Sadržaj

Uvod

Cilj ovog završnog rada je izrada GUI aplikacije koja će rješavati logičke zagonetke *Pentomino*. Nakon što korisnik ručno unese zagonetku u aplikaciju, program će riješiti problem jednostavnim DFS algoritmom koji primjenjuje propagaciju ograničenja te ispisati rješenje ako ono postoji.

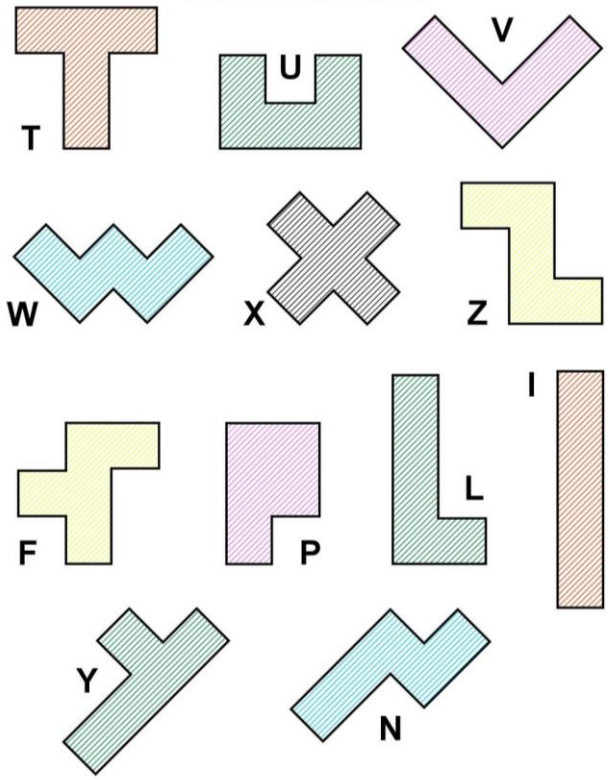
U prvom ćemo poglavlju pričati o tome što su to pentomina, o njihovoj povijesti i područjima uporabe.

Kroz drugo ćemo se poglavlje upoznati s tehnologijama korištenim za razvitak ove aplikacije, nakon čega ćemo se u trećem posvetiti implementaciji rješenja. U četvrtom poglavlju ćemo rješenje testirati. Analizirat ćemo rezultate i ocijeniti učinkovitost razvijenih metoda, a predložit ćemo i moguća poboljšanja.

Rad je namijenjen svima koji su zainteresirani za logičke zagonetke, metode njihova rješavanja te primjeni tih metoda. Očekujemo da će rezultati ovog rada pružiti dublje razumijevanje zagonetke Pentomino, ali i poslužiti kao temelj za daljnji razvoj i unapređenje metoda za rješavanje ove i sličnih zagonetki.

1. Pentomino

Pentomino je geometrijski lik koji se sastoji od pet kvadrata koji su međusobno spojeni po dužinama svojih stranica. Ova vrsta oblika spada u grupu *poliomina*, koja obuhvaća sve geometrijske oblike koji se sastoje od više kvadrata koji su spojeni zajedničkim stranicama. Postoji ukupno 12 različitih pentomina ako svaki od njih gledamo kao lik koji se smije rotirati i zrcaliti. Svaki od njih ima svoje ime kao što je prikazano na slici (Sl. 1.1).



Sl. 1.1 Pentomino oblici

Pentomina se, kao i ostala poliomina, koriste u mnogim poznatim matematičkim i logičkim zagonetkama, a koriste se i u raznim igrama. Jedne od najpoznatijih društvenih igri koje koriste poliomina su: *The Isle of Cats*, *Cartographers*, *A Feast for Odin* [10].

1.1. Povijest pentomina

Koncept oblika kao što su pentomina postoji u rekreativnoj matematici od početka 1900-ih, a popularnost im je znatno porasla kad su dobili imena i priznanje u drugoj polovici tog

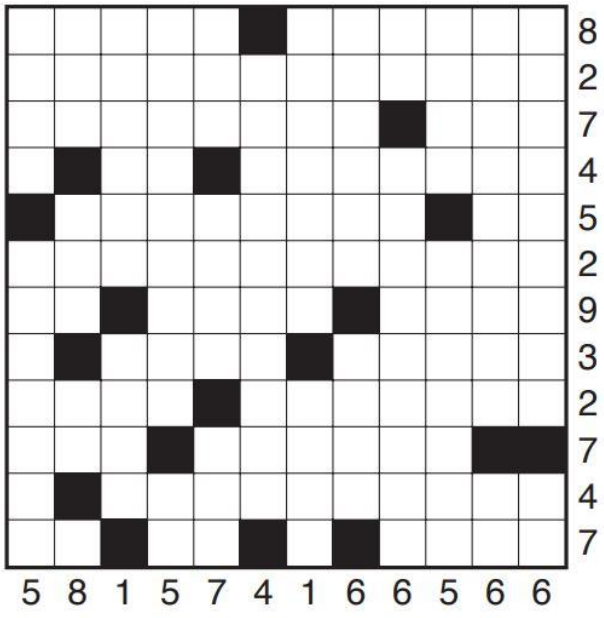
stoljeća [3]. Solomon Wolf Golomb, američki profesor, prvi je 1953. godine predstavio njihova imena i opisivao njihove mogućnosti matematičarima, koji su ih sa značajnim interesom prihvatili [3]. Do opće su se javnosti probili 1957. godine, kada je Martin Gardner prvi puta pisao o njima u svojoj poznatoj kolumni u časopisu Scientific American [6].

