SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 340

ALGORITMI ZA ISPITIVANJE RJEŠIVOSTI DIGITALNE IGRE ZA PROIZVOLJNO ZADANU POČETNU SITUACIJU

Adam Kolar

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 340

ALGORITMI ZA ISPITIVANJE RJEŠIVOSTI DIGITALNE IGRE ZA PROIZVOLJNO ZADANU POČETNU SITUACIJU

Adam Kolar

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

ZAVRŠNI ZADATAK br. 340

Pristupnik:	**Adam Kolar (0036516309)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Maja Matijašević

Zadatak: **Algoritmi za ispitivanje rješivosti digitalne igre za proizvoljno zadanu početnu situaciju**

Opis zadatka:

Zadana je digitalna igra na ploči sa 16 (4 x 4) polja. Početna situacija na ploči proizvoljno se zadaje putem korisničkog sučelja, pri čemu se svakom polju može dodijeliti vrijednost (prikazana kao objekt koji nosi neki broj "bodova") te funkcija koja, po dolasku igrača na to polje, mijenja vrijednost polja te potencijalno ograničava buduće kretanje igrača. Nakon što je ploča zadana, igrač se kreće po ploči skupljajući bodove po poljima kojima prolazi. Vaš zadatak je proučiti postupke konstruiranja i algoritme pretraživanja stabla igre te ih primijeniti na zadanu igru, kako biste utvrdili je li igra za proizvoljno zadanu početnu situaciju na ploči rješiva. Za potrebe ovoga rada, igra se smatra "rješivom" ako postoji niz koraka kojima (pretpostavljeni, ljudski) igrač može ostvariti cilj igre, a koji može biti zadan kao: a) posjećivanje svih polja na ploči, ili, b) prikupljanje najvećeg postizivog broja bodova, neovisno o broju posjećenih polja. Za način b) dodatno treba odrediti i koji je najveći postizivi broj bodova. U osnovnoj inačici igre razmotrite sljedeće postavke: početna vrijednost (broj bodova) pojedinog polja može biti 0, 1 ili 2, a funkcija izmjene vrijednosti polja s dolaskom igrača na polje smanjuje vrijednost za 1; ako je vrijednost polja nula, polje je ili postaje "zabranjeno", tj. igrač više ne može stupiti na njega. U naprednoj inačici igre predložite drugačiju(e) funkciju(e) izmjene vrijednosti polja (primjerice, pad i rast vrijednosti, "teleportiranje" i slično). Usporedite broj stanja i složenost pretraživanja za različite inačice igre.

Rok za predaju rada: 11. lipnja 2021.

Sadržaj

Uvod

U današnjem svijetu, digitalne igre su izrazito popularne. Kod nekih igara, postoje dijelovi igre, tzv. razine (engl. Levels) te je ponekad te razine moguće „odigrati“ tj. riješiti računalom. To je poželjno pogotovo ako su razine kompliciranе tj. njihovo testiranje (rješavanje) je repetitivan posao koji računala mogu obavljati umjesto ljudi.

Cilj ovoga rada je proširiti postojeću igru „ZR 2“ s mogućnošću rješavanja njenih razina računalom. To se postiže uporabom različitih algoritama koji su i glavna tema ovog rada.

Rad se sastoji od 5 poglavlja u kojima se detaljnije opisuje problem, dizajn i razvoj igre te sami algoritmi i njihovi rezultati.

1. Specifikacija problema

Cilj ovog rada bio je proširiti osnovnu inačicu vlastite (autorove) igre „ZR 2“ s mogućnošću algoritamske provjere „rješivosti igre“. U nastavku je opisana ideja igre, njena početna implementacija i definiran problem rješivosti igre.

1.1. Igra

Igra je aplikacija/program koji se izvodi na računalu, odvija se na ploči (Level) koja predstavlja dvodimenzionalno polje ($m \times n$) brojeva. Originalna ideja igre uključuje i višedimenzionalne „ploče“. Ploča se sastoji od polja te se može reprezentirati tablicom. Igrač počinje na određenom polju te u potezima u vodoravnom ili okomitom smjeru posjećuje susjedna polja čija je brojčana vrijednost veća od nule. Posjećivanjem polja umanjuje se njegova vrijednost, a povećava rezultat igrača. Cilj igre je posjetiti sva ili što više moguće polja za što veći rezultat. Tablica 1.1 prikazuje primjer ploče 3x3 sa zadanim vrijednostima polja i označenim početnim položajem igrača u žutoj boji.

0	1	1
1	2	2
1	1	0

Tablica 1.1 Prikaz 3x3 ploče sa igračem u gornjem lijevom polju

Slika 1.1 prikazuje izgled igraće ploče unutar aplikacije za ploču zadanu tablicom 1.1. Sam prikaz polja u igri nisu brojevi, već kockice naslagane jedna na drugoj ovisno o njenom broju. Posjećivanjem polja, njihova se vrijednost umanjuje za jedan, no moguće je imati i polja koja pri posjeti prebacuju igrača na neko drugo polje i/ili umanjuju vrijednosti drugih polja. Igrač je predstavljen žutom kockicom koja stoji na vrhu stoga kockica na polju u kojem se nalazi.

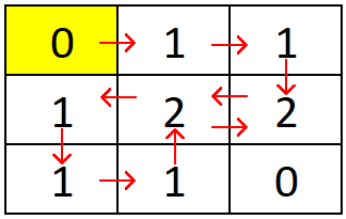
Slika 1.1 Prikaz polja u igri temeljenog na tablici 1.1

Ploča se može zadati „ručnim“ uređivanjem datoteke u kojoj je pohranjena ili s pomoću alata Level Editor. Level Editor u početnoj inačici igre nudi samo uređivanje ploča, bez ikakve druge provjere.

1.2. Ispitivanje rješivosti ploče

Za potrebe ovog rada, rješivost ploče je pitanje koje poteze poduzeti da bi se suma svih vrijednosti u ploči (suma ploče) smanjila na najmanji moguću broj, idealno nula. Rješenje (ploče) je put po ploči kojim se dolazi do te sume.

Slika 1.2 prikazuje jedno od rješenja ploče sa slike 1.1.



Slika 1.2 Rješenje ploče iz tablice 1.1

Pošto je ploča diskretna, te broj stanja za svaku ploču konačan, rješivost ploče u igri se može ispitati pomoću raznih algoritama za pretraživanje. Cilj algoritama je pronaći rješenje ploče. Pri samom posjećivanju polja stvara se stablo pretraživanja koje je konačno i nema ciklusa.

2. Dizajn igre

Dizajn igre obuhvaća podatkovne strukture te specifikaciju i razradu njenih komponenti.

2.1. Zapis podataka o ploči i poljima

Za spremanje podataka o ploči koristi se generičko višedimenzionalno polje koje je interno realizirano kao jednodimenzionalno polje. Elementima se pristupa na dva načina: preko višedimenzionalnih koordinata i indeksa elementa u internom jednodimenzionalnom polju. Za svako takvo polje prvo potrebno je definirati dimenzije D, formula (1):

$$D = (d_1, d_2, d_3, \dots, d_N) \quad d_{i\in\{1,2,\dots,N\}}, N \in \mathbb{N} \tag{1}$$

Veličina jednodimenzionalnog polja L definiranja je kao produkt svih dimenzija formulom (2):

$$L = \prod_{i=1}^{N} d_i \tag{2}$$

Višedimenzionalne koordinate C definirane su formulom (3):

$$C = (c_1, c_2, c_3, \dots, c_N) \quad c_{i\in\{1,2,\dots,N\}} \in \{0, 1, \dots, d_i - 1\} \tag{3}$$

Jednodimenzionalne koordinate (indeks elemenata) J definirane su formulom (4):

$$J \in \{0, 1, 2, \dots, L - 1\} \tag{4}$$

Iz navedenih izraza se može vidjeti da indeksiranje počinje od nule.

Formule za pretvorbu iz jednog oblika koordinata u drugi definirane su kao funkcije:

Jednodimenzionalni u višedimenzionalni oblik:

$$f_{J\to C}(J) = (J \,\%\, d_1, \left\lfloor \frac{J}{d1} \right\rfloor \,\%\, d_2, \left\lfloor \frac{J}{d1\times d2} \right\rfloor \,\%\, d_3, \dots, \left\lfloor \frac{J}{\prod_{i=1}^{N-1} d_i} \right\rfloor \,\%\, d_N) \tag{5}$$

Gdje operator % predstavlja operaciju modulo.

Višedimenzionalni u jednodimenzionalni oblik:

$$f_{C\to J}(C) = c_1 \times d_0 + c_2 \times (d_0 \times d_1) + c_3 \times (d_0 \times d_1 \times d_2) + \cdots + c_N \times \prod_{i=0}^{N-1} d_i \tag{6}$$

Ili kraći zapis: $f_{C\to J}(C) = \sum_{i=1}^{N}(c_i \times \prod_{j=0}^{i-1} d_j)$ (7)

Za formule (6) i (7) vrijedi: $d_0 = 1$

Primjeri mapiranja tih funkcija su sljedeći:

D = (3,3) = 3x3 – 2D polje

Jednodimenzionalne Dvodimenzionalne

6	7	8
3	4	5
0	1	2

(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)
(0,0)	(1,0)	(2,0)

Tablica 2-1 Primjer indeksiranja 2D polja

D = (2,2,3) = 2x2x3 – 3D polje

Jednodimenzionalne Trodimenzionalne

c_3=0		c_3=1		c_3=2	
2	3	6	7	10	11
0	1	4	5	8	9

(0,1,0)	(1,1,0)	(0,1,1)	(1,1,1)	(0,1,2)	(1,1,2)
(0,0,0)	(1,0,0)	(0,0,1)	(1,0,1)	(0,0,2)	(1,0,2)

Tablica 2-2 Primjer indeksiranja 3D tablice

Ovakav zapis, općenito, podržava i polja s dimenzijom većom od 2.

Zapis podataka o poljima

Polja mogu biti definirana s 3 tipa.

