sistem dostignuća (achievements.clj) predstavlja složen sistem pravila. U njemu je definisano 22 različitih kriterijuma koji prikazuju brzinu i upornost igrača. Funkcija **evaluate** koristi **cond –>** da proveri niz uslova nad trenutnim stanjem igre. Jedan od najinteresantnijih i najkreativnijih detalja je upotreba **java.time.LocalTime** klase za “Time-based” dostignuća, kao

što su “Night Owl” za igranje igrice noću u realnom vremenu ili “Early Bird” za igranje igrice u jutarnjim satima.

```
(defn evaluate [game-state round-data player-stats total-db-count] 14 usages
  (let [{:keys [result time-left total-time hints-used points]} round-data
        {:keys [streak difficulty-mode lives]} game-state


        total-solved-count (count (:solved-ids player-stats))
        category-counts (:category-counts player-stats)
        total-score (:total-score player-stats)
        no-hint-count (:no-hint-count player-stats)
        sessions (:sessions player-stats)
        hour (get-current-hour)]

    (cond-> #{}
            (= total-solved-count total-db-count) (conj :completionist)


            (>= (get category-counts :nature 0) 5)  (conj :nature-lover)
            (>= (get category-counts :objects 0) 5) (conj :tech-wizard)
            (>= (get category-counts :places 0) 5)  (conj :globetrotter)

            (and (= result :perfect) (< (- total-time time-left) 5)) (conj :speed-demon)
            (and (= result :perfect) (< (- total-time time-left) 2)) (conj :sonic)

            (>= streak 5)  (conj :streak-master)
            (>= streak 10) (conj :double-digits)
            (>= streak 20) (conj :unstoppable)

            (>= points 2000) (conj :big-winner)
            (>= points 3000) (conj :high-roller)
            (>= total-score 100000) (conj :centurion)

            (and (= result :perfect) (= difficulty-mode :hardcore)) (conj :hardcore-hero)
            (and (= result :perfect) (= difficulty-mode :hardcore) (= lives 3)) (conj :invincible)

            (and (= result :perfect) (zero? hints-used)) (conj :eagle-eye)
            (>= no-hint-count 10) (conj :hint-hater)

            (and (>= hour 0) (< hour 5)) (conj :night-owl)
            (and (>= hour 5) (< hour 8)) (conj :early-bird)
            (>= sessions 5) (conj :regular)
            (>= total-solved-count 10) (conj :cyber-sentry))))
```

Slika 7. Logika za evaluaciju uslova i dodelu dostignuća (achievements.clj)

5. TESTIRANJE KODA

Razvoj projekata a posebno igrica zahteva rigorozan pristup osiguranju kvaliteta. Kako bi se garantovala stabilnost igrice i tačnost algoritama, implementiran je sistem automatizovanih testova korišćenjem standardne biblioteke **clojure.test**.

Testovi su locirani u poseban direktorijum pod nazivom „test“ i pokrivaju sve aspekte sistema, a to su: procesiranje teksta, matematički model bodovanja, validacija baze podataka i logika dostignuća.

5.1 TESTIRANJE ALGORITAMA I LOGIKE IGRE

Osnovni preduslov za ispravno funkcionisanje igre je tačnost algoritma za prepoznavanje reči. U modulu **game_logic_test.clj** definisani su testovi **(Unit tests)** koji proveravaju funkciju **levenshtein** i **judge-answer**. Testovi pokrivaju različite granične slučajeve:

- **Identičnost:** Reči moraju imati distancu 0.
- **Tolerancija grešaka:** Provera da li sistem prepoznaje sitne greške (npr. „tigr“ umesto „tiger“) kao :close odgovor.
- **Imunost na velika i mala slova:** Unos „TigEr“ mora biti tretiran potpuno isto kao „tiger“.
- **Prazni stringovi:** Provera ponašanja sistema kada korisnik ne unese ništa.

Ujedno i modul scoring_engine_test.clj verifikuje matematičku ispravnost dodele poena. Nad funkcijom update-streak testovi osiguravaju da se niz pogodaka ispravno uvećava nakon tačnog odgovora, ali i resetuje na nulu u slučaju greške, a time se održava integritet takmičarskog dela igre.