1.2. Zagonetka Pentomino

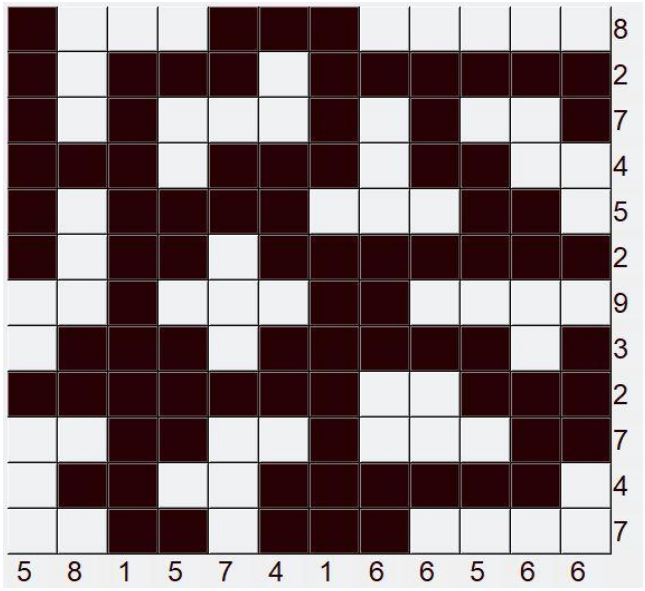
Još od početka dvadesetog stoljeća razni ljubitelji logike i matematike okušavaju se u stvaranju različitih zagonetki i igara temeljenih na pentominima [8]. Za ovaj rad je odabrana jedna od poznatijih varijanti logičke zagonetke Pentomino.

Pentomino se rješava postavljanjem svih 12 pentomina u mrežu tako da se međusobno ne dodiruju, čak ni dijagonalno [5]. Zrcaljenje i rotiranje je dozvoljeno [5]. Brojevi uz mrežu govore koliko je dijelova pentomina u pripadnom retku, odnosno stupcu [5]. Crno polje nije dio nekog pentomina [5].

Kako izgleda jedan primjer zagonetke Pentomino prikazano je na slici (Sl. 1.2), čije je rješavanje opisano. Taj primjer zagonetke unesen je u aplikaciju, koja ga je uspješno riješila za samo pola sekunde. Na slici (Sl. 1.3) je prikazano rješenje te zagonetke.



Sl. 1.2 Primjer zagonetke Pentomino [5]



Sl. 1.3 Rješenje zagonetke Pentomino

1.3. Optimalna metoda rješavanja zagonetke

Rješavanje logičkih zagonetki započinje se pronalaskom informacija najvećeg značaja u njima. U ovoj varijanti zagonetke Pentomino to su redovi tj. stupci koji trebaju sadržavati najviše dijelova pentomina i sigurno prazna polja u tim redovima tj. stupcima. Uz malo logičkog promišljanja o mogućim kombinacijama rasporeda polja popunjenih dijelovima pentomina i praznih polja zaključujemo da neka polja u svakom slučaju moraju sadržavati dio nekog pentomina, a neka moraju biti prazna. Popunjavanjem polja ćemo moći stvarati nove zaključke o rasporedu crnih i bijelih polja na ploči. Ovdje nije cilj odmah ucrtati neki pentomino za kojeg nam se „čini da bi dobro odgovarao“ na nekom mjestu, nego je bitno pažljivo i ispravno popunjavati ploču kako bismo izbjegli nepotrebno brisanje.

U jednom ćemo trenutku doći do pozicije u kojoj više nećemo moći pronaći polja za koja smo sigurni sadrže li dio pentomina ili ne. Tada tražimo neki pentomino koji još nismo postavili, a koji se na ploču može postaviti na samo nekoliko mjesta. Pretpostavljamo da se do točnog rješenja dolazi postavljanjem tog pentomina na mjesto koje nam izgleda kao manje povoljno jer je cilj doći do kontradikcije, tj. do toga da taj pentomino kasnije možemo unijeti sa sigurnošću umjesto preko pretpostavke. Nakon toga ćemo za neka polja opet moći sa sigurnošću znati jesu li popunjena ili ne. Takvim unosom ćemo nekad već

nakon samo jednog kruga eliminacija pretpostavki moći samo logičkim razmišljanjem doći do rješenja, a nekad će taj postupak biti potrebno ponoviti još nekoliko puta.

2. Korišteni programski jezici i biblioteke

Pentomino zagonetke predstavljaju intrigantnu sferu izazova unutar svijeta logičkih zagonetki, stvarajući potrebu za učinkovitom i intuitivnom aplikacijom za njihovo rješavanje. U ovom radu, ključni alat u implementaciji takve aplikacije bio je programski jezik Python. Python je odabran zbog svoje svestranosti, jednostavnosti kôda i bogate podrške za biblioteke.

2.1. Python

Python je visokorazinski interpretirani programski jezik koji se sve više koristi u raznim granama računarstva [12]. Gudio van Rossum objavio je prvu verziju 1991. godine, od kada njegova popularnost samo raste, a trenutno je jedan od najpopularnijih programskih jezika [12]. Sveprisutnost Pythona proizlazi iz njegove jednostavnosti i lakoće korištenja, što omogućava brzo i lako učenje te razvijanje programa. Objektno je orijentiran i slabo tipiziran. Tipovi podataka su mu dinamični, što znači da programer ne mora deklarirati tip varijable, već se tip određuje prilikom izvršavanja programa.