Tipovi polja

- Default
  - Predodređen tip svakog polja, omogućuje da se polje umanji za jedan pri posjeti igrača
- Teleporter
  - Teleportira igrača na određeno mjesto pri posjeti polja
- Increaser
  - Umanjuje vrijedosti određenih polja za određenu vrijednost u određenoj grupi

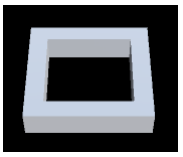
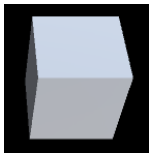
Svako polje u svom zapisu sadrži sljedeće podatke: tip polja, vrijednost polja, lokacija teleportiranja (za Teleporter), dodatak novih vrijednosti (za Increaser), boja polja, indeks ciljne grupe polja (za Increaser, grupe polja opisane su u sljedećem potpoglavlju). Sve navedene vrijednosti su cijeli brojevi (veličine 4 bajta) i svi osim tipa polja se zapisuju kao cijeli brojevi u bazi 10. Tip polja se zapisuje u bazi 2.

Korištene vrijednosti bitova za tip polja su:

1. bit	Default (uvijek 1)
2. bit	Ne koristi se
3. bit	Teleporter
4. bit	Ne koristi se
5. bit	Increaser
ostali	Ne čitaju se

Tablica 2-3 Bitovi za tip polja

Vrijednosti polja se za brojeve manje ili jednake nuli prikazuju kao „prazno“ polje. Za pozitivne vrijednosti, polje dobiva onoliko kockica kolika je njegova vrijednost, naslaganih jedna na drugu.

Vrijednosti polja:	≤ 0	1	2
Prikaz polja:			

Tablica 2-4 Prikaz vrijednosti polja

U datoteci se polje zapisuje na sljedeći način: (tip, vrijednost, lokacija, dodatak, boja, grupa), Primjer je sljedeći: (1001, 1, 0, -1, 3, 0)

Polja tipa Teleporter i Increaser funkcioniraju ako i samo ako su ispravno zadana i njihova brojčana vrijednost prije posjećivanja veća od nula.

Zapis ploče u datoteci

Ploča se u datoteci zapisuje kao obični tekst (UTF-8) u sljedećem formatu:

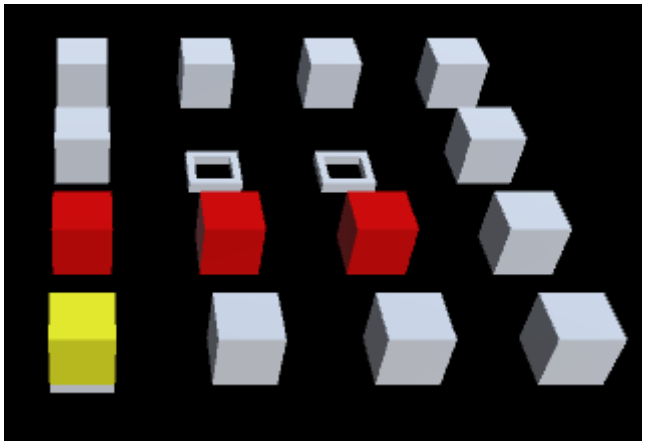
```
1: name - string
2: Target=TargetSum
3: type - {SumToZero 1 ReachPoints}
4: player position: p1 p2 p3 ...
5: cellGroupCount - int
   cellGroup0 - array in {}
   ...
   cellGroupN-1 - array in {}
   dimensions: d1 d2 d3 ...
   (cellData1) (cellData2) ...
```

Slika 2.1 Format zapisa ploče u datoteci

U prvom retku nalazi se ime ploče. Drugi redak sadrži broj koliko je polja potrebno posjetiti ili umanjiti da ploča bude riješena. U trećem retku je tip ploče što je uvijek SumToZero, ReachPoints se ne koristi. Četvrti redak se sastoji od niza brojeva koji reprezentiraju višedimenzionalnu lokaciju igrača. Peti redak sadrži cijeli broj koji označava broj grupa polja na ploči. Ovisno o vrijednosti četvrtog retka, neka to bude broj N, slijedi N redaka sa zapisom indeksa polja između vitičastih zagrada odvojenih zarezom. Ti zapisi označavaju grupe polja, brojevi u vitičastim zagradama označavaju jednodimenzionalne lokacije tih polja. Nakon toga slijedi redak koji definira dimenzije ploče i na kraju se nalaze zapisi podataka polja u proizvoljnom broju redaka. Lokacija polja na ploči je njegov redni broj u zapisu oduzet sa 1. Slika 2.2 prikazuje zapis ploče u datoteci, a slika 2.3 prikaz iste ploče u igri.

```
44ti2
Target=13
SumToZero
(0, 0)
1
{8, 9, 10, 11}
4 4
(1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
(1001, 1, 0, -1, 3, 0) (1001, 1, 0, -1, 3, 0) (1001, 1, 0, -1, 3, 0) (1, 1, 0, 0, 0, 0)
(1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
(1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
```

Slika 2.2 Primjer zapisa ploče u datoteci



Slika 2.3 Izgled ploče sa slike 2.2

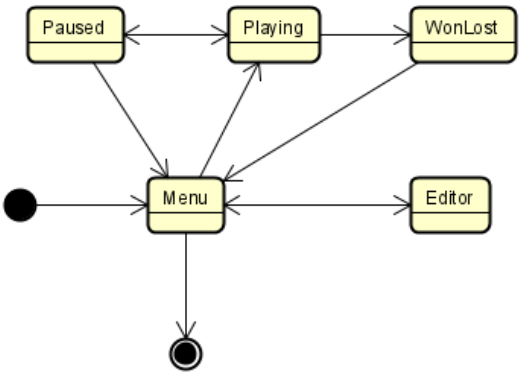
2.2. Načini rada aplikacije

Za kontrolu određenih komponenti igre koriste se automati stanja. Sama igra se sastoji od određenog broja stanja u kojima može biti.

Stanja aplikacije

- Playing
    - Stanje kada igrač igra igru
- Paused
    - Stanje kada je igra privremeno zaustavljena
- Menu
    - Početno stanje igre, početni meni
- Editor
    - Uređivanje ili izrada ploča za igranje
- WonLost
    - Stanje kada je igranje nad pločom gotovo

Ulazak u finalno stanje je izlaz iz igre. Povezanost navedenih stanja je ilustrirana slikom 2.4.



Slika 2.4 Povezanost stanja aplikacije

Bitan dio interakcije korisnika s aplikacijom čini korisničko sučelje, pri čemu se za kontrolu korisničkog menija koriste se stanja.

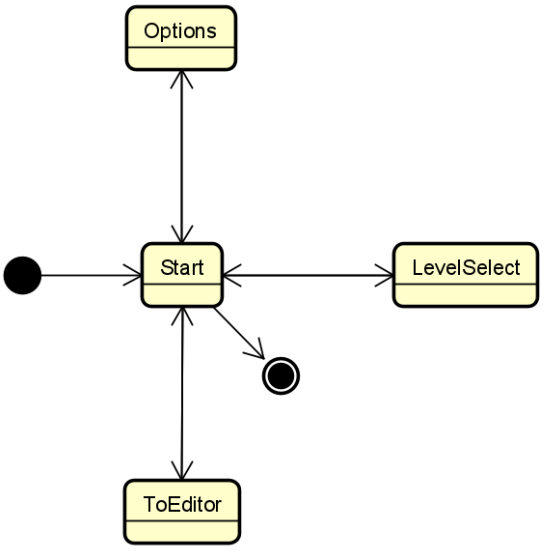
Stanja Menija

- Start

    - Početni meni, ujedno i početno stanje

- LevelSelect
    - Odabir ploča za igru
- Options
    - Opcije igre
- ToEditor
    - Pokretanje alata Level Editor

Ulazak u finalno stanje reprezentira izlazak iz glavnog menija igre. Navedena stanja su povezana na sljedeći način:



Slika 2.5 Povezanost stanja menija

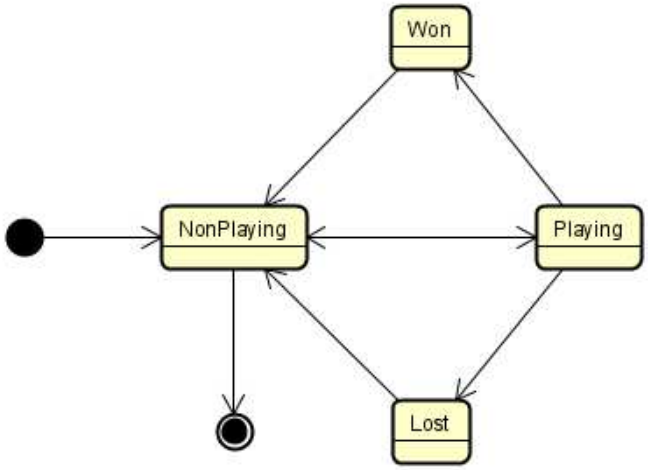
Kao što je ranije rečeno, igrač je i sam prikazan kao objekt u igri.

Stanja igrača

- NonPlaying
    - Igrač ne igra, početno stanje
- Playing
    - Igrač igra igru
- Won
    - Igrač je riješio ploču

- Lost
    - Igrač je napravio grešku u rješavanju polja i izgubio

Ulazak u finalno stanje reprezentira izlazak iz igre. Stanja igrača su povezana prema slici 2.6.



