Razvoj 3D igre koristeći Unity okruženje

Nöthig, Nikola

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

NIKOLA NÖTHIG

RAZVOJ 3D IGRE KORISTEĆI UNITY OKRUŽENJE

Završni rad

Pula, rujan, 2023. godine

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

NIKOLA NÖTHIG

RAZVOJ 3D IGRE KORISTEĆI UNITY OKRUŽENJE

Završni rad

JMB: 0303096784, redoviti student
Studijski smjer: Sveučilišni preddiplomski studij Računarstva

Predmet: Inženjerska grafika i konstruiranje
Znanstveno područje: Tehničke znanosti
Znanstveno polje: Računarstvo
Mentor: doc.dr. sc.tech Marko Kršulja

Pula, rujan, 2023. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Nikola Nöthig, kandidat za prvostupnika, smjera Računarstva ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, Nikola Nöthig dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Razvoj 3D igre koristeći Unity okruženje, koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

Sadržaj

1. Uvod

1.1. Hipoteza

Hipoteza je da se uporabom Unity programa može kreirati jednostavna 3D igra "šah" s prikladnim vizualnim efektima te računalnom logikom koja omogućava funkcionalnost sukladno usporedbi s tržišnim zahtjevima.

1.2. Predmet istraživanja

Predmet istraživanja je Unity program i programski jezik c# u segmentu primjene logičkog i vizualnog sadržaja radi potrebe izrade igre za razvoj logičkog razmišljanja. Problem istraživanja je kreiranje vizualnih efekata i logistike za konkurentno stvaranje sadržaja koje zadovoljava logičko razmišljanje. Sama uporaba softvera Unity te C# u jednoj instanci nadopunjavanja i zajedničkog funkcioniranja predstavlja određene izazove.

1.3. Problemi istraživanja

Problemi mogu nastati zbog performansi, odnosno pisanja neučinkovitog ili ne optimiziranog koda koji tada dovodi do lošijih performansi same aplikacije. Također napredno programiranje grafike u Unityju može biti izazovno. Programiranje shadera i potpuna upotreba cjevovoda za renderiranje može zahtijevati dublje razumijevanje od onoga što se nudi odmah.

Unity ima značajnu krivulju učenja, posebno za pojedince koji su novi u razvoju igara ili programiranju. Shvaćanje temeljnih koncepata stroja, zajedno sa složenošću C# programiranja, može biti zastrašujuće za početnike.

1.4. Ciljevi i struktura rada

- Prvi cilj rada je kreacija osnovne 3D šahovske igre unutar Unity okruženja gdje će svaka figura imati slobodu kretanja po ploči bez ikakvih ograničenja - apsolutna sloboda svakog poteza.

- Kao drugi cilj postavit ću si zadatak definiranja koji je tim na potezu te definiranja specifičnih pravila kretanja za svaku figuru, osiguravajući da svaka

figura slijedi tradicionalna šahovska pravila kretanja. Odnosno izradit ćemo logiku kretanja svih specifičnih figura pojedino.

- Treći i završni cilj obuhvaća implementaciju specijaliziranih poteza poput "en passant" i rokade. Također, optimizirat ću logiku igre za situacije kada se kralj nalazi u šahu, uvesti ću mehaniku šahmata te dodati korisničko sučelje koje obavještava igrača o pobjedi.

1.5. Metodologija rada

U procesu izrade 3D šahovske igre, primjenjuje se složena metodologija rada koja kombinira nekoliko pristupa. Prije svega, koristi se eksperimentalna metodologija jer se kroz različite faze razvoja neprestano testiraju i optimiziraju dijelovi koda. Osim toga, uključen je i modelirajući pristup s obzirom na to da se radi o programskoj aplikaciji koja služi kao dinamički model šahovske igre.

Tijekom razvoja projekta, svaka iteracija igre prolazi detaljnu analizu i evaluaciju. Ova konstantna analiza i usporedba različitih verzija predstavljaju metodu sustavnog promatranja. Uz upotrebu metode promatranja, kroz sve faze razvoja, dokumentiraju se sve promjene, nailazeći izazovi i njihova rješenja, pružajući uvid u razvojni put i omogućavajući dublje razumijevanje procesa.

2. Povijest Unitya i programskog jezika C#

Osnovan 2004., Unity je započeo svoj razvoj primarno služeći se Mac OS X. Do 2007. proširio je svoj opseg na Windows i ušao u mobilnu sferu do 2010. Njegova predanost širenju horizonata bila je očita kada je predstavio izvorne alate za razvoj 2D igara u 2013. Izdanje Unity 5 2014. hvaljeno je zbog naprednog mehanizma za renderiranje, postavljajući nove standarde za vizualnu vjernost u igrama. Osim igara, Unity je također napravio prodor u druge industrije poput filmske i automobilske. Svjedočanstvo njegovog trajnog uspjeha i vizionarskog vodstva, Unity Technologies je napravio svoj istaknuti debi na Njujorškoj burzi 2020., najavljujući novu fazu u svom rastu.

S druge strane, programski jezik C# se pojavio 2000. godine kao Microsoftov odgovor na sve veće zahtjeve modernog programiranja. Ovaj robustan i svestran jezik brzo je postao sinonim za učinkovit razvoj softvera, uključujući transformativne značajke kao što su LINQ i async/await podrška. Njegova prilagodljivost učinila ga je omiljenim među programerima za mnoštvo aplikacija, od weba do softvera za stolna računala. Odabir Unityja da integrira Mono .NET runtime u svoje temelje za skriptiranje pokazao se ključnim, postavljajući C# kao najistaknutiji skriptni jezik za mnoštvo programera. Ovo uparivanje kombiniralo je jednostavnost Unityjevog pogona sa snagom C#, što je kulminiralo u skladnom partnerstvu koje nastavlja utjecati i oblikovati krajolik razvoja igara.

2.1. Primjena u industriji

Unity, izvorno razvijen za dizajn igara, neprimjetno se integrirao u razne industrije zahvaljujući svom svestranom i intuitivnom skupu alata. U području filma i animacije, pomaže u vizualizaciji i renderiranju u stvarnom vremenu za kratke animirane filmove. Automobilski sektor koristi Unity za virtualnu izradu prototipa vozila i AI obuku za autonomnu vožnju. U arhitekturi i graditeljstvu profesionalci stvaraju virtualna uputstva za projekte, pružajući klijentima sveobuhvatne preglede.

2.2. Primjena u razvoju igara

Unity se čvrsto uspostavio kao temeljan u razvoju igara, nudeći alate koji služe i malim indie programerima koliko i velikim kompanijama. Njegova snaga leži u njegovoj sposobnosti da pruži korisničko sučelje u kombinaciji sa snažnim mogućnostima renderiranja, što ga čini prikladnim za izradu svega, od jednostavnih 2D igara do zamršenih 3D naslova. Unityjeva podrška za više platformi osigurava da se igre mogu implementirati na mnoštvo uređaja, od računala i konzola do mobitela i VR hardvera. Dodatno, njegov asset store i ogromna online zajednica pružaju neprocjenjive resurse i podršku, olakšavajući lakši proces razvoja. Fleksibilnost, skalabilnost i opsežan set alata Unity učinili su ga najboljim izborom za programere igara diljem svijeta.

