Automat za igranje igre "Poveži 4"

Šišić, Ilija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

AUTOMAT ZA IGRANJE IGRE "POVEŽI 4"

Diplomski rad

Ilija Šišić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 13.07.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Ilija Šišić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-964R, 29.09.2019.
OIB studenta:	87227756248
Mentor:	doc. dr. sc. Ivan Aleksi
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Prof.dr.sc. Željko Hocenski
Član Povjerenstva 1:	doc. dr. sc. Ivan Aleksi
Član Povjerenstva 2:	Doc.dr.sc. Tomislav Matić
Naslov diplomskog rada:	Automat za igranje igre "Poveži 4"
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	13.07.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.07.2020.

Ime i prezime studenta:	Ilija Šišić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-964R, 29.09.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Automat za igranje igre "Poveži 4"**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Aleksi

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Ilija Šišić_____, OIB: __87227756248_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Diplomski sveučilišni studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _Automat za igranje igre "Poveži 4"**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 21.07.2020._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Ovaj diplomski rad ima u cilju simulirati popularnu igru Poveži 4 na mikrokontroleru tako da se na LE diodama prikazuje stanje na ploči i da se omogući igranje igre u dva načina, čovjek protiv čovjeka i čovjek protiv računala, uz dvije težinske razine (teško i lako) za igru protiv računala. Igra protiv računala će biti odrađena pomoću *minimax* algoritma s alfa-beta rezanjem.

Potrebno je dizajnirati tiskanu pločicu (engl. *printed circuit board*) za što se koristi EAGLE aplikacija. Za to će biti potrebno dizajnirati električnu shemu te poredati sve komponente na pločicu i povezati ih kao na shemi.

Igra Poveži 4 je igra za dva igrača koji igraju naizmjenično. Cilj igre je, kao što i samo ime predlaže, da se povežu 4 žetona vodoravno, okomito ili dijagonalno. Igra sama po sebi nije komplicirana za igrati, ali je komplicirano za mikrokontrolere da simuliraju igranje igre jer ima 4,531,985,219,092 mogućih kombinacija na igraćoj ploči i u svakom potezu možemo igrati 7 različitih polupoteza.

Rad se sastoji od pet poglavlja. Prvo poglavlje nakon uvoda (drugo poglavlje u radu) opisuje aktuelne znanstvene i praktične dosege u području rada (engl. *State of the Art*). Zatim slijedi poglavlje koje govori o pravilima i povijesti igre „Poveži Četiri“. Četvrto poglavlje opisuje sve komponente, alate i algoritme korištene u izradi rada. U petom poglavlju je detaljno opisan programski kod, kao i dizajniranje i izrada tiskane pločice. Posljednje poglavlje je testiranje sustava u kojem je vidljivo kako izgleda jedna partija igre Poveži 4 na realiziranom automatu.

2. PREGLED PODRUČJA TEME

Umjetna inteligencija (engl. *Artificial Intelligence*) je znanost koja se bavi proučavanjem „racionalnih agenata“ tj. svakog stroja ili uređaja koji dobiva podatke iz okoliša i poduzima sve potrebne korake kako bi maksimizirao šansu uspješnog izvršavanja zadatka. Predstavlja inteligenciju prikazanu od strane strojeva i/ili uređaja za razliku od prirodne inteligencije koju posjeduju ljudi i životinje. Danas, umjetna inteligencija je jako napredna grana računarstva iako svi dosadašnji oblici umjetne inteligencije spadaju u ograničenu umjetnu inteligenciju. Ograničena umjetna inteligencija podrazumijeva svaku umjetnu inteligenciju koja je ograničena na rješavanje samo određenih problema, odnosno svaku koja nema vlastitu svijest ili razumijevanje. [1]

Kako strojevi vremenom postaju sve sposobniji tako rastu i mogućnosti umjetne inteligencije. Moderni strojevi su sposobni razumjeti i simulirati ljudski govor [1], igrati, pa i natjecati se sa velemajstorima raznih strateških igara, kao što su šah ili Go. Također autonomno upravljanje vozilima i vojne simulacije su realizirane pomoću umjetne inteligencije. [2]

Početkom 1980. godina, povećanjem broja tranzistora u digitalnom sklopovlju omogućene su veće procesorske brzine, a time i razvoj umjetnih neuronskih mreža (engl. *Artificial neural network*). Umjetna neuronska mreža predstavlja skup međusobno povezanih umjetnih neurona rađenih po uzoru na ljudski mozak. Među posljednjim dostignućima u upotrebi umjetne inteligencije u društvenim igrama je *Leela Chess Zero* koji je 2019. godine pobijedio *Stockfish 19050918* u meču od 100 igara sa rezultatom 53.5 naprema 46.5. *Leela Chess Zero* je besplatni, *open source* program za igranje šaha (engl. *chess engine*) baziran na neuronskoj mreži. Kako sama neuronska mreža nema nikakvo znanje igranja igre potrebno je da ista nauči. Tako je *Leela Chess Zero* odigrala preko 300 milijuna partija sama protiv sebe. [3] Razlika između *Leela Chess Zero* i *Stockfish*-a jest ta što je *Stockfish* „konvencionalni“ program za igranje šaha, tj. on je baziran na naprednom *alfa-beta* rezanju uz korištenje *bitboard*-a. [4]

U ovom radu se koristi *minimax* algoritam sa *alfa-beta* rezanjem koji spada u početke razvoja umjetne inteligencije no još uvijek se koristi za određivanje poteza najsnažnijih *chess engine-a*. Taj algoritam se koristi za pretraživanje igraćih stabala te, na osnovu heurističkih funkcija, bira najbolji potez. [1] Najveći ograničavajući faktor implementacije umjetne inteligencije na mikrokontroler jest mala radna frekvencija i ograničena količina memorije.

3. POVEŽI ČETIRI

3.1. Općenito o igri

Poveži četiri (engl. *Connect Four*) je igra namijenjena za dva igrača koja je prvi put prodavana od strane Milton Bradley-a, danas poznatiji pod imenom Hasbro, početkom 1974. godine. Postoje sumnje da je nastala još u 18. stoljeću, ali nema dovoljno dokaza da bi se sumnje potvrdile. [5]

Igra se sastoji od uspravne igraće ploče i 42 žetona, od kojih je podjednak broj žutih i crvenih žetona. Na ploči se nalazi 7 stupaca i 6 redaka i igra se na način da što se žetoni ubacuju u stupce kroz utore koji se nalaze na vrhu igraće ploče tako da isti padaju na dno ploče, odnosno prvo slobodno mjesto u stupcu, gledano odozdo.

Poveži Četiri je igra sa „savršenom informacijom“ (engl. *game of perfect information*) što znači da svaki igrač vidi sve svoje i protivničke žetone i poznate su mu sve mogućnosti protivnika u tom trenutku. Jedna mjera složenosti igre Poveži Četiri je broj mogućih pozicija na igraćoj ploči. Za klasičnu igraću ploču od 6 redaka i 7 stupaca postoji 4,531,985,219,092 mogućih kombinacija. [6] Broj mogućih kombinacija se dobije tako što se za svaki potez izračuna broj mogućih kombinacija i onda se sve kombinacije zbroje. Unatoč velikom broju mogućih kombinacija igru je u potpunosti riješio Victor Allis 1988. godine [7], što znači da se ishod igre uvijek može sa sigurnošću predvidjeti pod pretpostavkom da oba igrača igraju savršeno.

3.2. Pravila

Na početku igre jedan igrač izabire boju žetona i redoslijed igranja poteza. Zatim oba igrača naizmjenično ubacuju žetone u cilju da povežu svoja četiri žetona i da spriječe protivnika da učini isto. Kao što i samo ime predlaže cilj igre je dijagonalno, okomito ili vodoravno povezati četiri žetona iste boje. Ako oba igrača ostanu bez žetona tj. nakon 42 odigrana polupoteza (21 polupotez po igraču) i da niti jedan nije ostvario pobjedu igra se završava neriješeno.

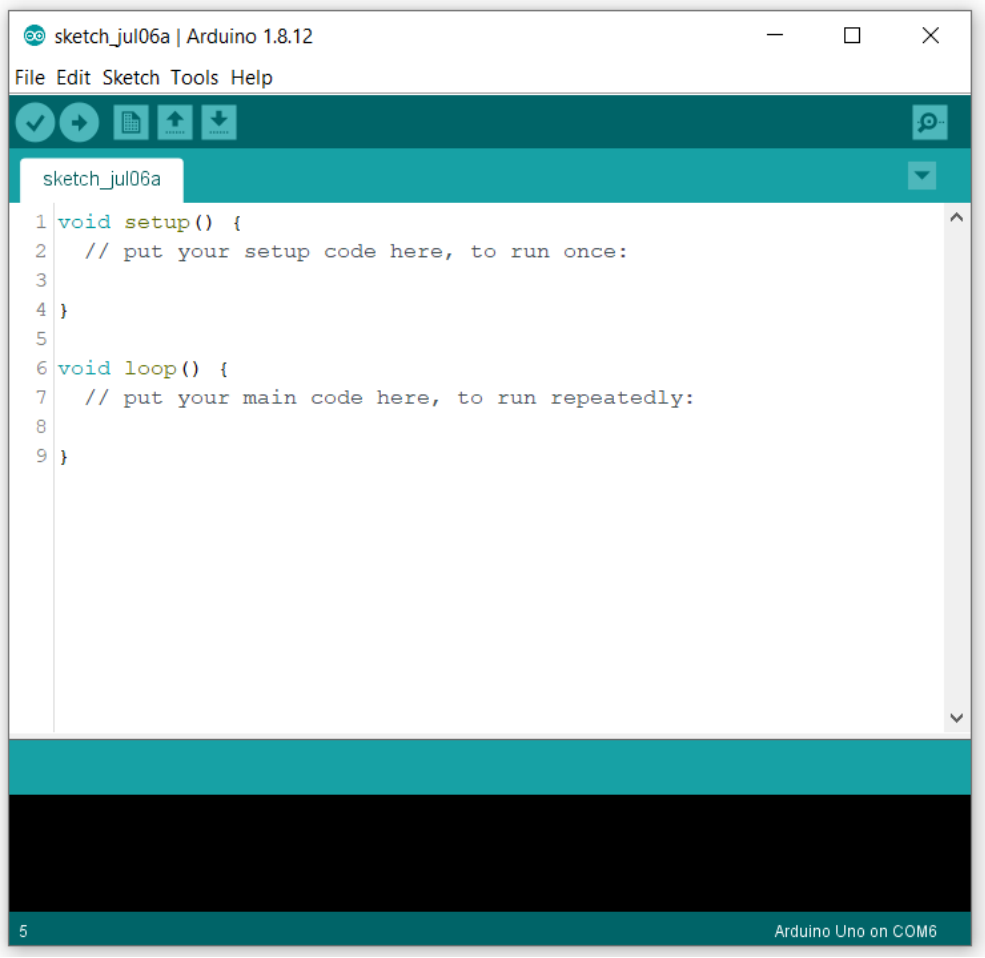
4. ALATI, KOMPONENTE I AKGORITMI

4.1. Alati

4.1.1. Arduino IDE

Arduino IDE ili Arduino integrirano razvojno okruženje (engl. *Integrated Davelopment Environment*), razvijeno u programskom jeziku Java, je *open-source* softver koji olakšava pisanje i učitavanje koda na Arduino, ili neku drugu pločicu. Podržava C i C++ programske jezike uz posebna pravila strukturiranja koda. Svaki kod napisan u Arduino IDE je sastavljen od dvaju osnovnih funkcija:

- *Setup*() –izvodi se samo jednom kada se program pokrene prvi put ili kada se mikrokontroler resetira
- *Loop*() –u suštini beskonačna petlja



Slika 4.1. Arduino IDE

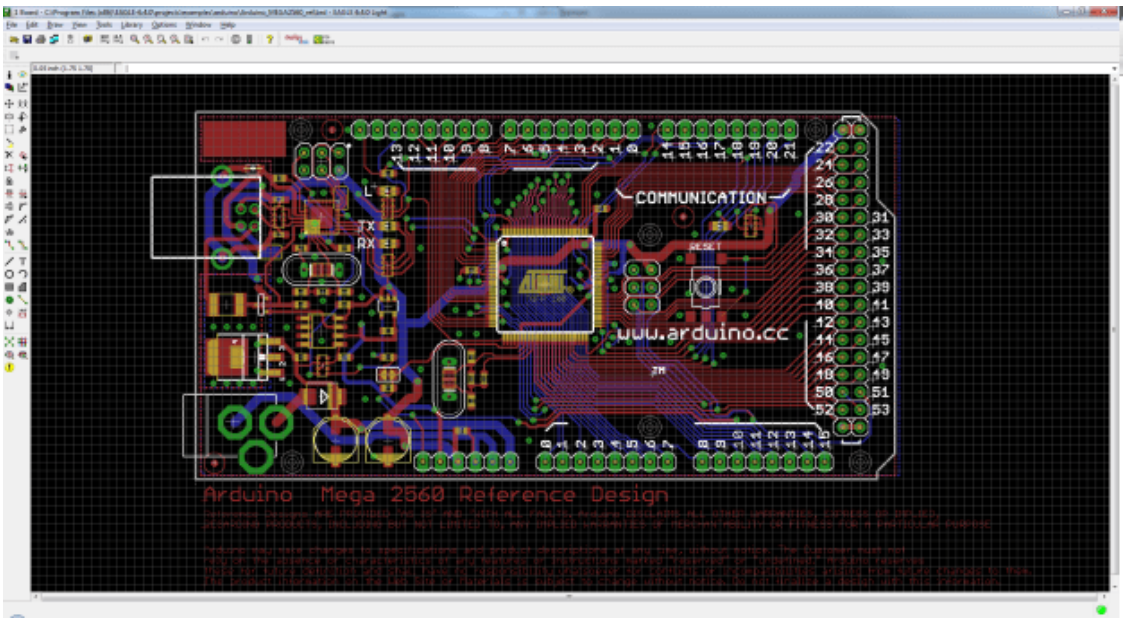
Kao i mnoga druga razvojna okruženja i Arduino IDE se može proširiti korištenjem *library*-a. Većina hardverskih komponenti imaju svoje vlastite *library*-e koji se mogu dodati razvojnom okruženju da bi se pojednostavnio rad s istim. Da bi se dodao *library* prvo se mora skinuti s interneta. Nakon toga se može odabrati *Sketch* > *Include Library*.

Za realizaciju ovog projekta koriste se sljedeći library-i:

- *Adafruit NeoPixel* – omogućava rad s adresabilnim LE diodama
- *Wire* – omogućava komunikaciju s I2C i TWI uređajima
- *Adafruit MPR121* – omogućava rad sa MPR121 kapacitivnim senzorom

4.1.2.Eagle

EAGLE je EDA (engl. *electronic design automation*) ili aplikacija za automatizaciju elektroničkog dizajna s mogućnosti dizajniranja shema i tiskanih pločica. EAGLE je skraćenica za jednostavno upotrebljiv grafički uređivač izgleda (engl. *Easily Applicable Graphical Layout Editor*) i razvijen je od *CadSoft Computer GmbH* tvrtke koju su osnovali *Rudolf Hofer* i *Klaus-Peter Schmidingler* 1988. godine. Tvrtku je kupio *Autodesk Inc.* 2016. godine. [8]



Slika 4.2. EAGLE strujna shema (engl. *circuit diagram*)

EAGLE se sastoji od dva glavna dijela:

- *Schematic editor* – ili shematski urednik je dio EAGLE-a zadužen za crtanje shema, što je i prvi korak u dizajniranju tiskane pločice.

- *Circuit diagrams* – ili strujna shema se može otvoriti nakon se završili dizajniranje sheme. Pri otvaranju se sve komponente koje se imaju u shemi postavljaju u donji lijevi kut te se odatle postavljaju na željeno mjesto na tiskanoj pločici.

4.2. Komponente

4.2.1. Wemos D1 mini Lite

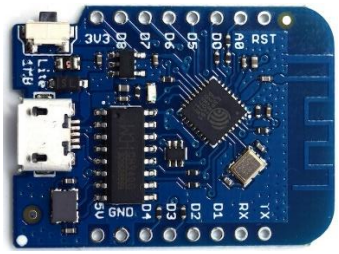
Kao „mozak“ projekta koristi se razvojna pločica Wemos D1 mini Lite iz razloga što ima veliku radnu frekvenciju što će biti potrebno da bi se program mogao izvršavati zadovoljavajućom brzinom.

Wemos D1 mini Lite je minijaturna razvojna pločica sa ESP8285 SoC (engl. *system on chip*). Radni napon mu je 3.3V. Ima 11 digitalnih I/O pinova i 1 analogni na kojim je maksimalni ulazni napon 3.2V. Radna frekvencija mu je do 160MHz.

Programiranje *Wemos D1 mini Lite*-a je jednostavno kao i programiranje bilo koje druge Arduino pločice pošto ima u sebi ugrađeno *microUSB* sučelje koje mu omogućuje da bude direktno programiran iz Arduino IDE razvojnog okruženja.