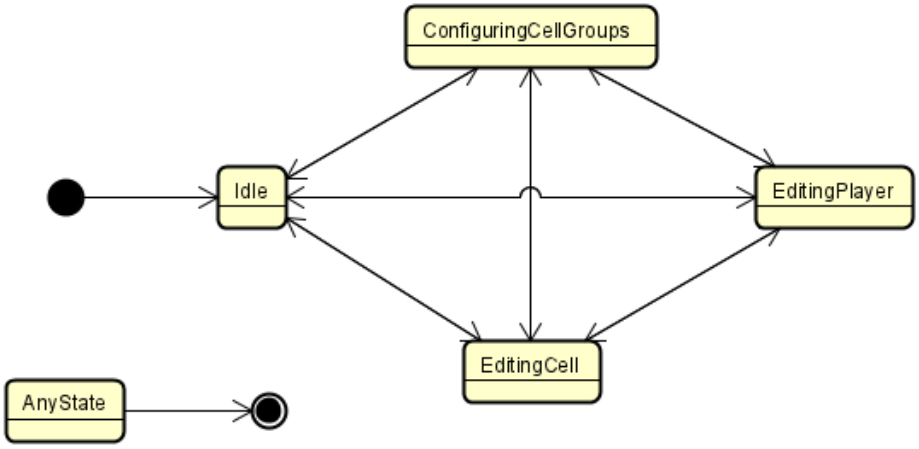
Slika 2.6 Povezanost stanja igrača

Alat Level Editor je također upravljan na temelju stanja. Njegova svrha je dodavanje novih ploča u igru ili uređivanje postojećih.

Stanja Level Editor-a

- Idle
    - Stanje kada ništa nije u uređivanju
- ConfiguringCellGoups
    - Konfiguracija grupa polja: dodavanje, brisanje i modificiranje ploče
- EditingCell
    - Uređivanje određenog polja na ploči
- EditingPlayer
    - Uređivanje svojstva igrača, njegova početna pozicija na ploči

Ulazak u finalno stanje reprezentira izlazak iz Level Editor-a. Prijelazi su definirani na sljedeći način:



Slika 2.7 Povezanost stanja Level Editor-a

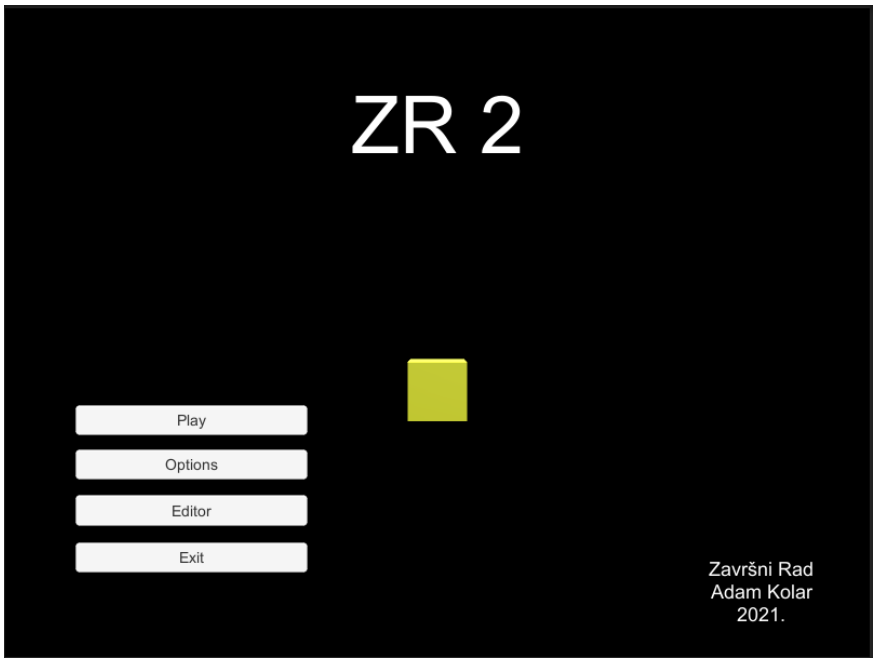
2.3. Igranje igre

Pri ulasku u igru, igraču se prikazuje početni meni igre, gdje su igraču ponuđena 4 gumba:

Gumbi početnog menija

- Start
    - Pokreće izbornik ploča za igranje
- Options
    - Ulaz u izbornik opcija, opcija nema jer je ovaj dio namijenjen za buduće inačice pa je jedino moguće izaći gumbom za povratak na glavni meni.
- Editor
    - Ulaz u Level Editor, nudi opciju odabira ploča za uređivanje postojeće, ili stvaranje nove ploče
- Exit
    - Izlaz iz igre

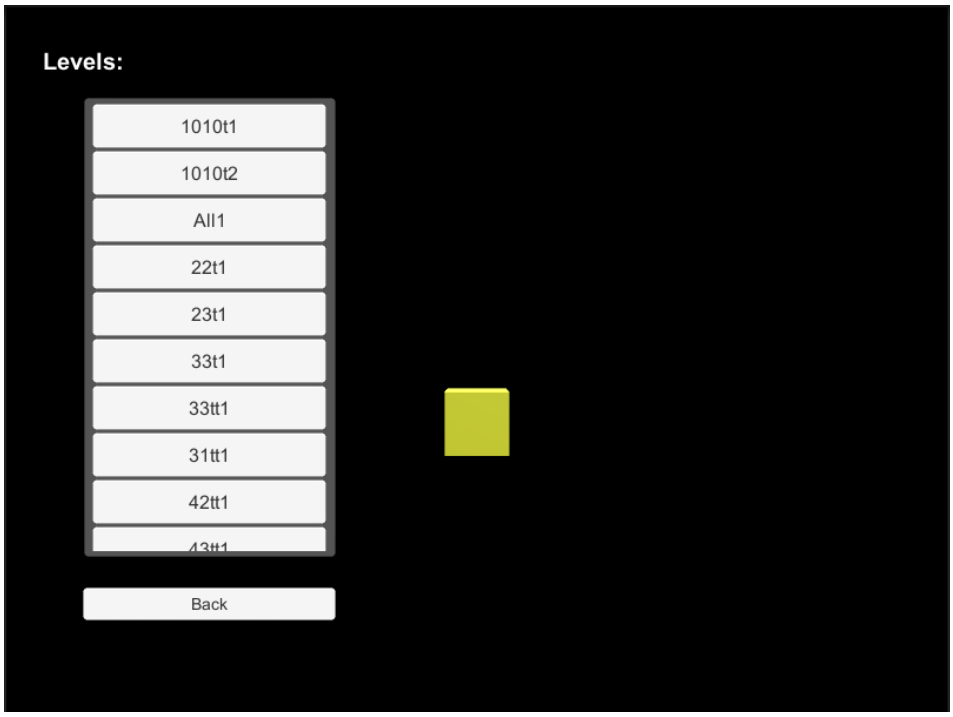
Slika 2.8 prikazuje početni meni igre.



Slika 2.8 Početni meni igre

Izbornik ploča za igru

Izbornik ploča za igru (slika 2.9) se sastoji od liste ploča s njihovim imenima te gumba za povratak na glavni meni. Lista ploča se može pomicati vertikalno ako ne može prikazati sve ploče od jednom. Klikom na ime ploče na navedenoj listi, pokreće se igranje na toj ploči.



Slika 2.9 Izbornik ploča

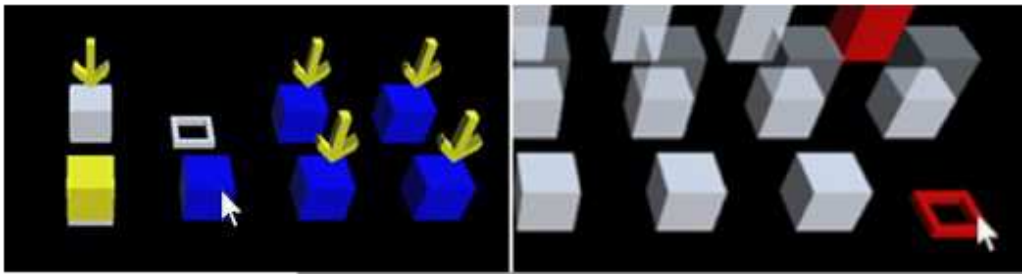
Igranje na ploči

Za upravljanje kretanjem igrača na ploči, koriste se sljedeće tipke na tipkovnici:

- W – pomak unaprijed s obzirom na pogled kamere
- S – pomak unazad s obzirom na pogled kamere
- A – pomak ulijevo s obzirom na pogled kamere
- D – pomak udesno s obzirom na pogled kamere
- Lijeva strelica (←) – rotiranje kamere oko igrača prema lijevo
- Desna strelica (→) – rotiranje kamere oko igrača prema desno
- Esc – privremeno zaustavlja (pauzira) igru
- B – poništava zadnje odrađenu radnju iz međuspremnika (*undo*)
- N – vraća zadnje poništenu radnju iz međuspremnika (*redo*)
- Kotrljanje kotačića miša – zumiranje prema igraču ili od njega (ovisno o smjeru kotrljanja)

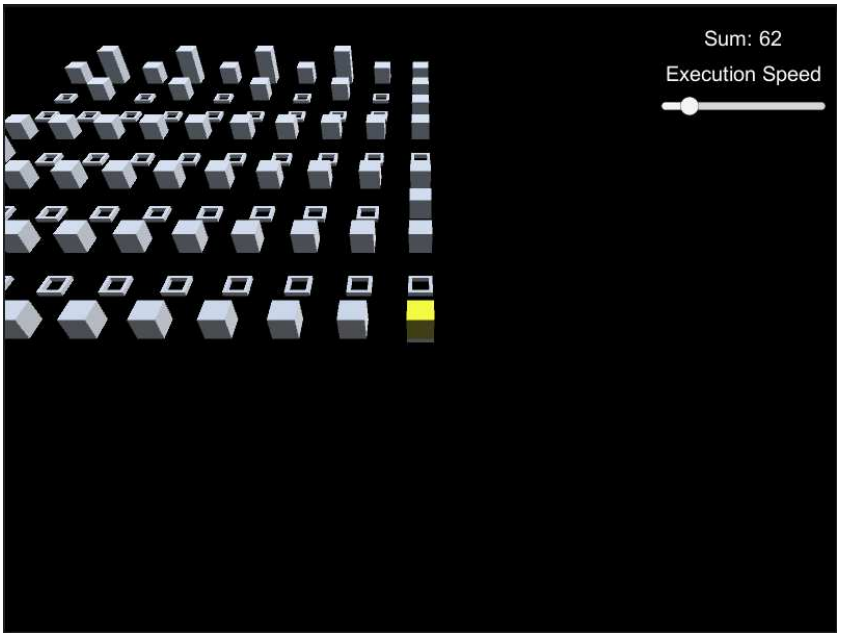
Navedene tipke ne djeluju ako ono što je za njih navedeno nije moguće provesti u skladu s pravilima igre. Na primjer, pomaci su mogući samo ako polje na koje se ide ima vrijednost veću od nule. Ako pomak nije moguć, neće se odigrati. Isto tako, radnje *undo* i *redo* ne rade ništa ako je međuspremnik radnji prazan ili nema više radnji za te akcije.