2.3. Filmovi i animacije

U filmskoj i animiranoj industriji Unity igra višestruku ulogu. Koristi se za vizualizaciju, omogućujući filmašima da mapiraju scene prije stvarnog snimanja. Sa svojim mogućnostima renderiranja u stvarnom vremenu, Unity ubrzava stvaranje animiranih sekvenci, osiguravajući brzu iteraciju i vizualno usavršavanje. Osim toga, alati poput sustava virtualne kamere omogućuju dinamičnu kompoziciju scene, premošćujući razmak između tradicionalnog snimanja filmova i digitalnih okruženja. Stoga Unity poboljšava tijek rada kinematografske produkcije, pružajući učinkovitost i fleksibilnost kreatorima.

2.4. Korisničko sučelje

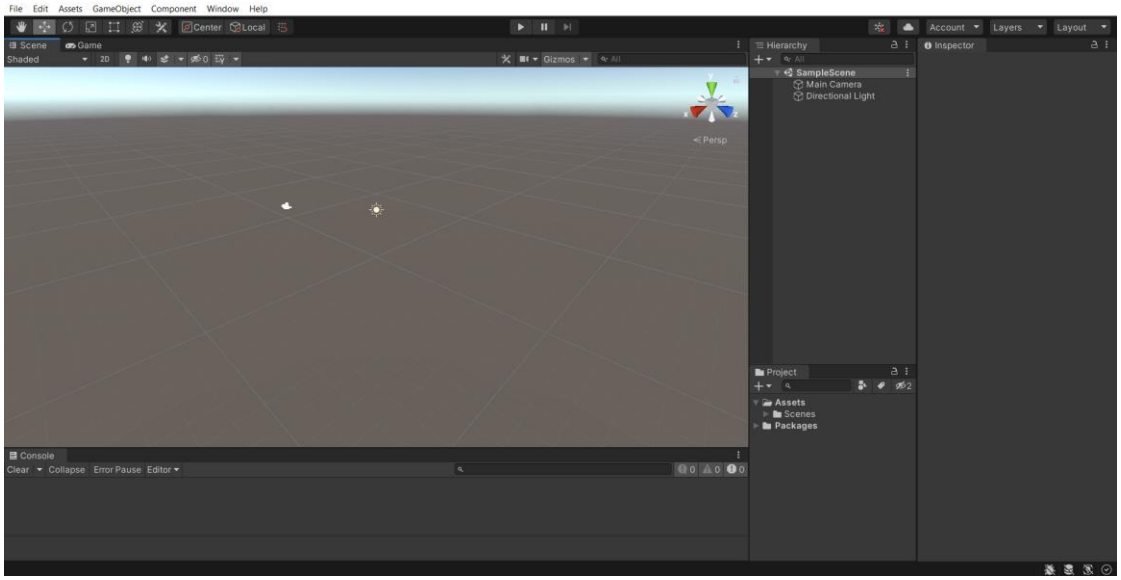
Korisničko sučelje (UI) u bilo kojem softveru, uključujući Unity, odnosi se na prostor i alate predstavljene korisniku za interakciju s programom, aplikacijama ili internet stranicom. Služi kao most između korisničkih naredbi i funkcionalnosti softvera.
U Unityju, korisničko sučelje je sveobuhvatan i prilagodljiv radni prostor koji integrira više panela i prozora. U svojoj srži, Unity UI se sastoji od prozora Scene, gdje programeri mogu manipulirati objektima igre u 3D prostoru, prozora Game, za pregled igre, prozora Hiearchy, s popisom svih objekata igre u sceni, prozora Projekt, koje prikazuje sredstva (assets) dostupna za korištenje. Inspektor, koji detaljno opisuje svojstva odabrane stavke i alatna traka na vrhu, s osnovnim alatima i kontrolama igranja.

Unity korisničko sučelje također dolazi sa prozorom Console, koji je neprocjenjiv alat za programere. Prikazuje poruke, upozorenja i pogreške koje se javljaju tijekom uređivanja i pokretanja igre. To uključuje sve, od pogrešaka u skripti i pojedinosti o iznimkama do prilagođenih dnevnika otklanjanja pogrešaka koje je izradio razvojni programer. Proučavanjem poruka u konzoli, programeri mogu identificirati probleme u svom projektu, čineći uklanjanje pogrešaka i optimizaciju učinkovitijim procesom. To je bitna komponenta u Unityju, koja pomaže programerima da osiguraju da njihove igre rade glatko i da nemaju problema povezanih s kodom.

Ovakav izgled korisničkog sučelja osigurava kreatorima cjelokupni pogled i kontrolu nad procesom razvoja igre, poboljšavajući učinkovitost i preciznost. Programer na zahtjev može lako i detaljno proučiti svaki element.

2.5. Početna scena u Unityu

Slika 1: Početni zaslon u Unityu



Izvor: Prikaz autora iz aplikacije Unity

Prilikom otvaranja programa u prvo što je prikazano u prozoru Scene je Zadana glavna kamera postavljena je okrenuta prema ishodištu. Ova kamera predstavlja igračevo gledište kada igra počne. U Sceni su također mreža, linije osi te usmjereno svjetlo, ono utječe na sve objekte u sceni i baca sjene.

Slika 2: Prikaz prozora Game

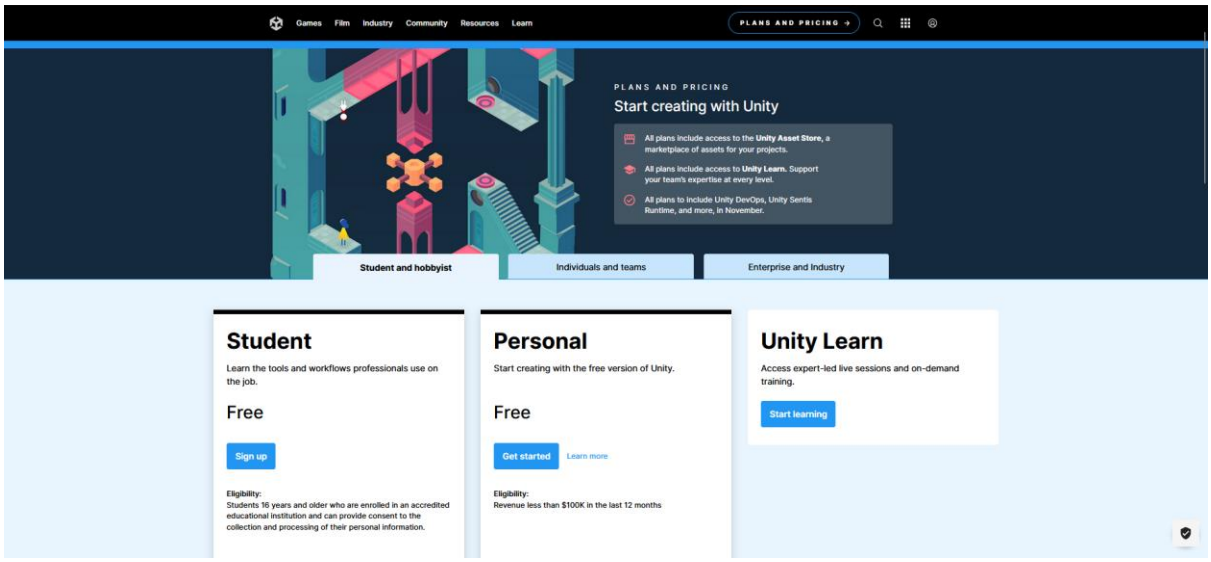
Izvor: Prikaz autora iz aplikacije Unity

U prozoru Game korisnici vide pregled u stvarnom vremenu kako će igra izgledati kada se igra. Prikazuje točku gledišta aktivne kamere u sceni. Važni elementi uključuju padajući izbornik razlučivosti i omjera širine i visine, koji osiguravaju da igra izgleda dobro na različitim zaslonima te kontrole pokretanja igre na vrhu.