2.2. Korištene biblioteke

Python ima mnogo biblioteka koje se mogu koristiti za rješavanje raznovrsnih problema. Vrlo su jednostavne za instalaciju. Sve što je potrebno napraviti jest otvoriti *Terminal* na računalu i unijeti naredbu `pip install <naziv_biblioteke>`.

2.2.1. Tkinter

Tkinter (Tk interface) je radni okvir za izradu grafičkog korisničkog sučelja (GUI) koji je dio standardne Python biblioteke [9].

Tkinter je objektno orijentirano sučelje za *Tcl/Tk GUI toolkit*, najpopularniji skup alata za stvaranje GUI-a u Python-u [13]. Tkinter podržava širok spektar grafičkih elemenata (engl. *widgets*), kao što su gumbi (engl. *buttons*), labele (engl. *labels*), polja za unos teksta (engl. *entry fields*) i sl., koji su neophodni za stvaranje interaktivnog sučelja. Također podržava

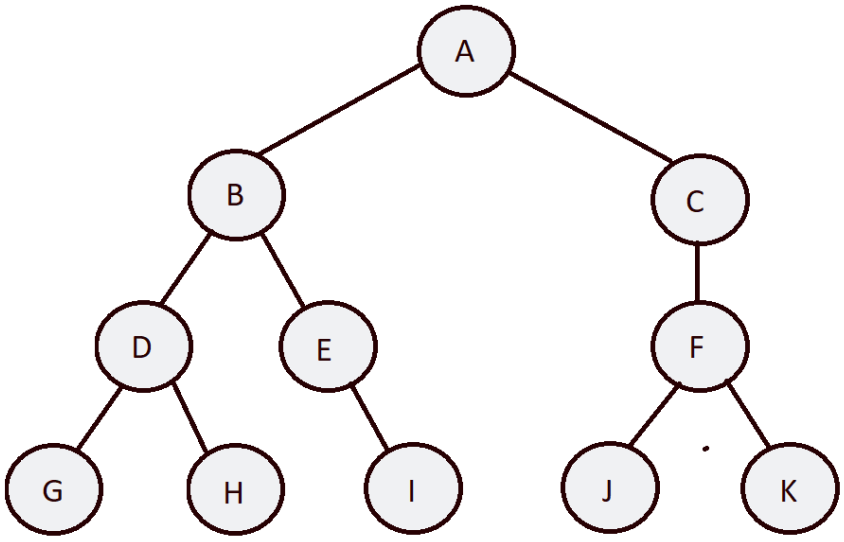
događaje (engl. *events*) i geometrijske menadžere (engl. *geometry manager*), koji omogućuju postavljanje grafičkih elemenata na prozoru.

2.2.2. Pillow

Pillow je napredna biblioteka za obradu slika u Pythonu, koja koristi depreciranu *Python Imaging Library* (PIL) biblioteku kao podlogu [11]. Jednostavna je za korištenje, a pokriva sve osnovne funkcije za digitalnu obradu slike [11].

2.3. Korišteni algoritmi

Za potrebe ove aplikacije odabran je algoritam pretrage u dubinu (DFS). To je algoritam za pretraživanje stabla ili grafa koji počinje od korijenskog čvora i istražuje što je moguće dalje duž određene grane, a kad više nema gdje za ići u tu stranu, krene se vraćati [14]. Graf sa slike (Sl. 2.1) će tako u ovom koraku krenuti iz čvora A i posjetiti redom čvorove B, D, G. Vraća se dok ne pronađe neistraženu djecu nekog čvora, nakon čega nju krene istraživati [14]. To će u ovom slučaju biti čvor D tj. njegovo dijete H. Postupak se ponavlja sve dok se ne istraži cijeli graf [14]. Ovaj graf (Sl. 2.1) će se pretraživati ovim redoslijedom: A, B, D, G, H, E, I, C, F, J, K.



Sl. 2.1 Primjer grafa

Neke prednosti DFS algoritma su to što nam garantira dolazak do rješenja ako ono postoji, jednostavan je za uporabu i ima malu prostornu složenost.

Vremenska složenost za ovaj algoritam iznosi $O(b^m)$, gdje *m* predstavlja maksimalnu dubinu stabla, koja u slučaju zagonetke Pentomino tek 12, jer postoji samo 12 pentomina koji se moraju postaviti. U ovom zadatku nam je problem *b*, što je faktor grananja stabla. Na zadatku veličine 12x12 koji nema postavljeno nijedno crno polje, za pentomino X je maksimalni faktor grananja $(12 - 2)^2 = 100$, za I je $2 \cdot (12 - 4) = 128$, a za pentomino F iznosi $8 \cdot (12 - 2)^2 = 800$. Znači da vremenska složenost algoritma za ovaj primjer iznosi $100 \cdot 128 \cdot \ldots \cdot 800$. To je velik problem, stoga su dodana razna ograničenja tako da algoritam brzo podreže granu koja ga sigurno ne vodi do točnog rješenja. Dodavanje svakog od ograničenja dovodilo je do višestrukog ubrzanja rješavanja zagonetke Pentomino.

