SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mario Nižić

IZRADA IGRE ŠAHA U UNITY OKRUŽENJU

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mario Nižić

JMBAG: 0023120330

Studij: Informacijski sustavi

IZRADA IGRE ŠAHA U UNITY OKRUŽENJU

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, rujan 2021.

Mario Nižić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom završnom radu prikazana je izrada računalne igre šaha u 2D okruženju korištenjem programskog alata Unity. Početkom rada opisane su metode i alati korišteni u izradi ovog rješenja te plan provedbe i proces razmišljanja kojim se došlo do konačnog rezultata projekta. Potom su detaljno opisani alati, točnije dva glavna, Unity i Microsoft Visual Studio 2019. Idućim poglavljem upoznajemo se s igrom šaha odnosno kako se igra i koje uloge koja figura na ploči ima. Također se zaključuje koji je cilj igre i kad je ista gotova. U najvažnijem dijelu razrađujemo proces izrade programskog rješenja. Poglavljima „Scena i UI“ te „Ploča“ više se dotičemo Unity-a te prikaza izgleda igre, dok u poglavljima „Upravljanje i logika igre“ i „Figure i mehanika“ ulazimo u suštinu rada te se kroz kod dotičemo najvažnijeg dijela implementacije. Na koncu rada zaključujemo čitavu stvar i remiziramo cjelokupan rad.

Ključne riječi: Unity, C#, Microsoft Visual Studio 2019, šah, računalna igra, 2D, programiranje

Sadržaj

1. Uvod

Svakom tko je ikada igrao video igrice barem je jednom u životu kroz misli prošla ideja samostalne izrade i implementacije igre koja bi bila plod vlastite mašte i koja bi ugodila svim osobnim željama. Zamišljajući njenu atmosferu, izgled, kontrole i na koncu uživanje u satima provedenim igrajući, ne tako davno takve misli i želje u većini slučajeva ostajale bi zbog nedostatka adekvatnih vještina ili neimanja resursa i tehnologije za samu provedbu izrade igre na kraju upravo to, misli i želje.

Međutim razvojem jednog moćnog i modernog alata, Unity-a, programeru se omogućava velika podrška u izradi vlastitog projekta jer softver pruža rješenja na mnoge temeljne probleme niže razine kojima pritom ušteđuje kreatoru mnogo vremena, ali i uvelike mu olakšava posao. Nadalje, vještine i znanja samog programera nisu neophodno napredna, već prosječnim shvaćanjem programiranja kreator može veoma brzo pohvatati sve koncepte i aspekte ovog intuitivnog softvera i na koncu dobiti željeno rješenje.

U ovom radu provest ću vas kroz izradu igre šaha u 2D okruženju. Započet ću opisom dva najvažnija alata koje sam koristio u izradi, Unity i Microsoft Visual Studio 2019, a zatim ću ukratko objasniti šah kao igru, njene figure te konačni cilj i završetak igre. Zatim ću detaljno objasniti korištene koncepte na kodu i proces implementacije završnog rješenja.

Za pisanje rada najviše sam se koristio online izvorima literature od kojih bi najviše izdvojio Unity dokumentaciju koja je davala odgovore na sva pitanja te se pokazala kao odličan vodič kroz shvaćanje čitavog programa i njegove logike. Prilikom same izrade igrice najviše sam se koristio „YouTube“ izvorima koji su kroz vlastite implementacije dali mi ideje te pomogle u razvoju načina razmišljanja i u kumulativnom procesu skriptiranja igre. Isti su navedeni u poglavlju 7.2. kao literatura korištena u programskom kodu.

Rad je podijeljen na šest dijelova: uvod (u kojem se govori o temi, navode najvažniji izvori te opisuje struktura rada), metode i tehnike (gdje se spominju alati i dočarava svojevrsni misaoni proces izrade), opis korištenih alata (jasno se definiraju uloge alata i njihove karakteristike), o šahu (progovara se o šahu kao igri i njenim najvažnijim aspektima), razvoj igre šaha (najvažniji dio rada koji će detaljno objasniti izradu igre), zaključak (u kojem se remizira čitav rad i izvodi zaključak) te na kraju rada popis literature, slika i korištenih resursa.

2. Metode i tehnike rada

Glavni alati upotrebljavani u implementaciji programskog rješenja ovog rada su „Unity“, softver za razvoj igara u 2D i 3D okruženju, i „Microsoft Visual Studio“, integrirano razvojno okruženje korišteno za uređivanje skripti napisanih C# programskim jezikom. Uz njih, korišteni su poneki online alati poput „ResizeImage.net“ uspomoć kojeg su figure i ploča adekvatno skalirani, pravilno izrezani te optimizirani za što bolji izgled igre. Za poboljšanje kvalitete figura pobrinuo se alat „Pinetools“ pomoću kojeg su slike izoštrene.

Ranije poznavanje programskog jezika C# uvelike je pomoglo prilikom samog skriptiranja funkcionalnog dijela, no nepoznavanje „Unity-a“ kao alata i njegovu logiku povezanosti objekata na sceni s njihovim pripadajućim komponentama predstavljao je početnu točku savladavanja zadatka ovog rada. Proučavao sam „Unity“ kroz „YouTube“ tutorijale koji su mi dali osnovno shvaćanje programa. Zatim sam prolazio primjere implementiranih 2D igara kako bih dobio viđenje koncepta implementacije igre u 2D okruženju. Za sve nepoznanice vezane uz terminologiju poslužio sam se „Unity“ dokumentacijom. Potom sam odlučio potražiti druge slične pokušaje implementacije same igre šaha kako bih formirao određeni plan provedbe i riješio poneke sumnje koje sam imao vezane za implementaciju logike ponekih pozicija poput šaha i šah mata te primjenu specifičnih poteza poput „en passanta“.

Prilikom samostalne provedbe rješenja, krenuo sam s najjednostavnijim točnije odabirom izgleda figura. Promijenio sam nekolicinu setova prije konačnog odabira. Kad sam konačno postavio početnu scenu i hijerarhiju elemenata, dao sam se na skriptiranje svih potrebnih aspekata igre. Krenuo sam od općih poput „Igra“, „Polje“ i „Figura“, ali s vremenom se svelo na konstantno skakanje s jedne na drugu, s općih skripti na specifične figure. Metode kojima sam u konačnici došao do završnog rješenja objašnjene su u petom poglavlju ovog rada.

Završetkom funkcionalnog dijela odnosno skriptiranja, igra je bila gotova te je preostalo samo napisati dokumentaciju. U samu izradu uloženo je dosta vremena i truda te sam zadovoljan ostvarivanjem željenog ishoda koji sam prilikom odabira ove teme zacrtao. Naučio sam mnogo toga kroz samostalan rad što je rezultiralo kvalitetnim rješenjem i praktičnim znanjem.

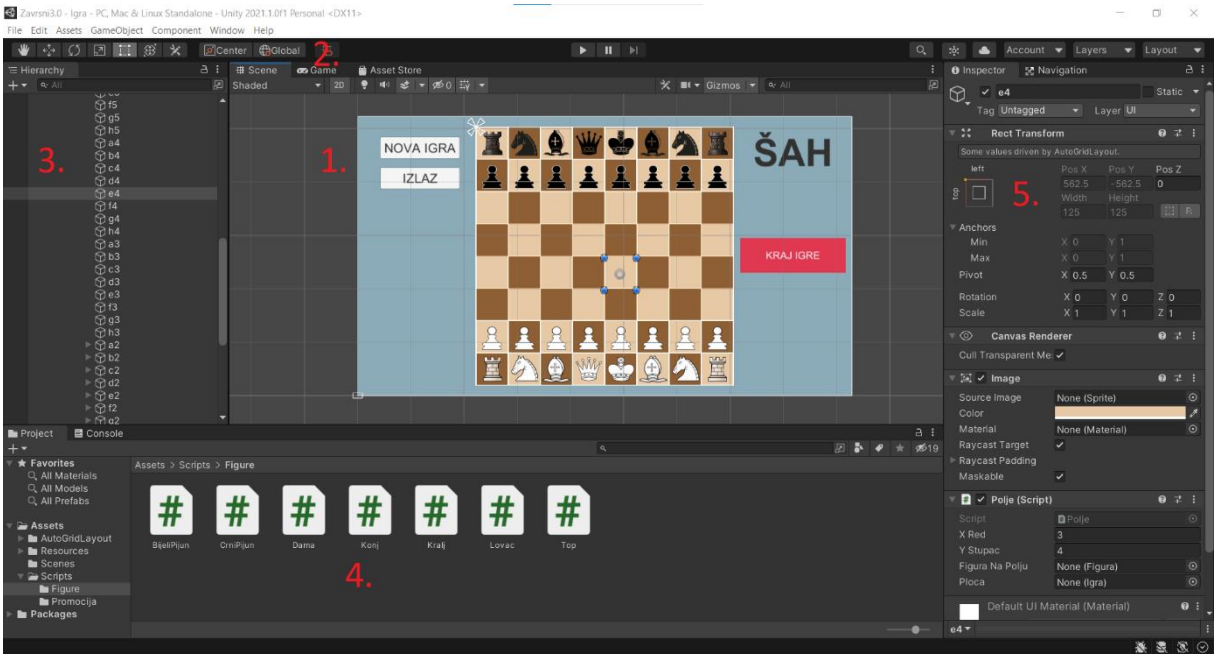
3. Opis korištenih alata

U prethodnom poglavlju dotakli smo se dva glavna alata za implementaciju programskog rješenja: „Unity“ i „Microsoft Visual Studio“, a u ovom poglavlju pobliže će se definirati svaki od njih te njihove najvažnije karakteristike.

3.1. Unity

Unity je softver za razvoj igara u 2D i 3D okruženju te je kreiran od strane tvrtke Unity Technologies 2005. godine. Kao „game engine“, Unity stavke koje igru čine funkcionalnom pruža već ugrađenima u program. Stvari poput fizike, 3D renderinga te collision detectiona razvojnom programeru omogućuju zdrave temelje za razvoj vlastite ideje i svog zamišljenog svijeta. Osim što je „game engine“, Unity je i integrirano razvojno okruženje što znači da sadrži sve alate na jednom mjestu. Također daje mogućnost kreatoru da jednostavno metodom „drag-and-drop“ uredi raspored elemenata na sceni te uređuje njihova svojstva ovisno o svojim željama.[1]

Sučelje Unity-a možemo podijeliti na pet sekcija:[2]



Slika 1: Prikaz sučelja Unity-a

1. Scenski prikaz: Mjesto gdje se kreira igra ili neki drugi 2D/3D projekt. Svi objekti igre postavljeni su i izmjenjivani na sceni.

2. Prikaz igre: Mjesto gdje se prikazuju konačni rezultati odnosno kako izgleda gotovo rješenje vaše implementacije.
3. Hijerarhija: Prozor koji prikazuje sve objekte igre postavljene na scenu. Ovisno o rasporedu objekata, tako se ti objekti „renderaju“ unutar igre pa je raspored veoma bitan. Uključuje sve vizualne i nevizualne objekte unutar igre.
4. Projekt: Prozor u kojem se nalaze mape koje sadržavaju objekte igre, skripte, teksture, modele, zvučne zapise itd. Također je moguće pristupiti gotovim resursima kupljenima u „Assets Store“ gdje postoje mnoge datoteke i implementacije koje mogu uvelike pomoći razvojnom programeru u izradi vlastite kreacije.
5. Inspektor: Panel koji prikazuje atribute i svojstva označenih objekata na sceni. Ovisno o odabiru, bit će navedeni odgovarajući atributi i komponente.

Na koncu ovog poglavlja bitno je izdvojiti i kvalitetno napisanu dokumentaciju na „Unity“ web stranici.[3] Ova detaljno razrađena dokumentacija veoma je dobro organizirana pa se izuzetno lako snalaziti i nastaviti proučavati sekcije koje su od važnosti za razvoj vlastitog projekta.

3.2. Microsoft Visual Studio 2019

Microsoft Visual Studio 2019 je integrirano razvojno okruženje (*eng. Integrated Development Environment*) koje pruža način kreiranja i uređivanja Unity skripti u C# programskom jeziku. Uz to nudi i brojne korisne alate koje pomažu u kompatibilnosti programskog koda sa samim Unityem. IntelliSense code-completion omogućava brzo dopunjavanje koda željenim Unity API porukama uključujući i njihove parametre. Debuggiranje koda znatno je pojednostavljeno, a integrirane su i sugestije „najboljih praksi“ i poboljšanja performansi. Nadalje, Visual Studio sadržava i CodeLens podršku koja odvaja kod omogućen od strane Unity-a od koda napisanog od strane razvojnog programera. Na koncu optimiziran je pregled svih skripti kako bi što više izgledao onom u Unityu. Pomoću Unity Project Explorera (UPE) dobiva se ljepši i pregledniji način hijerarhijskog prikaza skripti od recimo uobičajenog Solution Explorera.[4]

4. O šahu

Šah je jedna od najstarijih i najraširenijih igara na ploči. Ime je dobilo po perzijskoj riječi šah što znači vladar, kralj ili car. Igra se na šahovskoj ploči odnosno šahovnici koja se sastoji od osam redova numeriranih od 1 do 8 te osam stupaca označeni slovima od a do h tako da svako od 64 polja ima jedinstvenu oznaku stupca i retka (npr. e4 – peti stupac i četvrti redak). Početkom partije na jednoj strani nalazi se 16 bijelih, a na drugoj strani 16 crnih figura. Sa svake strane imamo po jednog kralja, jednu damu, dva topa, dva lovca, dva konja i osam pješaka.[5]

Igrači vuku naizmjenično po jedan potez, a prvi na potezu je bijeli. Cilj je igre matirati protivničkog kralja. Pješaci ili pijuni se kreću isključivo prema naprijed. Iz početne pozicije mogu se pomaknuti za dva ili jedno polje unaprijed, a nakon tog samo za jedno. Uzimanje pješakom vrši se dijagonalno prema naprijed, osim u slučaju kad protivnički pješak preskoči branjeno polje, tada pješak može uzeti protivničkog potezom „en passant“. Dolaskom do kraja ploče, pješak se mora promovirati u novu figuru, najčešće u damu.

Konj ili skakač jedna je od dvije „lake“ figure. Kreće se dva polja ravno potom lijevo ili desno, u obliku slova „L“. Ima mogućnost preskakanja drugih figura, bile one protivničke ili vlastite. U pravilu njegova vrijednost odgovara vrijednosti tri pješaka. Druga „laka“ figura je lovac ili laufer. Svaki igrač ima jednog lovca koji se kreće po crnim poljima i jednog koji se kreće po bijelim, a često zbog položaja ih nazivaju damin odnosno kraljev lovac. Kreće se izričito po poljima boje na kojem se nalazi na početnoj poziciji dijagonalno po šahovskoj ploči. Vrijednost lovca također odgovara tri pješaka.

Top ili kula je druga po jačini figura u šahu. Razlikujemo daminog i kraljevog topa. Kreće se uvijek prema ravno, okomitim i vodoravnim poljima. Vrijedi pet pješaka. Dama ili kraljica najjača je figura u šahu te uz topa pripada kategoriji „teških“ figura. Može se kretati u svim pravcima te čini svojevrsnu kombinaciju topa i lovca. Vrijednost dame jednaka je devet pješačkih jedinica.