Dodatna radnja korisnika je pregled posjete nekog polja, specifično za polja tipa Teleporter (teleportira igrača na drugo mjesto na ploči, prikazano strelicom) i Increaser (umanjuje vrijednosti drugih polja, prikazano efektom transparentnosti ili umanjenjem polja na prazno). Prikaz se dobiva pomicanjem kursora miša na kocku polja te se može vidjeti na slici 2.10.



Slika 2.10 Prikaz pregleda posjeta polja

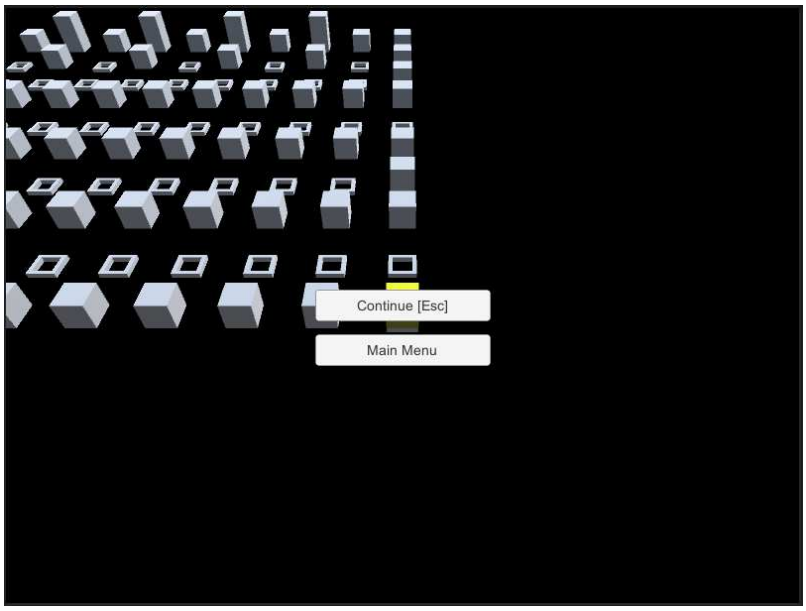
Uz mogući ulaz od njegove strane, korisnik može vidjeti koliko polja još mora posjetiti da bi završio uz labelu „Sum“, te ima mogućnost postavljanja brzine izvođenja rješenja iz međuspremnika rješenja (slika 2.11). Za pokretanje izvođenja rješenja iz međuspremnika potrebno je istodobno pritisnuti tipke Ctrl+V. Dodatni uvjeti da se rješenje iz međuspremnika počne izvoditi su: ime ploče treba biti jednako kao ime rješenja iz međuspremnika, trenutna suma treba biti jednaka onoj iz međuspremnika i igrač treba biti na istoj početnoj lokaciji kao i lokacija prvog koraka međuspremnika. Ako ijedan od tih uvjeta nije zadovoljen, izvođenje neće započeti.



Slika 2.11 Prikaz igranja igre

Pauza

Pod pauzom je igra „zamrznuta“. Sve kontrole igrača od korisnika su ignorirane te su jedine opcije koje se korisniku nude nastavak igranja i povratak na glavni meni. Nastavak igranja se može ostvariti i pritiskom tipke Esc. Primjer prikaza igre u stanju pauze je slika 2.12.

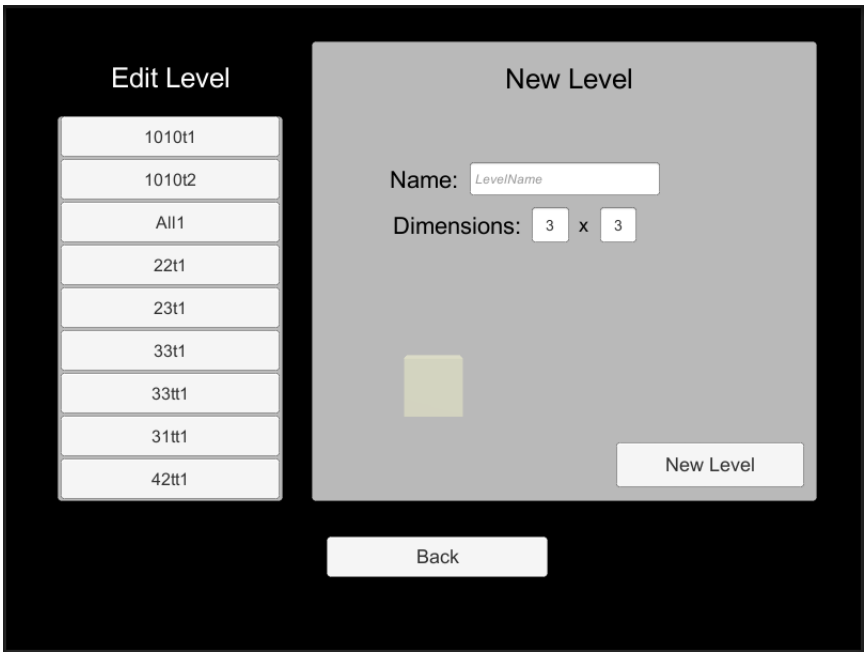


Slika 2.12 Pauza u igri

2.4. Level Editor

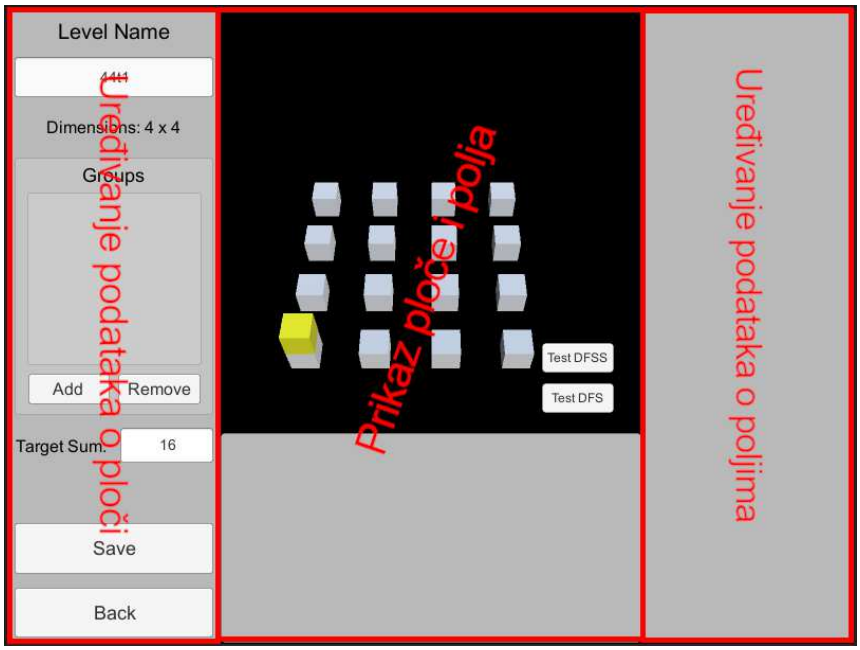
Ulaz u Level Editor

Pri ulazu u Level Editor, korisnik može urediti određenu ploču u listi Edit ili stvoriti novu ploču. Za stvaranje nove ploče potrebno je navesti ime ploče i njene dimenzije. Ako unos nedostaje ili je neispravan (navedene dimenzije nisu cijeli brojevi), korisnik dobiva poruku o pogrešnom unosu. Nakon ispravno unesenih svojstava ploče, klikom na gumb New Level otvara se Level Editor s novom praznom pločom. Ulaz u Level Editor prikazan je slikom 2.13. Otvaranjem Level Editora briše se međuspremnik rješenja.



Slika 2.13 Početni ekran (ulaz) u Level Editor

Level Editor se sastoji od 3 glavna dijela: lijevi, centralni i desni što je prikazano slikom 2.14.

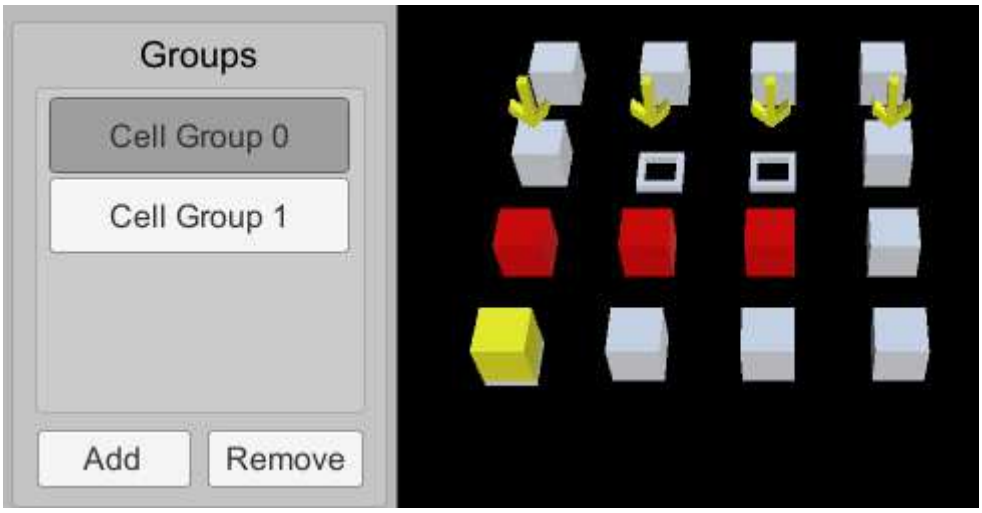


Slika 2.14 Glavni dijelovi Level Editora

Lijevi dio služi za uređivanje podataka o ploči. Ti podaci su: ime ploče, dimenzije ploče (samo informacija o njima), grupe polja i ciljna suma ploče. Ime ploče je tekst koji se prikazuje pri izboru ploča za igranje na za to predviđenoj listi. Grupe polja služe za postavljanje grupa nad kojima utječu polja tipa Increaser. Ciljna suma ploče (Target Sum)

je broj koji označava koliko koraka na ploči igrač treba proći da bi riješio ploču, vrijednost 0 znači da se taj broj automatski određuje pri učitavanju ploče. Centralni dio je glavno mjesto uređivanja i prikaz same ploče. Desni dio služi za uređivanje podataka o poljima što uključuje: tip polja (Default (uvijek da, čak i ako se ne označi), Teleporter, ReachCell (ne koristi se), Increaser), Vrijednost polja (Cell Health), boja kocke polja (Cell Look). Ako je polje tipa Teleporter, na desni dio se dodaje novo svojstvo za upis zvano Teleport to. Ako je polje tipa Increase, na desni dio se dodaju dva nova polja za upis: Increase (prihvaća vrijednosti ≤ 0) i Cell Group. Level Editor ne podržava potpuno uređivanje polja s dimenzijom većom od 2.