5.2 TESTIRANJE PERZISTENCIJE I INTEGRITETA PODATAKA

S obzirom na to da igra zavisi od eksternih fajlova, **modul db_test.clj** služi za validaciju strukture podataka. Testovi u ovom modulu proveravaju da li se baza **ASCII crteža (drawings.edn)** uspešno učitava i da li svaki entitet poseduje obavezne atribute **(:id, :solution, :category)**.

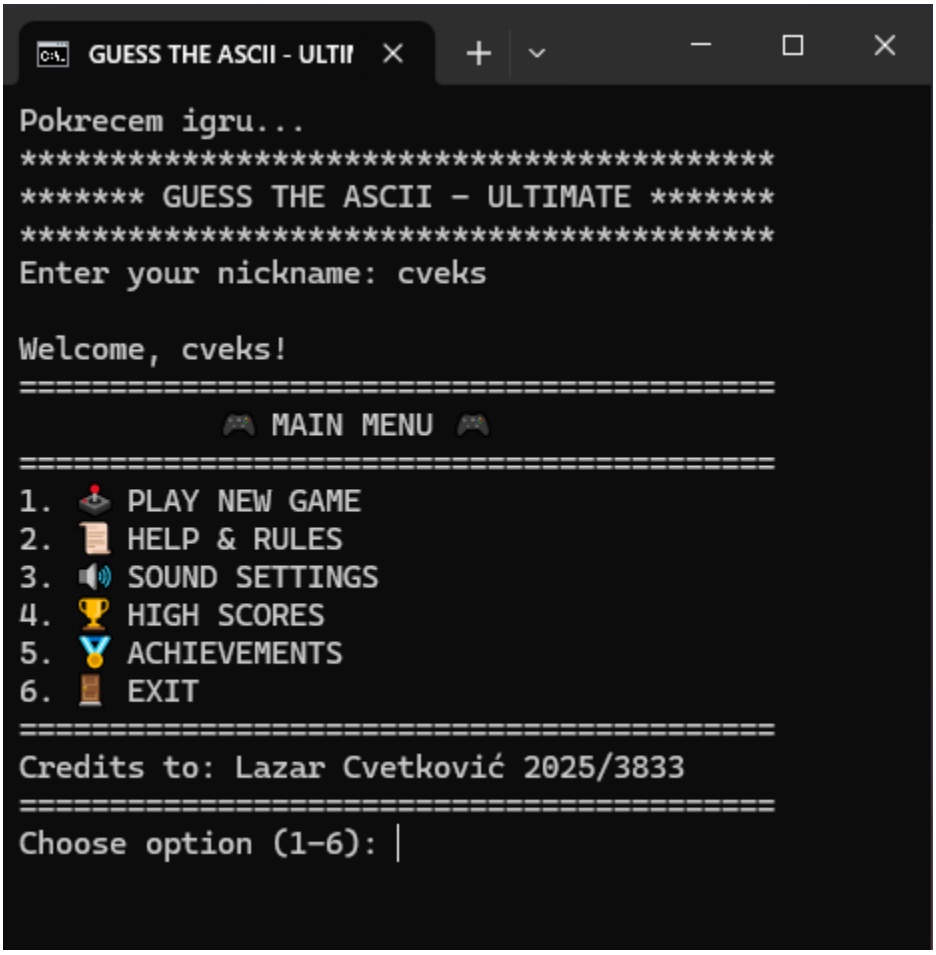
Najkorisniji aspekt ovog modula je implementacija vizuelnog testa **(preview-all-drawings-test)**. Pored svih testova koji se oslanjaju na logičke iskaze, ovaj test je ključan jer prolazi kroz celu bazu i renderuje svaki **ASCII crtež** u konzoli, što omogućava programeru brzu vizuelnu inspekciju ispravnosti svih crteža pre aktiviranja nove verzije igre.