Kralj je najvažnija i najdragocjenija figura u šahu. Cilj igre je zarobiti protivničkog kralja tako da je protivniku nemoguće pobjeći ili obraniti se. Ako kralja napada neka figura odnosno prijeti uzimanjem, tada se kaže da je kralj u šahu i igrač tu prijetnju mora otkloniti idućim potezom. Ako je to neizvedivo, tada je kralj matiran i igra je završena.[6]

Postoje tri načina obrane od mata. Prvi je da se napadačka figura uzme, drugi je da se postavi vlastita figura na liniju napada tako da je pritom napadnuta ta figura umjesto kralja i treći način je uzmicanjem kralja na polje koje nije pod napadom.[7]

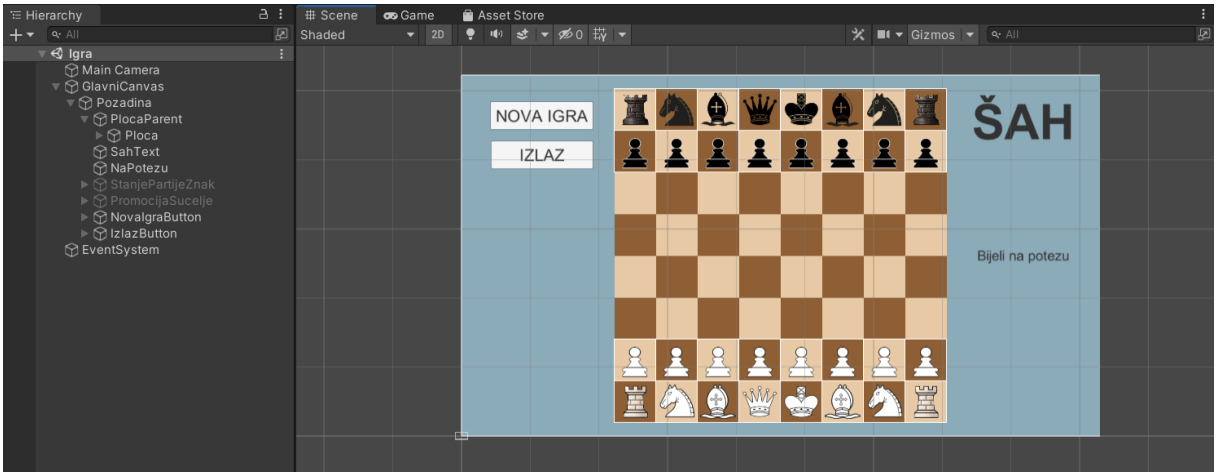
5. Razvoj igre šaha

U sljedećem poglavlju detaljno će se razraditi svi aspekti vezani za razvoj i implementaciju igre šaha što ujedno predstavlja i najvažniji dio rada. Igra je implementirana korištenjem dosad opisanih alata. Poglavlje je podijeljeno na četiri dijela:

- „Scena i UI“ ponajviše definira sredstva korištena unutar samog Unity-a vezana za sučelje i izgled same igre
- „Ploča“ detaljnije predočava izradu vizualnog dijela igre te povezanost između polja i figura
- „Upravljanje i logika igre“ tumači razne koncepte korištene za implementaciju logike poput stanja partije pomoću kojeg se definira je li partija završena ili ne
- „Figure i mehanika“ fokusira se na figure i njihovo međudjelovanje na šahovskim poljima

5.1. Scena i UI

Scene predstavljaju mjesto unutar kojeg se radi sa sadržajem i elementima Unity okruženja.[8] Scena „Igra“ jedina je scena u čitavoj implementaciji igre šaha i sastoji se od glavne kamere, canvasa „GlavniCanvas“ te „EventSystem“ koji za ulogu ima obradu i rukovanje događajima.[9] Najvažnija od te tri stavke, „GlavniCanvas“, predstavlja Game Object unutar kojeg se nalaze svi elementi korisničkog sučelja (*eng. User Interface*) te pritom svi elementi na sučelju moraju biti „djeca“ (*eng. Children)* zadanog Canvasa.[10]



Slika 2: Hijerarhija elemenata i scena

Child element „Glavnog Canvasa“ predstavlja „Pozadina“ koja je Game Object tipa „Slika“. „Slike“ (*eng. Image*) za ulogu ima prikazivanje Spriteova [11] koji predstavljaju 2D grafičke objekte korištene u samoj igri ili u njenom korisničkom sučelju.[12] „Pozadina“ je svijetloplave boje i roditelj element (*eng. Parent element*) je svim ostalim elementima prikazanim na sučelju. Unity koristi koncept roditelj-dijete kako bi se grupirali Game Objecti upotrebljavani u sceni. Hijerarhija elemenata vrlo je važna jer gornji elementi prvi se učitavaju te se nalaze iza odnosno u pozadini, dok se donji elementi učitavaju kasnije pa se iz tog razloga nalaze ispred tj. u prvom planu.[13]

Pod elemente na „Pozadini“ ulaze dugmi „NovaIgraButton“ i „IzlazButton“, „slike“ pod nazivima „PlocaParent“, „StanjePartijeZnak“ i „PromocijaSucelje“ te tekstovi „SahText“ i „NaPotezu“. „NovaIgraButton“ za ulogu ima resetiranje kompletne scene, a „IzlazButton“ služi za izlaz iz aplikacije.

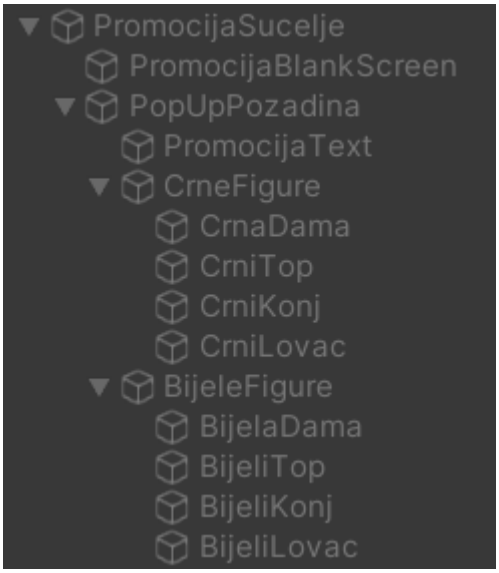
```
public void NovaIgraButtonHandler()
{
UnityEngine.SceneManagement.SceneManager.LoadScene(UnityEngine.SceneManagem
ent.SceneManager.GetActiveScene().buildIndex);
}

public void IzlazButtonHandler()
{
        Application.Quit();
}
```

„StanjePartijeZnak“ ima za ulogu prikazati ukoliko je igrač u šahu ili ukoliko je na ploči šah mat i igra je završena. Sam proces kad se taj „znak“ prikazuje opisat će se u dijelu rada vezanim za stanja partije (šah, pat, šah mat).

Slika 3: Prikaz znaka „Šah mat“ i proglašenje pobjednika

„PromocijaSucelje“ prikazuje se samo prilikom „posebnog“ poteza promocije prilikom koje pješak dolaskom do kraja ploče, u slučaju crnog to biva prvi red, a u slučaju bijelog osmi red polja, igraču se nudi opcija odabira figure u koju želi tog pješaka promovirati. Igrač može promovirati u bilo koju figuru po želji, ali ni u kom slučaju figura ne smije ostati pješak.

Slika 4: Prikaz sučelja promocije



Slika 5: Hijerarhija sučelja promocije

„PromocijaSucelje“ za dijete ima dva elementa „PromocijaBlankScreen“ i „PopUpPozadina“. „PromocijaBlankScreen“ predstavlja transparentni Image objekt koji se nalazi preko cijelog ekrana u pozadini skočnog prozora sučelja i za svrhu ima onemogućiti igrača da klikne izvan okvira sučelja te nekim slučajem nastavi partiju bez da odabere neku od ponuđenih opcija. Unutar „PopUpPozadine“ nalaze se „PromocijaText“ koji glasi „Odaberite figuru“ te same figure, bijele i crne, ovisno o tome koji igrač je na potezu.

„SahText“ predstavlja naslov u gornjem desnom kutu i sam je po sebi očigledan, međutim tekstualni element „NaPotezu“ koji za ulogu ima prikaz koji je igrač, bijeli ili crni, trenutno na potezu ima za sebe vezanu funkcionalnost izmjenjivanja sadržaja svaki potez.

```
bijeliNaPotezu = (bijeliNaPotezu == true) ? false : true;
if (bijeliNaPotezu) NaPotezuText.GetComponent<Text>().text = "Bijeli na potezu";
if (!bijeliNaPotezu) NaPotezuText.GetComponent<Text>().text = "Crni na potezu";
```

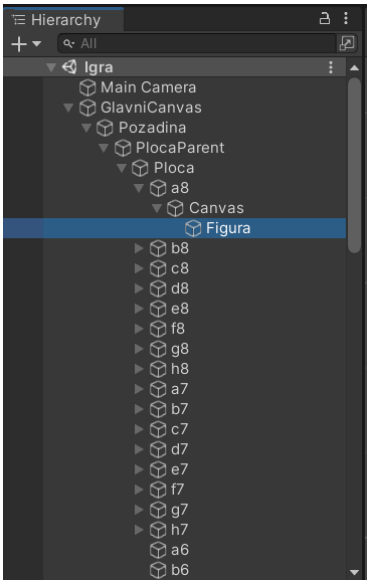
Slika 6: Prikaz tekstualnih natpisa

Varijabla tipa bool „bijeliNaPotezu“ svakim potezom mijenja svoju vrijednost. Ukoliko je bijeli na potezu, poprima vrijednost *true*, a ukoliko nije, vrijednost *false*, što podrazumijeva da je trenutno na potezu igrač s crnim figurama. Sukladno tome dohvaćamo tekstualnu komponentu „NaPotezu“ te joj alociramo adekvatan string.

Podelement „Pozadine“ pod nazivom „PlocaParent“ detaljno će se obraditi u idućem potpoglavlju „Ploca“.

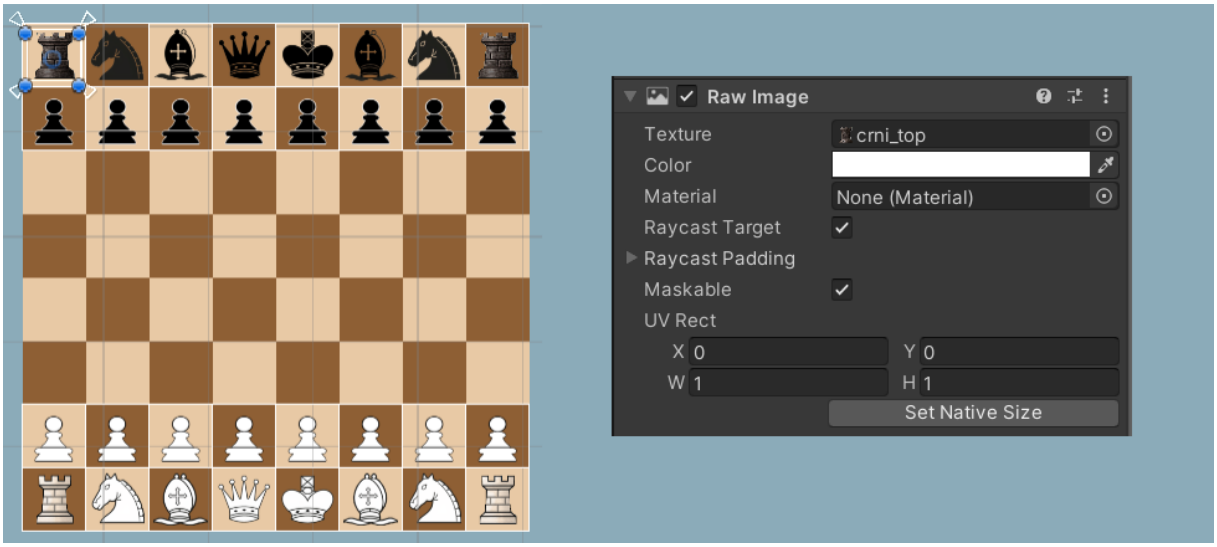
5.2. Ploča

Igraća ploča kreirana je tako da je napravljena prazna „slika“ pod nazivom „PlocaParent“ u obliku kvadrata veličine zamišljene ploče, međutim funkcionalno gledajući važniju ulogu sadrži podelement „Ploca“ koji posjeduje komponentnu „Auto Grid Layout Group“.[14] Pomoću te javno dostupne skripte za Unity UI, možemo skalirati ploču na 8 redova i 8 stupaca poput te automatski dobiti adekvatnu veličinu polja u odnosu na element roditelj „PlocaParent“.



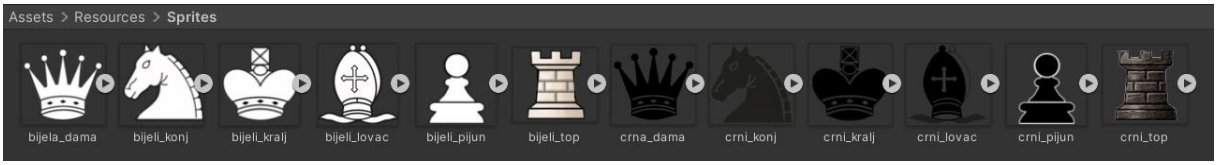
Slika 7: Hijerarhija elementa „PlocaParent“

Zatim elementu „Ploca“ dodajemo 64 elementa koji predstavljaju polja ploče, a nazvani su prema pozicijama šahovske notacije. Slova predstavljaju stupac, a brojevi redove. Primjerice, kod polja „a8“, „a“ označava prvi stupac slijeva, a „8“ posljednji redak odozdo odnosno najgornji redak. „Auto Grid Layout Group“ tih 64 polja grupira u mrežu veličine 8x8. Kako bi se postigla tema drvene ploče, za crna polja korištena je nijansa smeđe boje (#8E5F34), a za bijela nijansa bež (*eng. beige*) to jest blijedo pješčane boje (#E8C9A5). Bitno je i napomenuti da na početku igre 32 polja sadrže i figure. Kao što „Slika 6“ prikazuje, na polju „a8“ nalazi se figura na canvasu čiji okviri predstavljaju granice polja koje tu figuru sadrži. Pojedincima kojima je šahovska notacija strana će biti jasnije ukoliko uvide da figura na polju „a8“ predstavlja početnu poziciju crnog topa s damine strane. Također, možete uočiti da sva polja od „a8“ do „h7“ sadrže „djecu“ te da ta „djeca“ predstavlja čitavu kolekciju crnih figura, dok primjerice polje „a6“ ne sadrži nijedan podelement jer je s početkom partije to je polje prazno.