Dodavanje grupa polja se radi klikom na gumb Add u Groups podizborniku (slika 2.15). Nakon toga se u istom podizborniku pojavljuje gumb naziva „Cell Group N“ gdje je N broj postojećih grupa. Klikom na taj gumb počinje uređivanje grupe. Za uklanjanje grupe može se kliknuti na gumb Remove. Za dodavanje polja u grupu potrebno je kliknuti lijevom tipkom miša na željena polja, a za uklanjanje polja iz grupe potrebno je kliknuti desnom tipkom miša na željena polja. Trenutna polja u grupi naznačena su strelicom iznad njih. Prekid uređivanje grupe polja postiže se ponovnim klikom na gumb polja ili lijevim/desnim klikom miša „u prazno“ tj. izvan polja.



Slika 2.15 Uređivanje grupe polja 0

Ciljna suma ploče se također ne može uređivati ručno, već pozivom algoritama za pregled rješivosti ploče.

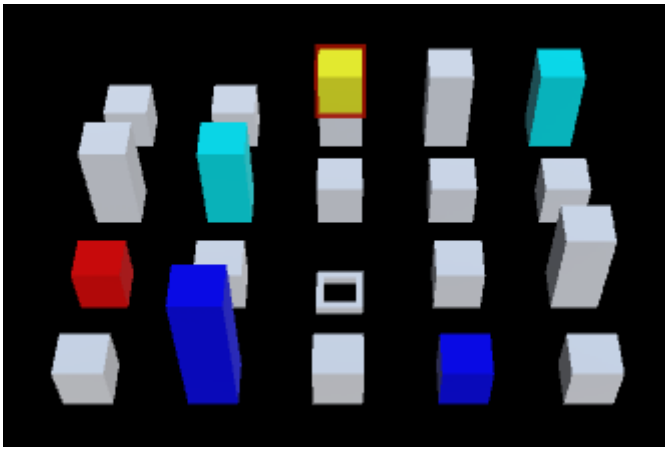
Donji dio centralnog dijela je prazan panel i ne koristi se.

Gornji dio centralnog dijela je namijenjen za uređivanje i pogled na ploču. Radnje koje se mogu odraditi su povezane sa pritiskom sljedećih tipki:

- Lijeva strelica (←) – rotiranje kamere oko igrača prema lijevo
- Desna strelica (→) – rotiranje kamere oko igrača prema desno
- Shift + Lijevi klik miša na polje – uvećava vrijednost tog polja za 1
- Shift + Desni klik miša na polje – umanjuje vrijednost tog polja za 1
- Shift + pritisak kotačića miša + pomicanje miša – pomicanje kamere po XZ plohi
- Kotrljanje kotačića miša – zumiranje prema ili od ploče (ovisno o smjeru kotrljanja)

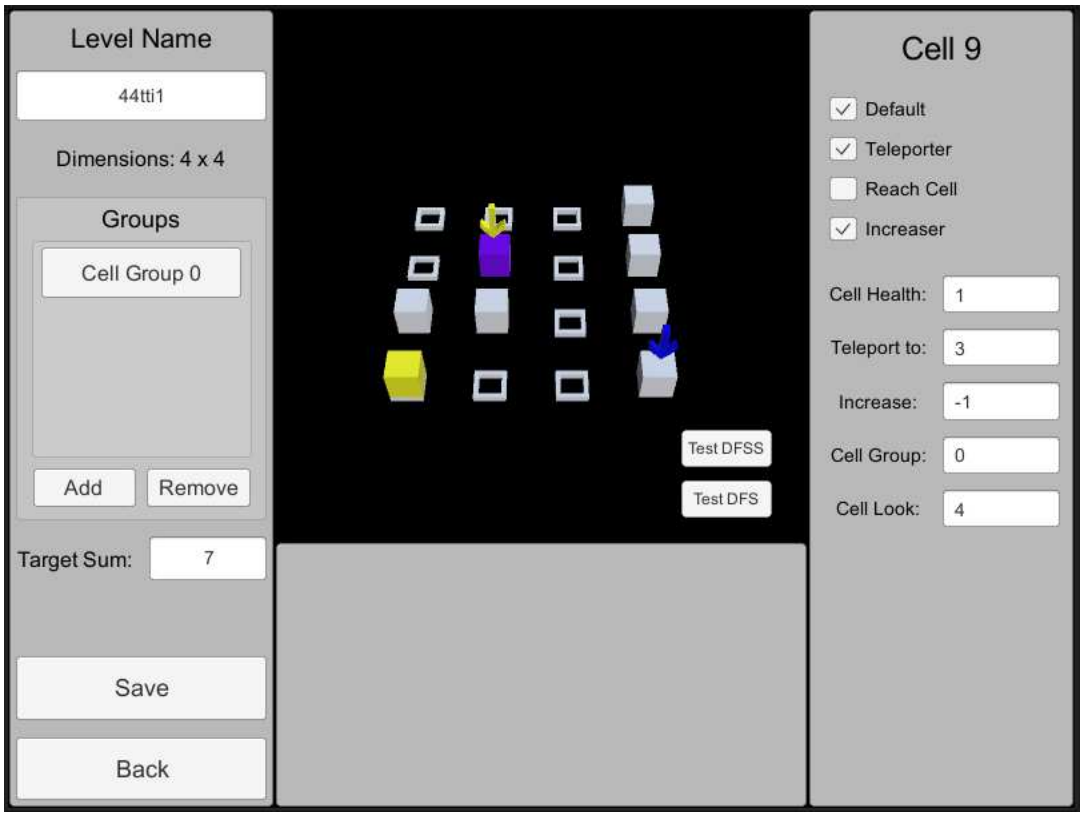
Za određivanje rješivosti ploče koriste se gumbi Test DFS (za algoritam DFS) i Test DFSS (za algoritam DFS sa selekcijom), nakon što algoritmi završe, korisnik dobije poruku o tome. Algoritmi su opisani u 3. poglavlju ovog rada.

Pozicija igrača (žute kocke) se može mijenjati klikom na njega, nakon čega se oko njegove kocke iscrtava narančasti okvir kao naznaka da je označen (slika 2.16). Nakon toga se pozicija igrača može mijenjati klikom na polje na koje se igrač postavlja. Za prestanak uređivanja pozicije igrača potrebno je kliknuti „u prazno“.



Slika 2.16 Selektirani objekt igrača

Uređivanje polja odrađuje se i u desnom dijelu Level Editor-a (slika 2.17), kako bi se desni dio editora prikazao potrebno je označiti željeno polje za uređivanje lijevim klikom na njega. Za prestanak uređivanja polja potrebno je kliknuti „u prazno“. Kao što je već navedeno, u desnom dijelu moguće je uređivati tip polja, a polje može imati više tipova odjednom.



Slika 2.17 Uređivanje polja

Unos za vrijednost Teleport je cijeli broj koji reprezentira jednodimenzionalni indeks polja na ploči na koje se igrač teleportira, što je dodatno prikazano plavom strelicom iznad tog polja u centralnom dijelu sučelja.

Vrijednost ulaza Increase može biti cijeli broj manji ili jednak nuli. Za taj broj se posjećivanjem selektiranog polja, umanjuju polja grupe čiji je indeks zapisan u Cell Group ulazu.

Za ulaz Cell Group uzima se cijeli broj koji reprezentira indeks grupe polja na koju trenutno polje tipa Increaser ima utjecaj, ako je indeks nevaljan (manji od nula ili veći od najvećeg indeksa u grupama polja) onda polje nema nikakav utjecaj kao Increaser.

Ulaz Cell Look definira boju polja, moguće vrijednosti su: 0 (bijela), 1 (plava), 2 (tirkizna), 3 (crvena) i 4 (ljubičasta).

3. Algoritmi za pretraživanje prostora stanja

U ovom poglavlju se nalazi opis prostora stanja te kako se pretražuje. Njegovim pretraživanjem dolazi se do rješenja ploče.

3.1. Generalna ideja

Opći pojmovi i opis problema pretraživanja stanja preuzeti su iz predmeta „Uvod u umjetnu inteligenciju “ [1].

Velik broj problema se može riješiti pretraživanjem prostora stanja. Prostor stanja je skup svih mogućih stanja u kojima može biti neki problem, između određenih stanja postoje određeni prijelazi. Za pretraživanje potrebno je zadati početno stanje i definirati ciljno stanje. Rješenje je niz koraka kojim se iz početnog stanja došlo u ciljno stanje. Jedan od problema koji se javlja je kada broj mogućih stanja postaje prevelik, pa zato pretraživanje treba raditi na sustavan način.