2.6. Unity web stranica i preuzimanje Unity okruženje

Unity Technologies nudi raznolik broj planova i programa koji zadovoljavaju širok raspon potreba, od pojedinačnih studenata i hobista do profesionalnih programera i velikih poduzeća. Svaki je plan pomno osmišljen kako bi pružio potrebne alate, resurse i podršku, omogućujući korisnicima stvaranje, razvoj i objavljivanje impresivnog i interaktivnog 3D sadržaja u raznim industrijama.

Slika 3: Prikaz besplatnih planova koje nudi Unity Technologies



Izvor: https://unity.com/pricing#plans-student-and-hobbyist

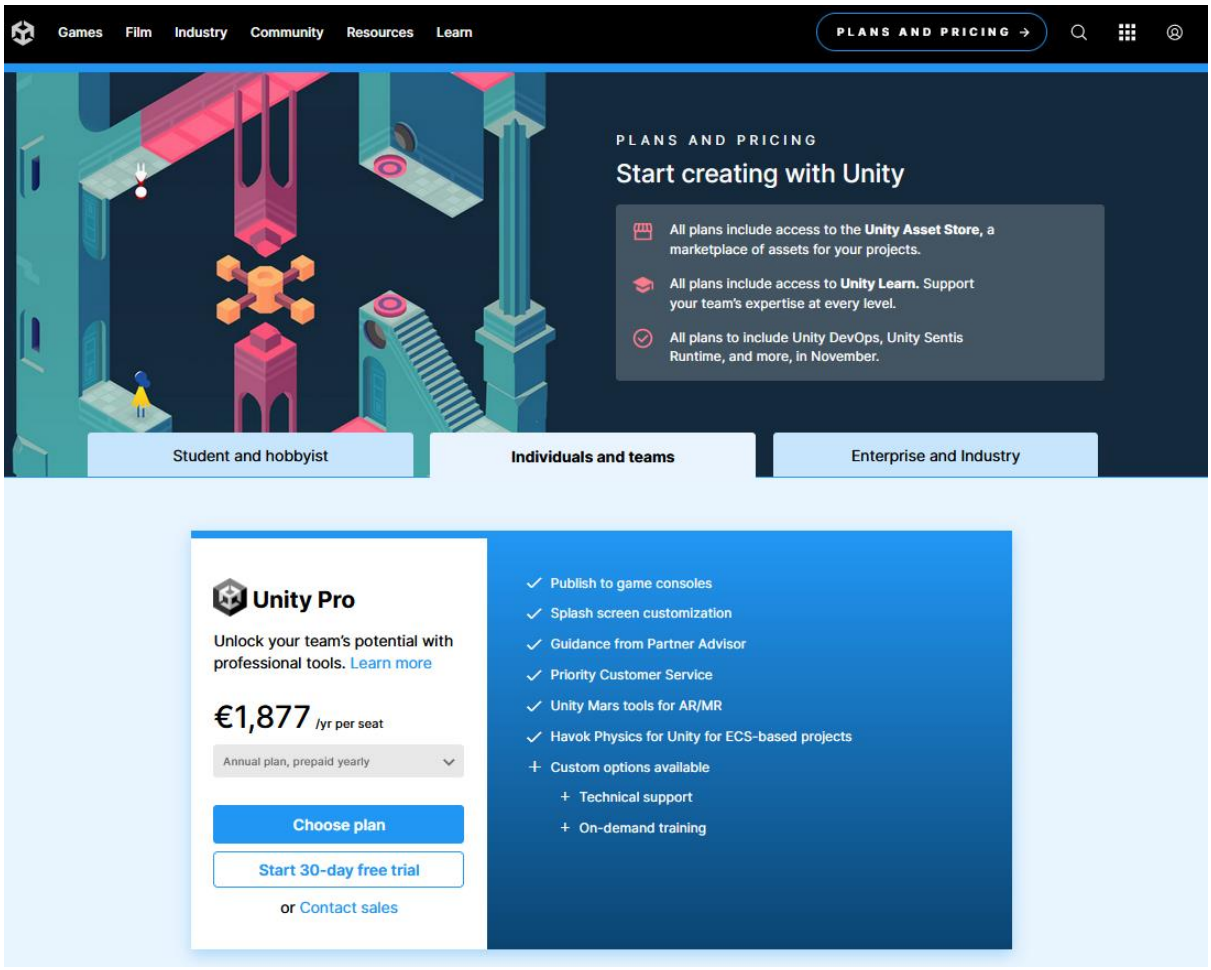
Unityjev studentski plan posebno je skrojen kako bi se prilagodio individualnim potrebama studenata. Omogućuje pristup platformi Unity zajedno s mnoštvom dodatnih resursa za učenje, olakšavajući studentima kultiviranje i poboljšanje njihovih vještina u razvoju igara. Ovaj je plan obično dostupan besplatno studentima koji ispunjavaju uvjete, čime se potiče okruženje za učenje i razvoj u Unityju bez ikakvih financijskih ograničenja.

Personl plan koji nudi Unity je pristupačna verzija platforme, dizajnirana za pojedince koji su krenuli u razvoj igara te za hobiste i mala poduzeća. Ovaj plan daje pristup

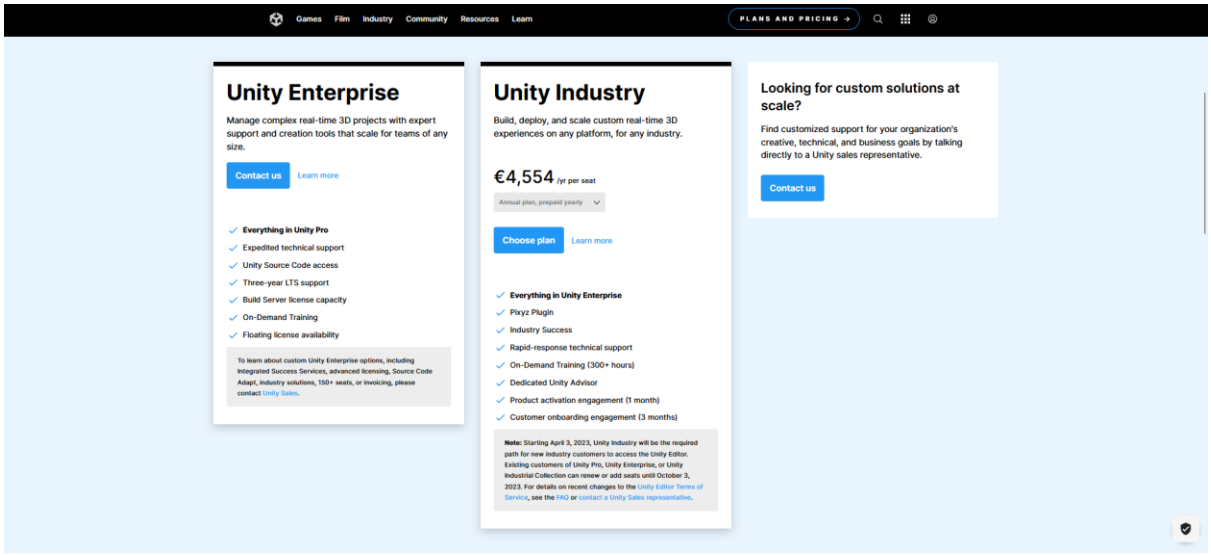
temeljnoj platformi i većini njezinih značajki, dopuštajući korisnicima da razvijaju i objavljuju svoje igre i interaktivni sadržaj. Iako može imati neka ograničenja u usporedbi s premium planovima, služi kao idealno polazište za osobe koje tek započinju učenje i rad u Unityju ili one koji rade na projektima manjeg opsega.