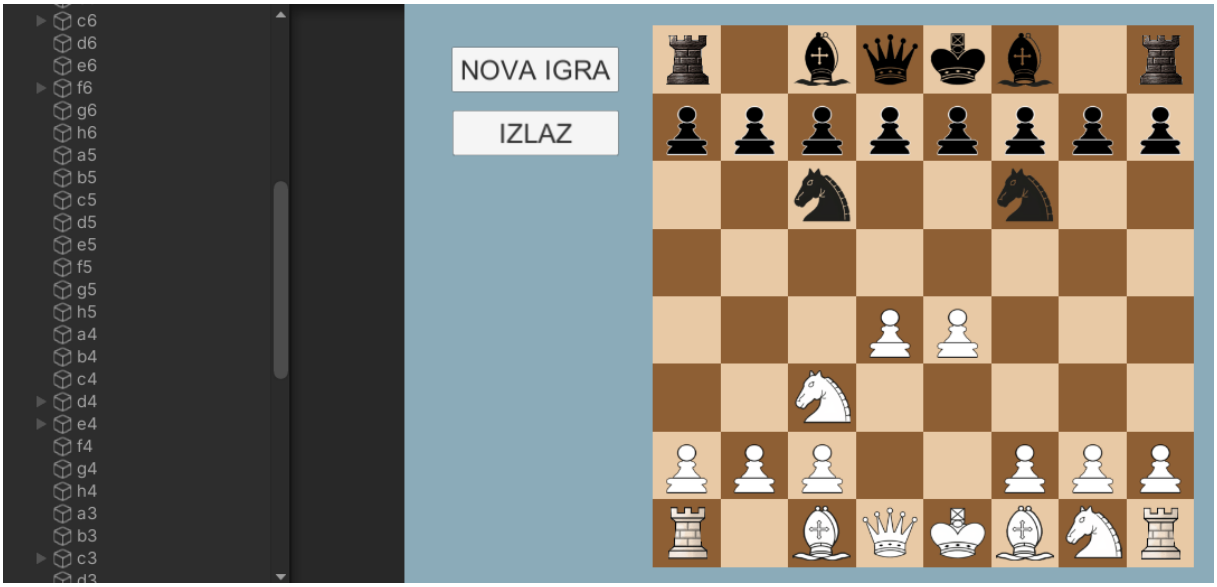
Slika 8: Crni top na polju a8

Kao što je prikazano na „Slici 7“, figura je element tipa „Raw Image“[15] koji može poprimiti bilo koji format 2D teksture, u ovom slučaju to su .png datoteke transparentne pozadine. Figure su izabrane pojedinačno pa u ovoj kombinaciji predstavljaju unikatan set. Korišteni su online alati za uređivanje[16] i izoštravanje[17] slika kako bi figure bile najbolje moguće kvalitete te su iste skalirane da budu polju pripadajuće veličine.



Slika 9: Slike figura

Prilikom pomicanja figura po ploči tokom igre, mijenja se i hijerarhija elemenata. Implementiran je koncept na temelju kojeg svako polje sadrži sliku figure koja prilikom njenog pomicanja se obriše iz izvornog i postavi u novo, ciljano polje. Primjerice pomicanjem bijelog konja s polja „b1“ na „c3“, polje „b1“ postaje prazno, a polje „c3“ koje dosad nije imalo „djece“ poprima za podelement canvas koji sadrži figuru iz polja „b1“.



Slika 10: Prikaz hijerarhije polja tijekom igre

Na primjeru vidimo kako polja „c6“ i „f6“ sadrže za „djecu“ crne konje, „e4“ i „d4“ su okupirali bijeli pijuni, a na polju „c3“ nalazi se bijeli konj“. Ostala polja prikazana na hijerarhiji su ostala prazna. Sam koncept pomicanja figura, njihovu logiku i način na koji sama igra funkcionira, obrađena je u iduća dva potpoglavlja „Upravljanje i logika igre“ te „Figure i mehanika“.

5.3. Upravljanje i logika igre

Temelj šaha predstavljaju polja i figure odnosno međudjelovanje figura i njihov utjecaj nad poljima ploče. Iako se šah naizgled čini igrom od koje se zahtjeva samo implementiranje mehaničke strane odnosno gdje se figura smije pomaknuti, a gdje ne smije, iznad te sfere postoji jedan upravljački aspekt koji sve te funkcionalne dijelove spaja u jednu cjelovitu sliku i čija uloga je konstantno motriti događanja na ploči te intervenirati ukoliko situacija zahtjeva šire tumačenje od onog implementiranog kroz sama polja i same figure.

Cjelovitu logičku stranu ploče pokrivamo skriptom „Igra“ koja ima razne zadaće te je najveća skripta u cijelom programu. Započinje deklaracijom enumeracije „StanjePartije“ koja će se konstantno ažurirati svakim potezom.

```
public enum StanjePartije
{
    Normalno,
    SahCrnom,
    SahBijelom,
    SahMatCrnomBijeliPobjednik,
    SahMatBijelomCrniPobjednik,
    Pat
}
```

„StanjePartije“ javna je enumeracija koja u sebi sadrži šest različitih mogućih situacija na ploči. Prvo i najčešće stanje je stanje „Normalno“ koje nagovještava da je igra u tijeku te da igrač koji je trenutno na potezu raspolaže svim legalnim kretnjama svih svojih figura u danoj poziciji. Iduća dva stanja „SahCrnom“ i „SahBijelom“ ograničuju igrača jer je u zadanoj poziciji šah igraču te je isti primoran pomaknuti kralja na njemu sigurno polje ili blokirati šah stavljanjem neke druge figure na putanju pod udarom od protivničke figure. Stanja „SahMatCrnomBijeliPobjednik“, „SahMatBijelomCrniPobjednik“ i „Pat“ označavaju da je igra završena to jest da je rezultat pobjeda bijelog, pobjeda crnog ili remi zbog nedovoljnog materijala ili nedostatka legalnih poteza jednog od igrača.

```
public StanjePartije StanjePartije = StanjePartije.Normalno;
```

Početkom partije, stanje partije se inicijalizira na „Normalno“ te se kasnije pozivanim metodama mijenja ovisno o poziciji na ploči.

```
public List<ObicanIliPosebanPotez> bijeliNapadackiPotezi = new
List<ObicanIliPosebanPotez>();
public List<ObicanIliPosebanPotez> crniNapadackiPotezi = new
List<ObicanIliPosebanPotez>();


public List<Figura> bijeleFigureNapadajuKralja = new List<Figura>();
public List<Figura> crneFigureNapadajuKralja = new List<Figura>();


public Polje BijeliKraljPozicija = null;
public Polje CrniKraljPozicija = null;


GameObject NaPotezuText;
GameObject StanjePartijeZnak;
GameObject StanjePartijeText;



public bool bijeliNaPotezu;
public Polje[,] SvaPolja = new Polje[8, 8];
public PrikazSuceljaPromocije prikazSuceljaPromocije;
public Polje TrenutnoOznacenoPolje;
public Figura TrenutnoOznacenaFigura;
public bool DopustiOznacavanjePolja = false;
public Figura PosljednjaPomaknutaFigura = null;
```

Skripta sadržava veliki broj atributa koje uglavnom sačinjavaju objekti i liste objekata. Prve dvije liste koje su inicijalizirane nazivaju se „bijeliNapadackiPotezi“ i „crniNapadackiPotezi“, a sadrže objekte klase „ObicanIliPosebanPotez“ odnosno poteze jedne i druge strane. Upravo ti potezi imat će ključnu ulogu u utvrđivanju obrane od šaha i na koncu ograničavanju kretnje kralja preko nedopuštenih polja.

Druge dvije inicijalizirane liste su „bijeleFigureNapadajuKralja“ i „crneFigureNapadajuKralja“ koje sadržavaju objekte klase „Figura“. Njihova primarna uloga je definiranje figure koja napada kralja kako bi se protiv te figure organizirala „obrana“. Recimo ako top napada kralja po ravnoj liniji, znamo da taj šah možemo blokirati postavljanjem figure na tu liniju ili pomicanjem kralja. Međutim, postavlja se pitanje zašto je to tip podataka lista? Odgovor leži u tome da postoje pozicije kad dvije ili više figure napadaju kralja istovremeno. U takvim situacijama možemo zanemariti činjenicu tko je napadač jer ne postoji obrana odnosno potez kojim bi se moglo blokirati dva napad iz dva različita smjera istovremeno. Tada znamo da je jedini legalni bijeg iz šaha pomicanje kralja, a ukoliko kralj nema legalnih poteza, tad je riječ o kraju partije.

Iduća dva atributa predstavljaju dva objekta tipa „Polje“ pod imenima „BijeliKraljPozicija“ i „CrniKraljPozicija“ čija uloga je konstantno praćenje polja na kojima se nalaze kralji. Čitav koncept implementacije ove igre do neke mjere bazira se na poljima koja ulaze u sferu poteza ovisno o pozicijama određenih figura na određenim poljima. Premda je kralj daleko najvažnija figura na ploči, konstantno je važno pratiti njegovu poziciju, a polje na kojem se on nalazi ne smije nikad biti na listi legalnih poteza drugih figura i, pojednostavljeno gledano, to polje je polje oko kojeg se cijela igra vrti.

Atributi tipa „GameObject“ poput „NaPotezuText“, „StanjePartijeZnak“ i „StanjePartijeText“ inicijalizirani su jer ovisno o stanju na ploči mijenja se njihov sadržaj ili vidljivost. Ovisno o igraču koji je na potezu mijenja se tekst koji navodi boju koja je na redu za odigrati potez, a ukoliko je stanje partije različito od „Normalno“ prikazuje se „StanjePartijeZnak“ s tekstom koji varira ovisno o stanju o kojem je riječ.

Atribut „bijeliNaPotezu“ tipa bool za zadaću ima odrediti koja je boja trenutno na potezu. Ukoliko mu je alocirana vrijednost *true*, tada znamo da je riječ o bijelom igraču, a ako je *false*, radi se o crnom. Atribut „SvaPolja“ dvodimenzionalno je polje koje se sastoji od 64 polja te predstavlja čitavu ploču. Prvi indeks predstavlja broj redova kojih je osam te je njegov raspon od 0 do 7 kao i kod drugog indeksa koji predstavlja broj stupaca kojih je također osam. Objekt „prikazSuceljaPromocije“ predstavlja instancu klase „PrikazSuceljaPromocije“ koju ćemo koristiti u jednoj od metoda, a predstavlja pristup određenim atributima i metodama potrebnih za provedbu promocije pijuna. Iduća dva atributa pod nazivom „TrenutnoOznacenoPolje“ i „TrenutnoOznacenaFigura“ pomažu nam kako bismo konstantno bili svjesni koja je figura na kliknutom polju odnosno koje je polje podno kliknute figure. Promjenom pozicije figure mijenja se i polje koje je podno nje, a pomicanjem figure polje koje je dotad tu figuru sadržavalo postaje prazno. Za potrebe funkcionalnosti igre treba postojati konstantna provjera događanja na svim poljima i poznavanje pozicija svih figura.

Atribut „DopustiOznacavanjePolja“ tip je podataka *bool* i za ulogu ima onesposobiti označavanje polja koje sadržavaju protivničke figure kad je igrač na potezu. Uz to njegova vrijednost varira ovisno o tome i sadrži li figura na polju koje želimo kliknuti legalne poteze ili ne. Posljednji atribut pod nazivom „PosljednjaPomaknutaFIgura“ objekt je klase „Figura“ te za ulogu ima pamtiti onu figuru koja je posljednja pomaknuta. Važnost ovog atributa temelji se na pravilu „en passant“ pomoću kojeg primjerice bijeli pijun koji se nalazi na petom redu može uzeti protivničkog pijuna koji je pomaknut sa sedmog reda na peti te se nalazi pored spomenutog bijelog pijuna. Jedan od uvjeta za izvođenje posebnog poteza „en passant“ jest da je taj protivnički pijun posljednja pomaknuta figura jer ukoliko nije, „en passant“ nije legalan potez.

```
void Start()...

private void PrikaziStanjePartije()...

private StanjePartije ProvjeriStanjePartije()...

private bool MoguceBlokiratiSahFigurom(Figura blokFigura, List<Figura>
figureKojeNapadajuKralja, Figura napadnutiKralj)...

private bool ListaSadrziPolje(List<ObicanIliPosebanPotez> Lista, Polje
polje)...

private bool ListaSadrziPosebanPotez(List<ObicanIliPosebanPotez> Lista,
Polje polje)...

private void UkloniFiguruIzPolja(Polje ParentPolje)...

private void PomakniFiguruNaPolje(Figura Figura, Polje Polje)...

private void KrajPoteza()...

private void ObicnoKretanjeFigura()...

public void ProvjeriOznaceno()...

public void NovaIgraButtonHandler()...

public void IzlazButtonHandler()...
```

Skripta „Igra“ započinje metodom „Start()“ koja se pokreće pri pokretanju aplikacije odnosno odmah po početku partije. Atribut „bijeliNaPotezu“ inicijalizira se na *true* budući da bijeli uvijek igra prvi. Zatim se odgovarajuća polja alociraju na odgovarajuće vrijednosti indeksa dvodimenzionalnog polja „SvaPolja“ kako bismo ga mogli kasnije koristi u ovoj i u ostalim skriptama za prolazak petljom kroz sva polja na ploči.

Privatna metoda „PrikaziStanjePartije()“ ima za ulogu dohvaćanje slikovne i tekstualne komponente stanja partije iz Unitya te omogućavanje prikaza istog. Budući da je početkom partije znak koji prikazuje stanje partije skriven te se prikazuje samo ako je stanje različito od „Normalno“, pozivom ove metode taj znak postaje vidljiv.

Privatna metoda „ProvjeriStanjePartije()“ koja vraća vrijednost enumeracije „StanjePartije“ za ulogu ima ispitati kakvo je trenutno stanje na partije. Svakim pozivom ove metode inicijaliziramo dvije prazne liste „bijeliPotezi“ i „crniPotezi“ koje će sadržavati u sebi sve poteze jedne i druge strane. Liste „bijeliNapadackiPotezi“, „crniNapadackiPotezi“, „bijeleFigureNapadajuKralja“, „crneFigureNapadajuKralja“ ispraznimo, a atribute „BijeliKraljPozicija“ i „CrniKraljPozicija“ alociramo na *null*. Zatim petljom prolazimo kroz „SvaPolja“ te određujemo poziciju bijelog kralja, sve poteze u „bijeliPotezi“ te sve poteze u „bijeleFigureNapadajuKralja“ ukoliko isti postoje i na koncu punimo listu „bijeliNapadackiPotezi“. Nakon toga isto činimo i za crne figure.

Potom inicijaliziramo varijablu „AzurirajStanjePartije“ tipa „StanjePartije“ koju ćemo na koncu metode „vratiti“ odnosno nakon prolaska svih zadanih uvjeta. Prvi uvjet provjerava sadrži li lista „bijeliNapadackiPotezi“ polje koje ima vrijednost istu kao atribut „CrniKraljPozicija“. Ukoliko je to slučaj, „AzurirajStanjePartije“ poprima vrijednost „SahCrnom“ te se poziva metoda „PrikaziStanjePartije()“ koja prikazuje znak s natpisom koji navodi da je crni kralj u šahu. Ista procedura je i u idućem uvjetu samo se provjeravaju „crniNapadackiPotezi“ i polje „BijeliKraljPozicija“. Uvjet koji provjerava je li na ploči pat prvo provjerava koja je stanja na potezu te ukoliko je to bijeli, ispituje je li lista „bijeliPotezi“ prazna. Ukoliko bijeli, koji je na potezu, nema poteza, igra završava patom. Isto vrijedi i za slučaj da je crni na potezu te ako lista „crniPotezi“ ne sadrži nijedan potez. Ako nijedan od navedenih uvjeta nije zadovoljen, „AzurirajStanjePartije“ alociramo na „Normalno“ te znak, ukoliko je bio vidljiv, postavimo ponovno nevidljivim.