Tablica 4.1. Specifikacije *Wemos D1 mini Lite*

Radni napon	3.3V
Digitalni ulazno/izlazni pinovi	11
Analogni ulazni pinovi	1 (3.2V Max)
Radna frekvencija	80/160 MHz
Flash memorija	1M Bytes



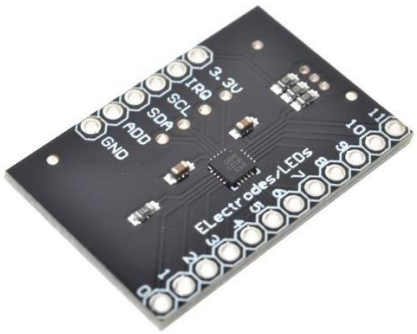
Slika 4.3. Wemos D1 mini Lite

4.2.2. MPR121

Za obradu ulaznih podataka koristi se MPR121QR2 breakout pločica koja olakšava rad sa MPR121 integriranim kontrolerom jer je sam integrirani kontroler jako mali i problematičan za lemljenje. MPR121 je kapacitivni senzor dodira. Integrirani kontroler može kontrolirati do 12

individualnih elektroda. Za komunikaciju sa mikrokontrolerom koristi I2C sabirnicu što je još jedan od razloga odabira ovog senzora jer se može spojiti veliki broj dodirnih pločica, a da se iskoristi minimalan broj ulaznih pinova na mikrokontroleru.

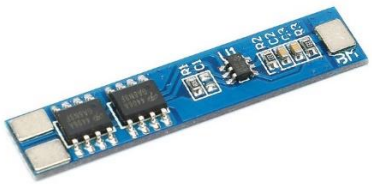
Ovaj integrirani kontroler radi na ulaznom naponu od 3 V i, kako na breakout pločici nema naponske regulacije, da bi integrirani kontroler radio ulazni napon mora biti u granicama od 2.5 V do 3.6 V.



Slika 4.4. MPR121

4.2.3.HX 2S 01

HX-2S-01 je BMS ili sustav upravljanja baterijama (engl. Battery management system) koji služi da štiti bateriju od prepunjivanja i od pretjeranog pražnjenja. Tako da je maksimalan napon na koji se baterija može dovesti 4.25 V-4.35 V dok je najniži napon od 2.3 V do 3.0 V uz dozvoljeno odstupanje od 0.05 V. Također postavlja ograničenja i na jakost struje koju baterija može generirati tako da je maksimalna kontinuirana struja 5 A dok je maksimalna trenutna struja 7 A.



Slika 4.5. HX-2S-01 sustav upravljanja baterijama

Uz ovaj sustav mogu se puniti dvije Li-Ion baterije spojene u serijsku vezu tako da se na ulaz dovede napon od 8.4 V do 9.0 V. Imajući to na umu ovaj projekt će se napajati pomoću dvije Li-Ion baterije spojene u seriju.

4.2.4.LM2596 DC to DC

Pošto se za napajanje koriste dvije Li-Ion baterije povezane serijski potreban je *buck converter*. *Buck converter-i* služe za spuštanje napona istosmjerne struje. Također postoje i *boost converteri* koji služe za podizanje napona istosmjerne struje. Koristeći *buck converter* spušta se napon baterija sa 8.4 V na 5 V s kojim će se napajati LE diode i Wemos D1 mini Lite koji ima u sebi regulator napona tako da se može napajati i sa 5 V.

Slika 4.6. LM2596 DC to DC *buck converter*

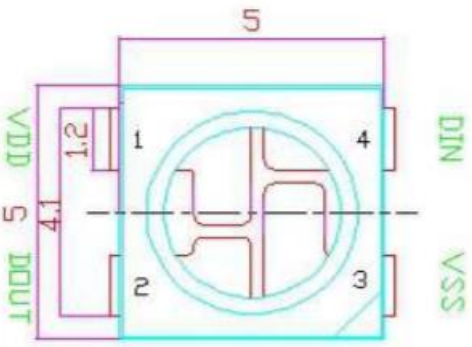
Za ovaj projekt je odabran LM2596 DC to DC zbog svoje velike efektivnosti što znači da će biti manja potrošnja i uz to će se rjeđe morati puniti baterije.

4.2.5.LE diode

Za prikaz stanja na igraćoj ploči koriste se WS2812b, adresabilne LE diode (engl. *Light emitting diode*). Ove LE diode su adresabilne što znači da se može upravljati s njima u stvarnom vremenu (engl. *real time*). To je bitno jer se pali nova LE dioda svaki put kada igrač odigra svoj potez. Također ove diode su i RGB (engl. *Red Green Blue*), dakle može se postavljati i proizvoljna boja.

Kao što je vidljivo na slici 3.8. WS2812b ima 4 pina:

- DI – ulazni upravljački signal
- DO – izlazni upravljački signal
- VDD – ulazni napon
- GND – uzemljenje



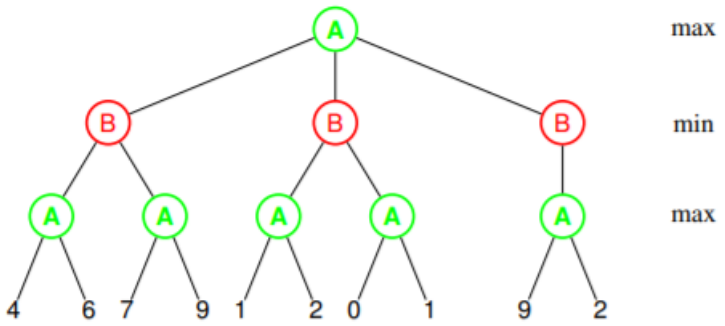
Slika 4.7. WS2812b LE dioda

4.3. Algoritmi

4.3.1. *Minimax* algoritam

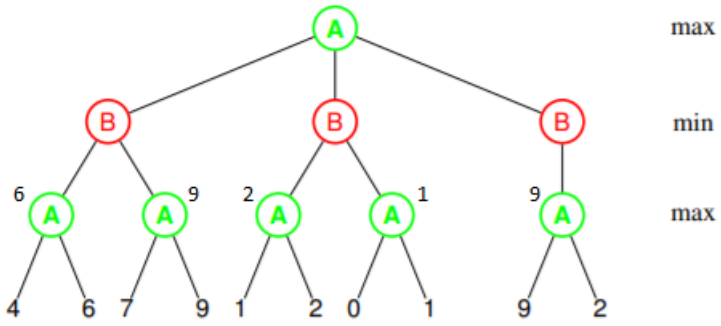
Budući da je „Poveži četiri“ igra s potpunim informacijama, što znači da su u potpunosti poznati svi mogući protivnički koraci, pretpostavlja se da će protivnik svaki put igrati potez koji će najviše škoditi njegovu protivniku odnosno njemu donijeti najveću dobit. Za pretraživanje stabla igre u tom slučaju koristiti se *minimax* algoritam. Ovaj algoritam je najlakše zamisliti kao dvije osobe koje igraju jedna protiv druge. Obje osobe znaju stanje na ploči u svakom trenutku i pokušavaju predvidjeti što će druga osoba uraditi uz pretpostavku da će ta druga osoba igrati u svoju korist odnosno da ostvari što veću dobit. [1]

Minimax algoritam radi na sličan način. Postoje dva igrača, jedan je nazvan A (računalo), a drugi B (protivnik). Igrač A nastoji maksimizirati svoj dobitak, dok igrač B nastoji minimizirati dobitak igrača A. Pojedine vrijednosti čvorova se dobiju iz heurističkih funkcija. U igri „Poveži četiri“ ta funkcija bi vraćala rezultat stanja ploče. Igrači igraju naizmjenično i svaki igrač koristi strategiju da minimizira maksimalan gubitak.



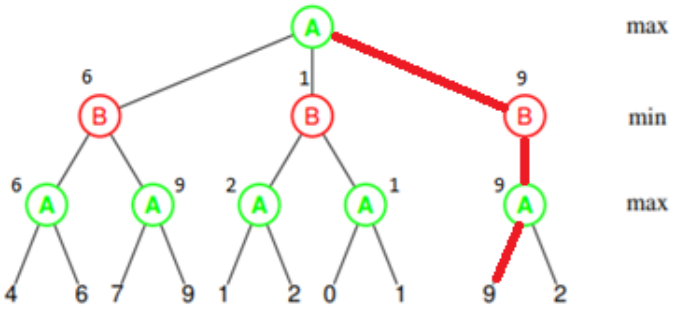
Slika 4.8.

Pretraživanje kreće od korijena stabla. Kao što je vidljivo na slici 3.9. čvor korijena stabla pripada igraču A. Kako igrač A nastoji maksimizirati svoj dobitak vrijednosti koje će poprimiti bit će maksimalne vrijednosti njegovih listova.



Slika 4.9.

Sada postoje nove vrijednosti listova (Slika 3.10.) no sada je na redu igrač B koji nastoji minimizirati dobitak igrača A što znači da će sada uzimati minimalne vrijednosti listova. Tako da se dobiva:



Slika 4.10.

Kao što je vidljivo, igrač B je izabrao sve minimalne vrijednosti, no sada je ponovno na redu igrač A koji će odabrati maksimalnu vrijednost od nova tri lista što je upravo 9. Na slici 3.11. crvenom je bojom prikazana najbolja strategija za igrača A.

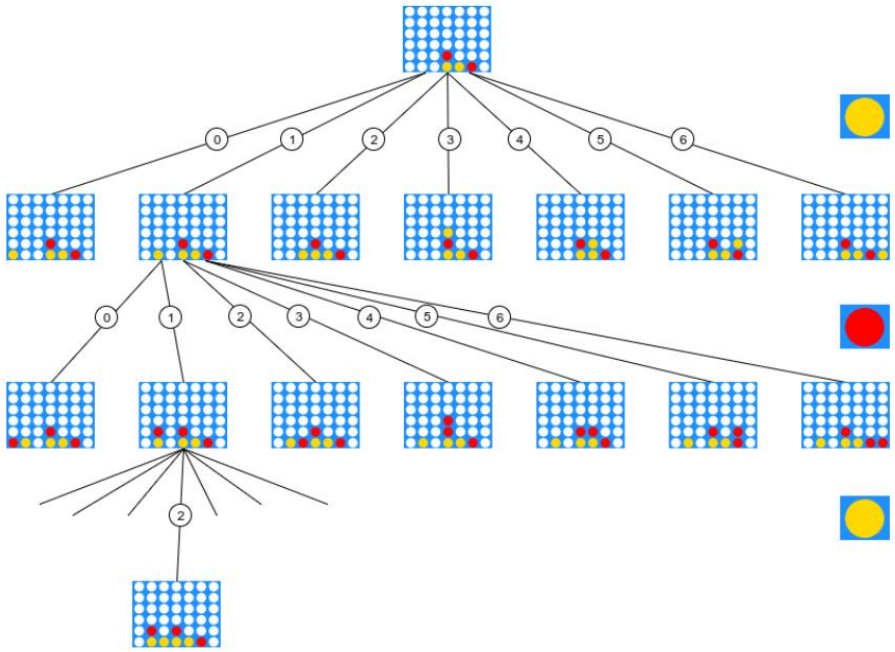
```
function min_max(cvor, dubina, maksimizirajuciIgrac) is
    if dubina = 0 or cvor is terminalni cvor then
        return heuristicku vrijednost of cvor
    if maksimizirajuciIgrac then
        vrijednost := -∞
        for each dijete of cvor do
            vrijednost := max(vrijednost, min_max(dijete, dubina - 1, FALSE))
        return value
    else (* minimizirajuci igrac *)
        vrijednost := +∞
        for each dijete of cvor do
            vrijednost := min(vrijednost, min_max(dijete, dubina - 1, TRUE))
```

```
        return vrijednost
```

Kod 4.1. Pseudo kod za *minimax* algoritam

Kod 3.1. je pseudo kod *minimax* algoritma po kojemu je implementiran kod u ovom radu. Kao što je vidljivo u kodu 3.1. *minimax* algoritam mora imati i heurističku funkciju koja će određivati vrijednosti čvorova stabla. Algoritam kao argumente prima *cvor*, što je čvor u stablu, *dubina*, što je dubina do koje da prolazi kroz stablo, i *maksimizirajuciIgrac* što je igrač koji želi maksimizirati svoj dobitak.

Nakon toga se provjerava je li *dubina* došlo do nule, tj. je li došlo do željene dubine i, ako jest, vraća heurističku vrijednost čvora. Zatim se provjerava koji je igrač na redu (A ili B) i na osnovu toga se traži maksimalna ili minimalna vrijednost čvora na sljedećoj dubini.



Slika 4.11. Grananje stabla u igri "Poveži 4"

4.3.2. Alfa-beta rezanje

Kao što se može primijetiti, da bi uspio pronaći optimalni put, *minimax* algoritam mora proći kroz svaki čvor stabla. Ova vrsta pretraživanja duže traje dok postoji jednostavan način da se skrati. Neki čvorovi ne mogu utjecati na konačan rezultat tako da nema nikakve potrebe da se obilaze. Na takav način radi alfa-beta rezanje. [1]

```
function alfa_beta(cvor, dubina, α, β, maksimizirajuciIgrac) is
    if dubina = 0 or cvor is terminalni cvor then
        return heuristicku vrijednost of cvor
    if maksimizirajuciIgrac then
        vrijednost := −∞
```

```
        for each dijete of cvor do
        vrijednost := max(vrijednost, alfa_beta(dijete, dubina - 1, α, β, FALSE))
            α := max(α, vrijednost)
            if α ≥ β then
                break (*β rezanje *)
        return vrijednost
    else
    vrijednost:= +∞
        for each dijete of cvor do
        vrijednost:= min(vrijednost, alfa_beta(dijete, dubina - 1, α, β, TRUE))
            β := min(β, vrijednost)
            if β ≤ α then
                break (*α rezanje *)
        return vrijednost
```

Kod 4.2. Pseudo kod za alfa-beta rezanje

Algoritam alfa-beta rezanja je u biti *minimax* algoritam koji nastoji da provjerava samo one čvorove koji mogu utjecati na konačan rezultat. Dobiva još dva argumenta, alfa i beta pomoću kojih određuje koje čvorove da posjeti. Kao što je vidljivo iz koda 3.2. ponovno na početku algoritma postoji provjeru dubine da se ne bi ostalo u beskonačnoj rekurziji. Nakon toga ponovno se provjerava koji igrač je na redu i ovisno o tome traži se minimalna ili maksimalna vrijednost čvora na sljedećoj dubini. Razlika ovog algoritma u odnosu na *minimax* je ta što alfa-beta rezanje traži vrijednosti alfe i bete i ovisno o njima nastavlja prolaziti kroz stablo.

Prednost alfa-beta rezanja u odnosu na *minimax* algoritam je u tome što *minimax* mora proći kroz sve čvorove stabla, za razliku od alfa-beta rezanja gdje se pojedini dijelovi stabla eliminiraju. To se radi pomoću gore navedenih novih varijabli, alfe i bete. Varijabla alfa označava najbolji potez koji maksimizirajući igrač može odigrati, dok je varijabla beta najbolji potez minimizirajućeg igrača. Svakim pozivanjem funkcije provjerava se je li alfa veća ili jednaka beti i, ako jest, izlazi iz funkcije. Ovim se okreće prema podstablima značajnijim za konačni rezultat i smanjuje se efektivna dubina za polovicu s obzirom na sam *minimax* algoritam. Uvjet su optimalni poredak čvorova ili čvorovi blizu optimalnom poretku.

Ako je h prosjek grananja, a b broj slojeva u stablu, u najlošijem slučaju broj listova u stablu koje se ocjenjuje je:

$$O(h \cdot h \cdot \cdots \cdot h) = O(h^b) \tag{3-1}$$

Prema izrazu (3-1) je vidljivo da je vrijeme izvođenja u najlošijem slučaju jednako kao i kod *minimax* algoritma za razliku od optimalnog poretka gdje su prvi pretraženi koraci istodobno i najbolji, broj listova koje je potrebno pretražiti u stablu bi bio:

$$O(h \cdot 1 \cdot h \cdot 1 \cdot \cdots \cdot h) \tag{3-2}$$

ako postoji parna dubina , i:

$$O(h \cdot 1 \cdot h \cdot 1 \cdot \cdots \cdot 1) \tag{3-3}$$

ako postoji neparna dubina.

Dakle, broj listova koje se ocjenjuje u stablu, uz optimalan poredak, jest:

$$O(\sqrt{h^b}). \tag{3-4}$$

Kao što je vidljivo, kod optimalnog poretka, uz alfa-beta rezanje može se pretraživati duplo više čvorova stabla. Kada su čvorovi poslagani nasumičnim redoslijedom, što je i u ovom radu slučaj, prosječan broj ocijenjenih čvorova je:

$$O(\sqrt[4]{h^{3b}}). \tag{3-5}$$

5. REALIZACIJA SUSTAVA

Ovo poglavlje detaljno će objasniti programski kod pisan u Arduino IDE, dizajn tiskane pločice u *EAGLE* aplikaciji i detaljno objasniti shemu.

Program je pisan tako da bude omogućena igra čovjek protiv čovjeka, kao i igra čovjeka protiv računala koja je dosta kompliciranija. Igra čovjeka protiv računala je bazirana na *minimax* algoritmu s alfa-beta rezanjem.