Unity Learn je namjenska platforma za učenje koja nudi niz tutorijala, tečajeva i lekcija koje pomažu korisnicima različitih razina znanja da poboljšaju svoje Unity i vještine razvoja igara. Uključuje i besplatan sadržaj i premium pretplatu za pristup naprednijim resursima.

Slike 4,5: Prikaz platnih planova koje nudi Unity Technologies



Izvor: https://unity.com/pricing#plans-individualsand-teams



Izvor: https://unity.com/pricing#plans-enterprise

Unity Pro dizajniran je za profesionalne programere i veće timove, pruža sveobuhvatan skup naprednih značajki, dodatne usluge u oblaku, poboljšanu podršku i alate za suradnju. Pogodan je za one koji rade na većim i složenijim projektima. Cijena licence je €170 mjesečno po licenci ili €1,877 godišnje.

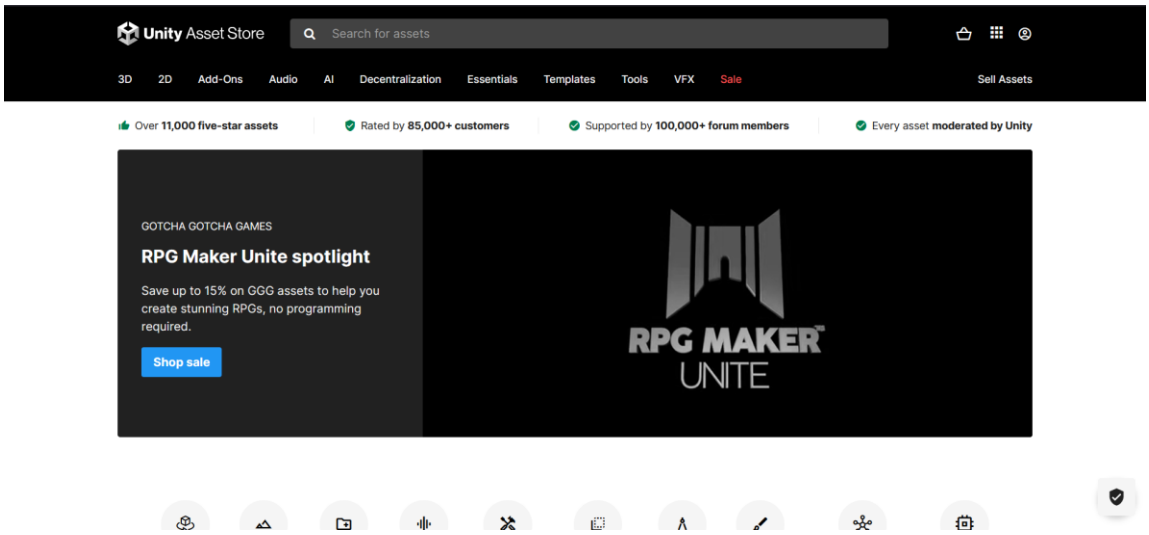
Unity Enterprise opslužuje organizacije i tvrtke sa značajnim razvojnim potrebama. Uključuje sve značajke Unity Pro, s dodanim uslugama, podrškom i opcijama prilagodbe za prilagodbu velikim projektima i jedinstvenim organizacijskim zahtjevima.

Planovi Unity Industry osmišljeni su tako da služe industrijama izvan područja igara, nudeći specijalizirana rješenja za područja kao što su automobilska industrija, film, arhitektura, inženjerstvo i građevinarstvo. Ovi planovi pružaju alate, značajke i podršku specifične za industriju, omogućujući profesionalcima stvaranje zadivljujućih 3D iskustava u stvarnom vremenu. Cijena ove licence €414 mjesečno po licenci, ili €4,554 godišnje.

2.7. Unity Asset Store

Unity Asset Store digitalno je tržište integrirano unutar platforme Unity, koje nudi široku zbirku sredstava, alata i usluga za programere i kreatore igara. U rasponu od 3D modela, animacija i tekstura do potpunih predložaka projekata i proširenja za uređivače, Asset Store služi kao repozitorij gdje programeri mogu kupovati, prodavati ili pronaći besplatne resurse kako bi ubrzali svoj proces razvoja. Ne samo da pomaže u ubrzavanju vremenskih okvira projekta, već također olakšava suradnju među Unity zajednicom, jer kreatori iz cijelog svijeta dijele svoju stručnost i kreacije.

Slika 6: Prikaz web stranice Unity Asset Store

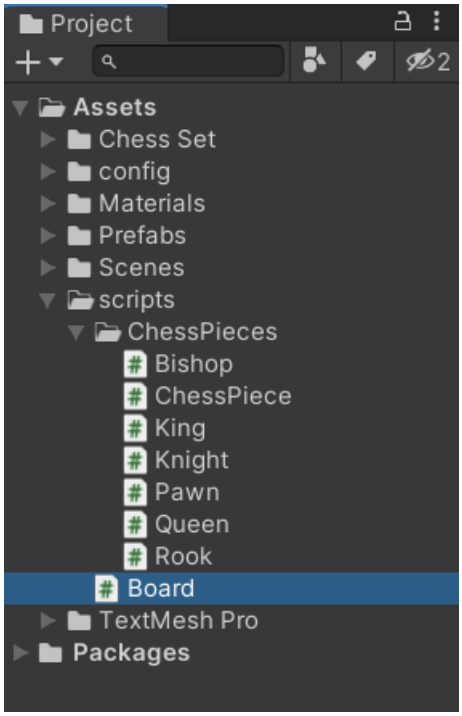


Izvor: https://assetstore.unity.com/

3. Izrada početne verzije 3D igre „šah“

Kod početne verzije prvo je bilo bitno razmisliti o logici šahovske polče, odnosno na koji način ću kreirati virtualnu ploču na kojoj će se nalaziti igraće figure. Koristeći Unity izradio sam novu mapu „scripts“ koja će sadržavati glavnu skriptu „board“ te ostale skripte koje će biti potrebne za pomicanje figura. Skripte za figure će se nalaziti u podmapi ChessPieces. Skripta Board će se baviti generiranjem igrače ploče dimenzija 8x8.