2.3.1. DFS s propagacijom ograničenja

Pri uvođenju propagacije ograničenja, algoritam primjenjuje ograničenja na svaki čvor kako bi podrezao broj grana iz njega. To se radi tako da se ograničenja propagiraju kroz stablo, što znači da se primjenjuju na sve čvorove koji su izravno ili neizravno povezani s čvorom na kojem se ograničenje primjenjuje. Time se smanjuje broj mogućih rješenja koje treba provjeriti, što može znatno utjecati na brzinu dolaska do rješenja. U slučaju da se neko ograničenje pogrešno postavi, algoritam gubi svoju osobinu dolaska do rješenja, stoga je bitno dobro provjeriti prolazi li svako željeno rješenje svako od zadanih ograničenja.

3. Programsko rješenje zagonetke

3.1. Struktura projekta

Struktura projekta osmišljena je s ciljem stvaranja modularne, održive i skalabilne aplikacije. Projekt se sastoji od četiri glavne Python datoteke: *pentomino.py*, *dfs_cp.py*, *zagonetka.py* i *upis_dimenzija.py*. Svaka od ovih datoteka obavlja specifičan skup funkcija unutar sustava, omogućujući kôdu da ostane pregledan te lak za održavanje i nadograđivanje. Unutar njih su također napravljene razne provjere tako da suptilne greške ne prouzročavaju daljnje komplikacije.

Datoteka *pentomino.py* sadrži definicije svih pentomino oblika. Ona sadrži klasu *Pentomino* koja omogućava manipulaciju oblicima putem metoda poput *rotate()*, *mirror_horizontal()* i *get_all_configurations()*. Dodatno, datoteka sadrži funkciju *get_pentominos()* koja omogućava lako dohvaćanje svih objekata pentomino za upotrebu u drugim dijelovima projekta.

upis_dimenzija.py služi kao glavno korisničko sučelje aplikacije. Ovaj modul učitava broj redaka i stupaca tablice kroz grafičko sučelje te poziva funkciju *create_table()* iz *zagonetka.py.* To je funkcija za izradu tablice koja omogućuje korisniku unos zagonetke. Nakon unosa se pokreće DFS algoritam iz *dfs_cp.py*.

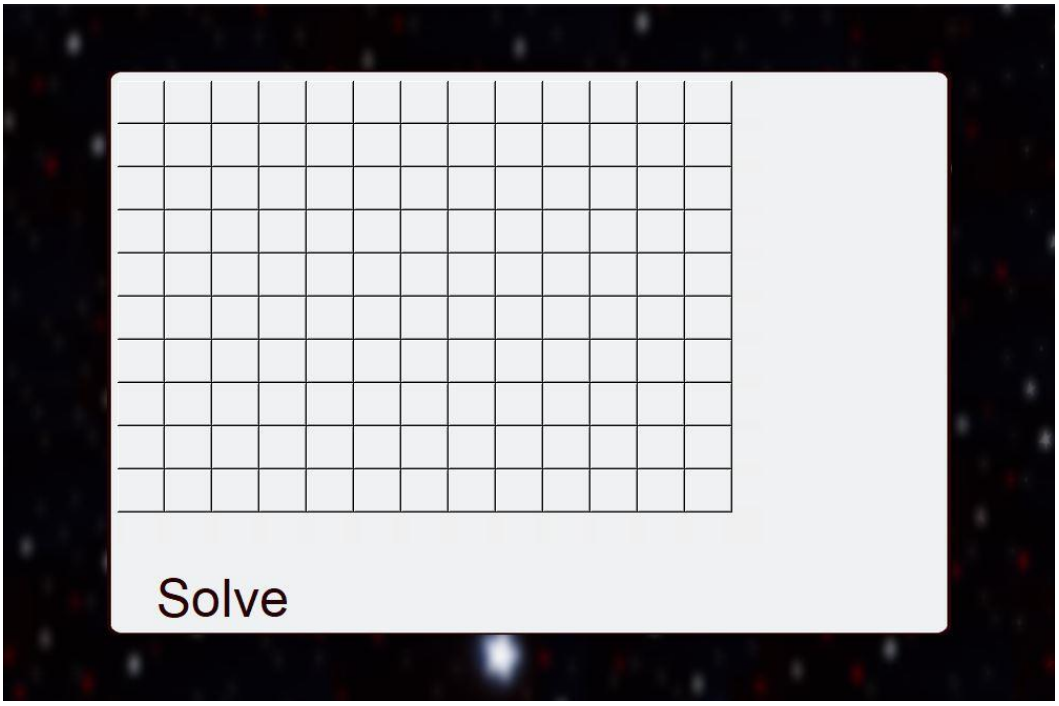
Datoteka *dfs_cp.py* sadrži algoritam DFS s propagacijom ograničenja za rješavanje pentomino zagonetki. Ova datoteka definira niz funkcija koje omogućuju manipulaciju i interakciju s pločom zagonetke, uključujući postavljanje i uklanjanje pentomina, postavljanje nula oko postavljenih oblika i provjeru valjanosti poteza.

3.2. Implementacija grafičkog sučelja

Datoteka *upis_dimenzija.py* učitava broj redaka i stupaca tablice kroz sučelje prikazano na slici (Sl. 3.1) i klikom na dugme *OK* poziva funkciju *create_table()* iz *zagonetka.py*. To je implementirano upotrebom ugrađenih funkcija i klasa iz Tkinter biblioteke. Instancirani su objekti klasa *Tk*, *PhotoImage*, *Label*, *Entry* i *Button*.

