ELEKTROTEHNIČKI FAKULTET

BEOGRAD

**PRAKTIKUM IZ PROGRAMIRANJA 2**

ODSEK ZA SOFTVERSKO INŽENJERSTVO

Projektni zadatak broj 20

Upustvo za korisnike

|  |  |  |
| --- | --- | --- |
| Autori: | Jovan Malović 14/ | () |
|  | Ivan Blažić 14/643 | (blazic.ivan@outlook.com) |
|  | Đorđe Perčobić 14/ | () |

Beograd, jun 2015.

Uvod

* **Opis projektnog zadatka**:

Igra „2048“ se igra na kvadratnoj tabli dimenzija 4x4 ili 5x5. Polje table može da bude prazno ili da sadrži pločicu određene vrednosti. Na svakoj pločici je ispisana njena vrednost. Na početku igre postoje samo dve pločice sa vrednostima 2 ili 4 na slučajno izabranim mestima. Pritiskom na taster neke od strelica na tastaturi sve pločice na tabli se pomeraju u odgovarajućem smeru, i ukoliko se jedna pločica pomera ka drugoj, a obe imaju istu vrednost, tada se one spoje i formiraju pločicu koja ima vrednost zbira dve pločice koje su se spojile. U jednom koraku može da se desi i više spajanja. Posledica ovog pravila jeste da pločice mogu da imaju vrednost koja je samo stepen broja dva. Nakon pomeranja pločica na tablu se dodaje nova pločica sa vrednošću 2 ili 4 (verovatnoće pojavljivanja: 0.8, 0.2) na slučajno odabranom mestu. Cilj igre je da se napravi pločica sa vrednošću 2048. Igraču je na raspolaganju „hint“ dugme koje će automatski odigrati optimalan potez kao i „undo“ dugme koje vraća potez maksimalno 5 koraka u nazad. Kraj igre je ukoliko se naprvi pločica zadate vrednosti ili kada nema više raspoloživih poteza, a nije napravljena pobednička pločica. Postoje različiti režimi igre: Klasičan, X-pločica, Brzopotezni, Autoplay i Autoplay X-pločica. Kod X-pločica režima posoji jedna pločica koja ne može da se spoji ni sa jednom, a kod Autoplay režima računar sam rešava igru. Određenim „cheat“ kôdovima dobijaju se prednosti koje ne poštuju pravila igre.

* **Alat korišćen za izradu projekta:**
  + Microsoft Visual Studio 2013
* **Spisak datoteka:**
  + Grafika: graphics.c, graphics.h, pdcurses.dll, menu.c, menu.h
  + Logika, I/O: IO.c, IO.h, logic.c, logic.h
  + AI: ai.c, ai.h, ai2.c
  + Zajedničke datoteke: stats.txt, hscore.dat, main.c, savegame.dat
* **Moduli projekta:**
  + Grafika/Meni/Glavni program: Jovan Malović
  + AI: Đorđe Perčobić
  + Logika/Datoteke/Cheat: Ivan Blažić

**Korišćene strukture podataka**

|  |  |
| --- | --- |
| **Struktura** | Matrica |
| **Ime** | matrix |
| **Definisana u** | Logic.h |
| **Koristi** |  |
| **Definicija** | typedef struct matrix  {  int \*\*set;  char size;  }matrix; |
| **Manipulacija** | Kreiranje matrice - newMatrix Ubacivanje novog broja – spawnNumber  Pomeranje matrice za korak – moveStep Pomeranje matrice bez animacije i računanja poena – snap  Kopiranje matrice – copyMatrix Oslobađanje matrice – freeMatrix  Provera kraja igre - checkGameOver Provera popunjenosti – checkFull |

|  |  |
| --- | --- |
| **Struktura** | Stanje igre |
| **Ime** | state |
| **Definisana u** | Logic.h |
| **Koristi** |  |
| **Definicija** | typedef struct state  {  int \*\*set;  unsigned int score;  }state; |
| **Manipulacija** | Dohvatanje trenutnog stanja igre – getState  Brisanje zapamćenog stanja - freeState |