Zatim kreće ispitivanje je li na ploči šah mat za slučajeve kad je na ploči šah. Ako je šah bijelom, odmah „pretpostavimo“ da je na ploči šah mat odnosno postavimo varijablu „AzurirajStanjePartije“ na „SahMatBijelomCrniPobjednik“. Zatim prolazimo petljom kroz sva polja te pokušavamo dokazati suprotno tako da pozivom metode „MoguceBlokiratiSahFigurom(Figura blokFigura, List<Figura> figureKojeNapadajuKralja, Figura napadnutiKralj)“ ispitamo može li se taj šah blokirati ili, iako nije u nazivu metode, izbjeći ako kralj ima legalne poteze. Ukoliko je pronađen barem jedan jedini legalni potez, bilo od blokirajuće figure ili samog kralja koji taj šah sprječava, „AzurirajStanjePartije“ alocira se natrag na „SahBijelom“. Ako nismo bili uspješni u „dokazivanju suprotnog“, „AzurirajStanjePartije“ ostaje „SahMatBijelomCrniPobjednik“ te se prikazuje znak koji deklarira kraj partije i proglašava pobjednika, u ovom slučaju crnog. Idućim uvjetom istom metodom provjeravamo i obrnuti slučaj odnosno je li samo šah ili šah mat crnom igraču. Na kraju metode vraćamo stanje koje je u varijabli „AzurirajStanjePartije“.

Najkompleksnija i najdulja metoda ove skripte je već spomenuta privatna metoda „MoguceBlokiratiSahFigurom(Figura blokFigura, List<Figura> figureKojeNapadajuKralja,

Figura napadnutiKralj)“ koja vraća tip podataka bool. Sastoji se od tri argumenata, prva je figura „blokFigura“ kojom prosljeđujemo figuru za koju želimo provjeriti ima li mogućnost blokiranja šaha ili ne, zatim lista „figureKojeNapadajuKralja“ koja sadrži sve figure koje daju šah kralju te na koncu „napadnutiKralj“ pomoću koje imamo mogućnost pristupa legalnim potezima napadnutog kralja kako bismo ustanovili da, ukoliko je nemoguće blokirati šah figurom, moguće je pomaknuti kralja na sigurno polje.

Metoda započinje kreiranjem bool varijable „mozeBlokirati“ koju alociramo na *false*. Potom kreiramo objekt tipa „Figura“ pod nazivom „figuraKojaNapadaKralja“ te joj alociramo prvi član liste „figureKojeNapadajuKralja“. Provjere će se izvršavati ukoliko je na listi samo jedna figura, ukoliko je više, odmah očistimo sve legalne poteze ispitivane „blokFigure“ jer je nemoguće blokirati jednom figurom šah koji dolazi iz dva ili više smjerova. Ukoliko je kralj napadnut od strane jedne figure, inicijaliziramo listu „blokFiguraLegalniPotezi“ kojoj alociramo legalne poteze te ispitivane „blokFigure“. Lista „blokFiguraLegalniPotezi“ za ulogu ima sadržavati samo one poteze koji blokiraju šah, a do njih se dolazi brisanjem svih ostalih poteza osim onih koji se poklapaju s legalnim potezima objekta „figuraKojaNapadaKralja“.

Kako bismo znali koje poteze tražimo, prvo moramo proći kroz uvjete koji redom ispituju je li „figuraKojaNapadaKralja“ top, lovac, dama, konj, bijeli ili crni pijun. Za različite figure vrijede različite putanje napada. U slučaju topa, promatramo horizontalna i vertikalna polja u odnosu na topa odnosno sva četiri smjera u koja se top može kretati. Zatim za svaki pojedinačni smjer prolazeći kroz listu „blokFiguraLegalniPotezi“ uspoređujemo ih sa legalnim potezima topa te ukoliko nisu isti i ne poklapaju se na putanji napada prema kralju, te poteze brišemo s liste i ostavljamo samo one koje su „na putu“ prema napadnutom kralju. S lovcem je ista procedura samo se promatraju sve četiri potencijalne dijagonale koje su u domeni kretanja lovca, a dama koja predstavlja spoj topa i lovca sadržava sve uvjete kao i kod te dvije figure. Konj nema putanju napada prema kralju budući da ima specifičan način kretanja te se njegov napad ne može blokirati od strane figure. No svejedno postoji uvjet koji provjerava sadrži li lista „blokFiguraLegalniPotezi“ potez kojim može pojesti figuru konja jer onda definitivno na ploči nije šah mat već je forsirano uzimanje konja. Kod pijuna je također specifična situacija. Nemoguće je blokirati napad pijuna tako da je jedini način za riješiti se te prijetnje tako da se figura pojede. Međutim razlikujemo dva uvjeta, jedan u kojem provjeravamo je li uzimanje pijuna legalan potez i drugi koji, ukoliko je „blokFigura“ protivnički pješak, ispituje radi li se možda o „en passantu“ koji je također legitiman način za otklanjanje pijuna koji daje šah kralju.

Nakon što su ispitani svi od tih uvjeta, na kraju se provjerava jesu li liste legalnih poteza kod „blokFigura“ i „napadnutiKralj“ prazne ili nisu. Ako nisu, bool koji smo inicijalizirali

na početku metode postavljamo na *true* te metoda vraća true, a ako jesu, vraća se izvorno inicijalizirani *false*.

Dvije kratke privatne metode „ListaSadrziPolje(List<ObicanIliPosebanPotez> Lista, Polje polje)“ i „ListaSadrziPosebanPotez(List<ObicanIliPosebanPotez> Lista, Polje polje)“ obe vraćaju bool vrijednost te imaju veoma jednostavne uloge. Prva metoda ispituje nalazi li se određeno polje u listi legalnih poteza određene figure, a druga metoda provjerava sadrži li polje koje se nalazi na listi legalnih poteza true vrijednost kod atributa „PosebanPotez“. Obe metode koristimo samo za provjeru i kreiranje određenih uvjeta kod većih i kompleksnijih metoda.

Privatna metoda „UkloniFIguruIzPolja(Polje ParentPolje)“ zaprima za argument polje koje želimo očistiti. Također figuru koja se nalazi na polju postavljamo na *null* vrijednost. Metoda „PomakniFiguruNaPolje(„Figura Figura, Polje Polje)„ također je privatna te za ulogu ima očistiti prethodno polje na kojem je bila figura te pozivom metoda „PostaviKoordinateFigura()“ iz skripte „Polje“ i „PostaviPoljeOdFigure()“ iz skripte „Figura“, postaviti nove vrijednosti figure na njenom novom polju. Na kraju metode se i atribut „odigraniPotezi“ dane figure inkrementira. Metoda koja se poziva odigravanjem poteza pod nazivom „KrajPoteza()“ privatna je metoda koja započinje tako da se atribut „DopustiOznacavanjePolja“ postavi na *false*, a atributi „TrenutnoOznacenaFigura“ i „TrenutnoOznacenoPolje“ na *null*. Zatim atribut „bijeliNaPotezu“ promjeni svoju vrijednost ovisno o njegovoj dotadašnjoj vrijednosti. Ukoliko je bio *true*, mijenja se u *false* i obrnuto. Također se ovisno o vrijednosti tog atributa i mijenja tekst koji navodi igrača čiji je potez. Na koncu metode poziva se metoda „ProvjeriStanjePartije()“ kako bi bili u toku svakim potezom kakva je situacija na ploči.

Ove tri metode čine okosnicu privatne metode „ObicnoKretanjeFigura()“. Metoda započinje pozivom metode „UkloniFiguruIzPolja“ kojoj se prosljeđuje atribut „TrenutnoOznacenoPolje“. Potom se poziva metoda „PomakniFiguruNaPolje“ koja za argumente poprima atribute „TrenutnoOznacenaFigrua“ i „TrenutnoOznacenoPolje“. Na kraju se zove metoda „KrajPoteza()“. „ObicnoKretanjeFigura()“ metoda kreirana je zbog čestog pozivanja u idućoj metodi koja je funkcionalno zahtjevnija.

Javna metoda „ProvjeriOznaceno()“ služi kao upravljač običnih i posebnih poteza odnosno provjera je li potez figuri svojstven zbog načina na koji se ta figura kreće ili je u trenutnoj poziciji moguće odigrati „poseban“ potez poput rokade, promocije i „en passanta“. Metoda započinje postavljanjem atributa „DopustiOznacavanjePolja“ na *true* te se zatim provjerava vraća li metoda „ListaSadrziPosebanPotez(List<ObicanIliPosebanPotez> Lista, Polje polje)“ *false* ukoliko joj se proslijede za argumente legalni potezi

„TrenutnoOznaceneFigure“ i polje „TrenutnoOznacenoPolje“. Tim uvjetom odmah možemo ustanoviti da se radi o običnom potezu i pozvati metodu „ObicnoKretanjeFigura()“. Međutim ako to nije slučaj, provjerava se pet uvjeta. Prvi uvjet ispituje je li trenutno označena figura bijeli pješak te nalazi li se „TrenutnoOznacenoPolje“ na šestom redu. Budući da otprije znamo da se ne radi o običnom već posebnom potezu, zaključujemo da se radi o „en passantu“. Prvo uklanjamo figuru iz trenutnog polja metodom „UkloniFiguruIzPolja“ te ju pomičemo na novo polje metodom „PomakniFiguruNaPolje“. Na koncu se poziva metoda „KrajPoteza()“. Ista procedura implementirana je i za crnog pješaka ukoliko je isti na trećem redu ploče.

Zatim slijede dvije provjere također za bijelog i crnog pješaka, ovog puta u slučaju promocije. Poziva se metoda „ObicnoKretanjeFigure()“ jer zapravo ne događa se nikakva posebna kretnja osim zamjene jedne figure drugom. To izvodimo pozivanjem metode „UpravljacBijelihFigura“ ili „UpravljacCrnihFigura“ ovisno o boji koju ispitujemo koja se nalazi u objektu „prikazSuceljaPromocije“. Sama implementacija promocije obradit će se u poglavlju 5.4.2.1.

Posljednja provjera proizlazi iz uvjeta ako je „TrenutnoOznacenaFigura“ kralj. Kralj ima samo jedan poseban potez pa znamo da se radi o rokadi. Metodom „PomakniFiguruNaPolje“ odmah pomičemo kralja na označeno polje, međutim potrebno je pomaknuti i topa na njemu pripadajuće polje. Ponovno koristeći metodu „PomakniFiguruNaPolje“, ako je označeno polje treći stupac slijeva znamo da se radi o dugačkoj rokadi te se pomiče top iz prvog u četvrti stupac. Ako to nije slučaj, radi se o kratkoj rokadi te se top pomiče iz osmog u šesti stupac ploče. Na kraju se poziva metoda „KrajPoteza()“.

S metodama „NovaIgraButtonHandler()“ i „IzlazButtonHandler()“ već smo se upoznali u poglavlju 5.1.

5.4. Figure i mehanika

5.4.1. Polje

Prije nego što započnemo opisivanjem pojedinačnih figura, njihovo kretanje i međudjelovanje, krenimo od osnovnog elementa šahovske igre pod nazivom polje. Skripta „Polje“ vezana je za svaki pojedinačni element definiran u prethodnom poglavlju te predstavlja veoma bitan faktor kod legalnosti poteza određenih figura.

```
public class Polje : MonoBehaviour, IPointerDownHandler
{
     public int xRed;
     public int yStupac;
     public Figura FiguraNaPolju = null;
     public Igra Ploca;

     private void Start()...

     private bool ListaSadrziPolje(List<ObicanIliPosebanPotez> Lista,
Polje polje)...

     public void PostaviKoordinateFigura()...

     public void OznaciPolje()...

     public void OnPointerDown(PointerEventData eventData)...
}
```

Skripta „Polje“ nasljeđuje „MonoBehaviour“ klasu i sučelje „IPointerDownHandler“. „MonoBehaviour“ predstavlja osnovnu klasu iz koje proizlazi svaka skripta u Unity-u.[18] Sadrži metode poput Start() i Awake() koje će se najčešće u ovom radu koristiti. Start() se poziva prvom prilikom kad je skripta omogućena (*eng. enabled*), dok se Awake() poziva kad je objekt skripte inicijaliziran, bez obzira je li omogućena ili ne.[19] „IPointerDownHandler“ sučelje je koje sadrži metodu „OnPointerDown“ čija uloga je detektirati klikove miša.[20]

Javni atributi „xRed“ i „yStupac“ cijelobrojnog tipa podataka (*eng. integer*) predstavljaju redak i stupac polja na ploči. Ploča se sastoji od 8 redaka i 8 stupaca međutim u programu su prvi redak i prvi stupac implementirani indeksom 0, a zadnji redak i zadnji stupac indeksom 7. Tom logikom možemo zaključiti da prvo donje lijevo polje na kojem se

nalazi bijeli damin top za vrijednost „xRed“ i „xStupac“ ima (0, 0), a zadnje gornje desno polje koje zauzima crni kraljev top za vrijednost nosi (7, 7).

Javni atribut „FiguraNaPolju“ objekt je klase „Figura“ koju ćemo kasnije detaljnije obraditi i predstavlja figuru koja se na zadanom polju u danom trenutku nalazi. Na početku sva polja inicijaliziramo na *null*, međutim kasnije kroz metodu „PostaviKoordinateFigura()“ odredimo nalazi li se figura na tom polju ili je polje prazno. Budući da sama funkcionalnost igre i međudjelovanje prostora i figura ovisi o tome nalaze li se i gdje točno figure na ploči, ovaj objekt veoma je važan u ovoj skripti.

Javni atribut „Ploca“ objekt je klase „Igra“ koja ne predstavlja ploču u doslovnom smislu već kompletnu logiku i upravljački aspekt igre koji će se detaljnije objasniti u idućem poglavlju.

Prva metoda ove skripte privatna je metoda Start() koju smo već spominjali. Uloga ove skripte je da se kod svakog elementa polja koje sadrži dijete (npr. polje a8) pokrene metoda „PostaviKoordinateFigura()“.

Druga metoda, privatna metoda „ListaSadrziPolje(List<ObicanIliPosebanPotez> Lista, Polje polje)“ vraća bool vrijednost i sastoji se od dva argumenta. Prvi argument „Lista“ predstavlja listu koja sadrži sve objekte klase „ObicanIliPosebanPotez“. Klasa „ObicanIliPosebanPotez“ sastoji se od tri atributa: „LegalanPotez“, „PosebanPotez“ i „MozePojestiFiguru“. Atribut „LegalanPotez“ objekt je klase „Polje“ i sadrži ono polje na koje se dana figura može pomaknuti. „PosebanPotez“ je atribut tipa bool koji služi kako bi se napravila distinkcija između običnih poteza poput pomicanja pijuna s polja e2 na polje e4 te specijalnih poteza poput rokade, en passanta i promocije koji se odvijaju samo u unaprijed određenim uvjetima. Atribut „MozePojestiFiguru“ također je bool te određuje može li se potezom pojesti druga figura. Većina poteza tu vrijednost ima true zato što primjerice ukoliko zamislimo početnu poziciju i uklonimo bijelog pješaka s a2, primjetit ćemo da u toj danoj poziciji bijeli top s a1 „napada“ crnog pijuna s a7. Taj top može pojesti pijuna na a7 te pritom taj „ObičanIliPosebanPotez“ za atribut „MožePojestiFiguru“ ima *true*. Međutim, uzmemo li u obzir pješake koji se mogu kretati naprijed, ali ne mogu pojesti figuru na taj način, u takvim slučajevima vrijednost atributa „MožePojestiFiguru“ bit će postavljena na *false*. Drugi argument metode „ListaSadrziPolje(…)“ je samo „Polje“ koje provjeravamo nalazi li se na listi. „Lista“ koja je prvi argument metode sadrži poteze čiji je najvažniji atribut „LegalanPotez“ objekt tipa polja. Metodom „ListaSadrziPolje(…)“ prolazimo kroz čitavu listu i ukoliko je traženo polje pronađeno vraća se *true*, a ukoliko nije *false*.