5.3 MOCKING I TESTIRANJE VREMENSKI ZAVISNIH FUNKCIJA

Najveći izazov u testiranju ispravnosti predstavljala je logika dostignuća koja zavisi od realnog vremena. Nije praktično niti normalno čekati ponoć da bi se neki test u igrici izvršio. Rešenje ovog problema je izvršeno tehnikom **Mocking-a (lažiranja funkcija)** korišćenjem makroa **with-redefs** u modulu **achievements_test.clj**. Ovaj makro omogućava da se privremeno, u okviru testa, redefiniše funkcija **get-current-hour** tako da vraća fiksnu vrednost. Na ovaj način je deterministički potvrđeno da sistem ispravno dodeljuje vremenska dostignuća, bez obzira na to kada se testovi zapravo pokreću.

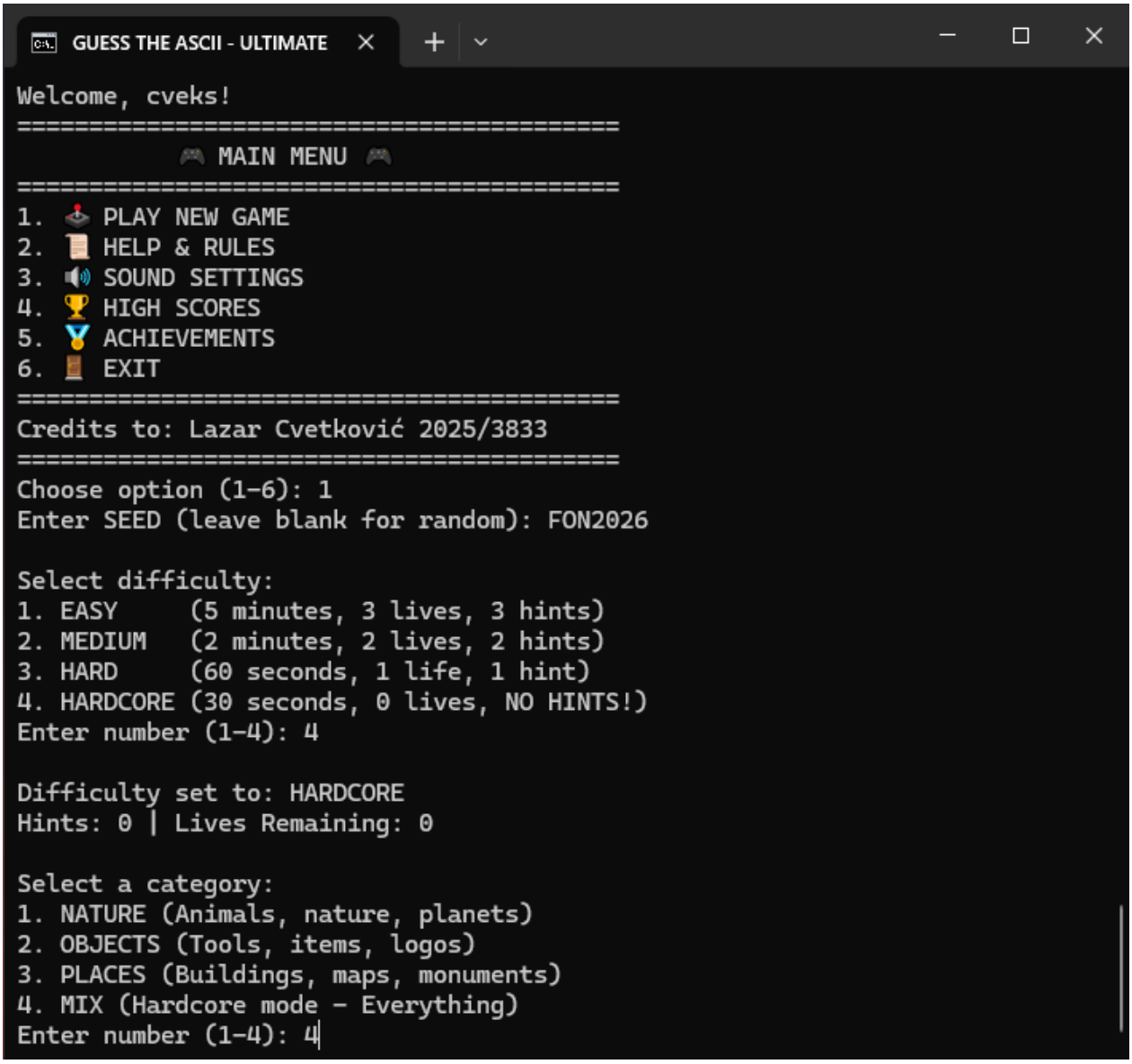
6. PRIKAZ RADNOG OKRUŽENJA I FUNKCIONALNOSTI IGRE