|  |  |
| --- | --- |
| **Struktura** | Istorija igre |
| **Ime** | history |
| **Definisana u** | Logic.h |
| **Koristi** |  |
| **Definicija** | typedef struct history  {  state \*stack;  matrix \*mat;  int depth,latest;  }history; |
| **Manipulacija** | Kreiranje nove istorije igre - newHistory Uništavanje strukture istorije igre - destroyHistory Brisanje istorije igre - clearHistory |

|  |  |
| --- | --- |
| **Struktura** | Highscore unos |
| **Ime** | entry |
| **Definisana u** | IO.h |
| **Koristi** |  |
| **Definicija** | typedef struct entry  {  char name[50];  unsigned int score;  }entry; |
| **Manipulacija** | Kreiranje novog unosa - newEntry Dodavanje unosa u highscore - addEntry |

|  |  |
| --- | --- |
| **Struktura** | Informacije o datoteci |
| **Ime** | file\_info |
| **Definisana u** | IO.h |
| **Koristi** |  |
| **Definicija** | typedef struct file\_info  {  unsigned int bit\_count, entry\_count;  }file\_info; |
| **Manipulacija** |  |

|  |  |
| --- | --- |
| **Struktura** | Čvor stabla |
| **Ime** | T\_node |
| **Definisana u** | ai.h |
| **Koristi** |  |
| **Definicija** | typedef struct tnode  {  int \*\*table;  int table\_size;  int level;  float weight, possibility;  struct tnode \*next[MAX\_NUMBER\_OF\_NODES];  } T\_node; |
| **Manipulacija** | Pravljenje cvora-get\_node  Pravljenje stabla - make\_tree\_iterative  Oslobadjanje stabla - free\_tree  Ubacivanje na stek - push  Izbacivanje sa steka - pop |

**Spisak funkcija**

Datoteka: Logic.c

|  |  |
| --- | --- |
| **Ime funkcije** | setSeed |
| **Prototip** | void setSeed(); |
| **Argumenti** | Nema argumenata |
| **Povratna vrednost** | Nema povratnu vrednost, postavlja seed za radnom broj |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | setSeed(); |
| **Ograničenja i problemi** | U sličnim vremenskim intervalima dobija se sličan seed |

|  |  |
| --- | --- |
| **Ime funkcije** | newMatrix |
| **Prototip** | matrix newMatrix(int size) |
| **Argumenti** | **size** [in]: Dimenzija nove matrice |
| **Povratna vrednost** | Nova matrica sa dva broja na slučajnim mestima. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4); |
| **Ograničenja i problemi** | Ukoliko je size 0, matrica će biti kreirana ali neće biti rezervisana memorija za nju. |

|  |  |
| --- | --- |
| **Ime funkcije** | freeMatrix |
| **Prototip** | void freeMatrix(matrix \*M) |
| **Argumenti** | **M** [in]: Pokazivač na matricu koja se oslobađa |
| **Povratna vrednost** | Nema povratne vrednosti, vrši se oslobađanje memorije korišćene za datu matricu. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  freeMatrix(m); |
| **Ograničenja i problemi** | Memorija koja se oslobađa mora prethodno biti alocirana! |

|  |  |
| --- | --- |
| **Ime funkcije** | copyMatrix |
| **Prototip** | void copyMatrix(matrix \*dest, matrix M) |
| **Argumenti** | **dest** [out]: Pokazivač na matricu u koju se smešta kopija (odredište) **M** [in]: Matrica od koje se pravi kopija (izvor) |
| **Povratna vrednost** | Nema povratne vrednosti, vrši se kopiranje celog sadržaja jedne matrice u drugu. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  matrix N; copyMatrix(&N, M); |
| **Ograničenja i problemi** | Obe strukture matrice moraju biti kreirane. Matrica dest mora da bude prazna struktura. |