Problem pretraživanja nekog skupa stanja S definiran je kao:

$$problem = (start, goal, pass) \tag{8}$$

Gdje je:

$start \in S$ – početno stanje

$goal(S) \rightarrow \{1,0\}$ – definira je li stanje ciljno (1) ili nije (0)

$pass(s_i) \rightarrow s_j$ – funkcija prijelaza stanja iz jednog u drugo (s_i i s_j su susjedi)

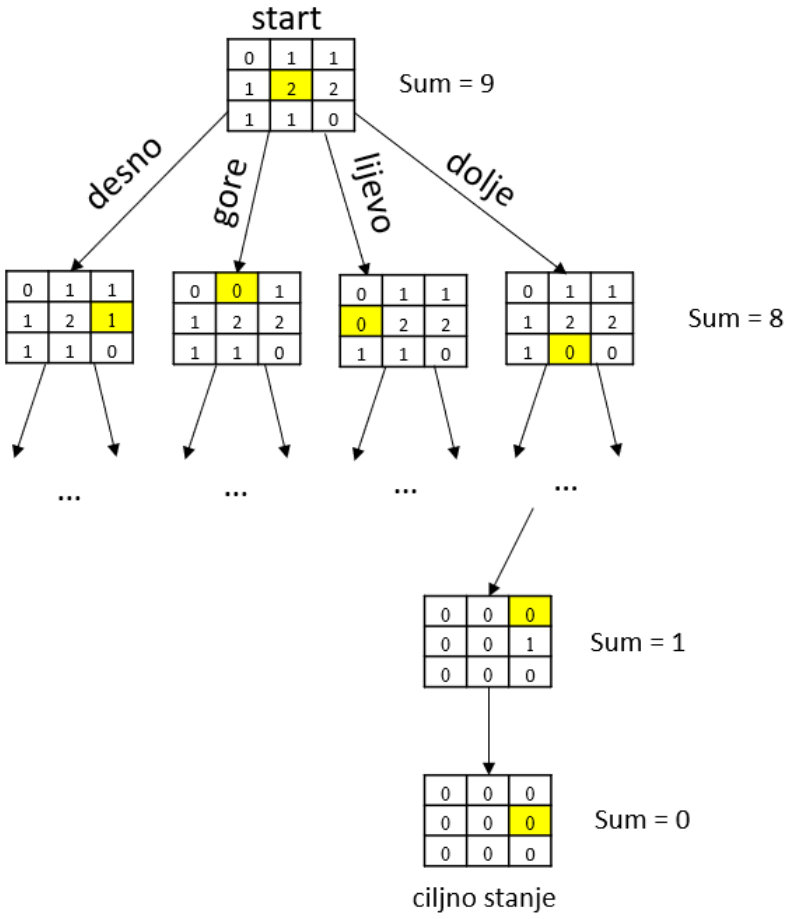
U igri ZR 2 funkcija *goal* daje 1 samo za stanja u kojima je zbroj svih polja ploče jednak 0.

Pretraživanje prostora stanja se općenito svodi na pretraživanje grafa. U igri ZR 2, graf pretraživanja prostora stanja je usmjeren graf za svaku ploču tj. nema ciklusa. Prema logici igre iz bilo kojeg stanja s_i nemoguće je ponovo doći u to stanje putovanjem po prostoru stanja jer se suma (zbroj vrijednosti polja na ploči) uvijek umanjuje svakom novim posjetom. Prostor stanja također nije beskonačan, iz istog razloga. Problem kod prostora stanja igre je da se za proizvoljno zadanu ploču ne mora postići ciljno stanje. U tom slučaju potrebno je pronaći stanje s minimalnom sumom, a pošto nije moguće odmah utvrditi ima li neko stanje minimalnu sumu, potrebno je pregledati sva stanja.

Pretraživanjem prostora stanja nastaje graf, odnosno stablo pretraživanja. Stablo pretraživanja u sebi sadrži čvorove koji u sebi sadrže stanja. U igri je za svaku ploču stablo konačno, ali ako nema ciljnih stanja, onda se cijelo stablo mora pretražiti što dovodi do traženja najduljeg puta u stablu odnosno postaje problem traženja najduljeg puta. Za generiranje čvorova stabla u igri koristi se funkcija (9) koja generira djecu nekog čvora *n* na temelju njegovih sljedbenika.

$$expand(n, pass) \rightarrow \{(s) \mid s \in pass(state(n))\} \tag{9}$$

Uz to, početni čvor sa stanjem „start“ nema roditelja (parent) odnosno vrijednost toga je prazna (null). Slika 3.1 prikazuje primjer stabla pretraživanja u igri.



Slika 3.1 Primjer stabla pretraživanja u igri

Za pretraživanje (što uključuje i generiranje) postoje različiti algoritmi, u okviru ovog rada primjenjuje se algoritam DFS (Depth First Search) i njegove manja modifikacije. Kod algoritama se gledaju sljedeća svojstva: potpunost (algoritam pronalazi rješenje ako i samo ako ono postoji), optimalnost (algoritam pronalazi optimalno rješenje tj. najkraći put), vremenska složenost i prostorna složenost.

3.2. Algoritam DFS

DFS je algoritam pretraživanja u dubinu te se koristi za pretraživanje prostora stanja u igri. Osnovni pseudokod je sljedeći [1]:

```
1. Function DFS(start, pass, end)
2.     open = stack(start)
3.     While !open.empty:
4.         n = open.pop()
5.         if n in end:
6.             Return n
7.         for m in expand(n, pass):
8.             open.push(m)
9.     Return Fail
```

Za potrebe ovog rada, tj. primjenu u igri ZR 2, uz zadržan način pretraživanja je algoritam DFS unaprijeđen tako da ne pohranjuje stanja ploče u čvorove, nego u čvorovima samo pamti roditelje, djecu i indeks polja na kojemu je igrač. Stanje ploče bilježi se u ranije opisanoj podatkovnoj strukturi. Time se umanjuje memorijska kompleksnost i zbog toga je odabran umjesto algoritama tipa BFS (Breadth first search [1]) ili A* (A star [1]). Zbog toga što je stablo uvijek konačno, algoritam se uz takvo ograničenje domene može smatrati potpunim. Algoritam nije optimalan jer ne garantira da je pronađeni put (do ciljnog stanja) najkraći mogući, ali optimalnost nije potrebna značajka za ovaj problem. Pseudokod opisuje algoritam:

```
1.   Function DFS(start, board):
2.       open = stack(Node(pos: start.position, parent: null))
3.       bestPath = (sum: int.MaxValue, path: [])
4.       previous = null
5.       #glavna petlja
6.       while !open.empty:
7.           current = open.pop()
8.           #počisti posječenu granu
9.           while(previous != current.parent)
10.              previous.unvisit()
11.              previous.clearChildren()
12.              previous = previous.parent
13.          #nije moguće posjećivati početno stanje
14.          if current.parent != null:
15.              current.visit()
16.          #traženje najduljeg puta
17.          if board.sum < bestPath.sum:
18.              bestPath.sum = board.sum
19.              bestPath.path = RetrackPath(current)
20.          if board.sum == 0:
```

```
21.                 Return bestPath
22.             for n in current.neighbours():
23.                 open.push(n)
24.             previous = current
25.         Return bestPath
```

Algoritam vraća najmanju pronađenu sumu vrijednosti polja na odigranoj ploči (`board.sum`) i put (`bestPath`) kojim se može doći do te sume. Kako prostorna složenost ne bi postala eksponencijalna, algoritam briše potpuno posjećene grane. Parametar `board` reprezentira stanje ploče koje se mijenja funkcijama `visit()` i `unvisit()`. Funkcija `RetrackPath` vraća put od početnog do danog čvora prolaskom po trenutnom stablu, vrijednosti tog puta su indeksi polja na ploči. Funkcija `neighbours()` vraća susjede koji se mogu posjetiti od trenutnog čvora. Funkcija `visit()` mijenja stanje ploče pri posjeti polja, a to uključuje: umanjivanje vrijednosti polja za jedan, obavljanje radnji Teleporter-a i/ili Increaser-a nad parametrom `board` čime indirektno utječe na parametar `board.sum`. Funkcija `unvisit()` vraća stanje ploče unatrag, tj. anulira učinak od funkcije `visit()`. Parametar board.sum prati kolika je suma svih polja na ploči, ako je 0, ploča je riješena i algoritam odmah može prekinuti sa izvođenjem glavne petlje te vratiti pronađeni put.

Vremenska kompleksnost algoritma je $O(g^d)$ gdje je g faktor grananja, a d maksimalna dubina stabla. Prostorna složenost je $O(gd)$. Najveći nedostatak algoritma je vremenska složenost te zbog toga algoritam na nekim pločama ne završava u vremenu prihvatljivom za interakciju s korisnikom, odnosno i u pozadinskom načinu rada vremenski zahtjevi se za pojedine ploče (za veliki d) mogu mjeriti u milijunima godina.

3.3. Algoritam DFS sa selekcijom čvorova

Algoritam DFS sa selekcijom čvorova, nazvan DFSS, je sličan opisanom DFS-u. Razlika je u tome što umjesto da posjeti prvi dostupni susjedni čvor, algoritam pregleda koliku dobit dostupni susjedni čvorovi imaju te posjećuje onaj s najvećom dobiti. Pod „dobit“ se misli na smanjenje sume, bolja dobit daje smanjenje sume veće od 1, dok lošija umanjuje sumu točno za 1. Pseudokod je sljedeći:

```
1.   Function DFSS(start, board):
2.       open = stack(Node(pos: start.position, parent: null))
3.       bestPath = (sum: int.MaxValue, path: [])
4.       previous = null
5.       #glavna petlja
6.       while !open.empty:
7.           current = open.pop()
8.           #počisti posječenu granu
9.           while(previous != current.parent)
10.              previous.unvisit()
11.              previous.clearChildren()
12.              previous = previous.parent
13.          #nije moguće posjećivati početno stanje
14.          if current.parent != null:
15.              current.visit()
16.          #traženje najduljeg puta
17.          if board.sum < bestPath.sum:
18.              bestPath.sum = board.sum
19.              bestPath.path = RetrackPath(current)
20.          if board.sum == 0:
21.              Return bestPath
22.          neighbours = list()
23.          for n in current.neighbours():
24.              neighbours.add((n, n.peek())
25.          sortBySecondNumber(neighbours)
26.          for n in neighbours:
27.              open.push(n.Item1)
28.          previous = current
29.      Return bestPath
```

Za ploče bez polja koja su tipa Teleporter i/ili Increaser ova varijanta algoritma ne nudi nikakvo ubrzanje. Algoritam zapravo tada ima negativan učinak pošto privremeno posjećuje polje prije nego što ga dodaje u stog, što rezultira time da radi dvostruko više posjeta od običnog DFS-a pa je za očekivati da je zato sporiji. Ali postoje slučajevi u kojima se, radi prioritiziranog posjećivanja, algoritam izvodi daleko brže od obične varijante, radi toga što odabere polja koja uvelike umanjuju sumu ploče i čine put kraćim. Funkcija `peek()` vraća sumu nakon posjete čvora djeteta te vraća `board` na prethodno stanje. Funkcija `sortBySecondNumber` sortira danu listu parova po drugom elementu para (novoj sumi nakon posjete) po silaznom redoslijedu zato što se nakon toga lista susjeda ubacuje u stog pa se tako čvorovi susjedi s lošijom dobiti posjećuju zadnji. Algoritam ima jednake vremenske i prostorne složenosti kao i obična varijanta.