Slika 7:Izrada skripte Board i podmape ChessPieces te skripti za svaku figuru



Izvor: Prikaz autora iz aplikacije Unity

U prozoru Project Dodane su C# skripte koje će služiti kao logika cjelokupne igrice „šah“

Slika 8: Pisanje koda za početnu logiku

```
using System;
using UnityEngine;

0 references
public class Board : MonoBehaviour
{
    [Header("Art")]
    [SerializeField] private Material tileMaterial;
    [SerializeField] private float tileSize = 1.0f;
    [SerializeField] private float yOffset = 0.2f;
    [SerializeField] private Vector3 boardCenter = Vector3.zero;

    //logic
    private const int TILE_COUNT_X = 8;
    private const int TILE_COUNT_Y = 8;
    private GameObject[,] tiles;
    private Camera currentCamera;
    private Vector2Int currentHover;
    private Vector3 bounds;

    0 references
    private void Awake()
    {
        GenerateGrid(tileSize, TILE_COUNT_X, TILE_COUNT_Y);
    }
    0 references
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Na slici je prikazan kod pisan C# programskim jezikoom.

Na slici su varijable namijenjene za kontrolu umjetnosti i nekih funkcija kroz sam Unity editor. Također su napisane logičke varijable koje će se koristiti za kreiranje same table i postavljanje figura u budućnosti.

Slika 9: Kreiranje funkcije koja generira jedno polje

```
78      private GameObject GenerateSingleTile(float tileSize, int x, int y)
79      {
80          GameObject tileObject = new GameObject(string.Format("X:{0},Y:{1}", x, y));
81          tileObject.transform.parent = transform;
82
83          Mesh mesh = new Mesh();
84          tileObject.AddComponent<MeshFilter>().mesh = mesh;
85          tileObject.AddComponent<MeshRenderer>().material = tileMaterial;
86
87          Vector3[] vertices = new Vector3[4];
88          vertices[0] = new Vector3(x * tileSize, yOffset, y * tileSize) - bounds;
89          vertices[1] = new Vector3(x * tileSize, yOffset, (y + 1) * tileSize) - bounds;
90          vertices[2] = new Vector3((x + 1) * tileSize, yOffset, y * tileSize) - bounds;
91          vertices[3] = new Vector3((x + 1) * tileSize, yOffset, (y + 1) * tileSize) - bounds;
92
93          int[] tris = new int[] { 0, 1, 2, 1, 3, 2 };
94          mesh.vertices = vertices;
95          mesh.triangles = tris;
96          mesh.RecalculateNormals();
97
98          tileObject.layer = LayerMask.NameToLayer("Tile");
99          tileObject.AddComponent<BoxCollider>();
00
01          return tileObject;
02      }
03
04      //operations
        1 reference
05      private Vector2Int LookupTileIndex(GameObject hitInfo)
06      {
07          for (int x = 0; x < TILE_COUNT_X; x++)
08              for (int y = 0; y < TILE_COUNT_Y; y++)
09                  if (tiles[x, y] == hitInfo)
10                      return new Vector2Int(x, y);
11
12          return -Vector2Int.one; // -1 -1
13      }
14  }
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Na slici je prikazana funkcija koja nam generira jedno polje veličina x*tileSize te y*tilesize. Polje također sadrži yOffset koji ću koristiti za poravnavanje igraćih figura u odnosu na asset ploče.

Slika 10: Prikaz funkcije za kreiranje koordinatne mreže „grida“ veličine 8x8

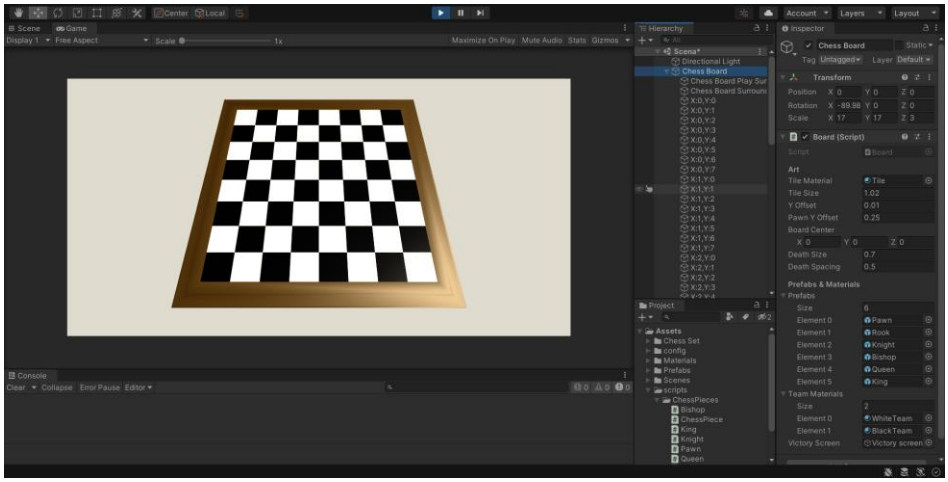
```
1 reference
private void GenerateGrid(float tileSize, int tileCountX, int tileCountY)
{

    yOffset += transform.position.y;
    bounds = new Vector3((tileCountX / 2) * tileSize, 0, (tileCountX / 2) * tileSize) + boardCenter;


    tiles = new GameObject[tileCountX, tileCountY];
    for (int x = 0; x < tileCountX; x++)
        for (int y = 0; y < tileCountY; y++)
            tiles[x, y] = GenerateSingleTile(tileSize, x, y);
}

1 reference
```

Izvor: Prikaz autora iz aplikacije Unity

Metoda GenerateGrid u skripti stvara rešetku (grid) veličine 8x8 za društvenu igru „šah“. Prilagođava svoj položaj na temelju objekta igre i izračunava središte ploče koristeći zadanu veličinu i dimenzije polja. Metoda zatim inicijalizira dvodimenzionalni niz koji predstavlja polja igre. Koristeći ugniježđene petlje, popunjava ovaj niz generiranjem pojedinačnih polja za svaku poziciju mreže, koristeći zasebnu metodu, GenerateSingleTile, za ovaj zadatak. Ova metoda postavlja temelje za vizualni izgled ploče za igru.

Nakon kreiranja virtualne ploče, potrebno je bilo ubaciti assete šahovske ploče te igraćih figura te u Unity programu podesiti veličine asseta prema našim dimenzijama polja.

Slika 11: Prikaz početnih generiranih polja



Izvor: Prikaz autora iz aplikacije Unity

Nakon podešavanja asseta sa našim kodom koristeći dijela koda za „art“ dobili smo početnu igraču ploču koja ima rešetku 8x8 polja. Svako polje ima svoje koordinate označene sa X i Y.

Slika 12:Prikaz update funkcije.