|  |  |
| --- | --- |
| **Ime funkcije** | spawnNumber |
| **Prototip** | void spawnNumber(matrix \*M) |
| **Argumenti** | **M** [in]: Pokazivač na matricu u koju se dodaje broj. |
| **Povratna vrednost** | Nema povratne vrednosti, dodaje se broj 2 ili 4 na nasumično prazno mesto u matrici. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  spawnNumber(&M); |
| **Ograničenja i problemi** | Ne treba pozivati funkciju ako je matrica puna, tada se ništa neće izmeniti. |

|  |  |
| --- | --- |
| **Ime funkcije** | randomInt |
| **Prototip** | int randomInt(int low, int high) |
| **Argumenti** | **low** [in]: Donja granica  **high** [in]: Gornja granica |
| **Povratna vrednost** | Vraća random ceo broj u opsegu od low do high, uključujući i high. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | int r = randomInt(0,10); |
| **Ograničenja i problemi** | Pre poziva ove funkcije treba pozvati setSeed |

|  |  |
| --- | --- |
| **Ime funkcije** | moveStep |
| **Prototip** | int moveStep(matrix \*M, int direction, int \*last\_merged, unsigned int \*score) |
| **Argumenti** | **M** [in]: Matrica u kojoj se vrši pomeranje **direction** [in]: Smer u kojem se vrši pomeranje (koriste se definisane konstante za smer iz hedera). **last\_merged** [out]: Vektor koji za svaku vrstu/kolonu pamti poslednji broj koji je sabran.  **score** [out]: Pokazivač na promenljivu u koju se smešta score |
| **Povratna vrednost** | Vraća broj promena u matrici i pomera matricu za jedan korak. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | int changes, moved, last\_merged[5] = { 0 };  changes = moved = moveStep(M, direction, last\_merged, score);  while (changes)  {  changes = moveStep(M, direction, last\_merged, score);  \_sleep(75);  displayMatrix(0, -1, \*M);  } |
| **Ograničenja i problemi** | Nema poznatih problema ili ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | snap |
| **Prototip** | int snap(unsigned int \*\*table, int table\_size, int direction, matrix \*M) |
| **Argumenti** | **M** [in]: Matrica na osnovu koje se procenjuje potez **direction** [in]: Smer u kojem se vrši pomeranje (koriste se definisane konstante za smer iz hedera). **table** [out]: Pokazivač na sadržaj kopije matrice.  **table\_size** [in]: Veličina kopirane matrice. |
| **Povratna vrednost** | Vraća 1 ako je bilo promena u matrici, 0 u suprotnom i daje pregled na potencijalni potez u datom smeru. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | Matrix M = newMatrix(4);  int c = snap(M.set,M.size,LEFT,&M); |
| **Ograničenja i problemi** | Nema poznatih problema ili ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | swipeSpeed |
| **Prototip** | void swipeSpeed(matrix \*M, int direction, unsigned int \*score) |
| **Argumenti** | **M** [in]: Matrica u kojoj se vrši pomeranje **direction** [in]: Smer u kojem se vrši pomeranje (koriste se definisane konstante za smer iz hedera).  **score** [out]: Pokazivač na promenljivu u koju se smešta score |
| **Povratna vrednost** | Nema povratnu vrednost, igra potez u brzopoteznom režimu. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | case KEY\_LEFT:  swipeSpeed(m, LEFT, &score);  break; |
| **Ograničenja i problemi** | Nema poznatih problema ili ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | swipeNoAnimation |
| **Prototip** | void swipeNoAnimation(matrix \*M, int direction, unsigned int \*score) |
| **Argumenti** | **M** [in]: Matrica u kojoj se vrši pomeranje **direction** [in]: Smer u kojem se vrši pomeranje (koriste se definisane konstante za smer iz hedera).  **score** [out]: Pokazivač na promenljivu u koju se smešta score |
| **Povratna vrednost** | Nema povratnu vrednost, igra potez bez animacije i postavlja dva broja. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | case KEY\_LEFT:  swipeSpeed(m, LEFT, &score);  break; |
| **Ograničenja i problemi** | Nema poznatih problema ili ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | checkFull |
| **Prototip** | int checkFull(matrix M) |
| **Argumenti** | **M** [in]: Struktura matrice koja se trenutno koristi u igri. |
| **Povratna vrednost** | Vraća 1 ako su sva polja matrice popunjena (različita od 0), u suprotnom vraća 0. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  int full = checkFull(M); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | newHistory |
| **Prototip** | history newHistory(short undo\_depth, matrix \*M) |
| **Argumenti** | **undo\_depth** [in]: Dozvoljeni broj vraćanja poteza u nazad (undo dubina)  **M** [in]: Matica sa kojom se pamti istorija igre |
| **Povratna vrednost** | Vraća novu strukturu istorije igre. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | Matrix M = newMatrix(4); History H = newHistory(20,&M); |
| **Ograničenja i problemi** | Ukoliko je undo\_depth 0, memorija neće biti alocirana i kreirana struktura će biti prazna. |