Sl. 3.1 GUI upis dimenzija

Nakon pritiska gumba događa se provjera unosa i ako je sve u redu izrađuje se prazna tablica željenih dimenzija kao što je prikazano na slici (Sl. 3.2).



Sl. 3.2 GUI ploča za unos zagonetke

Za prikaz tablice napravljena je klasa *Table*. Ona ima atribute kao što su *table_data* i *table_buttons*, koji su zapravo ove kvadratne čelije sa slike (Sl. 3.2). Ima i dvije liste koje

sadrže ograničenja za retke tj. stupce, gumb za pokretanje postupka rješavanja i poruku koja obavještava o ispravnosti zadane zagonetke vidljivu na slici (Sl. 3.3).

Sl. 3.3 Pokušaj unosa prazne ploče

3.3. Implementacija DFS-a s propagacijom ograničenja

Algoritam DFS s propagacijom ograničenja implementiran je u datoteci dfs_cp.py. Funkcionira tako da se prvo poziva funkcija dfs() koja kao argumente prima:

- ploču za igru (*board*)
- popis pentomina (*pentominos*)
- ograničenja za redove i stupce (*row_constraints*, *col_constraints*)

Prvo funkcija provjerava jesu li svi pentomini postavljeni (što znamo da je istina ako je lista pentominos prazna). Ako jesu, vraća kopiju trenutne ploče jer je problem uspješno riješen.

Ako postoji barem jedan pentomino koji još nije postavljen, funkcija uzima prvog iz liste i iterira kroz sve redove i stupce ploče za igru. Ako pentomino ima više konfiguracija, onda se za određenu poziciju prolazi kroz svaku od tih konfiguracija. Funkcija za svaku od njih provjerava je li moguće postaviti pentomino na ploču na taj način. To se provjerava pomoću funkcije *is_valid()*.

Ako je postavljanje pentomina valjano, pentomino se postavlja na ploču pomoću funkcije *place_pentomino().* Nakon postavljanja, funkcija provjerava jesu li ispunjena ograničenja za redove i stupce. Ako su sva ograničenja zadovoljena, funkcija rekurzivno poziva sama sebe (dfs) s preostalim pentominima.

Ako se pronađe rješenje u rekurzivnom pozivu (tj. funkcija ne vraća *None*), to rješenje se vraća. Ako rješenje nije pronađeno, pentomino se uklanja s ploče pomoću funkcije *remove_pentomino(),* a pretraživanje se nastavlja s drugim pozicijama i konfiguracijama.

Nakon što su sve moguće pozicije i konfiguracije isprobane, a rješenje nije pronađeno, funkcija vraća vrijednost *None*. To označava da zadana zagonetka nema rješenje.

3.3.1. Ograničenja

Sva se ograničenja pozivaju u *is_valid()* funkciji. Ona kao argumente prima:

- ploču za igru (*board*)
- konfiguraciju pentomina koji se želi postaviti (*pentomino*)
- polje na koje se pentomino pokušava postaviti (*row*, *col*)
- ograničenja za redove i stupce (*row_constraints*, *col_constraints*)

Prvo se za konfiguraciju pentomina provjerava bi li oblik izlazio iz ploče kad bi se postavio na odabrano mjesto. Ako bi, onda će funkcija vratiti vrijednost False.

```
for r, c in pentomino.coords:
    if row + r >= len(board) or col + c >= len(board[0]):
        return False
```

Nakon toga provjeravamo postoji li već neki drugi pentomino ili crno polje na nekom od mjesta na koja želimo postaviti pentomino:

```
for r, c in pentomino.coords:
    if board[row + r][col + c] != '1':
        return False
```

Sada kada znamo da su koordinate na koje želimo postaviti pentomino dostupne, simulirat ćemo postavljanje pentomina kako bismo lakše provjerili daljnja ograničenja.

```
temp_board = [row.copy() for row in board]
    temp_board = place_pentomino(temp_board, pentomino, row,
col)
```

Slijedeće ograničenje je malo komplicirnije pa ćemo ukratko prvo objasniti na primjeru. Prvi pentomino koji će doći do ovog koraka bit će pentomino X i on će ovdje biti postavljen na kopiju ploče na mjesta: (0,1), (1,0), (1,1), (1,2), (2,1).

Ovdje nam je potrebno nekoliko varijabli:

- Broj slobodnih mjesta u retku u koji se pokušava smjestiti novi pentomino:

```
        total_in_row = sum(1 for i in range(len(board[0])) if
temp_board[r][i] != '0')
```

- Broj mjesta zauzetih dijelovima pentomina u retku:

```
        pentomino_in_row = sum(1 for i in
range(len(board[0])) if temp_board[r][i] != '0' and
temp_board[r][i] != '1')
```

Prema slici (Sl. 1.1), u redu s indeksom 0 broj dijelova pentomina mora biti 8, a u redu s indeksom 1 je dozvoljeno zauzeti ploču sa samo 2 komadića pentomina. Ta su nam dva ograničenja prikazana u kôdu (Kôd 3.1), a ista provjera događa se naravno i za stupce. Zbog navedenih ograničenja funkcija će ovdje za prethodno navedeni primjer postavljanja pentomina X vratiti vrijednost False, čime će se pentomino X dalje pokušati postaviti na preostala mjesta.