U ovom poglavlju je prikazan finalni izgled igrice kroz sve njene funkcionalnosti, a to su: navigacija kroz meni, prikaz statistike, rezultata, komandi i interaktivnog rešavanja datih zagonetki. Svi prikazi su generisani direktno iz konzolnog okruženja, demonstrirajući mogućnosti **ui.clj** modula za formatiranje teksta, štampanje izlaza i prikaz **ASCII grafike**. Ovo poglavlje je u potpunosti posvećeno slikovnom prezentacijom igrice.

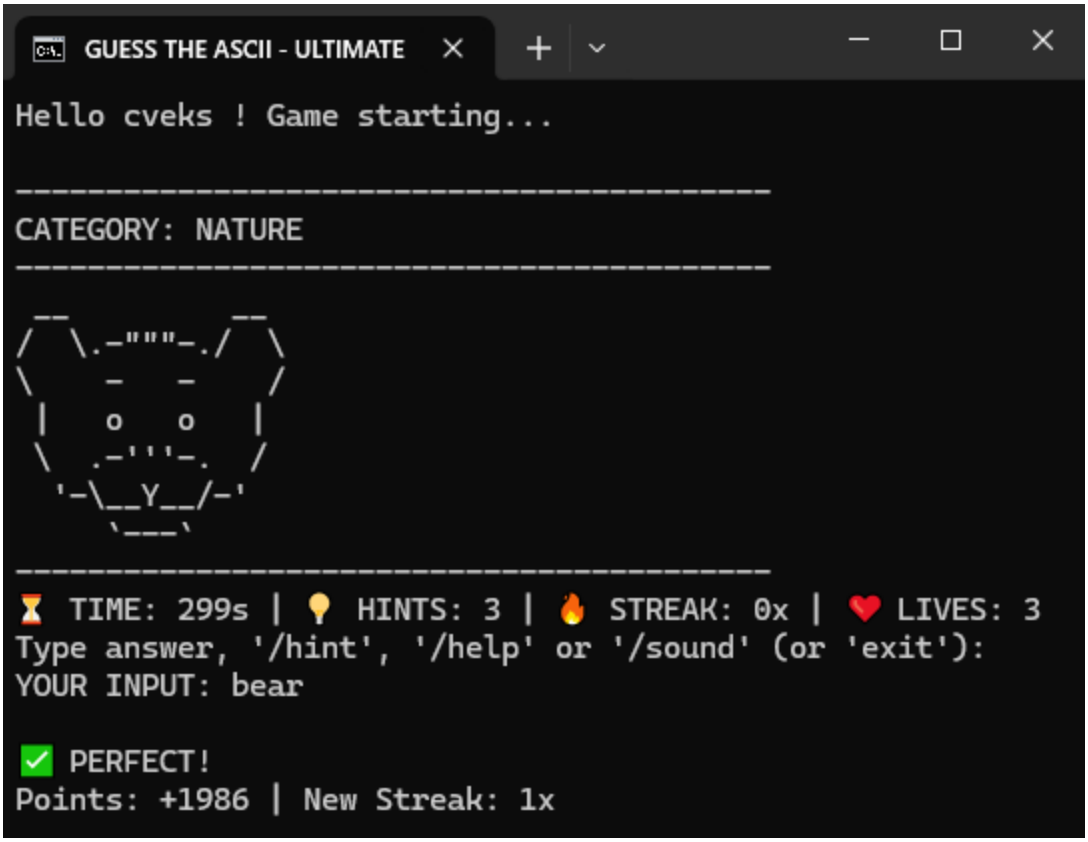


Slika 8. Prikaz glavnog menija

Slika 9. Prikaz pravila, komandi i spisak ostvarivih dostignuća

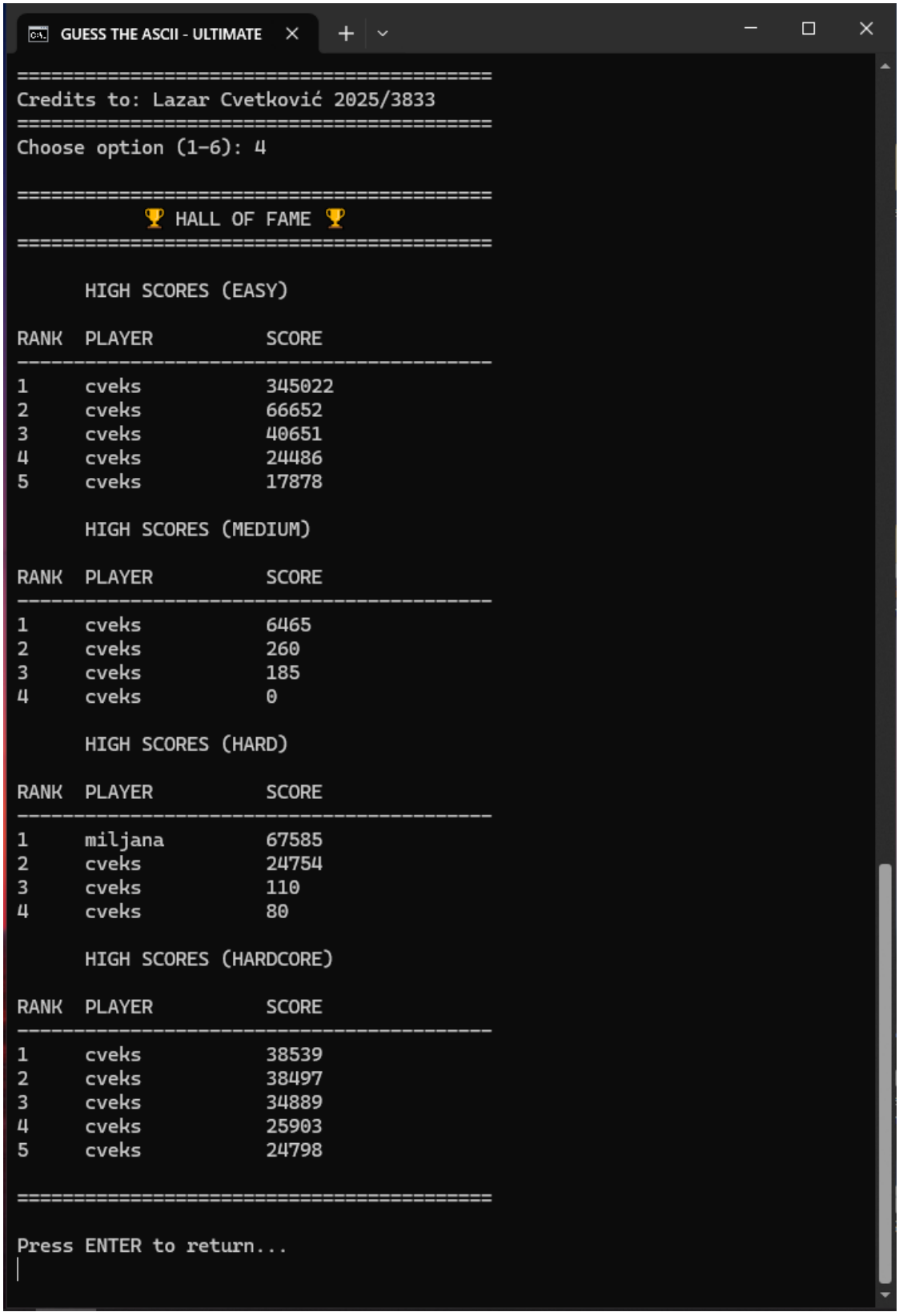


Slika 10. Prikaz konfiguracije parametara igre i unos SEED-a



Slika 11. Prikaz uspešnog rešavanja zagonetke u toku igre

Slika 12. Prikaz neuspešnog rešavanja zagonetke u toku igre



Slika 13. Prikaz tabele najboljih igrača u svim modovima

Slika 14. Prikaz kraja igre i evidencija postignutog score-a

7. ZAKLJUČAK

Cilj ovog seminarskog rada bio je prikaz praktične primene funkcionalne paradigme u razvoju interaktivnog softvera. Kroz implementaciju igre „guessing-game“, demonstrirano je kako jezik Clojure, uz podršku Java platforme, omogućava kreiranje stabilnih, održivih i proširivih aplikacija.