|  |  |
| --- | --- |
| **Ime funkcije** | clearHistory |
| **Prototip** | void clearHistory(history \*H) |
| **Argumenti** | **H** [in]: Pokazivač na strukturu istorije u kojoj se briše pamćen sadržaj. |
| **Povratna vrednost** | Nema povratne vrednosti, briše se istorija iz date strukture. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | History H = newHistory(20,&M);  clearHistory(&H); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | destroyHistory |
| **Prototip** | void destroyHistory(history \*H) |
| **Argumenti** | **H** [in]: Pokazivač na strukturu istorije koja se briše. |
| **Povratna vrednost** | Nema povratne vrednosti, oslobađa se memorija zazueta strukturom. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | History H = newHistory(20,&M);  destroyHistory(&H); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | popHistory |
| **Prototip** | void popHistory(history \*H, unsigned int \*score) |
| **Argumenti** | **H** [in]: Pokazivač na strukturu istorije u kojoj se čuvaju potezi.  **score** [out]: Pokazivač na promenljivu koja čuva score. |
| **Povratna vrednost** | Nema povratne vrednosti, briše se poslednje unet unos iz istorjie i matrica se vraća na prethodno stanje. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | popHistory(&hist, &score); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | pushHistory |
| **Prototip** | void pushHistory(history \*H, state S) |
| **Argumenti** | **H** [out]: Pokazivač na strukturu istorije u kojoj se čuvaju potezi.  **state** [in]: Struktura koja pamti trenutno stanje igre. |
| **Povratna vrednost** | Nema povratne vrednosti, u istoriju se upisuje trenutni potez. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | valid\_move = swipe(m, DOWN, &score);  if (valid\_move)  pushHistory(&hist, previous); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | freeState |
| **Prototip** | void freeState(state \*S, unsigned int tile\_size) |
| **Argumenti** | **S** [in]: Struktura stanja koja se oslobađa.  **tile\_size** [in]: Veličina matrice čije je stanje pamćeno. |
| **Povratna vrednost** | Nema povratne vrednosti, oslobđa se memorija zauzeta matricom. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  int score = 0;  state S = newState(M,score);  freeState(&S,M.size); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | freeMatrix |
| **Prototip** | void freeMatrix(matrix \*M) |
| **Argumenti** | **M** [in]: Pokazivač na strukturu matrice koja se oslobađa. |
| **Povratna vrednost** | Nema povratne vrednosti, oslobađa se memorija zauzeta strukturom. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  freeMatrix(&M); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | freeSet |
| **Prototip** | void freeSet(int \*\*set, int size) |
| **Argumenti** | **set** [in]: Pokazivač na sadržaj matrice koja se oslobađa.  **size** [in]: Veličina matrice čiji se sadržaj oslobađa. |
| **Povratna vrednost** | Nema povratne vrednosti, oslobađa se memorija zauzeta sadržajem matrice. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  freeSet(M.set,M.size); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | checkGameOver |
| **Prototip** | int checkGameOver(matrix M) |
| **Argumenti** | **M** [in]: Struktura matrice koja se trenutno koristi u igri. |
| **Povratna vrednost** | Vraća 1 ako igrač nema više mogućih poteza, u suprotnom vraća 0. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  int game\_over = checkGameOver(M); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | swipe |
| **Prototip** | int swipe(matrix \*M, int direction, unsigned int \*score) |
| **Argumenti** | **M** [in]: Matrica u kojoj se vrši pomeranje **direction** [in]: Smer u kojem se vrši pomeranje (koriste se definisane konstante za smer iz hedera).  **score** [out]: Pokazivač na promenljivu u koju se smešta score |
| **Povratna vrednost** | Igra potez sa datom matricom, postavlja dva broja i vraća 1 ako je potez validan. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | case KEY\_LEFT:  valid\_move = swipe(m, LEFT, &score);  if (valid\_move)  pushHistory(&hist, previous);  previous = getState(\*m, score);  break; |
| **Ograničenja i problemi** | Nema poznatih problema ili ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | copySet |
| **Prototip** | int \*\*copySet(matrix M) |
| **Argumenti** | **M** [in]: Struktura matrice čiji se sadržaj kopira. |
| **Povratna vrednost** | Nema povratne vrednosti, oslobađa se memorija zauzeta sadržajem matrice. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  freeSet(M.set,M.size); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