5.1. Programski kod

Prije nego što se počnu opisivati funkcije treba se upoznati s varijablama korištenim u daljnjem programu.

```
Adafruit_MPR121 cap = Adafruit_MPR121();
Adafruit_NeoPixel LED = Adafruit_NeoPixel(NUM_PIXELS, PIN, NEO_GRB + NEO_KHZ800);
int tkoJeNaRedu = 1;
int poceo = tkoJeNaRedu;
uint16_t posljednjiDotaknut = 0;
uint16_t trenutniDotaknut = 0;
int S[6][7] = {
  {41, 40, 39, 38, 37, 36, 35},
  {28, 29, 30, 31, 32, 33, 34},
  {27, 26, 25, 24, 23, 22, 21},
  {14, 15, 16, 17, 18, 19, 20},
```

```
  {13, 12, 11, 10, 9, 8, 7},
  {0, 1, 2, 3, 4, 5, 6},
};
int T[6][7] = {
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
};
int stp1 = -1, stp2 = -1, stp3 = -1, stp4 = -1, stp5 = -1, stp6 = -1, stp7 = -1;
int *Stupac[7] = {&stp1, &stp2, &stp3, &stp4, &stp5, &stp6, &stp7};
int AI_zeton = 1;
int igrac_zeton = 2;
bool flag_e = false, flag_h = false;
int y = 0, br_poteza = 0;
int St1 = -1;
```

Kod 5.1. Globalne varijable

U kodu 4.1. mogu se vidjeti sve korištene varijable. Najvažnije od svih su S[6][7] i T[6][7] dvodimenzionalna polja. Oba polja se sastoje od 6 redaka i 7 stupaca koji predstavljaju retke i stupce na igraćoj ploči. U polju S se nalaze sve adrese LE dioda i ono služi za paljenje odgovarajućih dioda nakon odigranih polupoteza (engl. *ply*). Polje T služi za spremanje trenutne vrijednosti igraće ploče. Započinje kao polje popunjeno s nulama i nakon svakog polupoteza se u njega upisuje 1 ili 2, ovisno koji igrač je odigrao polupotez.

```
bool provjeriPobjedu(int igrac) {
  // vodoravna provjera
  for (int j = 0; j < 7 - 3 ; j++ ) {
    for (int i = 0; i < 6; i++) {
      if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
== igrac) {
        return true;
      }
    }
  }
  // okomita provjera
  for (int i = 0; i < 6 - 3 ; i++ ) {
    for (int j = 0; j < 7; j++) {
      if (T[i][j] == igrac && T[i + 1][j] == igrac && T[i + 2][j] == igrac && T[i + 3][j]
== igrac) {
        return true;
      }
    }
  }
  // rastuca dijagonalna provjera
  for (int i = 3; i < 6; i++) {
    for (int j = 0; j < 7 - 3; j++) {
      if (T[i][j] == igrac && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == igrac && T[i
- 3][j + 3] == igrac)
        return true;
    }
  }
  // padajuca dijagonalna provjera
  for (int i = 3; i < 6; i++) {
    for (int j = 3; j < 7; j++) {
```

```
      if (T[i][j] == igrac && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == igrac && T[i
- 3][j - 3] == igrac)
        return true;
    }
  }
  return false;
}
```

Kod 5.2. Funkcija *provjeriPobjedu()*

Funkcija *provjeriPobjedu()*, kako joj i ime kaže, provjerava je li neki od igrača ostvario uvjet za pobjedu. Prima varijablu *igrac* koja može imati vrijednost '1' ili '2' ovisno o tome za kojeg igrača se provjerava pobjeda. Poziva se nakon svakog odigranog polupoteza. Radi na način da provjerava jesu li postavljena četiri žetona iste boje jedan do drugog tj. ima li u polju T četiri vrijednosti varijable *igrac* jednu uz drugu.

Prvo se provjerava je li ostvaren uvjet za pobjedu vodoravno. Radi se tako što se pomoću *for* petlje prolazi kroz polje T i za svaku vrijednost polja koja je jednaka varijabli *igrac* provjeri jesu li naredne 3 vrijednosti u retku jednake istom i ako jesu vraća *true*.

Ako nije postignuta pobjeda vodoravno kod ulazi u sljedeću *for* petlju koja provjerava imaju li četiri žetona okomito. Izvršava se po istoj logici kao i vodoravna samo što umjesto provjeravanja po recima provjeravamo po stupcima.

Za dijagonalnu provjeru (uzlazno i silazno) koristi se *for* petlja za kretanje po polju i pomoću *if* petlje provjerava uvjete.

Nakon što se izvrše sve četiri petlje bez pronalaska uvjeta za pobjedu funkcija vraća *false*.

```
int touchSens() {
  int a = -1;
  trenutniDotaknut = cap.touched();

  for (uint8_t i = 0; i < 12; i++) {
    if ((trenutniDotaknut & _BV(i)) && !(posljednjiDotaknut & _BV(i)) ) {
      a = i;
    }
    if (!(trenutniDotaknut & _BV(i)) && (posljednjiDotaknut & _BV(i)) ) {
    }
  }
  posljednjiDotaknut = trenutniDotaknut;
  return a;
}
```

Kod 5.3. Funkcija *touchSens()*

Da bi igrači uopće mogli odigrati poteze mora se imati neki način komunikacije mikrokontrolera s vanjskom okolinom. Za to služi MPR121 kapacitivni senzor dodira i ova funkcija. Funkcija *touchSens()* očitava vrijednosti s dodirnih pločica i vraća ju kao broj od 0 do 11.

```
void greenScreen() {

  for (int i = 0; i < NUM_PIXELS; i++) {
    LED.setPixelColor(i, 155, 0, 0);
    LED.show();
    delay(100);
  }
  while (1) {
    if (touchSens() > 0) {
      resetGame();
      changeigrac();
      delay(500);
      pickMode();
      break;
    }
    yield();
  }
}
```

Kod 5.4. Funkcija *greenScreen()*

Funkcija *greenScreen()* služi da, ako pobijedi igrač broj 1, postavi sve LE diode u zelenu boju. Također ako igrač nakon toga dotakne bilo koju dodirnu pločicu igra se ponovno pokreće i igrači zamijene redoslijed igranja. Također postoje još i *redScreen()* i *drowScreen()* funkcije koje se pozivaju ovisno o tome je li pobijedio igrač broj 2 ili je rezultat bio neriješen. Mogu se vidjeti u prilogu 2.

```
void pickMode() {
  LED.setPixelColor(2, 0, 0, 155);
  LED.setPixelColor(3, 155, 0, 0);
  LED.setPixelColor(4, 0, 155, 0);
  LED.show();
  while (1) {
    St1 = touchSens();
    if (St1 == 2) { //PvP

      flag_e = false;
      flag_h = false;
      St1 = -1;
      for (int i = 0; i < NUM_PIXELS; i++)
      {
        LED.setPixelColor(i, 0, 0, 0);
        LED.show();
      }
      delay(750);

      return;
    }
    if (St1 == 3) { //PvE HARD
      flag_e = false;
      flag_h = true;
      St1 = -1;
      for (int i = 0; i < NUM_PIXELS; i++)
      {
        LED.setPixelColor(i, 0, 0, 0);
        LED.show();
      }
```

```
      return;
    }
    if (St1 == 4) { //PvE EASY
      flag_h = false;
      flag_e = true;
      St1 = -1;
      for (int i = 0; i < NUM_PIXELS; i++)
      {
        LED.setPixelColor(i, 0, 0, 0);
        LED.show();
      }
      delay(750);

      return;
    }
    yield();
  }
}
```

Kod 5.5. ***pickMode()***

Ova funkcija služi za odabir željene razine i načina igre, čovjek protiv čovjeka ili čovjek protiv računala. Na početku svake partije prvo se mora odabrati hoće li se igrati protiv računala ili protiv čovjeka te, ako će se igrati protiv računala, treba se odabrati težina. Igra podržava dvije razine težine: lako i teško.

Funkcija postavlja tri različite boje, crvenu, zelenu i plavu, na tri LE diode. Dodirom dodirne pločice iznad plave diode odabire se igra čovjek protiv čovjeka, dodirom iznad zelene, čovjeka protiv računala na laganoj težini i dodirom iznad crvene, čovjek protiv računala na teškoj razini.

```
void odigrajPotez(int x, int y) {
  if (tkoJeNaRedu == 1) {
    T[x][y] = 1;
    br_poteza++;
    for (int i = 5; i > x; i--) {
      LED.setPixelColor(S[i][y], 155, 0, 0);
      LED.show();
      delay(20);
      LED.setPixelColor(S[i][y], 0, 0, 0);
      LED.show();
    }
    LED.setPixelColor(S[x][y], 155, 0, 0);
    LED.show();
    delay(DELAY_);

    tkoJeNaRedu = 2;
    if (provjeriPobjedu(1)) {
      delay(1000);
      greenScreen();
      return;
    }
    else if (br_poteza >= 42) {
      drawScreen();
      return;
    }
  }
  else {
```

```
    T[x][y] = 2;
    br_poteza++;
    for (int i = 5; i > x; i--) {
      LED.setPixelColor(S[i][y], 0, 155, 0);
      LED.show();
      delay(20);
      LED.setPixelColor(S[i][y], 0, 0, 0);
      LED.show();
    }
    LED.setPixelColor(S[x][y], 0, 155, 0);
    LED.show();
    delay(DELAY_);
    tkoJeNaRedu = 1;
    if (provjeriPobjedu(2)) {
      delay(1000);
      redScreen();
      return;
    }
    else if (br_poteza >= 42) {
      drawScreen();
      return;
    }

  }
}
```

Kod 5.6. Funkcija *odigrajPotez()*

Funkcija *odigrajPotez()* ima dvije ulazne varijable x i y. Varijabla x predstavlja redak dok varijabla y predstavlja stupac tako da se zna točno koju LE diodu je potrebno upaliti. Funkcija radi tako što s *if* petljom provjeri tko je na redu i, ovisno o tome, upiše 1 ili 2 u polje T. Također pomoću *for* petlje stvara dojam padanja žetona na mjesto tako što na vrlo kratko vrijeme pali LE diode u stupcu u koji je ubačen žeton.

```
void resetGame() {
  br_poteza = 0;
  flag_e = false;
  flag_h = false;

  for (int i = 0; i < NUM_PIXELS; i++)
  {
    LED.setPixelColor(i, 0, 0, 0);
    LED.show();
  }
  for (int i = 0; i < 6; i++)
  {

    for (int j = 0; j < 7; j++)
    {
      *Stupac[j] = -1;
      T[i][j] = 0;
    }
  }
}
```

Kod 5.7. Funkcija *resetGame()*

Funkcija *resetGame()* ponovno pokreće igru. Postavlja brojač poteza na 0, gasi sve LE diode, polje T popunjava s nulama. Na kraju svake partije ova se funkcija poziva da počisti igraću ploču za iduću igru i uz nju se poziva funkcija *changePlayer()* koja postavlja da onaj igrač koji je igrao prvi sada igra drugi.

```
void Play(int a) {
  if (a == 9) {
    stp1 = stp1 + 1;
    y = 0;
    if (stp1 < 6) {
      odigrajPotez(stp1, y);
      return;
    }
  }
  else if (a == 1) {
    stp2 = stp2 + 1;

    y = 1;
    if (stp2 < 6) {
      odigrajPotez(stp2, y);
      return;
    }
  }
  else if (a == 2) {
    stp3 = stp3 + 1;

    y = 2;
    if (stp3 < 6) {
      odigrajPotez(stp3, y);
      return;
    }
  }
  else if (a == 3) {
    stp4 = stp4 + 1;

    y = 3;
    if (stp4 < 6) {
      odigrajPotez(stp4, y);
      return;
    }
  }
  else if (a == 4) {
    stp5 = stp5 + 1;

    y = 4;
    if (stp5 < 6) {
      odigrajPotez(stp5, y);
      return;
    }
  }
  else if (a == 5) {
    stp6 = stp6 + 1;

    y = 5;
    if (stp6 < 6) {
      odigrajPotez(stp6, y);
      return;
    }
  }
```

```
  else if (a == 6) {
    stp7 = stp7 + 1;

    y = 6;
    if (stp7 < 6) {
      odigrajPotez(stp7, y);
      return;
    }
  }
  else if (a == 7) {
    if (St1 == 7) newGame();
  }
}
```

Kod 5.8. Funkcija *play()*

Ova funkcija služi da proslijedi vrijednost retka i stupca funkciji *odigrajPotez()*. Kao argument prima broj koji predstavlja stupac u koji igrač ili AI ubacuje žeton. Vrijednosti redaka dobiva tako što ima varijable za svaki stupac koje kreću od -1 i nakon svakog odigranog polupoteza se povećavaju za 1, tako da ako se za prvi polupotez pali LE dioda na indexu 3 (u sredini ploče) funkciji se predaje 3 i ona bi pozvala funkciju *odigrajPotez()* s y=3 i stp3=0 i ako bi se ponovno odigrao isti polupotez, pozvala bi se funkcija *odigrajPotez()* s argumentima y=3 i stp3=1. Također ova funkcija provjerava je li popunjen stupac u koji se pokušava ubaciti žeton tako da ako se slučajno stisne dodirna pločica neće biti registrirano kao odigran polupotez.

```
int AI_Random() {

  int randomNum = random(0, 7);
  while (T[5][randomNum] > 0) {
    randomNum = random(0, 7);
  }
  return randomNum;

}
```

Kod 5.9. Funkcija *AI_Random()*

AI_Random() je najjednostavnija funkcija umjetne inteligencije. Funkcija nasumično bira stupac i vraća njegov indeks. Funkcija također ima i beskonačnu petlju koja, ukoliko je stupac popunjen, ponovno nasumično bira stupac.

```
int AI_1()
{
  int pot = -1;
  for (int i = 0; i < 7; i++) {
    if ((*Stupac[i] + 1) < 6) {
      T[*Stupac[i] + 1][i] = AI_zeton;
      if (provjeriPobjedu(AI_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        return i;
      }
      if (provjeriPobjedu(AI_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
```

```
        pot = i;
      }
      T[*Stupac[i] + 1][i] = igrac_zeton;
      if (provjeriPobjedu(AI_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        return i;
      }
      if (provjeriPobjedu(igrac_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        pot = i;
      }

      T[*Stupac[i] + 1][i] = 0;
    }

  }
  return pot;
}
```

Kod 5.10. Funkcija *AI_1()*

Sljedeća razina umjetne inteligencije se može vidjeti u ovoj funkciji. Funkcija *AI_1()* prolazi kroz svih 7 mogućih polupoteza i provjerava hoće li, ako odigra taj polupotez, pobijediti ili uskratiti pobjedu protivniku s tim da funkcija prioritizira pobjedu. Funkcija radi tako što u *for* petlji upiše 1 ili 2, ovisno o tome tko je prvi počeo igrati, računalo ili čovjek, u polje T i provjerava hoće li pobijediti pozivanjem funkcije *provjeriPobjedu()*. Na kraju *for* petlje se vraćaju vrijednosti polja T na prvobitno stanje.

```
int AI_2() {
  int i = 0;
  while (i < 50) {
    int stupac = AI_Random();

    if ((*Stupac[stupac] + 2) < 6) {
      if (tkoJeNaRedu == AI_zeton) {
        T[*Stupac[stupac] + 2][stupac] = igrac_zeton;
        if (!provjeriPobjedu(igrac_zeton)) {
          T[*Stupac[stupac] + 2][stupac] = 0;
          return stupac;
        }
        else {
          T[*Stupac[stupac] + 2][stupac] = 0;
          i++;
        }
      }
      if (tkoJeNaRedu == igrac_zeton) {
        T[*Stupac[stupac] + 2][stupac] = AI_zeton;
        if (!provjeriPobjedu(AI_zeton)) {
          T[*Stupac[stupac] + 2][stupac] = 0;
          return stupac;
        }
        else {
          T[*Stupac[stupac] + 2][stupac] = 0;
          i++;
        }
      }
    }
```

```
  }
  return -1;
}
```

Kod 5.11. Funkcija *AI_2()*

Funkcija *AI_2()* radi tako da nasumično odabere stupac i nakon toga provjeri hoće li igrač pobijediti u sljedećem potezu u slučaju da računalo odigra taj polupotez. Ovo bi bila malo pametnija verzija funkcije *AI_Random()* koja se brine o tome da ti ne pokloni pobjedu.

```
int AI_3() {
  for (int i = 0; i < 6; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      if (tkoJeNaRedu == igrac_zeton) {
        // kombinacije 1-3
        if (T[i] [j] == 0 && T[i] [j + 1] == 0 && T[i][ j + 2] == AI_zeton && T[i] [j +
            3] == AI_zeton && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 1] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 1;
        }
        // kombinacije 4-6
        if (T[i][ j] == 0 && T[i][ j + 1] == AI_zeton && T[i][ j + 2] == 0 && T[i ][ j +
            3] == AI_zeton && T[i ][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 2] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 2;
        }
        // kombinacije 7-9
        if (T[i][ j] == 0 && T[i ][ j + 1] == AI_zeton && T[i][j + 2] == AI_zeton &&
            T[i][ j + 3] == 0 && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 3] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 3;
        }
      }
      else if (tkoJeNaRedu == AI_zeton) {
        // kombinacije 1-3
        if (T[i] [j] == 0 && T[i] [j + 1] == 0 && T[i][ j + 2] == igrac_zeton && T[i] [j
            + 3] == igrac_zeton && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 1] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 1;
        }
        // kombinacije 4-6
        if (T[i][ j] == 0 && T[i][ j + 1] == igrac_zeton && T[i][ j + 2] == 0 && T[i ][ j
            + 3] == igrac_zeton && T[i ][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 2] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 2;
        }
        // kombinacije 7-9
```

```
        if (T[i][ j] == 0 && T[i ][ j + 1] == igrac_zeton && T[i][j + 2] == igrac_zeton
            && T[i][ j + 3] == 0 && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 3] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 3;
        }
      }
    }
  }
  return -1;
}
```