Kôd 3.1 Ograničenja za retke

```
# Ako nema dovoljno mjesta u redu vrati False
        if total_in_row < int(row_constraints[r].get()):
            return False

# Ako broj zauzetih mjesta premašuje broj stvarnih zauzetih
mjesta vrati False
        if pentomino_in_row > int(row_constraints[r].get()):
            return False
```

Kad se pentomino X pokuša postaviti tako da mu središte bude na lokaciji (2,3) u tablici, zbog prva tri ograničenja za stupce taj pentomino ne smije doći na to mjesto. Dogodilo bi se da je treći stupac potpuno zauzet zbog tog pentomina. U prva dva stupca zbroj dijelova pentomina mora biti 13, što u situaciji kad bi se pentomino postavio više ne bi bilo moguće. Zaključujemo da ako bi se neki redak tj. stupac u potpunosti popunio unosom nekog pentomina, potrebno je provjeravati je li razlika zbroja ograničenja i zbroja već

postavljenih dijelova pentomina djeljiva s pet sa svake strane od u potpunosti zauzetog retka tj. stupca, kao što je to vidljivo u kôdu (Kôd 3.2).

Kôd 3.2 Provjera podjele ploče zbog popunjenih redaka

```
                    if pentomino_in_row == row_constraints[r]:
                constraints = sum(row_constraints[i] for i in
range(r))
                rows_count = sum(1 for j in range(r) for i in
range(len(board[0])) if temp_board[j][i] != '0' and
temp_board[j][i] != '1')
                if (constraints - rows_count) % 5 != 0:
                    return False
```

Posljednje dodano ograničenje radi provjeru „otoka“. U ovom kontekstu, otocima su nazvana područja veličine od dva do četiri polja koja su na ploči označena kao da se na njih može postaviti neki pentomino. Ta bi se polja trebala označiti kao nepovoljna tj. u njih bi se trebale postaviti nule i ona bi se trebala zacrniti, no mi smo ovdje išli još korak dalje jer su primjeri pokazali da takvi otoci ne postoje u rješenjima. Pretraga otoka odvija se jednostavnim DFS algoritmom, a ako se neki otok uistinu pronađe pokušajem postavljanja pentomina, funkcija *is_valid()* će vratiti vrijednost *False*.

3.3.2. Postavljanje i uklanjanje pentomina

Za postavljanje pentomina na skup polja koristimo funkciju place_pentomino(), a za otklanjanje pentomina funkciju remove_pentomino(). Obje funkcije kao argumente primaju:

- trenutnu verziju ploče (*board*)
- konfiguraciju pentomina koju treba postaviti tj. ukloniti (*pentomino*)
- broj redova i stupaca za koje je pentomino odmaknut od pozicije (0, 0) (*row*, *col*).

Funkcija *place_pentomino()* prvo za svaki dio pentomina kojeg postavljamo na ploču na željena mjesta postavi ime tog pentomina, nakon čega na ploči pentomino „okružimo“ nulama koje označavaju crno polje tj. da se na to mjesto više ne može postaviti niti jedan pentomino, kao što je prikazano u kôdu (Kôd 3.3).

Kôd 3.3 Postavljenje pentomina

```
def place_pentomino(board, pentomino, row, col):
    for r, c in pentomino.coords:
        board[row + r][col + c] = pentomino.name

    for r, c in pentomino.coords:
        board = set_zeros_around(board, pentomino, row + r,
col + c)

    return board
```

Suprotnu stvar radimo u funkciji *remove_pentomino()* prikazanoj u kôdu (Kôd 3.4). Mjesta koja smo prethodno označili kao dijelove nekog pentomina vraćamo na inicijalne vrijednosti jedan, a zatim isto radimo s nulama koje su postavljene zbog tog pentomina.

Kôd 3.4 Uklanjanje pentomina

```
def remove_pentomino(board, pentomino, row, col):
    for r, c in pentomino.coords:
        if board[row + r][col + c] == pentomino.name:
            board[row + r][col + c] = '1'
    for r, c in pentomino.my_zeros:
        board[r][c] = '1'

    pentomino.my_zeros.clear()
    return board
```

4. Testiranje i uporaba aplikacije

U ovom poglavlju prikazani su rezultati testiranja aplikacije za rješavanje zagonetki Pentomino i upute za korištenje aplikacije, a usput ćemo komentirati moguća poboljšanja. Testirani su unos zagonetke i algoritam koji se koristi pri njenom rješavanju. Prilikom izrade algoritma za rješavanje zagonetki, fokus je bio na točnosti i brzini pronalaženja rješenja. Rezultati testiranja pokazali su da algoritam konzistentno daje točne rezultate unutar očekivanog vremenskog okvira.

4.1. Testiranje unosa zagonetke

Pošto su standardne veličine ove zagonetke ili 10x13 ili 12x12, aplikacija prihvaća zagonetke veličina od 10x10 do 13x13. Ako se za broj redaka ili stupaca unese broj koji je manji od 10 ili veći od 13, ploča tih veličina se neće napraviti nego će se čekati da korisnik unese ispravne dimenzije. Ista stvar se događa ako korisnik unese neki znak koji nije broj ili polje za unos bude prazno.