|  |  |
| --- | --- |
| **Ime funkcije** | getState |
| **Prototip** | state getState(matrix M, unsigned int score) |
| **Argumenti** | **M** [in]: Struktura matrice koja se trenutno koristi u igri.  **score** [in]: Trenutan score. |
| **Povratna vrednost** | Vraća strukturu trenutnog stanja igre. |
| **Zahteva** | Logic.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  int score = 0;  state S = newState(M,score); |
| **Ograničenja i problemi** | Nema poznatih ograničenja. |

Datoteka: IO.c

|  |  |
| --- | --- |
| **Ime funkcije** | newEntry |
| **Prototip** | entry newEntry(char \*player\_name, unsigned int score) |
| **Argumenti** | **player\_name** [in]: Ime igrača  **score** [in]: Score do kojeg je igrač dospeo |
| **Povratna vrednost** | Vraća novu strukturu highscore unosa. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | char name[] = “Novi igrac“;  int score = 2048;  entry new\_player = newEntry(name,score); |
| **Ograničenja i problemi** | Ime igrača mora biti do 50 karaktera. |

|  |  |
| --- | --- |
| **Ime funkcije** | addEntry |
| **Prototip** | void addEntry(entry \*\*score\_list, unsigned int \*entry\_count, entry newScore) |
| **Argumenti** | **score\_list** [in]: Pokazivač na niz struktura tipa entry u kojem se čuva highscore lista.  **entry\_count** [in]: Pokazivač na promenljivu u kojoj se čuva broj unosa.  **newScore** [in]: Struktura unosa koja se dodaje u niz. |
| **Povratna vrednost** | Nema povratne vredosti, u niz struktura se dodaje nova i održava se poredak. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | entry \*score\_list = NULL;  int count = 0;  entry new\_player = newEntry(name,score);  addEntry(&score\_list,&count,new\_player); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | saveHsc |
| **Prototip** | void saveHsc(entry \*score\_list, unsigned int entry\_count) |
| **Argumenti** | **score\_list** [in]: Niz struktura tipa entry u kojem se čuva highscore lista.  **entry\_count** [in]: Promenljiva u kojoj se čuva broj unosa. |
| **Povratna vrednost** | Nema povratne vredosti, iz niza struktura se pravi ili ažurira highscore datoteka. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | entry \*score\_list = NULL;  int count = 0;  entry new\_player = newEntry(name,score);  addEntry(&score\_list,&count,new\_player);  saveHsc(score\_list,count); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | loadHsc |
| **Prototip** | entry \*loadHsc(unsigned int \*entry\_conunt, unsigned int \*bit\_check) |
| **Argumenti** | **entry\_count** [out]: Pokazivač na promenljivu u kojoj se čuva broj unosa.  **bit\_check** [out]: Proverava da li je datoteka menjana van programa (vraća 1 ako nije, a 0 ako jeste) |
| **Povratna vrednost** | Vraća pokazivač na početak niza highscore liste učitane iz datoteke, ako datoteka ne postoji ili ne može da se otvori vraća NULL. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | int count,check;  entry \*score\_list = loadHsc(&count,&check); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | saveGame |
| **Prototip** | void saveGame(matrix M, unsigned int score, unsigned int mode) |