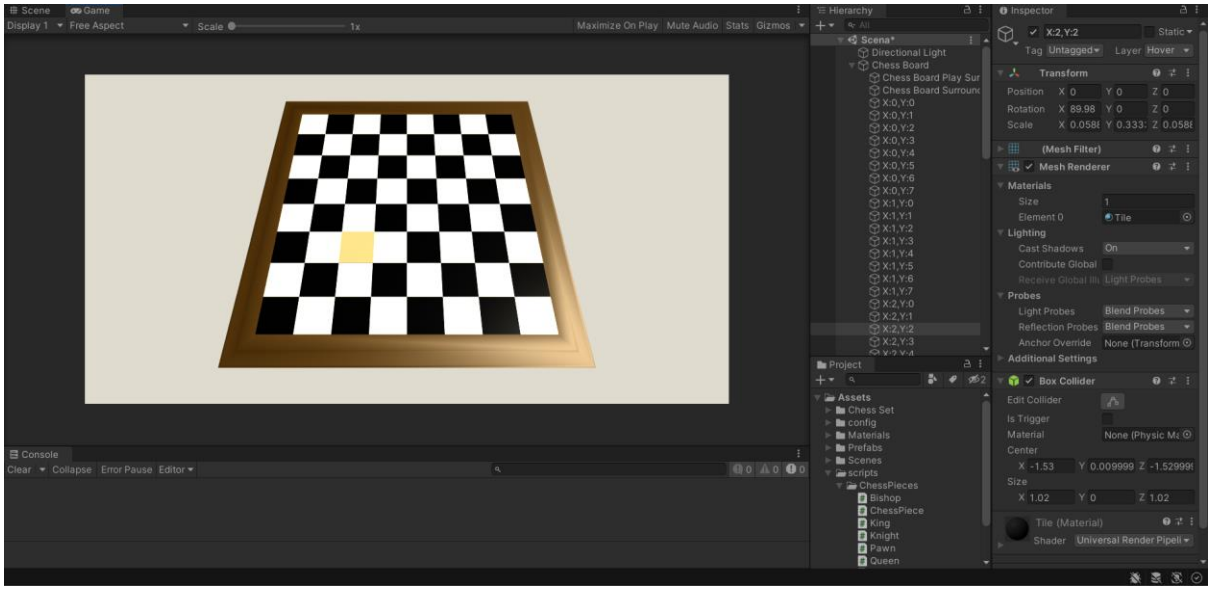
```
private void Update()
{
    if (!currentCamera)
    {
        currentCamera = Camera.main;
        return;
    }

    RaycastHit info;
    Ray ray = currentCamera.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, out info, 100, LayerMask.GetMask("Tile", "Hover")))
    {
        //Get the indexes of tile we hit
        Vector2Int hitPosition = LookupTileIndex(info.transform.gameObject);

        //If we are hovering any tile after not hovering any tile
        if (currentHover == -Vector2Int.one)
        {
            currentHover = hitPosition;
            tiles[hitPosition.x, hitPosition.y].layer = LayerMask.NameToLayer("Hover");
        }

        //if we were already hovernig a tile, change prewius
        if (currentHover != hitPosition)
        {
            tiles[currentHover.x, currentHover.y].layer = LayerMask.NameToLayer("Tile");
            currentHover = hitPosition;
            tiles[currentHover.x, hitPosition.y].layer = LayerMask.NameToLayer("Hover");
        }
    }
    else
    {
        if (currentHover != -Vector2Int.one)
        {
            tiles[currentHover.x, currentHover.y].layer = LayerMask.NameToLayer("Tile");
            currentHover = -Vector2Int.one;
        }
    }
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Metoda Update, koja se obično poziva jednom po okviru u Unity skripti, upravlja interakcijama s poljima igre pomoću pokazivača miša i glavne kamere. Ako trenutna

kamera nije postavljena, zadana je glavna kamera scene. Metoda zatim baca zraku od položaja kamere kroz trenutni položaj miša na zaslonu. Ako ova zraka siječe bilo koji objekt u slojevima "Tile", "Hover", ona određuje indeks polja koja je pogođena. Ako miš prijeđe iznad polja nakon što nije bio ni na jednom polju, trenutni sloj polja mijenja se u "Hover". Ako se miš pomiče s jednog polja na drugo, sloj prethodnog polja mijenja se natrag na „Tile“ temelju određenih uvjeta, a novo polje preuzima sloj "Hover". Logika učinkovito prati i ažurira vizualno stanje polja na temelju interakcija s mišem.

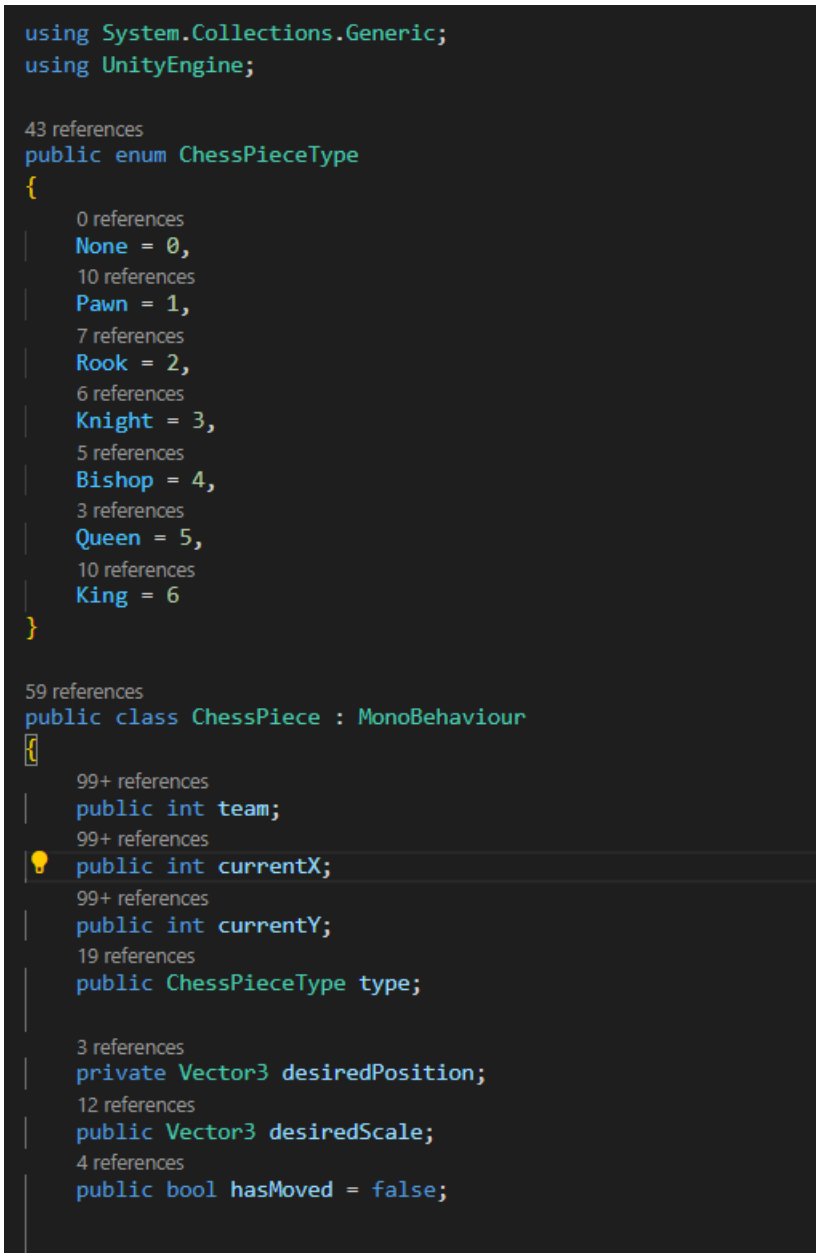
Slika 13: Prikaz pokrenutog koda u Unity programu:



Izvor: Prikaz autora iz aplikacije Unity

Na desnoj strani u prozoru Hierarchy možemo vidjeti da je odabrano poljesa koordinatama X:2 i Y:2. Gore desno u prozoru Inspector možemo vidjeti da se kreirani sloj (layer) promjenio u Hover. U sloju (layer) je podešeno da koristi dodani materijal Hover koji je smeđe žute boje. Na ploči možemo primjetiti da nam je polje na kojoj se nalazi miš promjenila boju.

Slika 14: Dodavanje nove skripte ChessPiece.

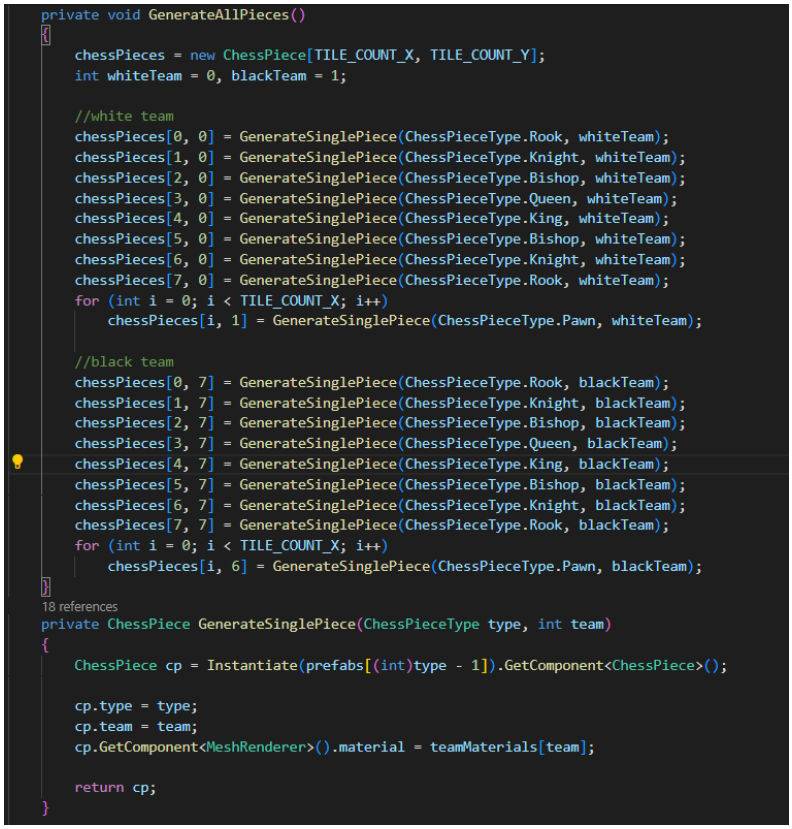