Nakon što korisnik ispravno unese dimenzije zagonetke, otvorit će se novi prozor u kojem se nalazi prazna ploča. Korisnik sad treba unijeti ograničenja za redove i stupce, a ako je potrebno može unijeti i crna polja. Ako pogrešno zacrni neko polje, to će lako popraviti ponovnim klikom na to polje. Ako neko od ograničenja za redak ili stupac ne bude uneseno ili u polje bude uneseno nešto što nije broj, ispisat će se obavijest o pogrešnom unosu i korisnik će moći ponovno unijeti zagonetku koju je potrebno riješiti.

Kad god se smatra da je zagonetka unesena pogrešno, korisniku je omogućeno da jednostavno nastavi s unosom zagonetke. Ako je zagonetka pogrešno unesena, a upisana je zagonetka rješiva, unos se neće smatrati pogrešnim i prvo pronađeno rješenje će biti ispisano. Nakon što je neka zagonetka riješena, unos u taj prozor više nije moguć jer se smatra da je brže otvoriti novi prozor i tamo upisati novu zagonetku.

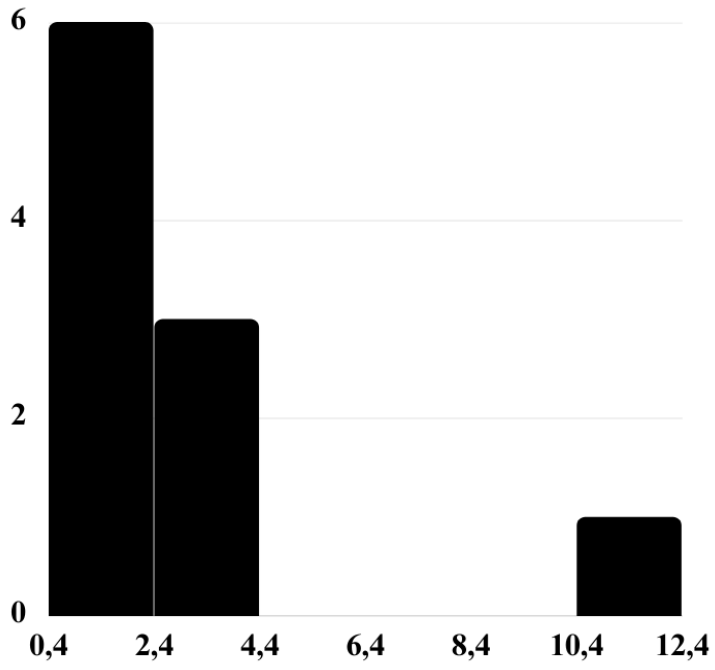
4.2. Testiranje algoritma

Testiranje algoritma provodeno je na uzorku od 10 zagonetki. Prvih 9 zagonetki su veličine 10x13, a posljednja je veličine 12x12. S obzirom na to da se za svaku zagonetku provodi isti algoritam i on se provodi svaki put na isti način, nije bilo potrebe za testiranjem iste zagonetke više puta jer bi rezultati bili identični.

Uvođenjem propagacije ograničenja rješavanje zagonetki se svelo s više minuta na trenutni prosjek od tek 3 sekunde, što se vidi u tablici 4.1 te na histogramu na slici 4.1. Sva dobivena rješenja su ispravna.

Tablica 4.1. Testiranje vremena rješavanja zagonetki

Redni broj	1	2	3	4	5	6	7	8	9	10	Prosjek
Vrijeme rješavanja [s]	3.6	1.7	11.2	1.6	1.7	3.9	4.1	1.3	0.4	0.5	3



Sl. 4.1 Vrijeme izvođenja prikazano pomoću histograma

4.2.1. Moguća poboljšanja

Dodatna ubrzanja mogla bi se postići tako da se prije pozivanja DFS algoritma koristi još jedna funkcija. Ona bi popunila dio tablice tako da se polja koja sigurno sadrže dio nekog pentomina ostave bijela, ali im se umjesto broja 1 pridijeli neka druga vrijednost (npr. 2). Sigurno prazna polja bi se zacrnila tj. u toj bi im funkciji bila dodijeljena vrijednost 1. Tako bismo postupak rješavanja ove zagonetke učinili mnogo sličnijim optimalnom načinu rješavanja kojeg koriste ljudi.

5. Zaključak

Aplikacija za rješavanje pentomino zagonetki dizajnirana je s namjerom da bude intuitivna i jednostavna za upotrebu, kako bi korisnicima bilo što lakše unijeti vlastite zagonetke i doći do njihova rješenja. Kad god je pogreška u unosu uočljiva, omogućen je i što jednostavniji oporavak od nje. Kada je zagonetka dobro unesena, korisnici klikom na gumb *Solve* pokreću proces rješavanja.

Za pronalazak rješenja koristi se DFS algoritam s propagacijom ograničenja. Osnovna ideja algoritma DFS je da se kreira stablo pretraživanja koje predstavlja sve moguće kombinacije rješenja problema, a propagaciju ograničenja primjenili smo kako bi se stablo podrezivalo tj. kako bi proces dolaska do rješenja bio što brži. Rješenje se prikazuje unutar originalne tablice, osiguravajući korisnicima jasan pregled rezultata.

Testiranje aplikacije provedeno je kako bi se osigurala njena učinkovitost i pouzdanost. Testovi su pokazali da aplikacija dosljedno i točno rješava različite Pentomino zagonetke unutar očekivanog vremenskog okvira. Kao rezultat, aplikacija pruža pouzdano rješenje za sve korisnike zainteresirane za rješavanje Pentomino zagonetki.