| **Argumenti** | **M** [in]: Struktura matrice koja se trenutno koristi u igri.  **score** [in]: Trenutno osvojen score.  **mode** [in]: Režim igre. |
| **Povratna vrednost** | Nema povratnu vrednost, čuva trenutno stanje igre i score u datoteku. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | int score = 0;  matrix M = newMatrix(4);  saveGame(M,score); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | loadGame |
| **Prototip** | int loadGame(matrix \*Mp, unsigned int \*score, unsigned int mode, unsigned int new\_size) |
| **Argumenti** | **Mp** [out]: Pokazivač na matricu za koji se vezuje učitana ili nova matrica.  **score** [out]: Score osvojen u sačuvanoj igri.  **mode** [out]: Režim sačuvane igre.  **new\_size** [in]: Veličina nove matrice ukoliko datoteka ne postoji |
| **Povratna vrednost** | 0 - Ne postoji datoteka  1 - Postoji, ali je izmenjena van programa  2 - Postoji, nije izmenjena van programa |
| **Zahteva** | IO.h |
| **Primer korišćenja** | int score = 0, check;  matrix M = loadGame(&score,&check); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | checkMemError |
| **Prototip** | void checkMemError(void \*new\_pointer) |
| **Argumenti** | **new\_pointer** [in]: Pokazivač za koji se vrši provera. |
| **Povratna vrednost** | Nema povratnu vrednost, proverava ispravnost pokazivača i po potrebi izbacuje grešku. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | int \*p = malloc(sizeof(int));  checkMemError(p); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | checkFileError |
| **Prototip** | void checkFileError(FILE \*file\_pointer) |
| **Argumenti** | **file\_pointer** [in]: Pokazivač na datoteku nad kojim se vrši provera. |
| **Povratna vrednost** | Nema povratnu vrednost, proverava ispravnost pokazivača i po potrebi izbacuje grešku. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | FILE \*f = fopen(“file.dat“,“rb“);  checkFileError(p); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | writeAIstats |
| **Prototip** | void writeAIstats(matrix M) |
| **Argumenti** | **M** [in]: Matrica nakon odigranog Autoplay-a. |
| **Povratna vrednost** | Nema povratnu vrednost, upisuje podatke o AI statistici. |
| **Zahteva** | IO.h |
| **Primer korišćenja** | matrix M = newMatrix(4);  //Odigra se Autoplay writeAIstats(M); |
| **Ograničenja i problemi** | Nema poznatih problema. |

|  |  |
| --- | --- |
| **Ime funkcije** | findCode |
| **Prototip** | int findCode(char \*cheats[], char \*buffer, int prev\_code) |
| **Argumenti** | **cheats** [in]: Niz cheat stringova  **buffer** [in]: String koji je za sada unet.  **prev\_code** [in]: Kôd očekivanog cheat-a |
| **Povratna vrednost** | -1: Ne postoji string ili se ne poklapa ni sa jednim cheat stringom  -2: string se poklopio, treba da se izvrsi cheat sa indeksom prev\_code  0 ili vise: string se za sada poklapa sa ocekivanim cheat stringom |
| **Zahteva** | IO.h |
| **Primer korišćenja** | //c – Tekuci karakter  strcat(buffer, &c);  code = findCode(cheats, buffer, prev\_code); |
| **Ograničenja i problemi** | Nema poznatih problema. |