Kod 5.12. Funkcija *AI_3()*

Pod određenim okolnostima postoji mogućnost da protivnik može napraviti dvostruki napad u jednom retku. Ovo se događa kada se od pet mjesta u prostoru, u tri srednja nađu dva žetona protivnika i slobodno mjesto uz prazna mjesta na rubovima. Funkcija prepoznaje tu opasnost te se uspješno brani prolazeći kroz svih devet mogućih kombinacija i vraća '-1' ako opasnosti nema.

```
int AI_Easy () {
  int st;

  st = AI_1();
  if (st != -1) {
    if (st == 0) st = 9;
    return st;
  }
  st = AI_3();
  if (st != -1) {
    if (st == 0) st = 9;
    return st;
  }
  st = AI_2();
  if (st != -1) {
    if (st == 0) st = 9;
    return st;
  }
  st = AI_Random();
  if (st == 0) st = 9;
  return st;
}
```

Kod 5.13. Funkcija *AI_Easy()*

Funkcija *AI_Easy()* se sastoji od jednostavnih funkcija simuliranja umjetne inteligencije. Radi tako da jednostavno poziva AI funkcije. U kodu 4.13. može se vidjeti logiku rada ove funkcije.

```
bool imaLiPoteza() {
  for (int i = 0; i < 6; i++) {
    for (int j = 0; j < 7; j++) {
      if (T[i][j] == 0) {
        return true;
      }
    }
  }
  return false;
```

```
}
```

Kod 5.14. Funkcija *imaLiPoteza()*

Da bi se implementirao *minimax* algoritam s alfa-beta rezanjem mora se prvo definirati par funkcija. Ako se drži pseudo koda, odmah na početku može se vidjeti da postoje dva uvjeta za izlazak iz petlje. Prvi je je li došlo do željene dubine, dok drugi provjerava je li došlo do terminalnog čvorišta (engl. *terminal node*). Terminalna čvorišta su u ovom radu svi potezi koji bi završili igru.

Funkcija *imaLiPoteza()* provjerava ima li ijedno slobodno mjesto u polju T, koje predstavlja igraću ploču. Ako ima vraća *true*, dok ako nema, vraća *false*.

```
int evaluate() {

  if (provjeriPobjedu(AI_zeton)) {
    return 100000;
  }
  else if (provjeriPobjedu(igrac_zeton))
  {
    return -100000;
  }
  else if (!imaLiPoteza()) {
    return 0;
  }
  else return -1;
}
```

Kod 5.15. Funkcija *evaluate()*

Funkcija *evaluate()* provjerava je li čvor u kojem se nalazi terminalni. Kao što je već rečeno terminalna čvorišta su svi potezi koji bi završili igru tj. u ovoj igri su to:

- Pobjeda igrača
- Pobjeda računala
- Neriješen rezultat

Ova funkcija provjerava tko bi pobijedio ako se odigra taj potez i, ako je to računalo, vraća '100000', ako je to čovjek, '-100000' i ako je neriješeno vraća 0.

Ako niti jedan od gore navedenih uvjeta nije ispunjen, odnosno ako čvorište nije terminalno, vraća '-1'.

```
int score_ploca(int igrac) {
  int br = 0;
  int  score = 0;
  //************************************Center
  for (int i = 0; i < 6; i++) {
    int    j = 3;
```

```
    if (T[i][j] == igrac) {
      br++;
    }
  }
  score = score + (br * 5);


  for (int i = 0; i < 6; i++) {
    for (int j = 1; j < 7 - 4; j++) {
      if (T[i][j - 1] == 0 && T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] ==
          igrac && T[i][j + 3] == 0) {
        score = score + 20;
      }
    }
  }

  //************************************Vodoravno
  for (int i = 0; i < 6; i++) {
    for (int j = 0; j < 7 - 3; j++) {
      if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
          == igrac) {
        score = score + 100;
      }
      if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
          == 0
          || T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == 0 && T[i][j + 3]
          == igrac
          || T[i][j] == igrac && T[i][j + 1] == 0 && T[i][j + 2] == igrac && T[i][j + 3]
          == igrac
          || T[i][j] == 0 && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
          == igrac) {
        score = score + 5;
      }
      if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == 0 && T[i][j + 3] ==
          0
          || T[i][j] == igrac && T[i][j + 1] == 0 && T[i][j + 2] == 0 && T[i][j + 3] ==
          igrac
          || T[i][j] == 0 && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
          == 0
          || T[i][j] == 0 && T[i][j + 1] == 0 && T[i][j + 2] == igrac && T[i][j + 3] ==
          igrac

          || T[i][j] == igrac && T[i][j + 1] == 0 && T[i][j + 2] == igrac && T[i][j + 3]
          == 0
          || T[i][j] == 0 && T[i][j + 1] == igrac && T[i][j + 2] == 0 && T[i][j + 3] ==
          igrac)
      {
        score = score + 2;
      }
      if (T[i][j] == igrac_zeton && T[i][j + 1] == igrac_zeton && T[i][j + 2] ==
          igrac_zeton && T[i][j + 3] == 0
          || T[i][j] == igrac_zeton && T[i][j + 1] == igrac_zeton && T[i][j + 2] == 0 &&
          T[i][j + 3] == igrac_zeton
          || T[i][j] == igrac_zeton && T[i][j + 1] == 0 && T[i][j + 2] == igrac_zeton &&
          T[i][j + 3] == igrac_zeton
          || T[i][j] == 0 && T[i][j + 1] == igrac_zeton && T[i][j + 2] == igrac_zeton &&
          T[i][j + 3] == igrac_zeton) {
        score = score - 40;
      }
    }
  }
```

Kod 5.16. Dio Funkcije *scorePloca()*

Funkcija *scorePloca()* je heuristička funkcija za određivanje vrijednosti čvorova u stablu. Kao argument prima *igrac* tj. igrača za kojeg se radi bodovanje ploče. Radi na principu da provjerava stanje na ploči i daje određeni broj bodova za određenu poziciju. Bodovanje je sljedeće:

Za svaki žeton koji ima u srednjem stupcu dobiva broj žetona u stupcu pomnožen s dva bodova. Zatim, ako sastavi tri žetona u retku uz to je po jedno mjesto s obje strane prazno dobiva 20 bodova.

Nakon toga se ulazi u vodoravnu provjeru stanja ploče tako da, ako spoji četiri žetona u jednom retku (vodoravno), čime ujedno i pobjeđuje igru, dobiva 100 bodova. Ako u prostoru od tri mjesta ima tri žetona prisutna, uz jedno mjesto slobodno, dobiva 5 bodova. ako na prostoru od četiri mjesta ima dva žetona prisutna, uz dva mjesta slobodna, dobiva 2 boda. Također, ukoliko bi protivnik, na prostoru od četiri mjesta, imao tri žetona svoje boje i jedno mjesto slobodno, što znači da bi u sljedećem potezu mogao pobijediti, gubi 40 bodova (dobiva -40).

Provjere stanja ploče se izvode tako što se *for* petljom prolazi kroz polje T i pomoću *if* petlje provjeravaju gore navedeni slučajevi.

Ista logika provjere stanja ploče i bodovanja se primjenjuje na okomitu, dijagonalnu padajuću i dijagonalnu rastuću provjeru. Kod se može vidjeti u prilogu 2.

```
int alfa_beta(int dubina, int alfa, int beta, int maximisingIgrac) {
  yield();
  int krajnjiList = evaluate();
  if (krajnjiList != -1) {
    return krajnjiList;
  }
  else if (dubina == 0) {
    return score_ploca(AI_zeton);
  }

  if (maximisingIgrac == AI_zeton) {                              //***MAX***
    int bestScore = -10000000;
    for (int i = 0; i < 7; i++) {
      if (*Stupac[i] < 5) {
        if ( T[*Stupac[i] + 1][i] == 0) {
          T[*Stupac[i] + 1][i] = AI_zeton;
          *Stupac[i] = *Stupac[i] + 1;
          br_poteza++;
          bestScore = max(bestScore, alfa_beta(dubina - 1, alfa, beta, igrac_zeton));
          alfa = max(alfa, bestScore);
          if (alfa >= beta) {
            br_poteza--;
            *Stupac[i] = *Stupac[i] - 1;
            T[*Stupac[i] + 1][i] = 0;
            break;
          }
          br_poteza--;
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1][i] = 0;
        }
      }
```

```
    }
    return bestScore;
  }

  else {       //***MIN***
    int bestScore = 10000000;
    for (int i = 0; i < 7; i++) {
      if (*Stupac[i] < 5) {
        if ( T[*Stupac[i] + 1 ][i] == 0) {
          T[*Stupac[i] + 1][i] = igrac_zeton;
          *Stupac[i] = *Stupac[i] + 1;
          br_poteza++;
          bestScore = min(bestScore, alfa_beta(dubina - 1, alfa, beta, AI_zeton));
          beta = min(beta, bestScore);
          if (alfa >= beta) {
            br_poteza--;
            *Stupac[i] = *Stupac[i] - 1;
            T[*Stupac[i] + 1 ][i] = 0;
            break;
          }
          br_poteza--;
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1 ][i] = 0;
        }
      }
    }
    return bestScore;
  }
}
```

Kod 5.17. Funkcija *alfa_beta()*

Ova funkcija je implementacija *minimax* algoritma s alfa-beta rezanjem za igru „Poveži Četiri“. Funkcija prima četiri argumenta:

- *dubina* – označava koliko polupoteza će funkcija provjeravati, tj. koliko će maksimalno puta funkcija pozvati samu sebe (rekurzija)
- *alfa* – najbolji potez koji maksimizirajući igrač može odigrati
- *beta* – najbolji potez koji minimizirajući igrač može odigrati
- *maksimizirajuciIgrac* – igrač koji maksimizira

Na početku funkcije postavljaju se uvjeti za izlazak iz funkcije da se ne bi ostalo u beskonačnoj rekurziji. Prvo se provjerava je li došlo do krajnjeg lista ili terminalnog čvora pozivajući funkciju *evaluate()*, zatim, ako *evaluate()* vrati '-1', provjerava se je li varijabla *dubina* nula, tj. je li se došlo do zadane dubine.

Nakon provjere uvjeta provjera se koji je igrač na redu i ovisno o tome ulazi se u dio funkcije koji uzima maksimalne vrijednosti, ili u dio gdje se uzimaju minimalne vrijednosti.

U dijelu koda gdje se odabiru minimalne vrijednosti varijabla *bestScore* se postavlja na 10000000 dok, za dio gdje se bira maksimalna, postavlja na -10000000. Nakon postavljanja vrijednosti

varijable ulazi se u *for* petlju koja prolazi od 0 do 6, što predstavlja svih mogućih 7 poteza, i provjerava je li mjesto slobodno za odigrati potez. Ako je mjesto slobodno, funkcija postavlja vrijednost 1 ili 2 (ovisno koji igrač je na redu) u polje T, tj. odigrava polupotez nakon čega ponovno poziva samu sebe, s tim da argument *dubina* smanjuje za 1.

To se sve izvršava dok se ne dođe do terminalnog čvora ili do željene dubine, odnosno da je varijable *dubina* jednaka nuli. Tada se počne nastavljati program tako da funkcija koja je se zadnja pozvala prva završi i predaje vrijednost ocjene ploče koja se sprema u varijablu *score.* Nakon toga u varijablu *bestScore* se upisuje veći broj (u slučaju da je igrač minimizirajući, biramo manji) od *score* i *bestScore*. Također u varijablu *alfa* (*beta* kod minimizirajućeg igrača) upisuje se veća (u slučaju minimizirajućeg igrača manja) vrijednost od *bestScore* i *alfa*. Nakon toga provjerava je li *alfa* veći ili jednak kao *beta* i, ako jest, briše odigrani polupotez iz polja T, vraća brojač poteza na vrijednost prije ulaska u petlju i izlazi iz funkcije.

```
int najboljiPotezAlfaBeta(int dubina) {
  int stupac = 0;
  int bestVal = -100000000;
  int moveVal;
  for (int i = 0; i < 7; i++) {
    if (*Stupac[i] < 5) {
      if ( T[*Stupac[i] + 1][i] == 0) {

        T[*Stupac[i] + 1][i] = AI_zeton;
        *Stupac[i] = *Stupac[i] + 1;
        if (provjeriPobjedu(AI_zeton)) {
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1][i] = 0;
          return i;
        }
        moveVal = alfa_beta(dubina, -100000000, 100000000, igrac_zeton);
        T[*Stupac[i] + 1][i] = 0;
        if (moveVal >= bestVal) {

          stupac = i;
          bestVal = moveVal;
        }
      }
    }
  }
  return stupac;
}
```

Kod 5.18. Funkcija *najboljiPotezAlfaBeta()*

Funkcija *najboljiPotezAlfaBeta()* služi da odredi koji potez ima najveći broj bodova nakon odrađivanja *minimax* algoritma s alfa-beta rezanjem. Radi tako da, pomoću *for* petlje prolazi kroz svih 7 mogućih poteza i za svaki poziva funkciju alfa_beta(). Nakon toga jednostavno provjeri, pomoću *if* petlje, koji od poteza ima najveći broj bodova, tj. koji od poteza je najbolji put u stablu.

```
int AI_Hard(int dubina) {
```

```
  int x = najboljiPotezAlfaBeta(dubina); //15 sekundi po potezu(dubina 8), 5sec dubina 7
  if (x == 0) x = 9;
  return x;
}
```

***Kod 5.19. Funkcija* AI_Hard()**

Ova funkcija samo poziva funkciju *najboljiPotezAlpfBeta()* i postavlja dubinu do koje bi *alpf_beta()* funkcija trebala ići.

```
void setup() {

  Serial.begin(115200);
  while (!Serial) {
    delay(10);
  }
  LED.begin();
  LED.setBrightness(BRIGHTNESS);
  LED.show();
  Serial.println("Adafruit MPR121 Capacitive Touch sensor test");
  delay(100);
  if (!cap.begin(0x5A)) {
    Serial.println("MPR121 not found, check wiring?");
    while (1) {
      yield();
    }
  }
  Serial.println("MPR121 found!");
  pickMode();
}
//**************************************************************************
void loop() {

  if ((flag_e) && (tkoJeNaRedu == AI_zeton) && (!provjeriPobjedu(AI_zeton)) &&
(!provjeriPobjedu(igrac_zeton))) {
    Play(AI_Easy());
  }
  if ((flag_h) && (tkoJeNaRedu == AI_zeton) && (!provjeriPobjedu(AI_zeton)) &&
(!provjeriPobjedu(igrac_zeton))) {
    Play(AI_Hard(6));
  }

  St1 = touchSens();
  Play(St1);
}
```

Kod 5.20. Funkcije *setup() i loop()*

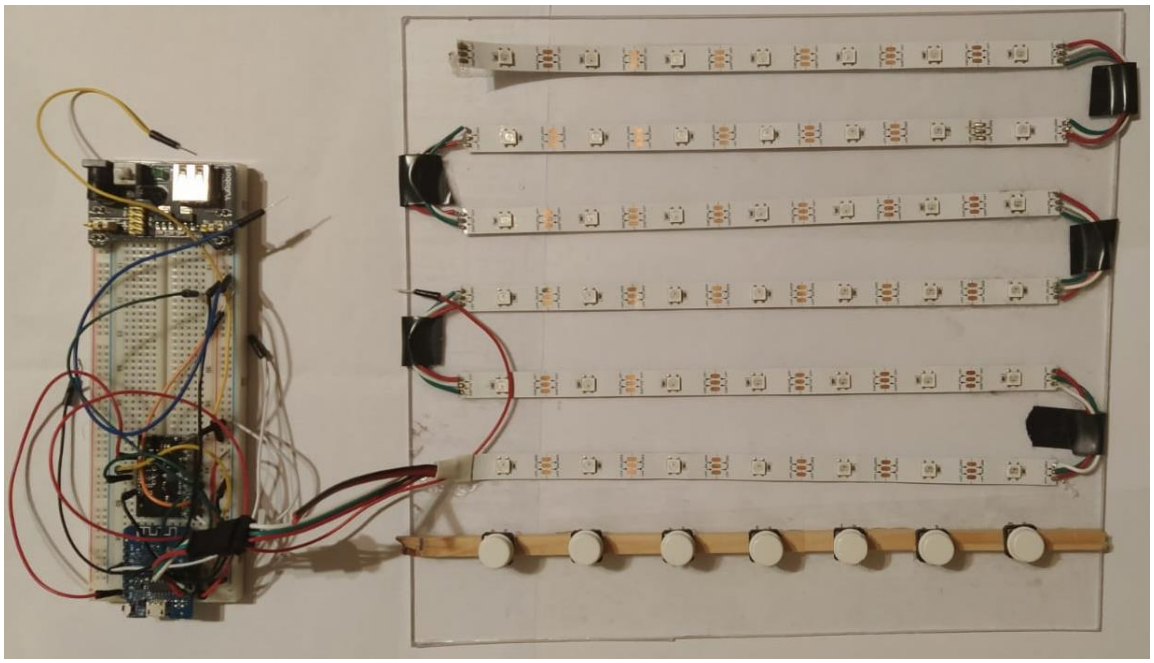
Bilo koji program koji je pisan u Arduino IDE sadrži dvije osnovne funkcije, a to su *loop()* i *setup()*. Funkcija *setup()* se izvodi samo jednom, kada program tek započne, i služi za inicijalizaciju varijabli, pin *mode*-ova, *library*-a itd. U ovom slučaju se u njoj poziva i funkcija *pickMode()*.