Treća metoda, javna metoda PostaviKoordinateFigura() alocira vrijednost atributu „FiguraNaPolju“ te povezuje figuru s poljem tako da se odredi kojem objektu tipa „Polje“ pripada objekt tipa „Figura“ i obrnuto.

Četvrta metoda, javna metoda OznaciPolje() dopušta označavanje polja odnosno klik na polje ukoliko je zadovoljen jedan od tri uvjeta. Prvi uvjet je da ukoliko je označena figura prikazuju se i omogućuju polja koja se nalaze na listi legalnih poteza. Drugi uvjet omogućuje klik na polje na kojem se nalazi figura koja je one boje koja je trenutno na potezu. Treći uvjet da bi polje bilo omogućeno za potez je da se na tom polju nalazi figura suprotne boje, a da se pritom na listi nalazi taj potez pod legalnim potezom i atribut „MožeUzetiFiguru“ je *true*. Ukoliko nijedan od ovih uvjeta nije zadovoljen odnosno klikom na sva ostala polja koja ne zadovoljavaju ovaj uvjet, resetira se cijela tako da ništa na njoj nije više označeno.

Metoda koja je nasljeđena iz sučelja „IPointerDownHandler“ pod nazivom „OnPointerDown(PointerEventData eventData)“ za svoju implementaciju ima samo poziv gore navedene metode „OznaciPolje()“.

5.4.2. Figura

Skripta „Figura“ služi kao osnovna klasa svim figurama za koje ćemo kasnije pojedinačno izvesti njihove distinktivne klase. Kao i uobičajeno kod nasljeđivanja, sve figure dijele neke zajedničke atribute i metode te su iste implementirane u ovoj skripti.

```
public class Figura : MonoBehaviour, IPointerDownHandler
{
    public bool bijelaFigura;
    public int odigraniPotezi;

    public List<ObicanIliPosebanPotez> legalniPotezi = new
List<ObicanIliPosebanPotez>();
    public List<ObicanIliPosebanPotez> KraljZabranjeniPotezi = new
List<ObicanIliPosebanPotez>();

    public Igra Ploca;
    public Polje PoljeOdFigure;

    void Awake()...

    public void OznaciFiguru()...
```

```
        public void PostaviPoljeOdFigure()...

        public virtual void ProvjeriLegalnePoteze()...

        public void Inicijaliziraj(bool _bijelaFigura, int _odigraniPotezi =
    1)...

        public bool ListaSadrziPolje(List<ObicanIliPosebanPotez> Lista,
    Polje polje)...

        public void OnPointerDown(PointerEventData eventData)...
}
```

„Figura“ kao i prethodna skripta „Polje“ nasljeđuje klasu „MonoBehaviour“ i sučelje „IPointerDownHandler“. Sadrži šest javnih atributa, od kojih je prvi „bijelaFigura“ tipa podataka bool. „bijelaFigura“ ima dva stanja, *true* ukoliko je figura bijele boje i *false* ukoliko je crne. Preko tog atributa određuje se koje se figure stavljaju na raspolaganja trenutnom igraču na potezu. Atribut „odigraniPotezi“ za ulogu ima prebrojavanje koliko je poteza odigrano zadanom figurom. Ta vrijednost veoma je važna u slučajevima rokade, en passanta i općenitom kretanju pješaka. Primjerice, da bi se rokada smatrala legalnim potezom u danoj poziciji, s kraljem i s topom se prije rokade ne smije odigrati nijedan jedini potez. Također, igrač može uzeti pješaka „en passant“ samo u slučaju da je protivnik svog pješaka pomaknuo za dva polja s početne pozicije što se može ustanoviti samo ako se pritom brojač odigranih poteza danog pješaka nijednom nije inkrementirao. Nakon tog poteza svi ostali potezi pješaka limitirani su za samo jedno polje unaprijed.

Skriptu uključuju i dvije veoma bitne liste koje sadrže za tip podataka objekte klase „ObicanIliPosebanPotez“. Lista „legalniPotezi“ akumulira sve poteze dane figure koje zadovoljavaju uvjete legalnosti u trenutnoj poziciji. Ovisno o kojoj je figuri riječ, ta lista sadrži različita polja odnosno poteze. Druga lista pod nazivom „KraljZabranjeniPotezi“ veliku ulogu igra u limitiranju kretanja kralja i utvrđivanju je li na ploči šah, pat ili šah mat. Tu listu ćemo podrobnije obraditi u poglavlju „Upravljanje i logika igre“ u kojoj će se objasniti kako se promjenjivost stanja partija ogleda kroz status kraljeva na ploči.

Iduća dva atributa su objekt klase „Igra“ pod nazivom „Ploca“ koji predstavlja logični i upravljački dio igre te objekt klase „Polje“ pod imenom „PoljeOdFigure“. „PoljeOdFigure“ ima sličnu ulogu kao i objekt „FiguraNaPolju“ u skripti „Polje“. Njen zadatak je da se konstantno zna koje polje sadrži zadanu figuru te kao i „FiguraNaPolju“ čini most i poveznicu između pojmova figura i polja.

Prva metoda ove skripte je „Awake()“ koja za zadatak ima alocirati početne vrijednosti atributima „odigraniPotezi“, „Ploca“ i „PoljeOdFigure“. Ulaskom u metodu poziva se funkciju „PostaviPoljeOdFigure“ putem koje se inicijalizira vrijednost polja objektu „PoljeOdFigure“. To se izvršava tako da se naredbom „GetComponentInParent<Polje>“ pronađe element roditelj koji je ujedno i polje na kojem se figura nalazi kao što možemo vidjeti na hijerarhiji prikazanoj na Slici 6. „odigraniPotezi“ se inicijaliziraju na 0, a „Ploca“ dohvaća objekt tipa „Igra“.

Metoda „OznaciFiguru“ dopušta označavanje samo figura boje koja je trenutno na potezu. Zatim, ukoliko se lista „legalniPotezi“ sadrži barem jedan objekt, foreach petljom se prolazi kroz listu i svako polje s te liste se označava dok sva ostala polja na ploči ostaju odznačena.

Najvažnija metoda ove skripte je virtualna metoda „ProvjeriLegalnePoteze“. Svaka zasebna figura tu metodu „pregazi“ (*eng. override*) s vlastitim implementacijama provjere poteza koji su za zadanu figuru legalni. Primjerice, kretnje lovca i njegovi legalni potezi bit će drugačiji od kretnji topa i njegovih dopuštenih poteza. Budući da ova metoda ima drugačiju implementaciju ovisno o kojoj je figuri riječ, pobliže će se obraditi u poglavljima posvećenim pojedinačnim figurama.

Metoda „Inicijaliziraj(bool _bijelaFigura, int _odigraniPotezi = 1)“ poprima dva argumenta koji se dodjeljuju promoviranoj figuri. Primjerice, pri promoviranju pješaka u topa, bitno je inicijalizirati boju zadane figure. Ukoliko je bijela, prosljeđuje joj se argument *true*. Također je važno i proslijediti inkrementirani broj odigranih poteza, odnosno broj 1 kako se ne bi slučajno dogodilo da se na promoviranu figuru gleda kao figuru koja se nalazi na početnoj poziciji što naravno prilikom promocije nije slučaj.

Metode „ListaSadrziPolje“ i „OnPointerDown“ imaju istu ulogu kao i kod skripte „Polje“. „ListaSadrziPolje“ metoda je kroz koju se provjerava nalazi li se na listi poteza zadano polje, a metoda „OnPointerDown“ reagira na klik miša na zadanu figuru odnosno polje od figure.

5.4.2.1. Bijeli i crni pijun

U ovom i sljedećim potpoglavljima, obradit će se svaka figura pojedinačno. Budući da su bijeli i crni pješaci po funkcionalnostima identični uz jednu jedinu razliku da se bijeli kreću prema „gore“, a crni prema „dolje“, razradit ćemo njihovu implementaciju na primjeru bijelih pješaka.

Implementacija skripte „Bijeli pijun“ započinje tako da klasa nasljeđuje osnovnu klasu „Figura“. Iz skripte „Figura“ preuzima metodu „ProvjeriLegalnePoteze()“ koju „pregazi“ i čiju

implementaciju u ovoj skripti mijenjamo sukladno kretnji zadane figure. Implementacija započinje tako da se očiste liste „legalniPotezi“ i „KraljZabranjeniPotezi“. Zatim se kroz foreach petlju prolazi kroz sva polja na ploči i provjeravaju uvjeti koji će se detaljnije pojasniti.

```
...
if ((this.odigraniPotezi==0) && (item.xRed==(this.PoljeOdFigure.xRed + 2))
&& (item.yStupac == this.PoljeOdFigure.yStupac)&&
(item.gameObject.transform.childCount == 0))
...
```

Polje koje se provjerava u implementaciji je označeno terminom „item“, a na ovom primjeru možemo vidjeti četiri različita uvjeta razdvojena logičkim operatorom AND. Prvi uvjet označava da pješak sadrži nula odigranih poteza odnosno da se figura nalazi na početnoj poziciji. Drugi uvjet naznačuje da se „item“ nalazi točno dva reda „iznad“ polja na kojem se nalazi figura, a treći uvjet da se to polje nalazi u istom stupcu kao i polje na kojem je pješak. Posljednji uvjet provjerava je li „item“ polje prazno polje.

Ukoliko je ovaj uvjet zadovoljen, to polje se dodaje kao atribut „LegalanPotez“ pri kreiranju novog objekta „ObicanIliPosebanPotez“ koji se pritom dodaje na listu „legalniPotezi“. Laički rečeno, taj potez omogućava pješacima s početne pozicije pomak za dva polja unaprijed.

```
...
else if ((item.xRed == 5) && ((Ploca.SvaPolja[item.xRed - 1,
PoljeOdFigure.yStupac].gameObject.transform.childCount != 0) &&
(Ploca.SvaPolja[item.xRed - 1, item.yStupac].FiguraNaPolju is CrniPijun) &&
(item.xRed == this.PoljeOdFigure.xRed + 1) &&((item.yStupac ==
this.PoljeOdFigure.yStupac- 1) || (item.yStupac ==
this.PoljeOdFigure.yStupac + 1)) && (Ploca.PosljednjaPomaknutaFigura ==
Ploca.SvaPolja[item.xRed - 1, item.yStupac].FiguraNaPolju &&
Ploca.SvaPolja[item.xRed - 1, item.yStupac].FiguraNaPolju.odigraniPotezi ==
1)))
...
```

Drugi uvjet malo je podulji i predstavlja implementaciju posebnog poteza „en passant“. Kod bijelih pješaka, taj potez uvijek pomiče pješaka s petog na šesti red ploče, no budući da brojimo od nultog reda, provjeru izvršavamo tako da provjeravamo je li atribut „xRed“ pronađenog „item-a“ jednak 5. Ukoliko jest, zatim provjeravamo nalazi li se u istom tom stupcu kao i „item“, ali redak niže bilo kakva figura te ako jest je li ta figura crni pješak. Također vršimo provjeru je li „item“ u redu „iznad“ označene figure odnosno bijelog pijuna te je pritom „item“ u stupcu lijevo od njega ili u stupcu desno od njega. Uz to, provjerava se je li

posljednja pomaknuta figura protivnika baš taj crni pješak kojeg želimo uzeti i je li mu je broj odigranih poteza jednak 1 jer samo tada i „en passant“ jest legitiman i dopušten potez u danoj poziciji.

Ako je uvjet zadovoljen, kreira se novi objekt „ObicanIliPosebanPotez“ kao u prethodnom uvjetu te se dodaje u listu „legalniPotezi“, međutim ovaj put atribut poteza „PosebanPotez“ postavljamo kao *true*.

```
...
else if ((((item.xRed == (this.PoljeOdFigure.xRed + 1)) &&
(item.yStupac == (this.PoljeOdFigure.yStupac - 1))) ||
((item.xRed ==(this.PoljeOdFigure.xRed + 1)) &&
(item.yStupac == (this.PoljeOdFigure.yStupac + 1)))))
...
```

Ovaj uvjet označava polja na koja inače bijeli pijun može pojesti figuru te ta polja dodajemo u listu „KraljZabranjeniPotezi“ budući da se kralju ograničava kretanje preko polja putem kojih se mogu kretati protivničke figure. Unutar njega nalaze se još dva poduvjeta.

```
...
   if ((item.gameObject.transform.childCount != 0) && (this.bijelaFigura !=
   item.FiguraNaPolju.bijelaFigura))
        {
                  if(item.xRed != 7)
                        {
                             ...
                        }
                        else
                        {
                             ...
                        }
        }
...
```

Prvi uvjet provjerava ima li na poljima na kojima bijeli pješak može pojesti figuru bilo koja figura te je li ta figura različite boje od našeg pješaka odnosno crna. Ukoliko je taj uvjet zadovoljen, provjerava se radi li se o posljednjem redu ploče jer se onda taj čin tretira kao poseban potez odnosno promocija pješaka. U oba slučajeva se dodaje potez na listu „legalniPotezi“ no jedan od njih je poseban dok je drugi običan potez.

```
...
else if ((item.xRed == (this.PoljeOdFigure.xRed + 1)) && (item.yStupac ==
this.PoljeOdFigure.yStupac) && (item.gameObject.transform.childCount == 0))
           {
              if (item.xRed != 7)
              {
                 ...
               }
               else
               {
                  ...
               }
...
```

Posljednji uvjet ove skripte predstavlja najobičniji potez pješaka, a to je pomicanje za jedno polje unaprijed. Najprije se mora provjeriti nalazi li se polje u istom stupcu kao i polje na kojem se nalazi figura te nalazi li se za jedan red ispred našeg bijelog pješaka. Na koncu bitno je i provjeriti je li zadano polje prazno. Ako su svi uvjeti zadovoljeni provjerava se kao i u prethodnom primjeru radi li se o promociji ili ne te se također kreira objekt „ObicanIliPosebanPotez“ koji se dodaje na listu „legalniPotezi“.

Nakon što smo detaljno obradili skriptu „Bijeli pijun“, u ovom potpoglavlju također ćemo se dotaknuti promocije pješaka te implementacije sučelja prikazanog na slici 3 u poglavlju „Scena i UI“. Funkcionalni dio tog sučelja implementiran je kroz skriptu „PrikazSuceljaPromocije“ koja kao gotove sve Unity skripte nasljeđuje osnovnu klasu „MonoBehaviour“.

```
public class PrikazSuceljaPromocije : MonoBehaviour
{
    public GameObject PromocijaSucelje;
    public GameObject BijeleFigure;
    public GameObject CrneFigure;

    public List<PromocijaOnClick> SveFigureOdabir = new
List<PromocijaOnClick>();

    void Awake()...

    public void UpravljacBijelihFigura(Polje poljePijun)...
```

```
    public void UpravljacCrnihFigura(Polje poljePijun)...

    public void PovratakNaIgru()...
}
```

Skripta sadrži tri atributa tipa GameObject koji predstavljaju elemente sučelja prikazanih prilikom odigravanja poteza promocije. Javni atribut „PromocijaSucelje“ predstavlja čitavo sučelje čiji prikaz se mora aktivirati kad do promocije dođe dok se javni atributi „BijeleFigure“ i „CrneFigure“ prikazuju ovisno o tome promovira li se bijeli ili crni pješak.