Datoteka: ai.c

|  |  |
| --- | --- |
| **Ime funkcije** | get\_node |
| **Prototip** | T\_node\* get\_node(int \*\*table, int table\_size, int level) |
| **Argumenti** | table [*in*]: kvadratna matrica, table\_size [in] - broj redova/kolona matrice, level [in] - nivo stabla |
| **Povratna vrednost** | cvor stabla |
| **Zahteva** | ai.h |
| **Primer korišćenja** | int \*\*matrix;  matrix = malloc(4\*sizeof(int\*);  int table\_size = 4;  for (i = 0; i < table\_size; i++)  matrix[i] = calloc(table\_size, sizeof(int));  int level = 0;  T\_node \*cvor;  cvor = get\_node(matrix, table\_size, level); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | free\_tree |
| **Prototip** | void free\_tree(T\_node \*root, T\_node \*\*\*stack, int \*stack\_space) |
| **Argumenti** | root [in]: koreni cvor, stack[*in*]: adresa pokazivaca na niz cvorova |
| **Zahteva** | ai.h |
| **Primer korišćenja** | T\_node \*root = NULL;  int stack\_size = 100;  T\_node \*\*stack = malloc(stack\_size\*sizeof(T\_node\*));  free\_tree(root, &stack, stack\_size); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | pop |
| **Prototip** | T\_node\* pop(T\_node \*\*stack, int \*top) |
| **Argumenti** | stack[in] - pokazivac na niz cvorova, top[in] - adresa vrha steka |
| **Povratna vrednost** | cvor sa vrha steka |
| **Zahteva** | ai.h |
| **Primer korišćenja** | T\_node \*elem;  elem = pop(stack, &top); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | push |
| **Prototip** | void push(T\_node \*\*\*stack, T\_node \*elem, int \*top, int \*stack\_space) |
| **Argumenti** | stack [*in*]: adresa pokazivaca na niz cvorova, elem [in] element koji se stavlja na stek, top [in]: adresa vrha steka, stack\_space [in]: adresa velicine alociranog prostora za stek. |
| **Povratna vrednost** |  |
| **Zahteva** | ai.h |
| **Primer korišćenja** | T\_node \*root;  int top = 0;  root = NULL;  int stack\_size = 100;  T\_node \*\*stack = malloc(sizeof(T\_node\*)\*stack\_size);  push(&stack, root, &top,&stack\_size); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | make\_tree\_iterative |
| **Prototip** | void make\_tree\_iterative(T\_node \*root, T\_node \*\*\*stack, int \*stack\_space) |
| **Argumenti** | root[in/out] - koreni cvor, stack[in] - adresa na pokazivac na niz cvorova, stack\_space[in] - adresa broja alociranog prostora za stack |
| **Povratna vrednost** |  |
| **Zahteva** | ai.h |
| **Primer korišćenja** | int level = 0;  int table\_size = 4;  int \*\*table = malloc(table\_size\*sizeof(int\*);  int i;  for (i = 0; i < table\_size; i++)  table[i] = calloc(table\_size, sizeof(int));  table[0][0] = 2;  T\_node \*root = get\_node(table, table\_size, level); |
| **Ograničenja i problemi** | Maksimalna velicina kvadratne matrice je 5. Testirana samo za velicine 4 i 5. |