Funkcija *loop()* se poziva nakon *setup()* funkcije i radi kao beskonačna *while* petlja. Znači beskonačno se pokreće. Kada god dođe do kraja funkcije, ponovno se pokrene od početka. Kao što se može vidjeti u kodu 4.20. *loop()* funkcija provjerava koji način igre je odabran, tj. protiv

računala (lagano i teško) ili protiv čovjeka. Ovisno o tome poziva funkcije *AI_Easy()*, *AI_Hard()* ili samo *Play()*. Također, kontinuirano provjerava je li igrač dodirnu ijednu od dodirnih ploča s funkcijom *touchSens()*.

5.2. Dizajniranje tiskane pločice

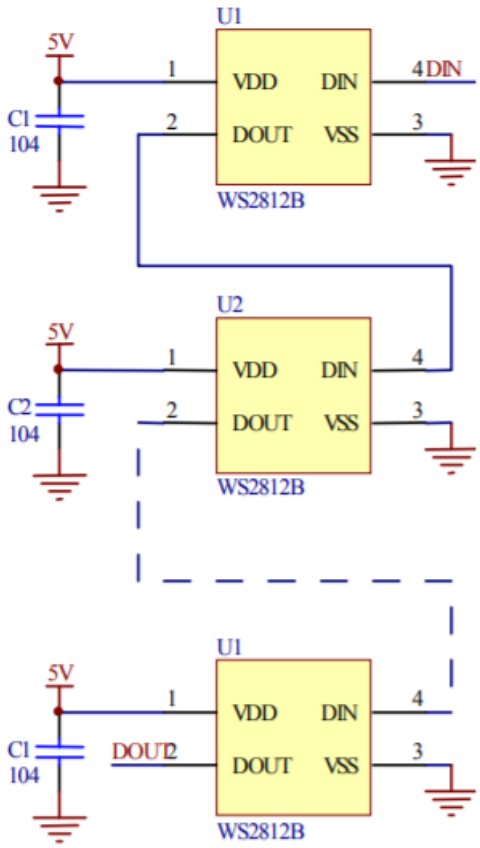
Za razvoj koda i razradu konačnog rješenja korištena je maketa sačinjena od LED traka spojenih na Wemos D1 mini Lite pločicu. U prvobitnom za igranje poteza su korištene tipke (engl. *button*), no zbog prevelikog broja kablova potrebnih za spajanje tipki i zbog nepreglednosti istih odlučeno je da se koristi kapacitivni senzor dodira koji ima I2C sučelje za komunikaciju. Sve je povezano preko eksperimentalne pločice (engl. *breadboard*).



Kod 5.21. Maketa za razvoj koda i razradu konačnog rješenja

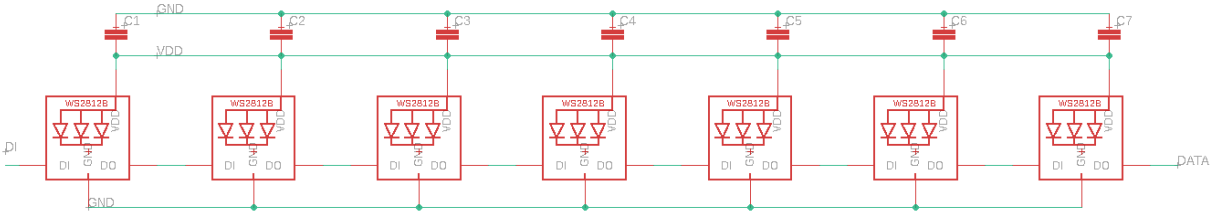
Nakon izrade programskog koda i testiranja istog maketu je trebalo zamijeniti s manjim i kompaktnijim rješenjem koje bi moglo biti napajano s punjivim Li-Ion baterijama. Iz tog razloga je dizajnirana tiskana pločica.

Najbolji program za izradu tiskanih pločica je EAGLE od Autodesk-a. Prvo se mora dizajnirati shema. Kako postoje 42 LE diode, moraju se nekako povezati. To je rađeno na način vidljiv na slici 4.1..



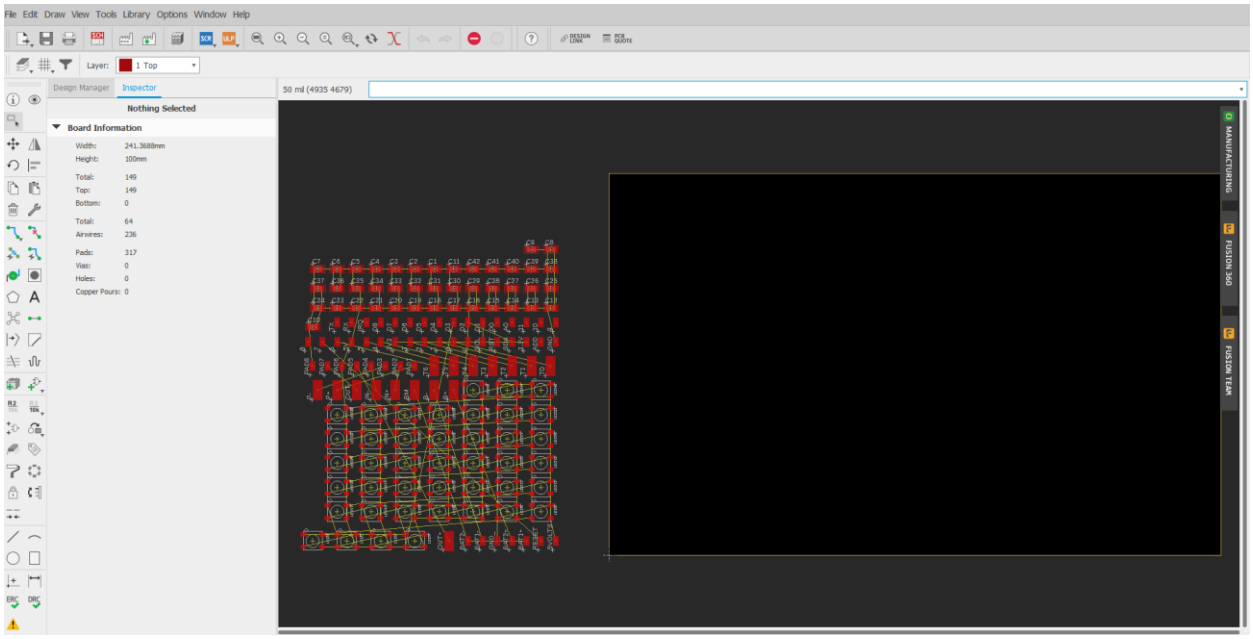
Slika 5.1. Shema spajanja WS2812B LE dioda

Da bi se dizajnirala shema u EAGLE-u prvo se moraju dodati sve komponente koje se žele imati na pločici. To se radi tako što se klikne na *Add Part* izbornik koji otvara popis svih komponenti koje se nalaze u osnovnom EAGLE-ovom *library*-u. Ako se željena komponenta ne nalazi na toj listi, potrebno je skinuti novi *library* i dodati ga u EAGLE. Nakon što se dodaju sve komponente mogu se spojiti na odgovarajući način. Na slici 4.2. je vidljivo kako je u spojen redak LE dioda za ovaj projekt.



Slika 5.2. Shema spajanja jednog retka

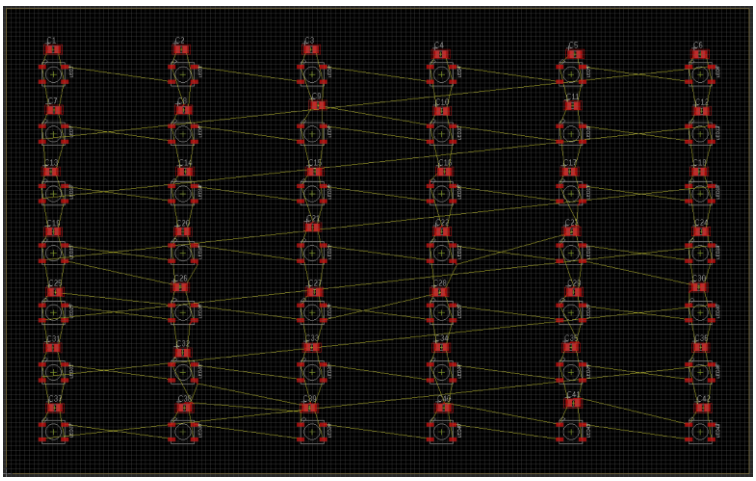
Nakon spajanja sheme može se otvoriti drugi dio EAGLE programa, a to je dizajn sklopovlja (engl. *circuit design*). Sve komponente koje su dodane shemi će biti poredane u donjem lijevom kutu kako je vidljivo na slici 4.3.. Puna shema je u prilogu 1.



Slika 5.3. EAGLE *circuit diagram*

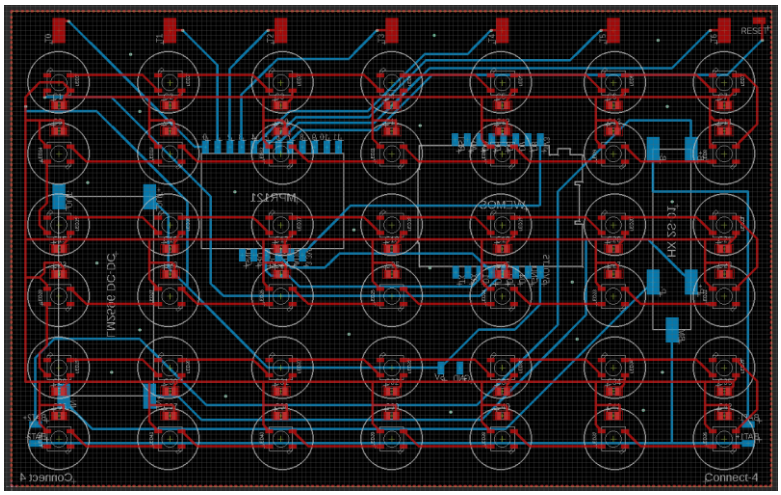
Na slici 4.3. može se vidjeti da su sve komponente povezane sa tzv. zračnim žicama (engl. *airwire*). One govore što je sve spojeno na shemi. Također EAGLE ne dopušta da se povežu komponente koje nisu povezane na shemi da bi se izbjegle nepotrebne pogreške.

Kada se otvori *circuit design* može se početi postavljati raspored komponenti na ploči. Ploča je prikazana na slici 4.3., desno od komponenti.



Slika 5.4. Primjer poredanih komponenti na tiskanoj pločici

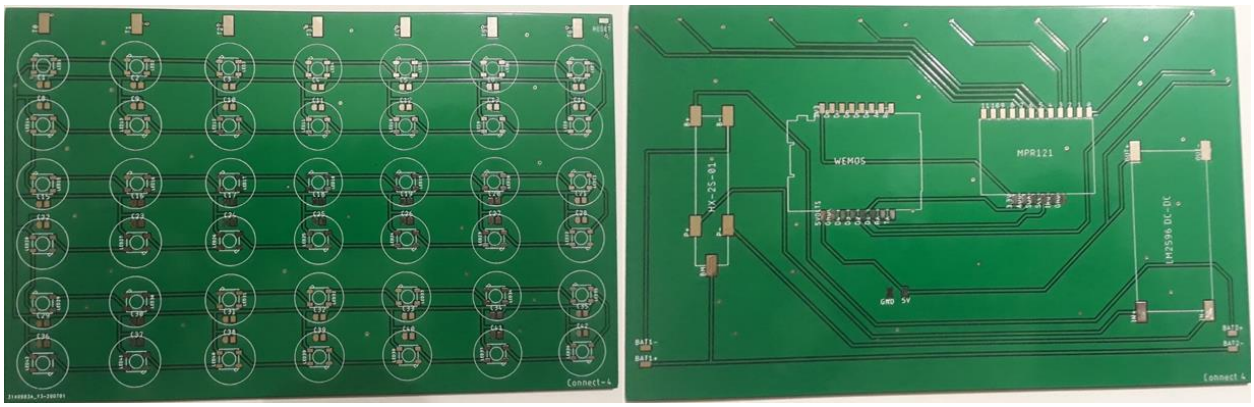
Kada se postave komponente na željena mjesta mogu se početi spajati iste. Pomoću alata *Route Airwire* određuje se kuda će prolaziti vodovi (engl. *tracers*). Sam program neće dopustiti pogrešno spajanje, tj. neće dopustiti da spoje drugačije nego što je specificirano na shemi preko koje je generirana pločica.



Slika 5.5. Gotov dizajn tiskane pločice

Na slici 4.5. je vidljivo kako izgleda gotov dizajn tiskane ploče. Crvena boja označava sve komponente i vodove s prednje strane dok su plavom bojom označeni vodovi i komponente sa stražnje strane. Umjesto da se spaja GND i dovode vodovi na svaku LE diodu, napravljene su dvije *ground* ravnine (engl. *ground plane*) i povezane su sa *via*-ma da ne bi postojala razlika potencijala nigdje. Također s druge strane se nalaze konektori za postavljanje Wemos D1 mini Lite, MPR121 i ostalih komponenata.

Kako bi se mogla izraditi tiskana pločica potrebno je napraviti *gerber* datoteke te ih poslati proizvođaču tiskanih pločica kako bi se mogla izraditi ista. *Gerber* datoteke su datoteke u kojima se nalaze određeni bitni dijelovi tiskane pločice, kao što su sloj bakra (predstavlja vodove kojima povezujemo komponente), oznake komponenti, pinovi i slično. Kako bi se generirale te datoteke u EAGLE-u prvo je potrebno provjeriti *design rules* odnosno pravila koja služe za dizajn pločice, a njih se može pronaći kod proizvođača pločica. Nakon što je to odrađeno mogu se poslati *gerber* datoteke proizvođaču da izradi tiskanu pločicu.

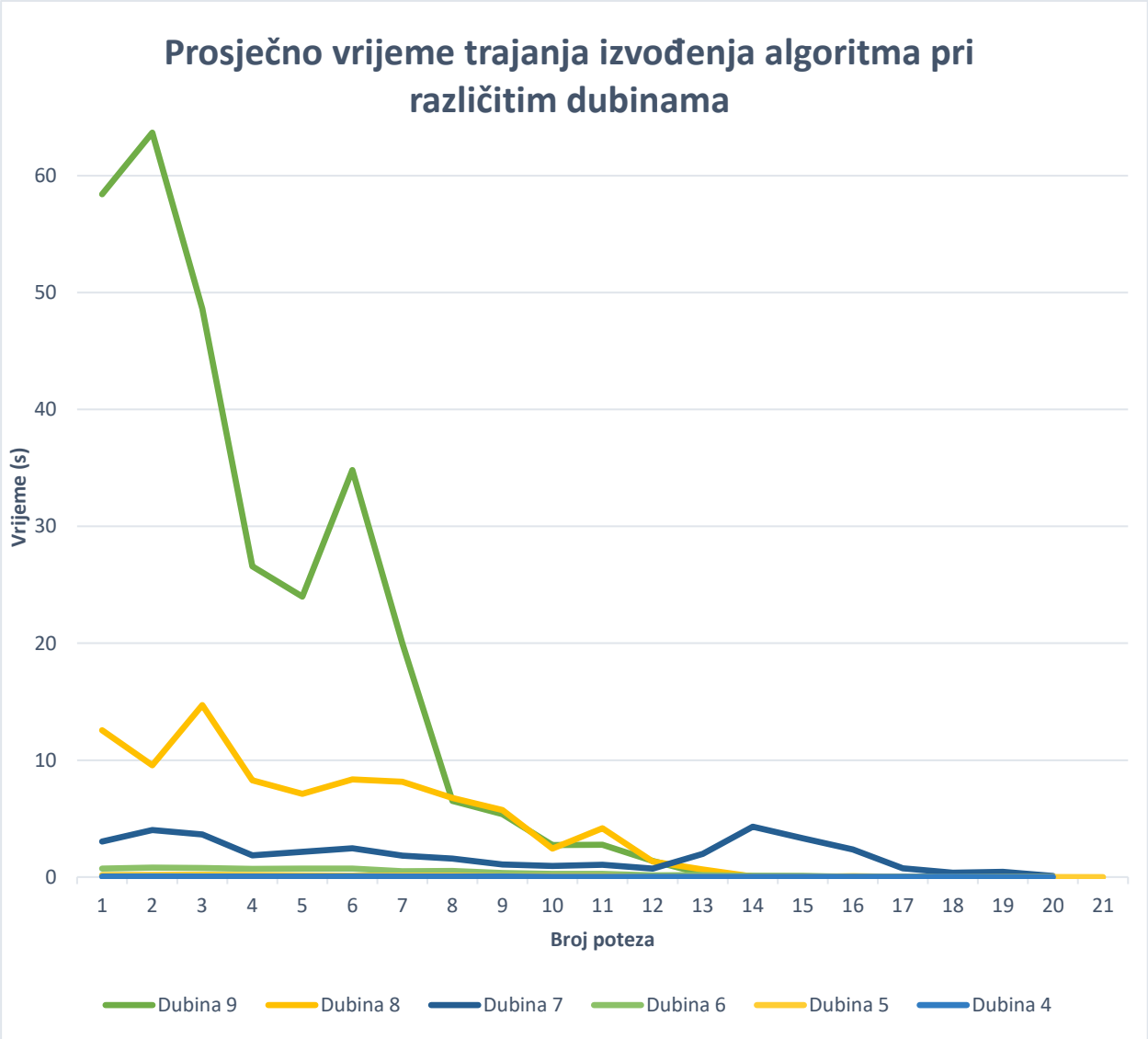


Slika 5.6. Izrađena tiskana pločica

Nakon izrade tiskane pločice potrebno je sve komponente zalemiti na za njih predviđena mjesta.