4. Razvoj igre

4.1. Razvojna okolina

Igra ZR 2 je izrađena u programu Unity (verzija 2020.1.12f1). Za modeliranje određenih objekata (strelica, prazno polje) korišten je program Blender (verzija 2.83). Za pisanje koda se koristio program Visual Studio 2019 (verzija 16.9.2). Sav razvoj je odrađen na računalu sa sljedećim specifikacijama: CPU: Intel Core i5-6500, GPU: Nvidia GeForce GTX 1050 Ti 4GB, RAM: 16 GB DDR4 2133 MHz, Sekundarna memorija: 2.25 TB SSD. Performanse igre na navedenom računalu u Unity Editor-u su 1200 sličica u sekundi.

4.2. Razvoj logike

Pod razvoj logike spada modeliranje ponašanja za postojeće komponente/objekte programa Unity te njihovo podešavanje. Svi objekti u igri su načinjeni od komponenti Unityja i samostalno izrađenih komponenti. Jedina „dodatna“ komponenta je besplatan dodatak „QuickOutline“ iz Unity Asset Store-a, koja se koristi za crtanje okvira oko igrača u Level Editor-u.

Programski kod aplikacije ima sveukupno 7578 linija. Cijeli kod pisan je u programskom jeziku C#. Glavne klase/skripte (ujedno i komponente) koje se koriste su: Scene, World i LevelEditor. Skripta Scene upravlja cijelom igrom, odnosno scenom (u igri se koristi jedna samo scena). Skripta World crta svijet odnosno ploče. Skripta LevelEditor upravlja Level Editorom. Za lakšu kontrolu nad igrom odnosno stanjima (automatima stanja) igre, koristi se generička klasa StateMachine. StateMachine kao generički argument prima definiciju enumeracije i na temelju tih vrijednosti stvara stanja. Na stvoreno stanje dodjeljuje se metoda stanja i moguće je definirati metode prijelaza između tog stanja i ostalih stanja. Primjer iz koda aplikacije je sljedeći:

```
1.    //definicija stanja
2.    public enum GameStates
3.    {
4.        Playing,
5.        Paused,
6.        Menu,
7.        Editor,
8.        WonLost,
9.    }
10.   //kreiranje automata stanja
```

```
11.   public StateMachine<GameStates> GameState;
12.
13.   //dodjela metoda stanjima u funkciji Awake()
14.   //stvaranje objekta s početnim stanjem Menu
15.   GameState = new StateMachine<GameStates>(GameStates.Menu);
16.   GameState.Methods[GameStates.Playing] = Playing;
17.   GameState.Methods[GameStates.Paused] = Paused;
18.   GameState.Methods[GameStates.Menu] = Menu;
19.   GameState.Methods[GameStates.Editor] = Editor;
20.
21.   //definiranje metoda prijelaza iz jednog stanja u drugo
22.   GameState.Switches[GameStates.Playing][GameStates.Paused] = Pause;
23.   GameState.Switches[GameStates.Paused][GameStates.Playing] = Unpause;
24.
25.   //Postavljanje stanja na Playing
26.   //ujedno poziva i zadanu funkciju prijelaza između stanja
27.   GameState.State = GameStates.Playing;
28.
29.   //Izvedi metodu trenutnog stanja
30.   GameState.Execute();
31.
```

Prikazani kod demonstrira rad sa instancom klase StateMachine. Prvo je potrebno definirati stanja enumeracijom, nakon toga stvoriti objekt pri čemu je potrebno zadati početno stanje. Nakon toga se zadaju metode za stanja i prijelaze između stanja. Konkretan rad s klasom se sastoji od promjena stanja (operatorom dodjeljivanja) i izvođenja stanja metodom `Execute()`.

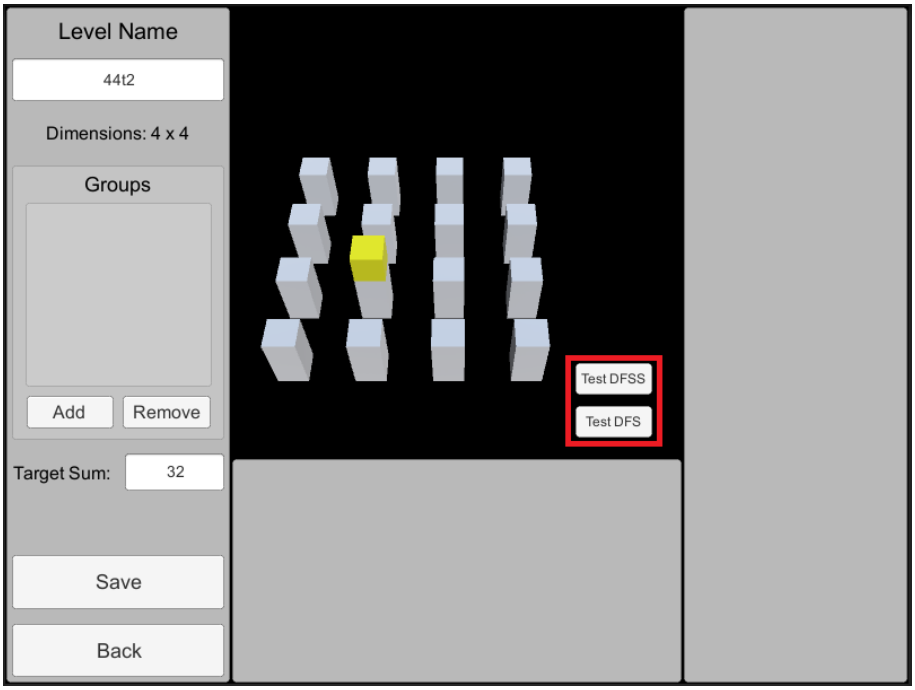
Za rad s višedimenzionalnim poljima (pločama) koristi se generička klasa MArray. Klasa interno pohranjuje podatke u jednodimenzionalno polje, no njima se može pristupiti preko jednodimenzionalnih i višedimenzionalnih koordinata. Tip podataka koji se pohranjuje definirani su generičkim argumentom. Primjer korištenja je sljedeći:

```
1.  //definiranje ploče, struktura Cell definira podatke za polje, ploča je dimenzija 2x3
2.  public MArray<Cell> Cells = new MArray<Cell>(2, 3);
3.
4.  //Višedimenzionalni pristup
5.  Cells[0,1] = new Cell();
6.  Cell cellAt01 = Cells[0,1];
7.
8.  //Jednodimenzionalni pristup
9.  Cells.OneDimensional[1] = new Cell();
10. Cell cellAt0O = Cells.OneDimensional[1];
11.
12. //Pretvorbe koordinata
13. //2D -> 1D
14. int OneDimensionalCoords = Cells.getIndex(0,1);
15. //1D -> 2D
16. int[] MultiDimensionalCoords = Cells.getCoords(1);
17.
```

Prikazani kod je primjer korištenja klase MArray. Elementima klase se može pristupiti preko višedimenzionalnih i jednodimenzionalnih koordinata, uz to, svaka instanca klase nudi pretvorbu iz jednog tipa koordinata u drugi.

5. Rezultati

Kao rezultat završnog rada, nova inačica igre ZR 2 ima ugrađen alat za ispitivanje rješivosti proizvoljne zadane ploče u igri. Za početne zadane kriterije zadatka gdje je ploča veličine 4x4 i vrijednosti polja su između 0 i 2, algoritmi se izvode u vremenu od najviše nekoliko minuta. Igra je dovoljno funkcionalna za demonstraciju rada. Procedura pokretanja algoritama nad proizvoljnom pločom je sljedeća: Otvoriti ili izraditi proizvoljnu ploču u Level Editor-u, te nakon toga pritisnuti gumb Test DFS (za obični DFS) ili Test DFSS (za DFS sa selekcijom čvorova).



Slika 5.1 Gumbi za pokretanje algoritama označeni u crvenom pravokutniku

U igri se tada „u pozadini“ tj. drugoj dretvi počinje izvoditi odabrani algoritam. Kada algoritam završi, korisniku se prikaže poruka sljedećeg oblika:

Slika 5.2 Poruka korisniku da je algoritam završio s izvođenjem

U poruci se vidi koji algoritam se izvodio, te koliko vremena je potrošio (u sekundama). Uz to, rješenje tj. put koji je algoritam pronašao spremljen je u međuspremnik rješenja, što znači da korisnik može vidjeti koji je to put bio pokretanjem testirane ploče iz izbornika ploča za igru te pritisnuti tipke Ctrl + V. Time se započinje izvođenje poteza iz međuspremnika rješenja.