|  |  |
| --- | --- |
| **Ime funkcije** | get\_hint |
| **Prototip** | int get\_hint(matrix table) |
| **Argumenti** | table - struktura koja sadrzi broj kolona/vrsta i matricu |
| **Povratna vrednost** | potez koji treba da se odigra(definisan u logic.h), broj 4 ako nema vise poteza |
| **Zahteva** | ai.h, logic.h |
| **Primer korišćenja** | matrix matrica;  int table\_size;  int \*\*table = malloc(table\_size\*sizeof(int\*);  int i;  for (i = 0; i < table\_size; i++)  table[i] = calloc(table\_size, sizeof(int));  table[0] = 2;  matrica->set = table;  matrica->size = table\_size;  get\_hint(matrica); |

Datoteka: ai2.c

|  |  |
| --- | --- |
| **Ime funkcije** | expectimax\_search |
| **Prototip** | void expectimax\_search(T\_node \*root, T\_node \*\*\*stack, int \*stack\_space) |
| **Argumenti** | root - koreni cvor stabla, stack - adresa pokazivaca na niz cvorova, stack\_space - broj alociranih elemenata stack-a |
| **Povratna vrednost** |  |
| **Zahteva** | ai.h |
| **Primer korišćenja** | int table\_size = 4;  int \*\*matrica = malloc(table\_size\*sizeof(int\*));  int i = 0;  for (i = 0; i < table\_size; i++)  matrica[i] = calloc(table\_size, sizeof(int));  matrica[0][0] = 2;  T\_node \*root = get\_node(matrica, table\_size);  make\_tree\_iterative(root, table\_size, stack, stack\_size);  expectimax\_search(root, &stack, &stack\_size); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | approximate\_position |
| **Prototip** | float approximate\_position(int \*\*table, int table\_size) |
| **Argumenti** | table[in] - matrica igre, table\_size[in] broj kolona/redova matrice |
| **Povratna vrednost** | ocjena pozicije |
| **Primer korišćenja** | int table\_size = 4;  int \*\*matrica = malloc(table\_size\*sizeof(int\*));  int i = 0;  for (i = 0; i < table\_size; i++)  matrica[i] = calloc(table\_size, sizeof(int));  matrica[0][0] = 2;  float weight = approximate\_position(matrica, table\_size); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | number\_of\_moves\_horizontally |
| **Prototip** | void number\_of\_moves\_horizontally(int \*\*table, int table\_size, float \*score) |
| **Argumenti** | table[in] - matrica, table\_size[in] - broj kolona/redova matrice, score[in/out] - adresa rezultata |
| **Povratna vrednost** |  |
| **Zahteva** |  |
| **Primer korišćenja** | int table\_size = 4;  int \*\*matrica = malloc(table\_size\*sizeof(int\*));  int i = 0;  for (i = 0; i < table\_size; i++)  matrica[i] = calloc(table\_size, sizeof(int));  matrica[0][0] = 2;  float score = 0;  number\_of\_moves\_horizontally(matrica, table\_size, &score); |
| **Ograničenja i problemi** | Nema poznatih problema |

|  |  |
| --- | --- |
| **Ime funkcije** | number\_of\_moves\_vertically |
| **Prototip** | void number\_of\_moves\_vertically(int \*\*table, int table\_size, float \*score) |
| **Argumenti** | table[in] - matrica igre, table\_size[in] - broj kolona/redova matrice, score[in/out] - adresa rezultata |
| **Povratna vrednost** |  |
| **Zahteva** |  |
| **Primer korišćenja** | int table\_size = 4;  int \*\*matrica = malloc(table\_size\*sizeof(int\*));  int i = 0;  for (i = 0; i < table\_size; i++)  matrica[i] = calloc(table\_size, sizeof(int));  matrica[0][0] = 2;  float score = 0;  number\_of\_moves\_vertically(matrica, table\_size, &score); |
| **Ograničenja i problemi** | Nema poznatih problema |