Najvažniji atribut je „SveFigureOdabir“ koja predstavlja javnu listu objekata „PromocijaOnClick“, klase koja će se kasnije u ovom potpoglavlju detaljnije obraditi. Prva metoda ove skripte je naslijeđena metoda „Awake()“ iz „MonoBehaviour“ klase, a njena uloga je napuniti tu listu objektima „PromocijaOnClick“ koja ujedno i sadrži funkcionalnosti figura koje biramo s liste.

Metode „UpravljacBijelihFigura()“ i „UpravljacCrnihFigura()“ su gotovo identične. Obje primaju za argument objekt klase „Polje“ pod nazivom „poljePijun“ čija je svrha poznavanje točnog polja na kojem se događa promocija. Metoda započinje tako da se „aktivira“ GameObject „PromocijaSucelje“ koji se pritom prikazuje zajedno sa svojim child elementima „PromocijaBlankScreen“ i „PopUpPozadina“ te njegov child element „PromocijaText“ čiju hijerarhiju možemo vidjeti na slici 4. „UpravljacBijelihFigura()“ i „UpravljacCrnihFigura()“ odudaraju u idućem koraku metode gdje se kod prve metode „aktivira“ GameObject „BijeleFigure“, a kod druge „CrneFigure“ što rezultira tome da se prikazuju figure one boje koja je na potezu. Posljednji korak je prolazak kroz listu „SveFigureOdabir“ te prosljeđivanje objekta „poljePijun“ odabranoj odnosno kliknutoj figuri.

Posljednja metoda „PovratakNaIgru()“ za ulogu ima „deaktivirati“ odnosno sakriti sve GameObject objekte što se postiže prosljeđivanjem argumenta *false* svakom od objekata korištenjem UnityEngine metode „SetActive(bool value)“.

Skripta „PromocijaOnClick“ predstavlja funkcionalni dio promocije odnosno njena uloga je da se ovisno o figuri na koju igrač klikne izvrši zamjena te figure i svih njenih karakteristika za promoviranog pješaka.

```
public class PromocijaOnClick : MonoBehaviour, IPointerDownHandler
{
    private Polje PoljePijun = null;
    public Texture SlicicaFigure;
    public PrikazSuceljaPromocije prikazSuceljaPromocije;

    public void DohvatiPoljePijuna(Polje _poljePijun)...

    public void UkloniPijuna(Polje ocistiPolje)...

    public virtual void PromovirajPijuna(Polje poljePijun)...

    public void OnPointerDown(PointerEventData eventData)...
}
```

Sastoji se od tri atributa i četiri metode. Prvi privatni atribut „PoljePijun“ tip je objekta „Polje“ te ga inicijaliziramo na *null*. Njegova uloga je preuzeti vrijednost polja na kojem se izvršava promocija. Drugi atribut pod nazivom „SlicicaFigure“ za tip sadrži objekt „Texture“ čija uloga je rukovanje teksturama unutar Unity-a[21], u ovom slučaju tekstura figure koju ćemo umjesto one od pješaka zamjeniti onom od izabrane figure. Treći atribut je instanca prethodno objašnjene klase „PrikazSuceljaPromocije“ putem koje ćemo morati pristupiti nekim od njezinih metoda.

Prva metoda „DohvatiPoljePijuna(Polje _poljePijun)“ zaprima za argument polje na kojem se nalazi pješak te ga alocira atributu „PoljePijun“. Metoda „UkloniPijuna(Polje ocistiPolje)“ prvo izvršava metodu iz biblioteke „UnityEngine“ pod nazivom „DestroyImmediate(Object obj)“ čiji cilj je odmah „uništiti“ u argumentu proslijeđeni objekt.[22] U ovom slučaju, „uništava“ se figura koja se nalazi u danom trenutku na polju argumenta metode „UkloniPijuna(Polje ocistiPolje)“ tj. na „ocistiPolje“. Idući korak metode je odmah nakon što je ta figura „uništena“, alocirati atribut „FiguraNaPolju“ od „ocistiPolje“ na *null*.

Virtualna metoda „PromovirajPijuna(Polje poljePijun)“ najvažnija je metoda u ovoj skripti. Kao i dvije metode prije nje, zaprima kao argument polje na kojem se promocija odvija. Započinje tako da se poziva metoda „UkloniPijuna(Polje ocistiPolje)“ te se zatim preko objekta „prikazSuceljaPromocije“ poziva metoda „PovratakNaIgru()“. Međutim, ta se metoda također „pregazi“ u osam različitih skripti čije uloge su dodavanje odabrane figure kao GameObject na ploču. Tih osam skripti dijelimo na četiri koje promoviraju u bijele figure i četiri koje promoviraju u crne. Redom glase: „PijunUBijelogKonja“, „PijunUBijelogLovca“,

„PijunUBijelogTopa“, „PijunUBijeluDamu“, „PijunUCrnogKonja“, „PijunUCrnogLovca“, „PijunUCrnogTopa“ te na koncu „PijunUCrnuDamu“.

```
public class PijunUBijeluDamu : PromocijaOnClick
   {
       public override void PromovirajPijuna(Polje poljePijun)
       {
   Dama DodanaFigura =
   poljePijun.GetComponentInChildren<Canvas>().gameObject.AddComponent
   <Dama>();
           DodanaFigura.Inicijaliziraj(true);
       }
   }
```

Uzmimo za primjer skriptu „PijunUBijeluDamu“ koja nasljeđuje klasu „PromocijaOnClick“. Sadrži jednu metodu, a to je „pregažena“ metoda „PromovirajPijuna(Polje poljePijun)“. Objekt „Dama“ koji predstavlja figuru dame, postavljamo kao GameObject za dijete element proslijeđenog elementa odnosno argumenta „poljePijun“. Zatim se preko te figure dohvaća metoda „Inicijaliziraj(bool _bijelaFigura, int _odigraniPotezi = 1)“ kojoj za argument tipa podataka bool prosljeđujemo *true* jer je riječ o promociji u bijelu figuru. Ukoliko bi bilo riječ o crnoj figuri, argument metode glasio bi *false*.

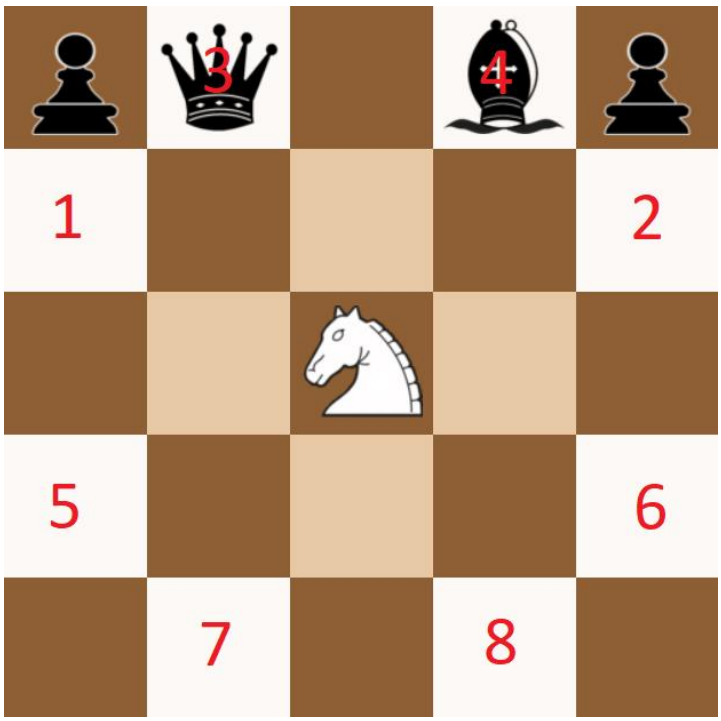
Posljednja metoda skripte „PromocijaOnClick“ naziva se „OnPointerDown (PointerEventData eventData)“ koja detektira klik miša i konačan odabir figure prilikom promocije. Započinje tako da se poziva metoda „PromovirajPijuna“ te mu se prosljeđuje atribut „PoljePijun“. Zatim se tekstura kliknute figure alocira na mjesto „RawImage“ prijašnje figura koja je bila na tom polju, u ovom slučaju pješak.

5.4.2.2. Konj

Skripta „Konj“ ima najkraću i najjednostavniju implementaciju od svih figura. Iako su kretnje konjem najupečatljivije te ujedno i najnepredvidljivije na šahovskoj ploči, konj ne sadrži nikakve posebne poteze i specifične metode.

Skripta započinje kao i prethodna, tako da se metoda „ProvjeriLegalnePoteze()“ „pregazi“ te se potom liste „legalniPotezi“ i „KraljZabranjeniPotezi“ očiste. Zatim se prolaskom kroz sva polja na ploči foreach petljom provjerava uvjet nad poljima koje su konju u danom trenutku dostupna.

```
if ((((item.xRed == (this.PoljeOdFigure.xRed + 1)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac - 2)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed + 1)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac + 2)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed + 2)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac - 1)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed + 2)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac + 1)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed - 1)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac - 2)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed - 1)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac + 2)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed - 2)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac - 1)))) ||
                ((item.xRed == (this.PoljeOdFigure.xRed - 2)) &&
((item.yStupac == (this.PoljeOdFigure.yStupac + 1))))))
            {
                KraljZabranjeniPotezi.Add(new
ObicanIliPosebanPotez(item));


                if(item.gameObject.transform.childCount == 0 ||
this.bijelaFigura != item.FiguraNaPolju.bijelaFigura)
                {
                    legalniPotezi.Add(new ObicanIliPosebanPotez(item));
                }
            }
```



Slika 11: Kretanje konja

Polje koje se provjerava u implementaciji označeno je terminom „item“ te se u prvom uvjetu provjerava 8 polja prikazanih na slici 9. Kao što otprije znamo, svako polje sadrži atribut „xRed“ i „xStupac“. Na početku uvjeta možemo uvidjeti da se provjerava je li vrijednost atributa „xRed“ za jedan veće od vrijednosti polja na kojem se nalazi konj. Točnije provjerava se polje na slici 9 označeno brojem 1. Provjera reda i stupca zadanog polja povezana je zagradama i logičkim operatorom AND te se redom provjerava osam potencijalnih polja kojima konj može pristupiti. Ukoliko provjeravano polje spada u kategoriju jednog od tih osam potencijalnih, stavlja se na listu „KraljZabranjeniPotezi“ odnosno to polje se oduzima protivničkom kralju jer bi inače na zadanom polju kralju bio šah.

Drugi uvjet odnosno poduvjet prethodnog uvjeta provjerava je li polje prazno ili nalazi li se na polju figura drugačije boje to jest protivnička figura. Ukoliko je to slučaj, to polje se dodaje na listu „legalniPotezi“.

5.4.2.3. Lovac

Skripta „Lovac“ specifična je zato što lovac teoretski može pokrivati mnogo polja istovremeno. Međutim, problem koji njihovo kretanje predstavlja je koliki je zapravo doseg dijagonale preko koje se lovčeva domena prostire. Budući da ne može preskakati figure, lovčeva kretnja limitirana je do polja na kojem se nalaze druge figure. Kako bi identificirali ta polja, uvodimo četiri različita atributa koji za tip podataka imaju objekt „Polje“.

```
public override void ProvjeriLegalnePoteze()
    {
        Polje provjeriGL = null;
        Polje provjeriGD = null;
        Polje provjeriDD = null;
        Polje provjeriDL = null;

        legalniPotezi.Clear();
        KraljZabranjeniPotezi.Clear();

        foreach (Polje item in Ploca.SvaPolja)
        {
...
```

Atributi su inicijalizirani na *null* vrijednosti te je njihova uloga da identificiraju gornja lijeva, gornja desna, donja desna i donja lijeva polja u odnosu na polje na kojem se nalazi

figura lovca. Također kao u prethodnoj skripti, očiste se liste „legalniPotezi“ i „KraljZabranjeniPotezi“ te se prolazi foreach petljom kroz sva polja na ploči.

```
...
     if (((item.xRed - this.PoljeOdFigure.xRed) == (item.yStupac -
   this.PoljeOdFigure.yStupac))||((item.xRed - this.PoljeOdFigure.xRed)
   == -(item.yStupac - this.PoljeOdFigure.yStupac)))
             {
                 legalniPotezi.Add(new ObicanIliPosebanPotez(item));

                 if(item.gameObject.transform.childCount != 0)
                 {
...
```

Potom se provjerava uvjet čija uloga je provjeriti je li razlika između atributa „xRed“ od provjeravanog polja i atributa „xRed“ od polja na kojem se nalazi figura jednaka razlici između atributa „yStupac“ od provjeravanog polja i atributa „yStupac“ od polja na kojem se nalazi figura. Prvim dijelom uvjeta upotpunjavamo sva polja koja su donja lijeva i gornja desna od figure te ih dodajemo na listu „legalniPotezi“ dok se drugim dijelom uvjeta obuhvaćaju i slučajevi negativnih razlika (npr. 2 == -2) odnosno dodaje se minus kako bi se obuhvatile apsolutne vrijednosti danih razlika. Tim dijelom uvjeta se obuhvaćaju polja koja su gornja lijeva te donja desna od figure lovca te se također dodaju na listu „legalniPotezi“. Ukoliko su uvjeti zadovoljeni vrši se provjera nalazi li se na polju koje provjeravamo bilo kakva figura.

```
...
if ((item.xRed < this.PoljeOdFigure.xRed && item.yStupac >
this.PoljeOdFigure.yStupac))
                    {
                        provjeriDD = item;
...
```

Ako je slučaj da se na tom polju nalazi figura, kreće provjera nalazi li se ta figura gore lijevo, gore desno, dolje lijevo ili dolje desno od figure lovca. U kodu je prikazana provjera za dolje desno. Kako bi se figura nalazila dolje desno od figure lovca, prvo mora imati manju vrijednost „xRed“ od polja na kojem se nalazi lovac odnosno mora se nalaziti „ispod“ njega. Uz to, mora sadržavati vrijednost atributa „yStupac“ veću od polja lovca što se ogleda u tome da je stupac provjeravanog polja figure „desniji“ od stupca u kojem se nalazi lovac. Zadovoljavanjem tog uvjeta, atributu „provjeriDD“ alocira se vrijednost provjeravanog polja.

```
...
if (item.FiguraNaPolju is Kralj && item.FiguraNaPolju.bijelaFigura !=
this.bijelaFigura)
           {
            foreach (ObicanIliPosebanPotez potez in
            item.FiguraNaPolju.legalniPotezi)
                          {
if (((potez.LegalanPotez.xRed - this.PoljeOdFigure.xRed) ==
(potez.LegalanPotez.yStupac - this.PoljeOdFigure.yStupac)) ||
((potez.LegalanPotez.xRed - this.PoljeOdFigure.xRed) == -
(potez.LegalanPotez.yStupac - this.PoljeOdFigure.yStupac)))
                               {
        if (this.PoljeOdFigure == potez.LegalanPotez)
                               {

            KraljZabranjeniPotezi.Remove(potez);
                               }
                               else
                               {

            KraljZabranjeniPotezi.Add(potez);
                               }
                           }
                       }
                   }
...
```