6. TESTIRANJE SUSTAVA

Da bi bili sigurni da programski kod radi kako je zamišljeno potrebno je testirati sustav. Sustav je testiran u načinu rada protiv računala na težoj razini, tj. testiran je algoritam s alfa-beta rezanjem do efektivne dubine 9. Testiranje je izvršeno tako da je igrano po 6 partija na svim dubinama počevši od 4 i provjeravano koliko dugo je trebalo računalu da odigra potez i do kojeg je poteza se došao i tko je pobijedio.



Grafikon 6.1. Prosječno vrijeme trajanja izvođenja algoritma pri različitim dubinama

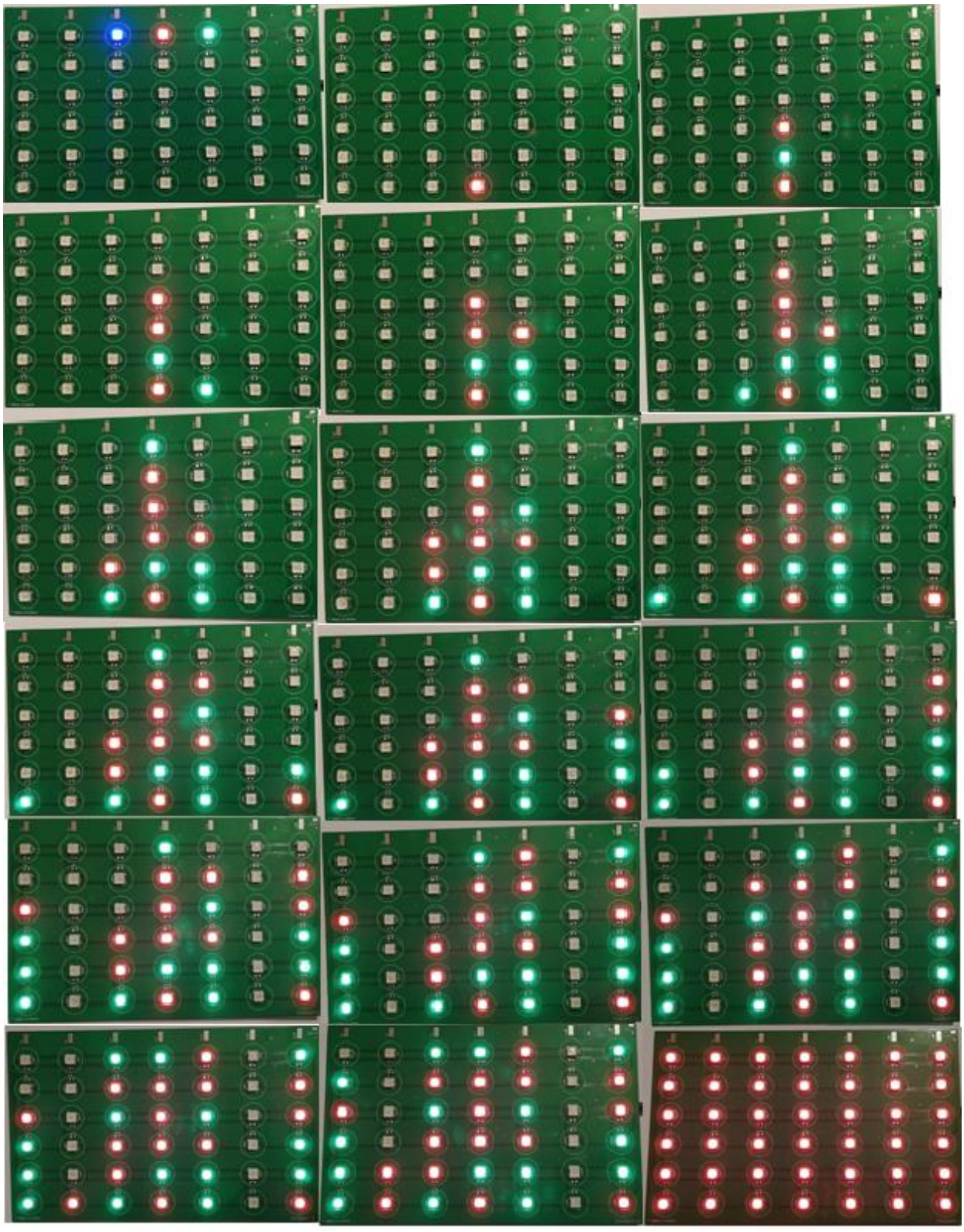
Kao što je vidljivo na grafikonu 6.1., vrijeme izvođenja poteza se drastično smanjuje sa smanjenjem dubine. Razlog tomu je taj što je funkcija alfa_beta() rekurzivna i s povećavanjem dubine se povećava i koliko puta će se pozvati ista. Na primjer za dubinu od 6 funkcija bi imala 7^6

ili 117649 čvorova stabla za posjetiti, dok za dubinu od 8, 7^8 što je 5764801 čvorova stala. Vidljivo je da vrijeme trajanja poteza ovisi o broju izvođenja funkcije. Također je vidljivo da se vrijeme izvođenja poteza smanjuje i s brojem poteza. Glavni razlog smanjenju jest taj što sa što više odigranih poteza smanjuje broj mogućih kombinacija u stablu igre i uz to postoji više šansi da se izađe iz funkcije s alfa-beta rezanjem.

Također, ako je najbolji potez za odigrati u najdesnijem stupcu, algoritmu će trebati duže da odredi koji je jer provjerava poteze s lijeva na desno. Isto vrijedi i obratno, ako je najbolji potez u nultom stupcu tada će trebati kraće vrijeme da se odigra isti. To vrijedi samo ako postoji implementirano alfa-beta rezanje, u samom *minimax* algoritmu bi se, bez obzira kako je stablo sortirano, proći kroz sve čvorove.

Na slici 5.1. vidi se primjer igre kroz poteze. U gornjem lijevom kutu slike nalazi se početni zaslon u kojem se bira način igre. Plava je čovjek protiv čovjeka, crvena protiv računala, teško i zelena protiv računala, lako. Na slici 5.1. odabrana je crvena, tj. dodirnuta je dodirna pločica T3.

Nakon toga računalo, koje igra prvo (crveni), igra polupotez. Potezi se mogu pratiti na sličicama od gornjeg lijevog kuta ka desnom. Na dnu slike 5.1. na srednjoj sličici je vidljivo da je računalo pobijedilo i na posljednjoj sličici (dolje, desno) se vide sve LE diode u crvenoj boji, što označava pobjedu igrača s crvenim žetonima.



Slika 6.1. Primjer igre protiv računala na "teško"

7. ZAKLJUČAK

U ovom radu je opisano kako napraviti automat za igranje igre Poveži 4 na Wemos D1 mini Lite mikrokontroleru pomoću Arduino IDE i u programskom jeziku C. Također je dizajnirana tiskana pločica u EAGLE programu.

Igra posjeduje opciju igranja računala protiv jednog igrača ili igranja dvaju igrača jednog protiv drugog. Igra protiv računala je realizirana u dvije težinske razine (lako i teško). Razina lako je implementirana tzv. *hard* kodiranjem tako da provjerava samo trenutno stanje na ploči i sprječava pobjedu protivnika. Razina teško je implementirana pomoći *minimax* algoritma s alfa-beta rezanjem. Testiranjem je ustanovljeno da je optimalna dubina, glede vremena potrebnog da računalo odigra potez i kvalitete odigranog poteza, 7.

Iako automat radi bez grešaka i samo sučelje je intuitivno za korisnike, postoje mogućnosti da se isti nadogradi. Zbog ograničene brzine korištenog mikrokontrolera u ovom radu postojalo je ograničenje s dubinom pretraživanja stabla jer je pretraživanje jednostavno predugo trajalo. Kao odgovor na taj problem mogao bi se koristiti isti mikrokontroler koji bi mogao slati stanje ploče na server koji bi odradio sve izračune i pretraživanja stabla te vratio najbolji potez mikrokontroleru.

LITERATURA

[1] S. Russel J., P. Norvig, Artifical Intelligence: A Modern Approach, Prentice Hall, New Jersey, 2003.

[2] G. Allen, AI.mil, Understanting AI Technology https://www.ai.mil/docs/Understanding%20AI%20Technology.pdf, srpanj 2020.

[3] Chessprogramming wiki, Leela Chess Zero, https://www.chessprogramming.org/Leela_Chess_Zero, srpanj 2020.

[4] Chessprogramming wiki, Stockfish, https://www.chessprogramming.org/Stockfish, srpanj 2020.

[5] J. P. Hinebaugh, More Board Game Education, Rowman & Littelfield Publishers, 2019.

[6] S. Edelkamp, BDD for Minimal Perfect Hashing: Merging Two State-Space Compression Techniques, Springer, 2017.

[7] V. Allis, A Knowledge-based Approach of Connect-Four The Game is Solved: White Wins, Department of Mathematics and Computer Science Vrije Universiteit Amsterdam, 1988.

[8] Wikipedia, EAGLE (program), https://en.wikipedia.org/wiki/EAGLE_(program), srpanj 2020.

[9] M. Maschler, E. Solan, S. Zamir, Game Theory, Cambridge University Press, Cambridge, 2013.

[10] M. Scarpino, Desingning Circuit Boards with EAGLE – Make High-Quality PCBs at Low Cost, Prentice Hall, 2014.

[11] Connect Four, http://web.mit.edu/sp.268/www/2010/connectFourSlides.pdf, lipanj 2020.

[12] D. McCann, Instructables circuits, Tic Tac Toe on Arduino With AI (Minimax Algorithm) ,https://www.instructables.com/id/Tic-Tac-Toe-on-Arduino-With-AI-Minimax-Algorithm/, lipanj 2020.

SAŽETAK

Glavni zadatak diplomskog rada je realizirati automat koji igra klasičnu logičku igru „Poveži 4“. Tehnologije korištene u radu su Arduino IDE (za razvoj programskog koda), Wemos D1 mini Lite, MPR121 i EAGLE (za dizajniranje tiskane pločice). Rad se sastoji od pet cjelina. Prvo poglavlje je uvod, u drugom su opisuje igra „Poveži 4“ i njena pravila. U trećem poglavlju su opisane tehnologije korištene u radu te je detaljno opisan korišteni algoritam. Četvrto poglavlje detaljno opisuje programski kod i dizajniranje tiskane pločice u EAGLE. Posljednje poglavlje je testiranje sustava u kojem je vidljivo kako izgleda jedna partija igre Poveži 4 na realiziranom automatu. Igranje protiv računala je realizirano pomoću *minimax* algoritma s alfa-beta rezanjem.

Ključne riječi: Alfa-beta rezanje, Arduino, EAGLE, Minimax, „Poveži 4“

ABSTRACT

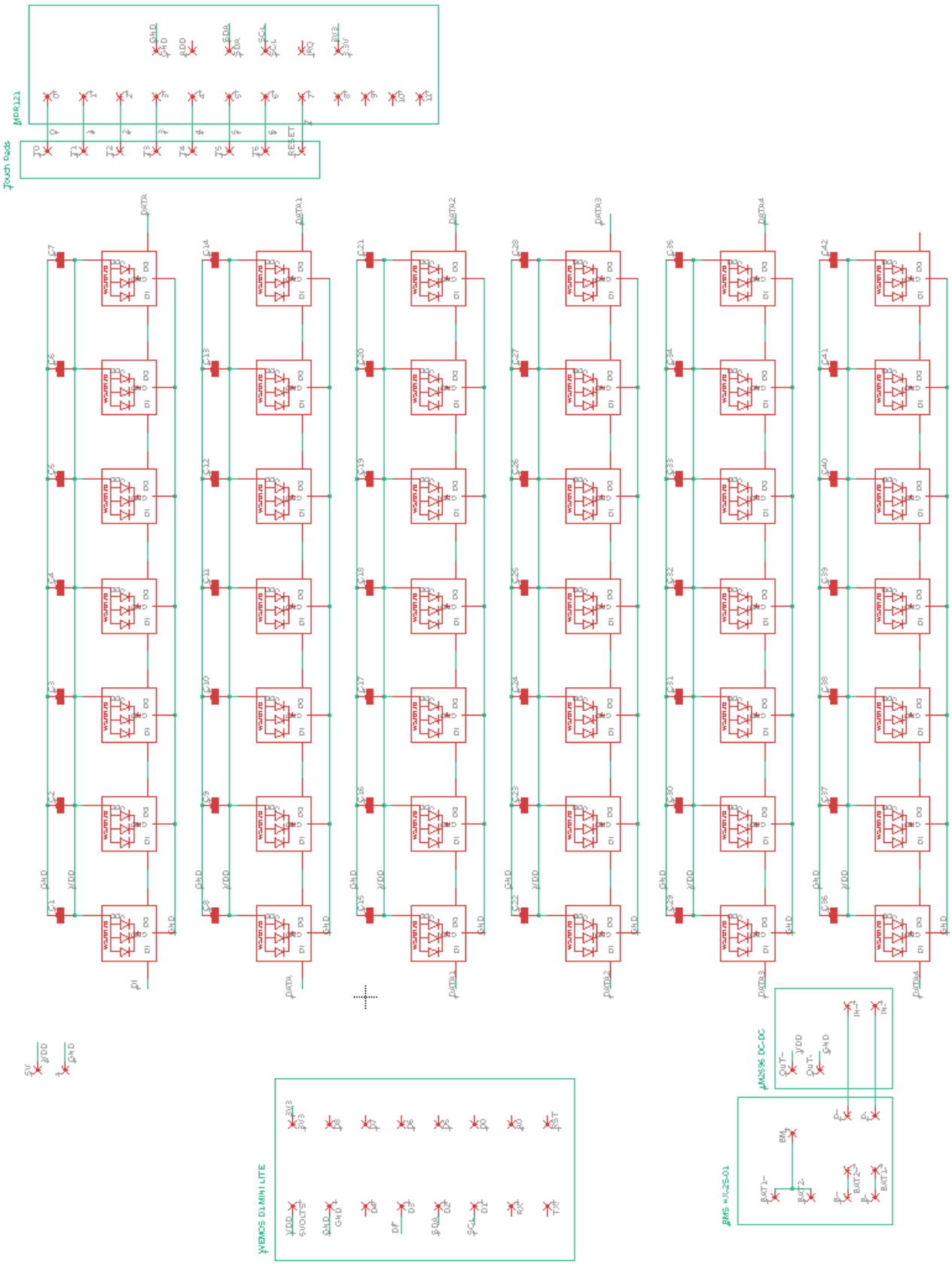
The main task of this thesis was to create an automat that would play the classic logic game of „Connect 4“. Technologies used in this environment are: Arduino IDE (IDE for code development), Wemos D1 mini Lite, MPR121 and EAGLE (used for the design of printed circuit board). The thesis is divided into five chapters. The first chapter is introduction, the second describes the game of „Connect 4“ and its rules. In the third chapter used technologies and algorithms are described and explained in detail. Forth chapter is explaining the code and the design of the printed circuit board in EAGLE in detail. Last chapter is testing, in which it can be seen how does one game of „Connect 4“ look like on the finished automat. Computer versus player mode is done with minimax algorithm and alpha-beta pruning.