Literatura

[1] Tutorialspoint. *Python - GUI Programming (Tkinter).* Poveznica: https://www.tutorialspoint.com/python/python_gui_programming.htm#:~:text=Tkinter%20is%20the%20standard%20GUI,to%20the%20Tk%20GUI%20toolkit.; pristupljeno 3. svibanj 2023.

[2] Tutorialspoint. *Python – Overview.* Poveznica: https://www.tutorialspoint.com/python/python_overview.htm; pristupljeno 3. svibanj 2023.

[3] Centre for Innovation in Mathematics Teaching. *PENTOMINOES – An Introduction*. Poveznica: https://www.cimt.org.uk/resources/puzzles/pentoes/pentoint.htm; pristupljeno 23. svibanj 2023.

[4] Golomb, S.W. *Polyominoes.* 2. izdanje. Princeton: Princeton University Press, 1994. Poveznica: https://www.google.hr/books/edition/Polyominoes/JdPgDwAAQBAJ?hl=en&gbpv=1&dq=Simon+W+Golomb+Princeton+University+Books+1994+ISBN+0-691-08573-0&printsec=frontcover; pristupljeno 25. svibanj 2023.

[5] Horváth, Z. *Pentomino by Zoltán Horváth*. Grandmaster Puzzles. Puzzle Championship, Mađarska, (2014) Poveznica: https://www.gmpuzzles.com/blog/2014/10/pentomino-zoltan-horvath/; pristupljeno 10. ožujak 2023.

[6] Mulcahy, C. *The Top 10 Martin Gardner Scientific American Articles*, Scientific American, (2014, listopad). Poveznica: https://blogs.scientificamerican.com/guest-blog/the-top-10-martin-gardner-scientific-american-articles/; pristupljeno 24. svibanj 2023.

[7] Mulcahy, C. *Five Martin Gardner eye-openers involving squares and cubes,* Plus, (2014, listopad) Poveznica: https://plus.maths.org/content/five-martin-gardner-eye-openers-involving-squares-and-cubes; pristupljeno 24. svibanj 2023.

[8] Henry Dudeney. *The Canterbury Puzzles and Other Curious Problems* (1907). The Broken Chessboard. Poveznica: http://djm.cc/library/The_Canterbury_Puzzles_Dudeney_edited.pdf; pristupljeno 24. svibanj 2023.

[9] Python™. *TkInter.* Poveznica: https://wiki.python.org/moin/TkInter; pristupljeno 3. svibanj 2023.

[10] Franklin, T. *Games Featuring Polyominoes.* Meeple mountain (2022). Poveznica: https://www.meeplemountain.com/topics/games-featuring-polyominoes/; pristupljeno 23. svibanj 2023.

[11] About. *Pillow (PIL Fork) 9.5.0 documentation*. Poveznica: https://pillow.readthedocs.io/en/stable/about.html; pristupljeno 3. svibanj 2023.

[12] Python Institute. *Python® – the language of today and tomorrow*. Poveznica: https://pythoninstitute.org/about-python; pristupljeno 3. svibanj 2023.

[13] Python™. *tkinter - Python interface to Tcl/Tk.* Poveznica: https://docs.python.org/3/library/tkinter.html; pristupljeno 4. svibanj 2023.

[14] Brilliant. *Depth-First Search (DFS).* Poveznica: https://brilliant.org/wiki/depth-first-search-dfs/; pristupljeno 8. svibanj 2023.

Sažetak

Tema ovog rada je upoznavanje s logičkom zagonetkom Pentomino, njenim pravilima i povijesti pentomina. Pričat ćemo o optimalnom načinu rješavanja ove zagonetke i implementirati jednostavno programsko rješenje korištenjem DFS algoritma s propagacijom ograničenja. Dobivene rezultate ćemo testirati i prokomentirati.

Ključne riječi: Pentomino, logička zagonetka, Python, GUI, aplikacija, kvadrat, poliomino

Pentomino; logička zagonetka; Python; GUI; aplikacija; kvadrat; poliomino

Summary

The topic of this paper is an introduction to the logic puzzle Pentomino, its rules and the history of pentominoes. We will discuss the optimal way to solve this puzzle and implement a simple software solution using the DFS algorithm with constraint propagation. The obtained results will be tested and commented upon.

Keywords: Pentomino, logic puzzle, Python, GUI, application, square, polyomino.

Skraćenice

GUI	*Graphical User Interface*	grafičko korisničko sučelje
DFS	*Depth First Search*	algoritam pretrage u dubinu
PIL	*Python Imaging Library*	biblioteka za manipulaciju slikama

Privitak

Upute za korištenje aplikacije

U radu je osim programskog jezika Python korišteno još nekoliko njegovih biblioteka. Korisnici bi taj proces instalacije mogli shvatiti kao zahtjevan proces, što bi moglo negativno utjecati na njihovu zainteresiranost za isprobavanjem ove aplikacije.

Stoga je korištenjem biblioteke *pyInstaller* i alata *NSIS* izrađena *.exe* datoteka, za čije pokretanje korisnik ne treba preuzimati ništa osim nje. Dobivenu *.exe* datoteku je potrebno instalirati što se radi njenim pokretanjem i pritiskom na dugme *Install*. U odabranoj će se mapi izraditi mapa *Pentomino* u koju je potrebno ući. Klikom na datoteku *upis_dimenzija.exe* pokrenut će se ova aplikacija.