Bitno je odrediti još jedan važan uvjet koji se odnosi na protivničkog kralja. Ukoliko se na listi legalnih poteza protivničkog kralja nalaze polja koje su u domeni lovca odnosno sva dijagonalna polja kojima lovac ima mogućnost migrirati, treba ih oduzeti odnosno dodati na listu „KraljZabranjeniPotezi“. Međutim, ukoliko je na listi legalnih poteza protivničkog kralja i polje na kojem se nalazi lovac odnosno ako kralj ima mogućnost pojesti tog lovca, to se polje briše s liste „KraljZabranjeniPotezi“ jer kralj može figuru koja je u njegovoj domeni kretanja pojesti.

```
...
List<ObicanIliPosebanPotez> pomocnaListaLegalnihPoteza = new
List<ObicanIliPosebanPotez>();
```

```
pomocnaListaLegalnihPoteza.AddRange(legalniPotezi);


foreach (ObicanIliPosebanPotez item in pomocnaListaLegalnihPoteza)
        {
if((item.LegalanPotez.gameObject.transform.childCount != 0 &&
item.LegalanPotez.FiguraNaPolju.bijelaFigura == this.bijelaFigura) ||
(provjeriGL && (item.LegalanPotez.xRed > provjeriGL.xRed &&
item.LegalanPotez.yStupac < provjeriGL.yStupac)) ||
(provjeriGD && (item.LegalanPotez.xRed > provjeriGD.xRed &&
item.LegalanPotez.yStupac > provjeriGD.yStupac)) ||
(provjeriDL && (item.LegalanPotez.xRed < provjeriDL.xRed &&
item.LegalanPotez.yStupac < provjeriDL.yStupac)) ||
(provjeriDD && (item.LegalanPotez.xRed < provjeriDD.xRed &&
item.LegalanPotez.yStupac > provjeriDD.yStupac)))
            {
                legalniPotezi.Remove(item);
            }
...
```

Kako bi uredili listu „legalniPotezi“ te odredili koje poteze treba iz nje izbaciti, inicijaliziramo pomoćnu listu „pomocnaListaLegalnihPoteza“. Prolazeći kroz nju brišemo poteze koji su inače u domeni lovca međutim su mu nedostupne jer se na putu nalazi figura koja ta polja blokira. Uvjet započinje tako da se provjerava nalazi li se na ispitivanom polju figura koja je identične boje lovcu. Ukoliko jest, taj se potez briše iz liste „legalniPotezi“. Ostatak uvjeta malo je dulji te provjerava nalazi li se bilo koji od legalnih poteza u bilo kojem od četiri pravca „iza“ figure. Budući da smo otprije ustanovili polja na kojima se nalaze figure gore desno, gore lijevo, dolje desno te dolje lijevo od lovca točnije atributi „provjeriGL“, „provjeriGD“, „provjeriDL“ i „provjeriDD“ pomažu nam utvrditi dokle točno seže domena lovca. Potezi na tom pravcu, ali „iza“ figura na ta četiri polja, brišu se jer lovac nema mogućnost preskakanja figura.

```
...
foreach (ObicanIliPosebanPotez item in legalniPotezi)
        {
            if (!ListaSadrziPolje(KraljZabranjeniPotezi,
item.LegalanPotez))
            {
                KraljZabranjeniPotezi.Add(item);
            }
        }
...
```

Nakon što smo ustanovili koji su potezi legalni, a koji nisu te uspješno uredili listu „legalniPotezi“, sve te legalne poteze dodajemo na listu „KraljZabranjeniPotezi“. Logično je da ukoliko lovac pokriva određeni set polja da kralj nema pristupa tim poljima.

```
...
if (provjeriDL && provjeriDL.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
        {
     KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriDL));
        }
if (provjeriDD && provjeriDD.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
        {
     KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriDD));
        }
if (provjeriGL && provjeriGL.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
        {
     KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriGL));
        }
if (provjeriGD && provjeriGD.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
        {
     KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriGD));
        }
     }
}
```

Na koncu skripte dodajemo u listu „KraljZabranjeniPotezi“ i polja na kojem se nalaze prijateljske „rubne“ figure odnosno gornje lijeve i desne te donje lijeve i desne figure koje su istobojne lovcu. Ovim pristupom postiže se motiv „branjene figure“ odnosno da kralj ne može uzeti figuru koja ga napada ukoliko je ta figura na polju koje „čuva“ lovac.

5.4.2.4. Top

Skripta „Top“ dijeli velike sličnosti sa skriptom „Lovac“. Dok se lovac kreće dijagonalno, top se kreće ravno u svim pravcima. Također kao i kod lovca, problem nastaje problem kolika je zapravo topova domena budući da ne može preskakati figure te je njegova domena ograničena ili rubom ploče ili figurom koja stoji kao meta ili prepreka. Kao i kod lovca, uvodimo četiri atributa tipa „Polje“.

```
public override void ProvjeriLegalnePoteze()
    {
        Polje provjeriPoljeIznad = null;
        Polje provjeriPoljeDesno = null;
        Polje provjeriPoljeIspod = null;
        Polje provjeriPoljeLijevo = null;

        legalniPotezi.Clear();
        KraljZabranjeniPotezi.Clear();

        foreach (Polje item in Ploca.SvaPolja)
        {
...
```

Atributi ovog puta služe za identificiranje lijevog, desnog, gornjeg i donjeg polja u odnosu na topa. Kao u prijašnjem potpoglavlju, inicijaliziraju se na *null*, očiste se liste „legalniPotezi“ i „KraljZabranjeniPotezi“ te se prolazi kroz sva polja na ploči foreach petljom.

```
...
if ((item.xRed == this.PoljeOdFigure.xRed) ||
    (item.yStupac == this.PoljeOdFigure.yStupac))
            {
                legalniPotezi.Add(new ObicanIliPosebanPotez(item));

                if (item.gameObject.transform.childCount != 0)
                {
...
```

Zatim za svako polje na ploči provjeravamo nalazi li se u istom redu ili u istom stupcu kao i polje na kojem se nalazi top. Prvi dio uvjeta provjerava sve horizontalne kretnje topa tako da uspoređuje je li atribut „xRed“ ispitivanog polja identičan atributu „xRed“ polja na

kojem se na nalazi figura topa. Ukoliko to jest slučaj, dodaje se novi potez na listu „legalniPotez“. Isto vrijedi i za sve vertikalne kretnje topa te se provjerava jesu li iste vrijednosti atributa „yStupac“ kod provjeravanog polja kao kod atributa „yStupac“ polja na kojem je figura. Ti se potezi također pridodaju na listu. Nakon što ustanovimo da se ispitivano polje nalazi na istom redu ili stupcu kao i figura, ispitujemo nalazi li se na tom istom polju i bilo koja figura.

```
...
if (item.xRed < this.PoljeOdFigure.xRed)
                    {
                        provjeriPoljeIspod = item;
...
```

Zatim to polje prolazi kroz četiri uvjeta koji provjeravaju nalazi li se to ispitivano polje s figurom na njemu lijevo, desno, iznad ili ispod figure topa. Uzmimo za primjer uvjet koji ispituje nalazi li se polje s figurom na njemu ispod figure topa. Kako bi se zadovoljio taj uvjet, to polje mora imati manju vrijednost atributa „xRed“ od topa odnosno mora se nalaziti u redu koji je „niži“ od reda u kojem je top. Ako je taj uvjet zadovoljen, alocira se vrijednost ispitivanog polja atributu „provjeriPoljeIspod“.

```
...
if(item.FiguraNaPolju is Kralj && item.FiguraNaPolju.bijelaFigura !=
this.bijelaFigura)
                        {
foreach(ObicanIliPosebanPotez potez in item.FiguraNaPolju.legalniPotezi)
                            {
          if(this.PoljeOdFigure.yStupac == potez.LegalanPotez.yStupac)
                                {
                    KraljZabranjeniPotezi.Add(potez);
                    if (this.PoljeOdFigure == potez.LegalanPotez)
                                    {
                        KraljZabranjeniPotezi.Remove(potez);
                                    }
                                }
                            }
                        }
...
```

Nakon što smo ustanovili na kojem se polju nalazi figura koja topu služi kao meta ukoliko je protivnička ili prepreka ako je prijateljska, bitno je provjeriti radi li se o protivničkom kralju. Ako se ustanovi da je ta figura protivnički kralj koji je trenutno pod šahom budući da se nalazi na meti topa, potrebno je sva polja koja udara top na tom pravcu dodati na listu „KraljZabranjeniPotezi“. U kodu možemo vidjeti primjer u kojem se razmatra opcija da je protivnički kralj figura koja se nalazi ispod topa te je prolaskom kroz sve legalne poteze kralja foreach petljom potrebno odstraniti sve legalne poteze koje se nalaze na toj vertikali odnosno u tom stupcu budući da i kralj i top dijele istu vrijednost atributa „yStupac“. Međutim ako je polje na kojem se nalazi top u listi legalnih poteza protivničkog kralja, taj se potez briše s liste „KraljZabranjeniPotezi“ jer kralj može tog topa pojesti budući da se nalazi u domeni kretanja kralja.

```
...
List<ObicanIliPosebanPotez> pomocnaListaLegalnihPoteza = new
List<ObicanIliPosebanPotez>();


pomocnaListaLegalnihPoteza.AddRange(legalniPotezi);


foreach (ObicanIliPosebanPotez item in pomocnaListaLegalnihPoteza)
            {
if ((item.LegalanPotez.gameObject.transform.childCount != 0 &&
item.LegalanPotez.FiguraNaPolju.bijelaFigura == this.bijelaFigura) ||
(provjeriPoljeDesno && item.LegalanPotez.yStupac >
provjeriPoljeDesno.yStupac) ||
(provjeriPoljeIznad && item.LegalanPotez.xRed > provjeriPoljeIznad.xRed) ||
(provjeriPoljeLijevo && (item.LegalanPotez.yStupac <
provjeriPoljeLijevo.yStupac)) ||
(provjeriPoljeIspod && (item.LegalanPotez.xRed < provjeriPoljeIspod.xRed)))
                {
                    legalniPotezi.Remove(item);
                }
            }
...
```

Koristeći pomoćnu listu pod nazivom „pomocnaListaLegalnihPoteza“, utvrđuje se koje poteze s liste „legalniPotezi“ treba izbaciti. Prvo ju inicijaliziramo te joj potom dodamo sve elemente liste „legalniPotezi“ kako bi prolazeći kroz nju mogli brisati elemente iz originalne liste. Cilj je obrisati one poteze koji su inače u domeni topa te se nalaze na njegovoj

horizontali i vertikali međutim su nedostupni zbog toga što se na putu do njih nalazi figura koja ih blokira. Uvjet počinje s ispitivanjem radi li se o prijateljskoj figuri na provjeravanom polju što znači da to polje automatski možemo izbrisati s liste „legalniPotezi“ jer je nemoguće pojesti vlastitu figuru. Ukoliko je riječ o protivničkim figurama, u nastavku uvjeta brišemo sve poteze s liste čija polja su „desnije“ od polja „provjeriDesnoPolje“ na kojem se nalazi figura koja ta polja blokira odnosno brišemo sve poteze čija vrijednost atributa „yStupac“ je veća od atributa „yStupac“ polja „provjeriDesnoPolje“. Isto vrijedi i za ostale smjerove te su oni obuhvaćeni provjerom nalaze li se potezi na tom ispitivanom pravcu „iza“ figura koja te poteze čini nedostupnima.

```
...
foreach (ObicanIliPosebanPotez item in legalniPotezi)
        {
        if (!ListaSadrziPolje(KraljZabranjeniPotezi, item.LegalanPotez))
            {
                KraljZabranjeniPotezi.Add(item);
            }
        }
...
```

Potom sve te poteze iz domene topa koje smo identificirali kao legalnima dodajemo na listu „KraljZabranjeniPotezi“ budući da protivnički kralj nikako ne bi imao mogućnost pristupa tim poljima jer bi inače bio u šahu.

```
...
if (provjeriPoljeIspod && provjeriPoljeIspod.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
          {
KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriPoljeIspod));
          }
if (provjeriPoljeIznad && provjeriPoljeIznad.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
          {
KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriPoljeIznad));
          }
if (provjeriPoljeLijevo && provjeriPoljeLijevo.FiguraNaPolju.bijelaFigura
== this.bijelaFigura)
          {
KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriPoljeLijevo));
```

```
            }
if (provjeriPoljeDesno && provjeriPoljeDesno.FiguraNaPolju.bijelaFigura ==
this.bijelaFigura)
            {
KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(provjeriPoljeDesno));
            }
        }
```

Za epilog skripte na listu „KraljZabranjeniPotezi“ dodajemo i poteze na kojem se nalaze figure koje „blokiraju“ topa. Iako ograničavaju kretanje topa i u jednu ruku mu „stoje“ na putu, dodavanje tog poteza na listu kralju zabranjenih poteza postižemo međudjelovanje figura koje je kasnije bitno kod recimo davanje šah mata protivniku jer top tu figuru koja ga „blokira“ ujedno i brani da ga kralj ne može pojesti.

5.4.2.5. Dama

Skripta „Dama“ najduža je skripta od svih skripti figura, ali s obzirom da predstavlja figuru koja je kombinacija lovca i topa te sadrži sve identične koncepte implementirane u protekla dva potpoglavlja, ukratko ćemo se na nju osvrnuti.

Skripta započinje tako da se inicijalizira osam atributa koji redom označavaju svih osam strana na koje se dama može kretati. Četiri su svojstvena topu te provjeravaju vertikalna i horizontalna „rubna“ polja, a četiri lovcu koji provjeravaju dijagonalna polja. Kao i u protekle dvije skripte proces utvrđivanja tih „rubnih“ polja je identičan, samo su uvjeti dvostruko dulji zbog dvostruko većeg opsega kretanja od samog topa i lovca.

Kao i u skriptama topa i lovca kreiramo pomoćnu listu kroz koju se rješavamo nedostupnih polja koji se nalaze iza figura koje „blokiraju“ kretanje. Potom ta polja dodajemo na listu kralju zabranjenih poteza te na koncu skripte sva „rubna“ polja koje sadrže figure također dodajemo na listu poteza koji su kralju nedostupna.