```
using System.Collections.Generic;
using UnityEngine;

43 references
public enum ChessPieceType
{
    0 references
    None = 0,
    10 references
    Pawn = 1,
    7 references
    Rook = 2,
    6 references
    Knight = 3,
    5 references
    Bishop = 4,
    3 references
    Queen = 5,
    10 references
    King = 6
}

59 references
public class ChessPiece : MonoBehaviour
{
    99+ references
    public int team;
    99+ references
    public int currentX;
    99+ references
    public int currentY;
    19 references
    public ChessPieceType type;

    3 references
    private Vector3 desiredPosition;
    12 references
    public Vector3 desiredScale;
    4 references
    public bool hasMoved = false;
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Ovaj isječak koda prvo definira enumeraciju, ChessPieceType, koja navodi sve moguće vrste šahovskih figura. Zatim se definira klasa ChessPiece, koja nasljeđuje Unityjev MonoBehaviour. Ova klasa predstavlja pojedinačne šahovske figure, pohranjujući atribute kao što je momčad figure, njen trenutni položaj na ploči (koristeći currentX i currentY), njen tip, željeni položaj u 3D prostoru, namjenjenu veličinu i zastavu hasMoved za praćenje je li figura pomaknuta u igri.

Slika 15: Dodavanje skripte za generiranje jedne figure, te funkcije za dodavanje svih figura



```
private void GenerateAllPieces()
{
    chessPieces = new ChessPiece[TILE_COUNT_X, TILE_COUNT_Y];
    int whiteTeam = 0, blackTeam = 1;

    //white team
    chessPieces[0, 0] = GenerateSinglePiece(ChessPieceType.Rook, whiteTeam);
    chessPieces[1, 0] = GenerateSinglePiece(ChessPieceType.Knight, whiteTeam);
    chessPieces[2, 0] = GenerateSinglePiece(ChessPieceType.Bishop, whiteTeam);
    chessPieces[3, 0] = GenerateSinglePiece(ChessPieceType.Queen, whiteTeam);
    chessPieces[4, 0] = GenerateSinglePiece(ChessPieceType.King, whiteTeam);
    chessPieces[5, 0] = GenerateSinglePiece(ChessPieceType.Bishop, whiteTeam);
    chessPieces[6, 0] = GenerateSinglePiece(ChessPieceType.Knight, whiteTeam);
    chessPieces[7, 0] = GenerateSinglePiece(ChessPieceType.Rook, whiteTeam);
    for (int i = 0; i < TILE_COUNT_X; i++)
        chessPieces[i, 1] = GenerateSinglePiece(ChessPieceType.Pawn, whiteTeam);

    //black team
    chessPieces[0, 7] = GenerateSinglePiece(ChessPieceType.Rook, blackTeam);
    chessPieces[1, 7] = GenerateSinglePiece(ChessPieceType.Knight, blackTeam);
    chessPieces[2, 7] = GenerateSinglePiece(ChessPieceType.Bishop, blackTeam);
    chessPieces[3, 7] = GenerateSinglePiece(ChessPieceType.Queen, blackTeam);
    chessPieces[4, 7] = GenerateSinglePiece(ChessPieceType.King, blackTeam);
    chessPieces[5, 7] = GenerateSinglePiece(ChessPieceType.Bishop, blackTeam);
    chessPieces[6, 7] = GenerateSinglePiece(ChessPieceType.Knight, blackTeam);
    chessPieces[7, 7] = GenerateSinglePiece(ChessPieceType.Rook, blackTeam);
    for (int i = 0; i < TILE_COUNT_X; i++)