Ključni doprinosi ovog projekta ogledaju se u sledećem:

1. Funkcionalna arhitektura: Korišćenjem čistih funkcija i nemutabilnih struktura podataka, minimizovana je mogućnost grešaka (bugova) izazvanih nekontrolisanom promenom stanja.
2. Algoritamska rešenja: Implementacija Levenshtein algoritma značajno je unapredila korisničko iskustvo, čineći igru tolerantnom na greške u kucanju.
3. Inovacija kroz Seed sistem: Inspirisan modernim igrama poput Minecraft-a, implementiran je deterministički generator nasumičnih brojeva, što je omogućilo fer takmičenje i ponovno igranje radi boljeg rezultata.
4. Java Interoperabilnost: Uspešno je iskorišćen postojeći Java sistem za implementaciju audio sistema, dokazujući da Clojure nije izolovan jezik, već moćan alat koji nadograđuje postojeće tehnologije.

Iako je trenutna verzija aplikacije konzolnog tipa, modularna arhitektura ostavlja prostor za buduća unapređenja ove igre. Logika igre je potpuno odvojena od korisničkog interfejsa, što znači da bi se u budućnosti mogao razviti grafički interfejs (GUI) ili web verzija igre bez potrebe za menjanjem osnovnog koda koji već postoji.

Ovaj projekat potvrđuje da funkcionalno programiranje nije samo akademski koncept, već efikasan pristup za rešavanje realnih problema u softverskom inženjerstvu.

8. LITERATURA

[1] D. Higginbotham, Clojure for the Brave and True: Learn the Ultimate Language and Become a Better Programmer., San Francisco: No Starch Press, 2015.

[2] S. &. B. A. Halloway, Programming Clojure, 2nd Edition. Pragmatic Bookshelf, 2012..

[3] M. &. H. C. Fogus, The Joy of Clojure. Manning Publications, 2014..

[4] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, USSR: Soviet Physics Doklady, 1966..

[5] D. Wampler, Functional Programming for Java Developers, O'Reilly Media, 2011..

[6] R. Hickey, „Clojure Rationale," 2024. Available: https://clojure.org/about/rationale. [Poslednji pristup 2026.].

[7] Oracle., „Java Sound API Documentation," 2024. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/sound/. [Poslednji pristup 2026.].

[8] Oracle., „Java Platform SE 8: Class Random (java.util.Random)," 2024. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html. [Poslednji pristup 2026.].

[9] R. Hickey, „EDN (Extensible Data Notation) Specification," 2023. Available: https://github.com/edn-format/edn. [Poslednji pristup 2026.].

[10] ClojureDocs., „Clojure Core API Reference: Atoms and State Management.," 2024. Available: https://clojuredocs.org/clojure.core/atom. [Poslednji pristup 2026.].

[11] Wikipedia., „Levenshtein distance," 2024. Available: https://en.wikipedia.org/wiki/Levenshtein_distance. [Poslednji pristup 2026.].

[12] S. Sierra, „Component: Managed Lifecycle of Stateful Objects," 2018. Available: https://github.com/stuartsierra/component. [Poslednji pristup 2026.].

[13] Cognitect., „clojure.test - A unit testing framework for Clojure.," 2024. Available: https://clojure.github.io/clojure/clojure.test-api.html. [Poslednji pristup 2026.].

[14] Mojang Studios., „Minecraft Official Site - Seed System Inspiration," 2026. Available: https://www.minecraft.net/. [Poslednji pristup 2026.].

[15] Nintendo., „Super Mario Official Site - Sound Design Inspiration," 2026. Available: https://mario.nintendo.com/. [Poslednji pristup 2026.].