5.4.2.6. Kralj

Skripta „Kralj“ predstavlja najvažniju figuru na ploči, figuru oko koje se cijela igra temelji i figure koja nije najjača na ploči, ali je najdragocjenija. Kralju je prirodno kretati se za jedno polje u svim smjerovima, međutim u kombinaciji s topom, kralj ima mogućnost posebnog poteza pod nazivom „rokada“. Rokadom se kralj pomiče za dva mjesta ulijevo ili za dva mjesta udesno, a da se pritom top sa svoje početne pozicije pomakne, ukoliko je riječ o lijevoj rokadi, na polje kralju s desne strane, a ukoliko o desnoj, kralju s lijeve strane.

```
public override void ProvjeriLegalnePoteze()
    {
        legalniPotezi.Clear();
        KraljZabranjeniPotezi.Clear();

        foreach(Polje item in Ploca.SvaPolja)
        {
...
```

Kao i svaka skripta implementirane figure i ova također nasljeđuje klasu „Figura“ te „pregazi“ metodu „ProvjeriLegalnePoteze()“. Identična procedura slijedi kao i kod ostalih, čišćenje listi „legalniPotezi“ i „KraljZabranjeniPotezi“ te pokretanje petlje kojom prolazimo kroz sva polja na ploči.

```
   ...
if(((((item.yStupac - this.PoljeOdFigure.yStupac) <= 1) && ((item.yStupac -
this.PoljeOdFigure.yStupac) >= -1)) &&
(((item.xRed - this.PoljeOdFigure.xRed) <= 1) && ((item.xRed -
this.PoljeOdFigure.xRed) >= -1))))
               {
                KraljZabranjeniPotezi.Add(new ObicanIliPosebanPotez(item));
...
```

Zatim slijedi uvjet koji uvjetuje da se protivničkom kralju zabrane svi potezi u domeni „našeg“ kralja. To se postiže tako da se provjerava da razlika između stupaca odnosno vrijednosti atributa „yStupac“ kod ispitivanog polja i polja na kojem je figura kralja ostane u rasponu od -1 do 1 čime je uvjetovano da će kralj potencijalno moći pomaknuti u lijevi stupac, ostati u stupcu u kojem jest ili u desni stupac. Identičan raspon mora vrijediti i za redove te je isti implementiran i u nastavku uvjeta koji nalaže da se kralj može potencijalno pomaknuti u red „niže“, ostati u istom redu ili u red „iznad“. Naglasak na potencijalno jer ovim uvjetom samo uskraćujemo poteze protivničkom kralju jer ni teoretski jedan kralj ne smije zagaziti u domenu drugog kralja.

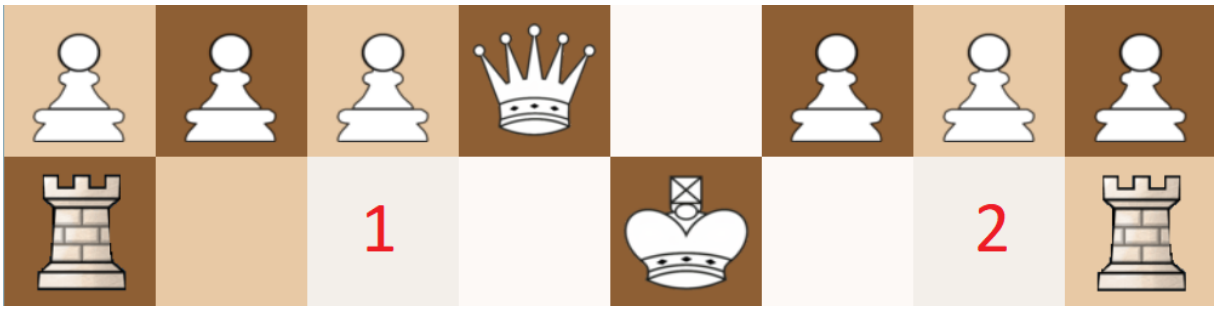
```
...
if(((item.gameObject.transform.childCount == 0) ||
(this.bijelaFigura != item.FiguraNaPolju.bijelaFigura)) &&
((this.bijelaFigura && !ListaSadrziPolje(Ploca.crniNapadackiPotezi, item))
|| (!this.bijelaFigura && !ListaSadrziPolje(Ploca.bijeliNapadackiPotezi,
item))))
                    {
                        legalniPotezi.Add(new ObicanIliPosebanPotez(item));
                    }
                }
...
```

Ovim poduvjetom zapravo određujemo je li potez legalan ili nije. Ako su zadovoljeni kriteriji prijašnjeg uvjeta, da bi potez bio legalan, to polje mora biti prazno ili sadržavati protivničku figura, ali da pritom to polje nije na listi napadačkih poteza protivničkih figura, bile to bijele ako je u pitanju crni kralj ili crne ako je u pitanju bijeli.

```
...
else if((Ploca.StanjePartije == StanjePartije.Normalno) &&
(this.odigraniPotezi == 0) && (item.xRed == this.PoljeOdFigure.xRed) &&
(((Ploca.SvaPolja[this.PoljeOdFigure.xRed, 0].FiguraNaPolju != null) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 0].FiguraNaPolju.odigraniPotezi ==
0) && (item.yStupac == 2) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 1].FiguraNaPolju == null) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 2].FiguraNaPolju == null) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 3].FiguraNaPolju == null)) ||
((Ploca.SvaPolja[this.PoljeOdFigure.xRed, 7].FiguraNaPolju != null) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 7].FiguraNaPolju.odigraniPotezi ==
0) && (item.yStupac == 6) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 5].FiguraNaPolju == null) &&
(Ploca.SvaPolja[this.PoljeOdFigure.xRed, 6].FiguraNaPolju == null))))
                {
            legalniPotezi.Add(new ObicanIliPosebanPotez(item, true, false));
                }
            }
        }
```

Nakon što dodamo obične kraljeve poteze odnosno njegove legalne kretnje za jedno polje u bilo kojem ovisno o poziciji dopuštenom smjeru, treba ispitati je li rokada u danoj poziciji legalna. Budući da rokada zahtjeva određeni broj uvjeta prije nego što je izvediva, te uvjete treba ispitati. Prvo ispitujemo da je stanje partije normalno to jest da kralj nije u šahu.

Rokada nije izvediv i legalan potez ako je kralj pod udarom. Uz to, kralj i top s kojim se rokira, bilo s lijeve ili s desne strane, ne smije biti pomaknut nijednom otpočetka partije.



Slika 12: Duga rokada (1) i kratka rokada (2)

Također kod duge rokade potrebno je ispitati da su polja u istom redu kao i kralj, a u stupcima pod indeksima 1, 2 i 3 prazna odnosno da se između lijevog topa i kralja ne nalazi nijedna figura. Isto radimo i s poljima koji su u stupcu 5 i 6 kako bismo omogućili kratku rokadu.

Slika 13: Prikaz duge rokade kod crnog i kratke rokade kod bijelog

6. Zaključak

Proces samostalnog razvoja vlastite računalne igrice zahtjeva podosta vremena i truda, ali u konačnici posvećenost i pažnja upućena na detalje rezultira kvalitetnim završnim proizvodom no još važnije znanjem koje je nemjerljivo u usporedbi s onim s početka izrade i dragocjenim iskustvom hvatanja u koštac s dugotrajnim projektom. Važno je napomenuti da iako je Unity uvelike olakšao provedbu određenih dijelova rada, proces promišljanja o problemima koji su se u nekim trenucima činili prevelikima i, zbog limitiranog poznavanja određenih sfera, nesavladivim, na koncu su se premostile uloženim trudom i željom za učenjem.

U radu sam se osvrnuo na najrelevantnije karakteristike alata koji su dali najveći obol izradi ovog projekta, ali sam se i dotakao procesa planiranja i promišljanja o implementaciji koja je iziskivala više pripreme od same provedbe. Nadalje, opisao sam igru što sam najbolje mogao u što kraćim crtama, kako bi čitatelju bilo što lakše spojiti teorijski dio šaha s praktičnim. Razvoj same igre u prva dva potpoglavlja pokazao se relativno jednostavnim i izravnim što je ponajviše zasluga intuitivnosti Unity-a.

Međutim u drugoj polovici razvoja stvari postaju kompleksnije te je šah za kojeg je potrebno nekoliko minuta za naučiti ga igrati, a mnogo dulje za postati vješt u njemu, pokazao zašto je podosta kompliciraniji nego što se to naoko čini. Krenuo sam od opisa upravljačkog aspekta te provjeravanje konstante situacije na ploči, a zatim sam se pozabavio konačnim krajem partije odnosno matom što bih izdvojio kao najteži dio rada. Kretanje figura bile su podosta intuitivne međutim kad sam počinjao uračunavati međudjelovanje tih figura i u konačnici njihov utjecaj na ograničavanje kretnji protivničkog kralja, shvatio sam da ovo nije samo igra u kojem se figure miču proizvoljno u okvirima svojih domena i uloga, već funkcionira kao svojevrsna mreža u kojem je svaki dio povezan sa svakim drugim dijelom ploče te ovisno o poziciji odabir poteza se širi i sužava.

Shodno tome, jednom kad su ti problemi bili svladani, implementirani i riješeni, došao sam do rezultata koji sam želio kad sam kretao s ovim projektom. Iskustvo konkretnog rada i praktične primjene objektno orijentiranog programiranja otpočetka studiranja predstavljalo mi je želju te sam zadovoljan da je kulminiralo ovakvim završnim projektom nakon tri godine preddiplomskog studija.

7. Popis literature

7.1. Literatura korištena za izvor informacija

[1] „What is Unity? Everything you need to know“, Android Authority, 20-ožu-2021. [Online]. Dostupno na: https://www.androidauthority.com/what-is-unity-1131558/ [Pristupljeno: 12-srp-2021]

[2] „Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There“, freeCodeCamp.org, 13-velj-2020. [Online]. Dostupno na: https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/ [Pristupljeno: 13-srp-2021]

[3] U. Technologies, „Unity - Manual: Unity User Manual 2020.3 (LTS)“. [Online]. Dostupno na: https://docs.unity3d.com/Manual/ [Pristupljeno: 15-srp-2021]

[4] therealjohn, „Visual Studio Tools for Unity“. [Online]. Dostupno na: https://docs.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity [Pristupljeno: 17-srp-2021]

[5] „šah | Hrvatska enciklopedija“. [Online]. Dostupno na: https://www.enciklopedija.hr/Natuknica.aspx?ID=59291 [Pristupljeno: 21-srp-2021]

[6] Pravila šaha FIDE. [Online]. Dostupno na: http://crochess.com/dokumenti/pdf/pravila_saha_fide_2009_hr.pdf [Pristupljeno: 21-srp-2021]

[7] „Šah – Wikipedija“. [Online]. Dostupno na: https://hr.wikipedia.org/wiki/%C5%A0ah [Pristupljeno: 21-srp-2021]

[8] U. Technologies, „Unity - Manual: Scenes“. [Online]. Dostupno na: https://docs.unity3d.com/Manual/CreatingScenes.html [Pristupljeno: 24-srp-2021]

[9] U. Technologies, „Unity - Scripting API: EventSystem“. [Online]. Dostupno na: https://docs.unity3d.com/2018.1/Documentation/ScriptReference/EventSystems.EventSystem.html [Pristupljeno: 24-srp-2021]

[10] „Canvas | Unity UI | 1.0.0“. [Online]. Dostupno na: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html [Pristupljeno: 24-srp-2021]

[11] U. Technologies, „Unity - Scripting API: Image“. [Online]. Dostupno na: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Image.html [Pristupljeno: 27-srp-2021]

[12] U. Technologies, „Unity - Scripting API: Sprite“. [Online]. Dostupno na: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/Sprite.html [Pristupljeno: 27-srp-2021]

[13] U. Technologies, „Unity - Manual: The Hierarchy window“. [Online]. Dostupno na: https://docs.unity3d.com/Manual/Hierarchy.html [Pristupljeno: 30-srp-2021]

[14] user, „Unity UI – Auto scaling grid layout code“, Just a Pixel. [Online]. Dostupno na: http://www.justapixel.co.uk/2014/12/08/ugui-auto-scaling-grid-layout-code/ [Pristupljeno: 02-lip-2021]

[15] „Raw Image vs Image? - Unity Answers“. [Online]. Dostupno na: https://answers.unity.com/questions/1070280/raw-image-vs-image.html [Pristupljeno: 03-kol-2021]

[16] „Online Image Resizer - Crop, Resize & Compress Images, Photos and Pictures for FREE“. [Online]. Dostupno na: https://resizeimage.net/ [Pristupljeno: 10-svi-2021]

[17] „Sharpen image online“, PineTools. [Online]. Dostupno na: https://pinetools.com/sharpen-image [Pristupljeno: 10-svi-2021]

[18] U. Technologies, „Unity - Scripting API: MonoBehaviour“. [Online]. Dostupno na: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html [Pristupljeno: 10-kol-2021]

[19] U. Technologies, „Unity - Scripting API: MonoBehaviour.Start()“. [Online]. Dostupno na: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html [Pristupljeno: 10-kol-2021]

[20] U. Technologies, „Unity - Scripting API: IPointerDownHandler“. [Online]. Dostupno na: https://docs.unity3d.com/2019.1/Documentation/ScriptReference/EventSystems.IPointerDownHandler.html [Pristupljeno: 10-kol-2021]

[21] U. Technologies, „Unity - Scripting API: Texture“. [Online]. Dostupno na: https://docs.unity3d.com/ScriptReference/Texture.html [Pristupljeno: 11-kol-2021]

[22] U. Technologies, „Unity - Scripting API: Object.DestroyImmediate“. [Online]. Dostupno na: https://docs.unity3d.com/ScriptReference/Object.DestroyImmediate.html [Pristupljeno: 11-kol-2021]

7.2. Literatura korištena u programskom kodu

[23] „Unity Chess Game Tutorial - YouTube“. [Online]. Dostupno na: https://www.youtube.com/playlist?list=PLXV-vjyZiT4b7WGjgiqMy422AVyMaigl1 [Pristupljeno: 02-ožu-2021]

[24] „Unity + Chess Tutorials - YouTube“. [Online]. Dostupno na: https://www.youtube.com/playlist?list=PLmc6GPFDyfw-xZ-hHEGLnV5yRF7_eM5w3 [Pristupljeno: 04-tra-2021]

[25] „Programming Chess - YouTube“. [Online]. Dostupno na: https://www.youtube.com/playlist?list=PLRb3gax7xN_UhbTqKUkDCQaNsYFSvF9Nc [Pristupljeno: 18-svi-2021]

[26] „Full Games - YouTube“. [Online]. Dostupno na: https://www.youtube.com/playlist?list=PLMQr0t2lnabATVpcjT0X_dWCvxawlLuEY [Pristupljeno: 15-ožu-2021]

[27] „Complete Chess Game Tutorial - YouTube“. [Online]. Dostupno na: https://www.youtube.com/playlist?list=PLmcbjnHce7SeAUFouc3X9zqXxiPbCz8Zp [Pristupljeno: 27-tra-2021]

8. Popis slika

9. Popis korištenih resursa

9.1. Korišteni programski alati

1. Unity [Online]. Dostupno na: https://unity3d.com/get-unity/download [Pristupljeno: 05-vel-2021]
2. Microsoft Visual Studio [Online]. Dostupno na: https://visualstudio.microsoft.com/downloads/ [Pristupljeno: 16-vel-2021]
3. „Online Image Resizer - Crop, Resize & Compress Images, Photos and Pictures for FREE“. [Online]. Dostupno na: https://resizeimage.net/ [Pristupljeno: 10-svi-2021]
4. [17] „Sharpen image online“, PineTools. [Online]. Dostupno na: https://pinetools.com/sharpen-image [Pristupljeno: 10-svi-2021]

9.2. Korišteni „Assets“-i

1. „Auto scaling grid layout code“ [Online]. Dostupno na: http://www.justapixel.co.uk/2014/12/08/ugui-auto-scaling-grid-layout-code/ [Pristupljeno: 02-lip-2021]