Keywords: Alpha-beta pruning, Arduino, „Connect 4“, EAGLE, Minimax

ŽIVOTOPIS

Ilija Šišić je rođen 28. ožujka 1995. godine u Zagrebu. Osnovnoškolsko obrazovanje stječe u OŠ Žepče. Srednju školu KŠC „Don Bosco“ upisuje 2009. godine i 2013. godine stječe zvanje Tehničar za mehatroniku. Iste godine upisuje Fakultet strojarstva i računarstva sveučilišta u Mostaru koji završava 2017. godine i stječe zvanje sveučilišni prvostupnik inženjer računarstva. Nakon toga upisuje diplomski studij Fakulteta elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOG 1



PRILOG 2

```
#include <Adafruit_NeoPixel.h>
#include <Wire.h>
#include "Adafruit_MPR121.h"

#ifndef _BV
#define _BV(bit) (1 << (bit))
#endif
#define PIN D3
#define BRIGHTNESS 5
#define NUM_PIXELS 42
#define DELAY_ 200
Adafruit_MPR121 cap = Adafruit_MPR121();
Adafruit_NeoPixel LED = Adafruit_NeoPixel(NUM_PIXELS, PIN, NEO_GRB + NEO_KHZ800);
int tkoJeNaRedu = 1;
int poceo = tkoJeNaRedu;
uint16_t posljednjiDotaknut = 0;
uint16_t trenutniDotaknut = 0;
//int S[6][7] = {
//
//
//  {0, 1, 2, 3, 4, 5, 6},
//  {13, 12, 11, 10, 9, 8, 7},
//  {14, 15, 16, 17, 18, 19, 20},
//  {27, 26, 25, 24, 23, 22, 21},
//  {28, 29, 30, 31, 32, 33, 34},
//  {41, 40, 39, 38, 37, 36, 35},
//
//};
int S[6][7] = {
  {41, 40, 39, 38, 37, 36, 35},
  {28, 29, 30, 31, 32, 33, 34},
  {27, 26, 25, 24, 23, 22, 21},
  {14, 15, 16, 17, 18, 19, 20},
  {13, 12, 11, 10, 9, 8, 7},
  {0, 1, 2, 3, 4, 5, 6},
};

int T[6][7] = {
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0},
};
int stp1 = -1, stp2 = -1, stp3 = -1, stp4 = -1, stp5 = -1, stp6 = -1, stp7 = -1;
int *Stupac[7] = {&stp1, &stp2, &stp3, &stp4, &stp5, &stp6, &stp7};
int AI_zeton = 1;
int igrac_zeton = 2;
bool flag_e = false, flag_h = false;
int y = 0, br_poteza = 0;
int St1 = -1;

//*************************************************************************
void setup() {

  Serial.begin(115200);
  while (!Serial) { // needed to keep leonardo/micro from starting too fast!
    delay(10);
  }
```

```
  LED.begin();
  LED.setBrightness(BRIGHTNESS);
  LED.show();
  Serial.println("Adafruit MPR121 Capacitive Touch sensor test");
  delay(100);
  // Default address is 0x5A, if tied to 3.3V its 0stp5B
  // If tied to SDA its 0x5C and if SCL then 0stp5D
  if (!cap.begin(0x5A)) {
    Serial.println("MPR121 not found, check wiring?");
    while (1) {
      yield();
    }
  }
  Serial.println("MPR121 found!");
  pickMode();
}
//**************************************************************************
void loop() {

  if ((flag_e) && (tkoJeNaRedu == AI_zeton) && (!provjeriPobjedu(AI_zeton)) &&
(!provjeriPobjedu(igrac_zeton))) {
    Play(AI_Easy());
  }
  if ((flag_h) && (tkoJeNaRedu == AI_zeton) && (!provjeriPobjedu(AI_zeton)) &&
(!provjeriPobjedu(igrac_zeton))) {
    Play(AI_Hard(6));
  }

  St1 = touchSens();
  Play(St1);

  //  for(int i=0; i<NUM_PIXELS; i++) {
  //    LED.setPixelColor(i, 0, 0, 0, 0);,
  //  }
  //  LED.show();
  //  for(int i=0; i<NUM_PIXELS; i++) {
  //    LED.setPixelColor(i, 155, 0, 0);
  //    LED.show();
  //    delay(DELAY_);
  //  }

}
//**************************************************************************
int touchSens() {
  int a = -1;
  trenutniDotaknut = cap.touched();

  for (uint8_t i = 0; i < 12; i++) {
    if ((trenutniDotaknut & _BV(i)) && !(posljednjiDotaknut & _BV(i)) ) {
      a = i;
    }
    if (!(trenutniDotaknut & _BV(i)) && (posljednjiDotaknut & _BV(i)) ) {
    }
  }
  posljednjiDotaknut = trenutniDotaknut;
  return a;

}

//**************************************************************************
void printArray( const int a[6][7] ) {
  // loop through array's redak
```

```
  for ( int i = 0; i < 6; ++i ) {
    // loop through columns of current row
    for ( int j = 0; j < 7; ++j )
      Serial.print (a[ i ][ j ] );
    Serial.print (" ");
    Serial.println (" " ) ; // start new line of output
  }
}
//**************************************************************************
void odigrajPotez(int x, int y) {
  if (tkoJeNaRedu == 1) {
    T[x][y] = 1;
    //    Serial.print(tkoJeNaRedu); Serial.print("  "); Serial.println(AI_zeton);
    br_poteza++;
    for (int i = 5; i > x; i--) {
      LED.setPixelColor(S[i][y], 155, 0, 0);
      LED.show();
      delay(20);
      LED.setPixelColor(S[i][y], 0, 0, 0);
      LED.show();
    }
    LED.setPixelColor(S[x][y], 155, 0, 0);
    LED.show();
    delay(DELAY_);

    tkoJeNaRedu = 2;
    //    for (int i = 0; i < 6; i++) {
    //      for (int j = 0; j < 7; j++) {
    //
    //        Serial.print(T[i][j]);
    //        Serial.print(" ");
    //      }
    //      Serial.println();
    //    }
    if (provjeriPobjedu(1)) {
      delay(1000);
      greenScreen();
      return;
    }
    else if (br_poteza >= 42) {
      drawScreen();
      return;
    }
  }
  else {
    //    Serial.print(tkoJeNaRedu); Serial.print("  "); Serial.println(AI_zeton);
    T[x][y] = 2;
    br_poteza++;
    for (int i = 5; i > x; i--) {
      LED.setPixelColor(S[i][y], 0, 155, 0);
      LED.show();
      delay(20);
      LED.setPixelColor(S[i][y], 0, 0, 0);
      LED.show();
    }
    LED.setPixelColor(S[x][y], 0, 155, 0);
    LED.show();
    delay(DELAY_);
    tkoJeNaRedu = 1;
    if (provjeriPobjedu(2)) {
      delay(1000);
      redScreen();
```

```
      return;
    }
    else if (br_poteza >= 42) {
      drawScreen();
      return;
    }

  }
  //  Serial.println(score_ploca(1));
  //  Serial.println("Odigraj potez Xs");
  //  for (int i = 0; i < 7; i++) {
  //
  //    Serial.print(i);
  //    Serial.print("=");
  //    Serial.print(*Stupac[i]);
  //    Serial.println(" ");
  //  }
}

//**********************************************************************
bool provjeriPobjedu(int igrac) {

  // vodoravna provjera
  for (int j = 0; j < 7 - 3 ; j++ ) {
    for (int i = 0; i < 6; i++) {
      if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
          == igrac) {
        return true;
      }
    }
  }
  // okomita provjera
  for (int i = 0; i < 6 - 3 ; i++ ) {
    for (int j = 0; j < 7; j++) {
      if (T[i][j] == igrac && T[i + 1][j] == igrac && T[i + 2][j] == igrac && T[i + 3][j]
          == igrac) {
        return true;
      }
    }
  }
  // rastuca dijagonalna provjera
  for (int i = 3; i < 6; i++) {
    for (int j = 0; j < 7 - 3; j++) {
      if (T[i][j] == igrac && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == igrac && T[i
          - 3][j + 3] == igrac)
        return true;
    }
  }
  // padajuca dijagonalna provjera
  for (int i = 3; i < 6; i++) {
    for (int j = 3; j < 7; j++) {
      if (T[i][j] == igrac && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == igrac)
        return true;
    }
  }
  return false;
}
//**********************************************************************
void greenScreen() {

  for (int i = 0; i < NUM_PIXELS; i++) {
```

```
    LED.setPixelColor(i, 155, 0, 0);
    LED.show();
    delay(100);
  }
  while (1) {
    if (touchSens() > 0) {
      resetGame();
      changeigrac();
      delay(500);
      pickMode();
      break;
    }
    yield();
  }
}
//*************************************************************************
void redScreen() {
  for (int i = 0; i < NUM_PIXELS; i++) {
    LED.setPixelColor(i, 0, 155, 0);
    LED.show();
    delay(100);
  }
  while (1) {
    if (touchSens() > 0) {
      resetGame();
      changeigrac();
      delay(500);
      pickMode();
      break;
    }
    yield();
  }
}
//*************************************************************************
void drawScreen() {
  for (int i = 0; i < NUM_PIXELS; i++) {
    LED.setPixelColor(i, 0, 155, 155);
    LED.show();
    delay(100);
  }
  while (1) {
    if (touchSens() > 0) {
      resetGame();
      changeigrac();
      delay(500);
      pickMode();
      break;
    }
    yield();
  }
  //if(button) then reset

}
//*************************************************************************
void pickMode() {
  LED.setPixelColor(2, 0, 0, 155);
  LED.setPixelColor(3, 155, 0, 0);
  LED.setPixelColor(4, 0, 155, 0);
  LED.show();
  while (1) {
    St1 = touchSens();
    if (St1 == 2) { //PvP
```

```
      flag_e = false;
      flag_h = false;
      St1 = -1;
      for (int i = 0; i < NUM_PIXELS; i++)
      {
        LED.setPixelColor(i, 0, 0, 0);
        LED.show();
      }
      delay(750);

      return;
    }
    if (St1 == 3) { //PvE HARD
      flag_e = false;
      flag_h = true;
      St1 = -1;
      for (int i = 0; i < NUM_PIXELS; i++)
      {
        LED.setPixelColor(i, 0, 0, 0);
        LED.show();
      }
      // delay(750);

      return;
    }
    if (St1 == 4) { //PvE EASY
      flag_h = false;
      flag_e = true;
      St1 = -1;
      for (int i = 0; i < NUM_PIXELS; i++)
      {
        LED.setPixelColor(i, 0, 0, 0);
        LED.show();
      }
      delay(750);

      return;
    }
    yield();
  }
}



//************************************************************************
void newGame() {
  br_poteza = 0;
  flag_e = false;
  flag_h = false;
  tkoJeNaRedu = poceo;
  for (int i = 0; i < NUM_PIXELS; i++)
  {
    LED.setPixelColor(i, 0, 0, 0);
    LED.show();
  }
  for (int i = 0; i < 6; i++)
  {

    for (int j = 0; j < 7; j++)
    {
      *Stupac[j] = -1;
```

```
      T[i][j] = 0;
    }
  }
  pickMode();
}
//*************************************************************************
void resetGame() {
  br_poteza = 0;
  flag_e = false;
  flag_h = false;

  for (int i = 0; i < NUM_PIXELS; i++)
  {
    LED.setPixelColor(i, 0, 0, 0);
    LED.show();
  }
  for (int i = 0; i < 6; i++)
  {

    for (int j = 0; j < 7; j++)
    {
      *Stupac[j] = -1;
      T[i][j] = 0;
    }
  }
}
//*************************************************************************
void changeigrac() {
  int temp;
  //  temp = AI_zeton;
  //  AI_zeton = igrac_zeton;
  //  igrac_zeton = temp;
  if (poceo == AI_zeton) {
    tkoJeNaRedu = igrac_zeton;
    poceo = igrac_zeton;
  }
  else {
    tkoJeNaRedu = AI_zeton;
    poceo = AI_zeton;
  }
  return;
}
//*************************************************************************
int AI_Easy () {
  int st;

  st = AI_1();
  if (st != -1) {
    if (st == 0) st = 9;
    return st;
  }
  st = AI_3();
  if (st != -1) {
    if (st == 0) st = 9;
    return st;
  }
  st = AI_2();
  if (st != -1) {
    if (st == 0) st = 9;
    return st;
  }
  st = AI_Random();
```

```
  if (st == 0) st = 9;
  return st;

}
//*************************************************************************
int AI_Hard(int dubina) {
  int x = najboljiPotezAlfaBeta(dubina); //15 sekundi po potezu(dubina 8), 5sec dubina 7
  if (x == 0) x = 9;
  return x;
}



//*************************************************************************
void Play(int a) {
  if (a == 9) {
    stp1 = stp1 + 1;
    y = 0;
    if (stp1 < 6) {
      odigrajPotez(stp1, y);
      return;
    }
  }
  else if (a == 1) {
    stp2 = stp2 + 1;

    y = 1;
    if (stp2 < 6) {
      odigrajPotez(stp2, y);
      return;
    }
  }
  else if (a == 2) {
    stp3 = stp3 + 1;

    y = 2;
    if (stp3 < 6) {
      odigrajPotez(stp3, y);
      return;
    }
  }
  else if (a == 3) {
    stp4 = stp4 + 1;

    y = 3;
    if (stp4 < 6) {
      odigrajPotez(stp4, y);
      return;
    }
  }
  else if (a == 4) {
    stp5 = stp5 + 1;

    y = 4;
    if (stp5 < 6) {
      odigrajPotez(stp5, y);
      return;
    }
  }
  else if (a == 5) {
    stp6 = stp6 + 1;
```

```
    y = 5;
    if (stp6 < 6) {
      odigrajPotez(stp6, y);
      return;
    }
  }
  else if (a == 6) {
    stp7 = stp7 + 1;

    y = 6;
    if (stp7 < 6) {
      odigrajPotez(stp7, y);
      return;
    }
  }
  else if (a == 7) {
    if (St1 == 7) newGame();
  }
}


//**************************************************************************
int AI_Random() {

  int randomNum = random(0, 7);
  while (T[5][randomNum] > 0) {
    randomNum = random(0, 7);
  }
  return randomNum;

}
//**************************************************************************
int AI_1()
{
  int pot = -1;
  for (int i = 0; i < 7; i++) {
    if ((*Stupac[i] + 1) < 6) {
      T[*Stupac[i] + 1][i] = AI_zeton;
      if (provjeriPobjedu(AI_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        return i;
      }
      if (provjeriPobjedu(AI_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        pot = i;
      }
      T[*Stupac[i] + 1][i] = igrac_zeton;
      if (provjeriPobjedu(AI_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        return i;
      }
      if (provjeriPobjedu(igrac_zeton)) {
        T[*Stupac[i] + 1][i] = 0;
        pot = i;
      }

      T[*Stupac[i] + 1][i] = 0;
    }

  }
  return pot;
```

```
}
//*************************************************************************
int AI_2() {
  int i = 0;
  while (i < 50) {
    int stupac = AI_Random();

    if ((*Stupac[stupac] + 2) < 6) {


      if (tkoJeNaRedu == AI_zeton) {
        T[*Stupac[stupac] + 2][stupac] = igrac_zeton;
        if (!provjeriPobjedu(igrac_zeton)) {
          T[*Stupac[stupac] + 2][stupac] = 0;
          return stupac;
        }
        else {
          T[*Stupac[stupac] + 2][stupac] = 0;
          i++;
        }
      }
      if (tkoJeNaRedu == igrac_zeton) {
        T[*Stupac[stupac] + 2][stupac] = AI_zeton;
        if (!provjeriPobjedu(AI_zeton)) {
          T[*Stupac[stupac] + 2][stupac] = 0;
          return stupac;
        }
        else {
          T[*Stupac[stupac] + 2][stupac] = 0;
          i++;
        }
      }
    }


  }
  return -1;
}
//*************************************************************************
int AI_3() {
  for (int i = 0; i < 6; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      if (tkoJeNaRedu == igrac_zeton) {
        // kombinacije 1-3
        if (T[i] [j] == 0 && T[i] [j + 1] == 0 && T[i][ j + 2] == AI_zeton && T[i] [j +
            3] == AI_zeton && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 1] != 0 && T[i -
              1][ j + 4] != 0))
            return j + 1;
        }
        // kombinacije 4-6
        if (T[i][ j] == 0 && T[i][ j + 1] == AI_zeton && T[i][ j + 2] == 0 && T[i ][ j +
            3] == AI_zeton && T[i ][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 2] != 0 && T[i -
              1][ j + 4] != 0))
            return j + 2;
        }
        // kombinacije 7-9
```

```
        if (T[i][ j] == 0 && T[i ][ j + 1] == AI_zeton && T[i][j + 2] == AI_zeton &&
            T[i][ j + 3] == 0 && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 3] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 3;
        }
      }
      else if (tkoJeNaRedu == AI_zeton) {
        // kombinacije 1-3
        if (T[i] [j] == 0 && T[i] [j + 1] == 0 && T[i][ j + 2] == igrac_zeton && T[i] [j
            + 3] == igrac_zeton && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 1] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 1;
        }
        // kombinacije 4-6
        if (T[i][ j] == 0 && T[i][ j + 1] == igrac_zeton && T[i][ j + 2] == 0 && T[i ][ j