Za dane algoritme i za određene ploče (opis je u poglavlju Prilog A) izmjerene performanse su prikazane u tablici 5-1:

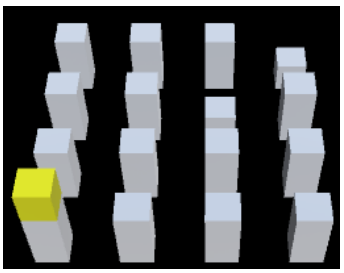
Ime ploče	Dimenzije ploče	DFS vrijeme (ms)	DFSS vrijeme (ms)
44t1	4x4	0.05	2.63
44t2	4x4	43.31	57.11
44t3	4x4	1.93	2.75
44t4	4x4	0.05	0.08
44t5	4x4	126209.70	183968.13
44tt8	4x4	0.23	5.85
44tt9	4x4	0.03	0.04
55ti1	5x5	--	0.03
66tti1	6x6	0.04	--
1010t1	10x10	1.16	3.00
99tt1	9x9	3.02	4.14
55t3	5x5	--	--

Tablica 5-1 Izmjerene performanse algoritama

Dvije crtice u tablici označavaju da algoritmu treba previše (u testovima > 10 min) da se izvrši, u tom slučaju je izvođenje algoritma bilo prekinuto.

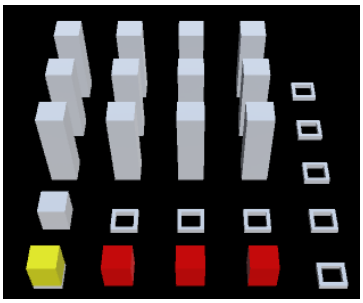
Iz Tablice 5.1 se vidi da se algoritmi izvode relativno brzo za 8 od 12 slučajeva te da je obični DFS uglavnom brži od DFSS-a. Posebni slučajevi su 44t5, 55ti1, 55t3 i 66tti1. Ploča 44t5 (Slika 5.3) je problemski složeniji oblik ploče za zadane kriterije zadatka. Razlog tomu

je to što u ploči nije moguće dovesti sumu na 0, a čvorova grana stabla je dosta velik čime vremensko izvođenje značajno raste.



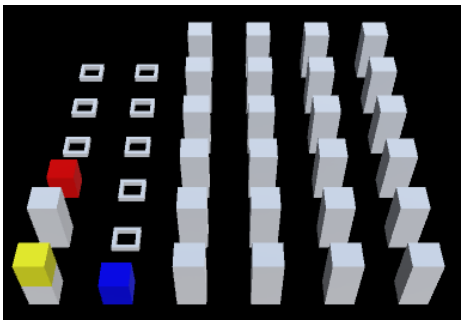
Slika 5.3 Ploča 44t5

Slučaj ploče 55ti1 gdje DFSS radi daleko bolje od običnog je takav da dozvoljava svojstvu selekcije DFSS-a da napravi odabir koji je za tu ploču ključan. Obični DFS „padne u jamu“ u kojoj pokušava pronaći sve puteve, a do izlaska iz „jame“ će proći više vremena, dok DFSS odmah odabere polja desno (Slika 5.4) tipa Increaser koja umanjuju „jamu“ do 0 i njihovim prolaskom završava se ploča.



Slika 5.4 Ploča 55ti1

Kod ploče 66tti1 događa se suprotna stvar, obični DFS „bez gledanja“ odabire bolji put dok DFSS odmah odabire polje do njega koje ga „baca u jamu“, a to je Teleporter prema desno (Slika 5.5).



Slika 5.5 Ploča 66tti1

Zadnji posebni slučaj 55t3 je primjer u kojem ni jedan od algoritama ne može riješiti u vremenu manjem od 10 minuta zbog prevelikog broja stanja ploče.

Zaključak

Kao rezultat ovog rada, postojeća igra „ZR 2“ je proširena s mogućnošću rješavanja njenih razina (ploča) pomoću algoritma DFS i njegove manje modifikacije. Algoritmi imaju performanse sukladne težinama tj. kompleksnosti ploča koje su im zadane. Vraćeni rezultat od algoritama tj. put po ploči je također moguće vidjeti. Sama igra je još uvijek u „ranoj“ fazi razvoja, odnosno moguće je nadograditi igru u aspektima poput grafike, priče na kojoj se igra temelji, kompleksnosti ploča, poboljšanje korisničkog sučelja i sl.

Literatura

[1] Jan Šnajder, Bojana Dalbelo Bašić: prezentacija *Pretraživanje prostora stanja* u sklopu predmeta „Uvod u umjetnu inteligenciju“ Fakultet elektronike i računarstva, Sveučilište u Zagrebu, 2020./2021.

[2] Unity Dokumentacija (https://docs.unity3d.com/Manual/index.html), pristupljeno 1. lipnja 2021.

Prilog A

Definicije ploča iz tablice 5-1:

```
1.  44t1
2.  Target=16
3.  SumToZero
4.  (0, 0)
5.  0
6.  4 4
7.  (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
8.
```

```
1.  44t2
2.  Target=32
3.  SumToZero
4.  (1, 1)
5.  0
6.  4 4
7.  (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0,
    0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
    (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0,
    0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
8.
```

```
1.  44t3
2.  Target=16
3.  SumToZero
4.  (3, 3)
5.  0
6.  4 4
7.  (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
8.
```

```
1.  44t4
2.  Target=17
3.  SumToZero
4.  (3, 2)
5.  0
6.  4 4
7.  (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
8.
```

```
1.  44t5
2.  Target=29
3.  SumToZero
4.  (0, 0)
5.  0
6.  4 4
7.  (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0,
    0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
```

```
    (1, 2, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0,
    0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
8.
```

```
1.  44tt8
2.  Target=16
3.  SumToZero
4.  (0, 0)
5.  0
6.  4 4
7.  (1, 1, 0, 0, 0, 0) (11, 1, 13, 0, 1, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1,
    0, 0, 0, 0) (11, 1, 10, 0, 1, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0,
    0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1,
    0, 0, 0, 0) (11, 1, 2, 0, 1, 0) (1, 1, 0, 0, 0, 0)
8.
```

```
1.  44tt9
2.  Target=9
3.  SumToZero
4.  (0, 0)
5.  0
6.  4 4
7.  (11, 1, 15, 0, 1, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1,
    6, 0, 1, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1, 12, 0, 1, 0) (1, 0, 0, 0, 0,
    0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1,
    0, 0, 1, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
8.
```

```
1.  55t3
2.  Target=0
3.  SumToZero
4.  (0, 0)
5.  0
6.  5 5
7.  (1, 4, 0, 0, 0, 0) (1, 4, 0, 0, 0, 0) (1, 4, 0, 0, 0, 0) (1, 4, 0, 0, 0, 0) (1, 5, 0,
    0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 5, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0)
    (1, 4, 0, 0, 0, 0) (1, 4, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 6, 0, 0, 0, 0) (1, 4, 0,
    0, 0, 0) (1, 5, 0, 0, 0, 0) (1, 5, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0)
    (1, 5, 0, 0, 0, 0) (1, 4, 0, 0, 0, 0) (1, 4, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 4, 0,
    0, 0, 0) (1, 5, 0, 0, 0, 0) (1, 5, 0, 0, 0, 0)
8.
```

```
1.  55ti1
2.  Target=40
3.  SumToZero
4.  (0, 0)
5.  1
6.  {10, 11, 12, 13, 18, 17, 16, 15, 20, 21, 22, 23, 5}
7.  5 5
8.  (1, 0, 0, 0, 0, 0) (1001, 1, 0, -1, 3, 0) (1001, 1, 0, -1, 3, 0) (1001, 1, 0, -1, 3,
    0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0,
    0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0,
    0) (1, 3, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3,
    0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0,
    0) (1, 3, 0, 0, 0, 0) (1, 3, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
9.
```

```
1.  66tti1
2.  Target=0
3.  SumToZero
4.  (0, 0)
5.  1
6.  {8, 9, 10, 11, 17, 16, 15, 14, 20, 21, 22, 23, 29, 28, 27, 26, 32, 33, 34, 35, 2, 3,
    4, 5, 1}
7.  6 6
```

```
8.  (1, 1, 0, 0, 0, 0) (11, 1, 2, 0, 1, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0,
    0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
    (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1001, 1, 0, -2, 3, 0) (1, 0,
    0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0,
    0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2,
    0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 2, 0, 0, 0,
    0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0,
    0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 2, 0, 0, 0,
    0)
9.
```

```
1.  99tt1
2.  Target=33
3.  SumToZero
4.  (1, 1)
5.  0
6.  9 9
7.  (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1, 24, 0, 1, 0) (1, 1, 0, 0, 0, 0) (1, 0,
    0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0,
    0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1, 56, 0, 1, 0) (1, 1, 0, 0, 0, 0) (1,
    0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0,
    0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1,
    0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0,
    0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1,
    0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1, 10, 0, 1, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0,
    0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (11, 1, 60, 0, 1, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0)
8.
```

```
1.  1010t1
2.  SumToZero
3.  (0, 0)
4.  0
5.  10 10
6.  (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0)
    (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 2, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 1, 0,
    0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 1, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0) (1, 0, 0, 0, 0, 0)
    (1, 1, 0, 0, 0, 0)
7.
```

Sažetak

Algoritmi za ispitivanje rješivosti digitalne igre za proizvoljno zadanu početnu situaciju

Problem ovog završnog rada je bio proširiti postojeću digitalnu igru s alatom za ispitivanje rješivosti razina i prikaz njihovih rješenja. U igri se osim igranja razina mogu uređivati i dodavati i nove pomoću alata Level Editor. Cilj rada je bio dodati novi alat za provjeru rješivosti razina i prikaz tog rješenja. Do rješenja razina se dolazi pretraživanjem prostora stanja. Za pretraživanje prostora stanja postoje razni algoritmi. U ovom radu obrađeni su algoritam DFS i njegova manja modifikacija.

Ključne riječi:

Unity; Razvoj digitalnih igara; Algoritmi; Prostor stanja; DFS

Summary

Algorithms for testing the solvability of digital game for an arbitrary starting situation

The topic of this final paper was to extend the existing digital game with a tool to examine the solvability of the levels and display their solutions. In the game, in addition to playing levels, you can also edit and add new ones using the Level Editor tool. The aim of the paper was to add a new tool to check the solvability of levels and display this solution. The solution of the levels is reached by searching the state space. There are various algorithms for searching the state space. In this paper, the DFS algorithm and its minor modification are discussed.

Keywords:

Unity; Game Development; Algorithms; State space; DFS