            + 3] == igrac_zeton && T[i ][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 2] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 2;
        }
        // kombinacije 7-9
        if (T[i][ j] == 0 && T[i ][ j + 1] == igrac_zeton && T[i][j + 2] == igrac_zeton
            && T[i][ j + 3] == 0 && T[i][ j + 4] == 0)
        {
          if ((i == 0) || (i > 0 && T[i - 1][ j] != 0 && T[i - 1][ j + 3] != 0 && T[i -
            1][ j + 4] != 0))
            return j + 3;
        }
      }
    }
  }
  return -1;
}
//*********************************************************************
int evaluate() {

  if (provjeriPobjedu(AI_zeton)) {
    return 100000;
  }
  else if (provjeriPobjedu(igrac_zeton))
  {
    return -100000;
  }
  else if (!imaLiPoteza()) {
    return 0;
  }
  else return -1;
}
//*********************************************************************
bool imaLiPoteza() {
  for (int i = 0; i < 6; i++) {
    for (int j = 0; j < 7; j++) {
      if (T[i][j] == 0) {
        return true;
      }
    }
  }
```

```
  return false;
}


//*************************************************************************
int miniMax(int dubina, int maximisingIgrac) {
  yield();
  int krajnjiList = evaluate();
  //  Serial.println(krajnjiList);
  //  yield();
  if (krajnjiList != -1) {
    return krajnjiList;
  }
  else if (dubina == 0) {
    return score_ploca(AI_zeton);
  }




  if (maximisingIgrac == igrac_zeton) {                              //***MAX***

    int bestScore = -10000;
    for (int i = 0; i < 7; i++) {
      if (*Stupac[i] < 5) {
        if ( T[*Stupac[i] + 1][i] == 0) {


          T[*Stupac[i] + 1][i] = igrac_zeton;
          *Stupac[i] = *Stupac[i] + 1;
          br_poteza++;
          int score = miniMax(dubina - 1, AI_zeton);
          bestScore = max(bestScore, score);
          //Serial.print("MAX");Serial.println(bestScore);
          br_poteza--;
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1][i] = 0;

        }
      }
    }
    return bestScore;

  }
  else if (maximisingIgrac == AI_zeton) {
//***MIN***
    int bestScore = 10000;
    for (int i = 0; i < 7; i++) {
      if (*Stupac[i] < 5) {
        if ( T[*Stupac[i] + 1 ][i] == 0) {


          T[*Stupac[i] + 1][i] = AI_zeton;
          *Stupac[i] = *Stupac[i] + 1;
          br_poteza++;
          int score = miniMax(dubina - 1, igrac_zeton);
          bestScore = min(bestScore, score);
          //Serial.print("MIN");Serial.println(bestScore);
          br_poteza--;
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1 ][i] = 0;
```

```
        }
      }
    }
    return bestScore;
  }
}


//*************************************************************************

int najboljiPotez(int dubina) {
  int stupac = 0;
  int bestVal = -10000;
  for (int i = 0; i < 7; i++) {
    if (*Stupac[i] < 5) {
      if ( T[*Stupac[i] + 1][i] == 0) {

        T[*Stupac[i] + 1][i] = AI_zeton;
        *Stupac[i] = *Stupac[i] + 1;
        int moveVal = miniMax(dubina, AI_zeton);  //SCORE - MOVEVAL
        // int moveVal = alfa_beta(dubina, -10000000, 10000000, AI_zeton);

        *Stupac[i] = *Stupac[i] - 1;
        T[*Stupac[i] + 1][i] = 0;

        if (moveVal > bestVal) {
          stupac = i;
          bestVal = moveVal;
        }
      }
    }
  }
  return stupac;
}
//*************************************************************************
int score_ploca(int igrac) {
  int br = 0;
  int  score = 0;
  //************************************Center
  for (int i = 0; i < 6; i++) {
    int    j = 3;

    if (T[i][j] == igrac) {
      br++;
    }
  }
  score = score + (br * 5);


  for (int i = 0; i < 6; i++) {
    for (int j = 1; j < 7 - 4; j++) {
      if (T[i][j - 1] == 0 && T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] ==
          igrac && T[i][j + 3] == 0) {
        score = score + 20;
      }
    }
  }

  //************************************Vodoravno
  for (int i = 0; i < 6; i++) {
    for (int j = 0; j < 7 - 3; j++) {
```

```
    if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
        == igrac) {
      score = score + 100;
    }
    if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
        == 0
        || T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == 0 && T[i][j + 3]
        == igrac
        || T[i][j] == igrac && T[i][j + 1] == 0 && T[i][j + 2] == igrac && T[i][j + 3]
        == igrac
        || T[i][j] == 0 && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
        == igrac) {
      score = score + 5;
    }
    if (T[i][j] == igrac && T[i][j + 1] == igrac && T[i][j + 2] == 0 && T[i][j + 3] ==
        0
        || T[i][j] == igrac && T[i][j + 1] == 0 && T[i][j + 2] == 0 && T[i][j + 3] ==
        igrac
        || T[i][j] == 0 && T[i][j + 1] == igrac && T[i][j + 2] == igrac && T[i][j + 3]
        == 0
        || T[i][j] == 0 && T[i][j + 1] == 0 && T[i][j + 2] == igrac && T[i][j + 3] ==
        igrac

        || T[i][j] == igrac && T[i][j + 1] == 0 && T[i][j + 2] == igrac && T[i][j + 3]
        == 0
        || T[i][j] == 0 && T[i][j + 1] == igrac && T[i][j + 2] == 0 && T[i][j + 3] ==
        igrac)
    {
      score = score + 2;
    }
    if (T[i][j] == igrac_zeton && T[i][j + 1] == igrac_zeton && T[i][j + 2] ==
        igrac_zeton && T[i][j + 3] == 0
        || T[i][j] == igrac_zeton && T[i][j + 1] == igrac_zeton && T[i][j + 2] == 0 &&
        T[i][j + 3] == igrac_zeton
        || T[i][j] == igrac_zeton && T[i][j + 1] == 0 && T[i][j + 2] == igrac_zeton &&
        T[i][j + 3] == igrac_zeton
        || T[i][j] == 0 && T[i][j + 1] == igrac_zeton && T[i][j + 2] == igrac_zeton &&
        T[i][j + 3] == igrac_zeton) {
      score = score - 40;
    }
  }
}
//***********************************Okomito
for (int i = 0; i < 6 - 3 ; i++ ) {
  for (int j = 0; j < 7; j++) {
    if (T[i][j] == igrac && T[i + 1][j] == igrac && T[i + 2][j] == igrac && T[i + 3][j]
        == igrac) {
      score = score + 100;
    }
    if (T[i][j] == igrac && T[i + 1][j] == igrac && T[i + 2][j] == igrac && T[i + 3][j]
        == 0
        || T[i][j] == igrac && T[i + 1][j] == igrac && T[i + 2][j] == 0 && T[i + 3][j]
        == igrac
        || T[i][j] == igrac && T[i + 1][j] == 0 && T[i + 2][j] == igrac && T[i + 3][j]
        == igrac
        || T[i][j] == 0 && T[i + 1][j] == igrac && T[i + 2][j] == igrac && T[i + 3][j]
        == igrac)
    {
      score = score + 5;
    }
    if (T[i][j] == igrac && T[i + 1][j] == igrac && T[i + 2][j] == 0 && T[i + 3][j] ==
        0
```

```
        || T[i][j] == 0 && T[i + 1][j] == igrac && T[i + 2][j] == igrac && T[i + 3][j]
        == 0
        || T[i][j] == 0 && T[i + 1][j] == 0 && T[i + 2][j] == igrac && T[i + 3][j] ==
        igrac
        || T[i][j] == 0 && T[i + 1][j] == igrac && T[i + 2][j] == 0 && T[i + 3][j] ==
        igrac
        || T[i][j] == igrac && T[i + 1][j] == 0 && T[i + 2][j] == igrac && T[i + 3][j]
        == 0
        || T[i][j] == igrac && T[i + 1][j] == 0 && T[i + 2][j] == 0 && T[i + 3][j] ==
        igrac)

    {
      score = score + 2;
    }
    if (T[i][j] == igrac_zeton && T[i + 1][j] == igrac_zeton && T[i + 2][j] ==
        igrac_zeton && T[i + 3][j] == 0
        || T[i][j] == igrac_zeton && T[i + 1][j] == igrac_zeton && T[i + 2][j] == 0 &&
        T[i + 3][j] == igrac_zeton
        || T[i][j] == igrac_zeton && T[i + 1][j] == 0 && T[i + 2][j] == igrac_zeton &&
        T[i + 3][j] == igrac_zeton
        || T[i][j] == 0 && T[i + 1][j] == igrac_zeton && T[i + 2][j] == igrac_zeton &&
        T[i + 3][j] == igrac_zeton) {
      score = score - 40;

    }
  }
}
//**************************************rastuce dijagonalno
for (int i = 3; i < 6; i++) {
  for (int j = 0; j < 7 - 3; j++) {
    if (T[i][j] == igrac && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == igrac && T[i
        - 3][j + 3] == igrac) {
      score = score + 100;
    }
    if (T[i][j] == igrac && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == igrac && T[i
        - 3][j + 3] == 0
        || T[i][j] == igrac && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == 0 && T[i
        - 3][j + 3] == igrac
        || T[i][j] == igrac && T[i - 1][j + 1] == 0 && T[i - 2][j + 2] == igrac && T[i
        - 3][j + 3] == igrac
        || T[i][j] == 0 && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == igrac && T[i
        - 3][j + 3] == igrac) {
      score = score + 5;
    }

    if (T[i][j] == igrac && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == 0 && T[i -
        3][j + 3] == 0
        || T[i][j] == 0 && T[i - 1][j + 1] == 0 && T[i - 2][j + 2] == igrac && T[i -
        3][j + 3] == igrac
        || T[i][j] == 0 && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == igrac && T[i
        - 3][j + 3] == 0
        || T[i][j] == igrac && T[i - 1][j + 1] == 0 && T[i - 2][j + 2] == 0 && T[i -
        3][j + 3] == igrac
        || T[i][j] == 0 && T[i - 1][j + 1] == igrac && T[i - 2][j + 2] == 0 && T[i -
        3][j + 3] == igrac
        || T[i][j] == igrac && T[i - 1][j + 1] == 0 && T[i - 2][j + 2] == igrac && T[i
        - 3][j + 3] == 0)
    {
      score = score + 2;
    }
    if (T[i][j] == igrac_zeton && T[i - 1][j + 1] == igrac_zeton && T[i - 2][j + 2] ==
        igrac_zeton && T[i - 3][j + 3] == 0
```

```
          || T[i][j] == igrac_zeton && T[i - 1][j + 1] == igrac_zeton && T[i - 2][j + 2]
          == 0 && T[i - 3][j + 3] == igrac_zeton
          || T[i][j] == igrac_zeton && T[i - 1][j + 1] == 0 && T[i - 2][j + 2] ==
          igrac_zeton && T[i - 3][j + 3] == igrac_zeton
          || T[i][j] == 0 && T[i - 1][j + 1] == igrac_zeton && T[i - 2][j + 2] ==
          igrac_zeton && T[i - 3][j + 3] == igrac_zeton) {
        score = score - 40;
      }
    }
  }
  //************************************ padajuce dijagonalno
  for (int i = 3; i < 6; i++) {
    for (int j = 3; j < 7; j++) {
      if (T[i][j] == igrac && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == igrac) {
        score = score + 100;
      }
      if (T[i][j] == igrac && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == 0
          || T[i][j] == igrac && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == 0 && T[i
          - 3][j - 3] == igrac
          || T[i][j] == igrac && T[i - 1][j - 1] == 0 && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == igrac
          || T[i][j] == 0 && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == igrac) {
        score = score + 5;
      }
      if (T[i][j] == igrac && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == 0 && T[i -
          3][j - 3] == 0
          || T[i][j] == 0 && T[i - 1][j - 1] == 0 && T[i - 2][j - 2] == igrac && T[i -
          3][j - 3] == igrac
          || T[i][j] == 0 && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == 0
          || T[i][j] == igrac && T[i - 1][j - 1] == 0 && T[i - 2][j - 2] == 0 && T[i -
          3][j - 3] == igrac
          || T[i][j] == 0 && T[i - 1][j - 1] == igrac && T[i - 2][j - 2] == 0 && T[i -
          3][j - 3] == igrac
          || T[i][j] == igrac && T[i - 1][j - 1] == 0 && T[i - 2][j - 2] == igrac && T[i
          - 3][j - 3] == 0)
      {
        score = score + 2;
      }
      if (T[i][j] == igrac_zeton && T[i - 1][j - 1] == igrac_zeton && T[i - 2][j - 2] ==
          igrac_zeton && T[i - 3][j - 3] == 0
          || T[i][j] == igrac_zeton && T[i - 1][j - 1] == igrac_zeton && T[i - 2][j - 2]
          == 0 && T[i - 3][j - 3] == igrac_zeton
          || T[i][j] == igrac_zeton && T[i - 1][j - 1] == 0 && T[i - 2][j - 2] ==
          igrac_zeton && T[i - 3][j - 3] == igrac_zeton
          || T[i][j] == 0 && T[i - 1][j - 1] == igrac_zeton && T[i - 2][j - 2] ==
          igrac_zeton && T[i - 3][j - 3] == igrac_zeton) {
        score = score - 40;
      }
    }
  }



  return score;
}

//*********************************************************************
//alfa-BETA
```

```
int alfa_beta(int dubina, int alfa, int beta, int maximisingIgrac) {
  yield();
  int krajnjiList = evaluate();
  // Serial.println(krajnjiList);
  //  yield();
  if (krajnjiList != -1) {
    return krajnjiList;
  }
  else if (dubina == 0) {
    return score_ploca(AI_zeton);
  }



  if (maximisingIgrac == AI_zeton) {                              //***MAX***


    int bestScore = -10000000;
    for (int i = 0; i < 7; i++) {
      if (*Stupac[i] < 5) {
        if ( T[*Stupac[i] + 1][i] == 0) {


          T[*Stupac[i] + 1][i] = AI_zeton;
          *Stupac[i] = *Stupac[i] + 1;
          br_poteza++;
          // int score = alfa_beta(dubina - 1, alfa, beta, igrac_zeton);


          bestScore = max(bestScore, alfa_beta(dubina - 1, alfa, beta, igrac_zeton));

          alfa = max(alfa, bestScore);


          if (alfa >= beta) {

            //            Serial.print(' ');
            //            Serial.println(beta);
            br_poteza--;
            *Stupac[i] = *Stupac[i] - 1;
            T[*Stupac[i] + 1][i] = 0;
            break;
          }
          //Serial.print("MAX");Serial.println(bestScore);
          br_poteza--;
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1][i] = 0;

        }
      }
    }
    return bestScore;

  }
  else {       //***MIN***

    int bestScore = 10000000;
    for (int i = 0; i < 7; i++) {
      if (*Stupac[i] < 5) {
        if ( T[*Stupac[i] + 1 ][i] == 0) {
```

```
          T[*Stupac[i] + 1][i] = igrac_zeton;
          *Stupac[i] = *Stupac[i] + 1;
          br_poteza++;
          //int score = alfa_beta(dubina - 1, alfa, beta, AI_zeton);
          bestScore = min(bestScore, alfa_beta(dubina - 1, alfa, beta, AI_zeton));
          beta = min(beta, bestScore);

          if (alfa >= beta) {

            //            Serial.print(' ');
            //            Serial.println(beta);
            br_poteza--;
            *Stupac[i] = *Stupac[i] - 1;
            T[*Stupac[i] + 1 ][i] = 0;
            break;
          }
          //Serial.print("MIN");Serial.println(bestScore);
          br_poteza--;
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1 ][i] = 0;
        }
      }
    }
    return bestScore;
  }
}
//**************************************************************************

int najboljiPotezAlfaBeta(int dubina) {
  int stupac = 0;
  int bestVal = -100000000;
  int moveVal;
  for (int i = 0; i < 7; i++) {
    if (*Stupac[i] < 5) {
      if ( T[*Stupac[i] + 1][i] == 0) {

        T[*Stupac[i] + 1][i] = AI_zeton;
        *Stupac[i] = *Stupac[i] + 1;
        if (provjeriPobjedu(AI_zeton)) {
          *Stupac[i] = *Stupac[i] - 1;
          T[*Stupac[i] + 1][i] = 0;
          return i;
        }
        //int moveVal = miniMax(dubina, AI_zeton);  //SCORE - MOVEVAL
        moveVal = alfa_beta(dubina, -100000000, 100000000, igrac_zeton);
        //        Serial.print(i);
        //        Serial.print('=');
        //        Serial.println(moveVal);
        *Stupac[i] = *Stupac[i] - 1;
        T[*Stupac[i] + 1][i] = 0;

        if (moveVal >= bestVal) {
          stupac = i;
          bestVal = moveVal;
        }
      }
    }
  }
  return stupac;
}
```