        chessPieces[i, 6] = GenerateSinglePiece(ChessPieceType.Pawn, blackTeam);
}
18 references
private ChessPiece GenerateSinglePiece(ChessPieceType type, int team)
{
    ChessPiece cp = Instantiate(prefabs[(int)type - 1]).GetComponent<ChessPiece>();

    cp.type = type;
    cp.team = team;
    cp.GetComponent<MeshRenderer>().material = teamMaterials[team];

    return cp;
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Metoda GenerateAllPieces inicijalizira dvodimenzionalni niz chessPieces, koji predstavlja figure. Zatim popunjava odgovarajuće pozicije na ploči bijelim i crnim šahovskim figurama ispravnim redoslijedom: topovi, skakači, lovci, dama, kralj i pješaci. To se postiže metodom GenerateSinglePiece.

Metoda GenerateSinglePiece, s obzirom na određenu vrstu šahovske figure i tim (bijeli ili crni), instancira figuru, postavlja njen tip i tim i primjenjuje odgovarajući vizualni materijal na temelju tima. Instancirana figura se zatim vraća da se postavi na ploču.

Također su dodane funkcije za pozicioniranje generiranih šahovskih figura, koje su pozvane u Awake funkciji, te se pomoću te funkcije figure stavljaju u početnu poziciju.

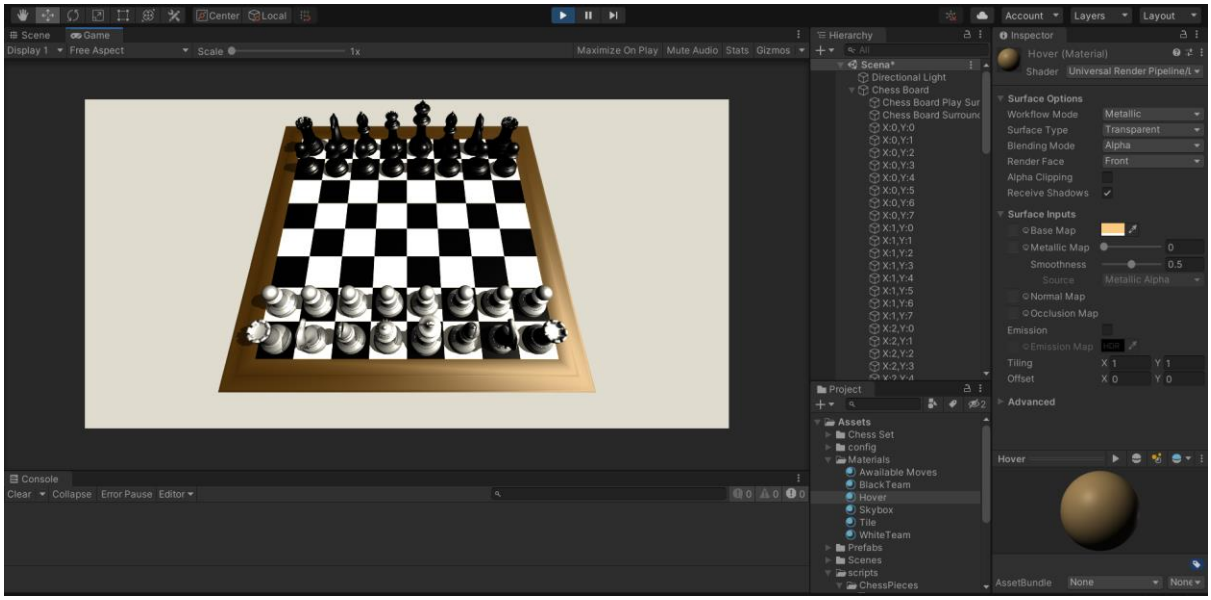
Slika 16: Prikaz koda za pozicioniranje figura

```
private void PositionAllPieces()
{
    for (int x = 0; x < TILE_COUNT_X; x++)
        for (int y = 0; y < TILE_COUNT_Y; y++)
            if (chessPieces[x, y] != null)
                PositionSinglePiece(x, y, true);

}
3 references
private void PositionSinglePiece(int x, int y, bool force = false)
{
    chessPieces[x, y].currentX = x;
    chessPieces[x, y].currentY = y;
    chessPieces[x, y].SetPosition(GetTileCenter(x, y), force);
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Kod je odgovoran za postavljanje početnog rasporeda šahovskih figura na šahovskoj ploči u Unityju. Te će služiti za pomicanje figura po ploči.

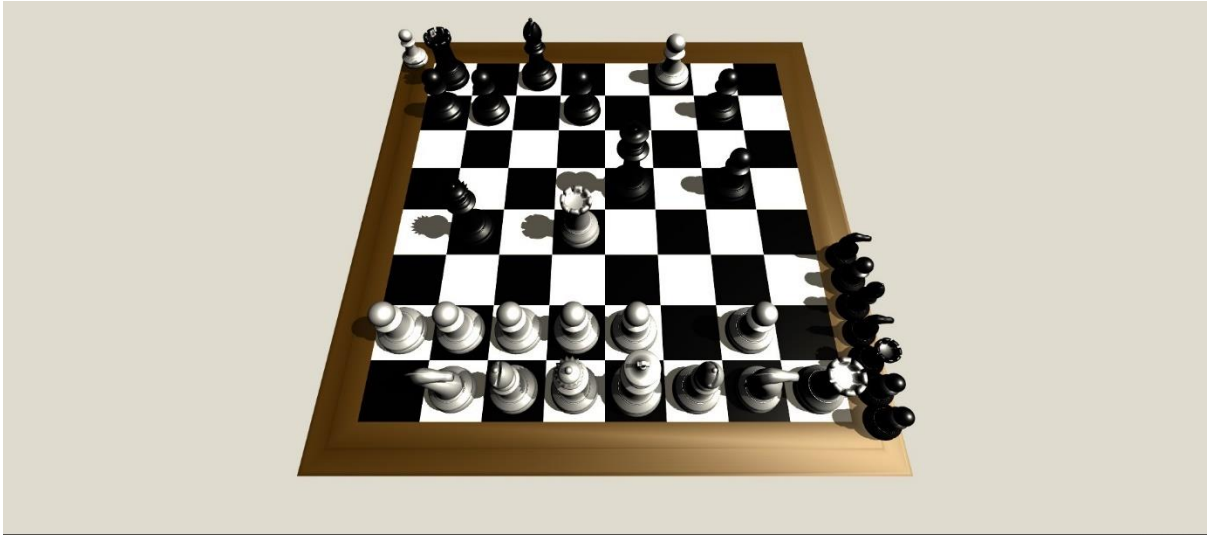
Slika 17: Prikaz generiranih šahovskih figura



Izvor: Prikaz autora iz aplikacije Unity

Zatim se u skripti Board dodala funkcija MoveTo, koja nam je služila za kretanje figura po ploči, te također je kreirana kako bi se riješio problem osvajanja protivničkih figura. U funkciji Update se poziva Boolean funkcija MoveTo za provjeru dali je potez dozvoljen. Hvatanje figure se odvija u Update funkciji, lijevim klikom miša, kada pustimo klik, pokuša pomaći figuru, ako potez nije validan, vraća figuru na početno mjesto. U MoveTo funkciju dodana je funkcionalnost koja se pobrine za događaj kada osvojim protivničku figuru. Tako je kreirana prva verzija igre „šah“ bez redoslijeda i pravila kretanja šahovskih figura.

Slika 18: Prikaz prve verzije kreirane igre



Izvor: Prikaz autora iz aplikacije Unity

4. Izrada logike kretanja figura

Prilikom izrade i pisanja koda za svaku figuru, treba voditi računa o tome kako se svaka figura može kretati po pravilima društvene igre „šah“. Također treba paziti na to da nam figure ne izađu izvan igrače ploče (out of bounds), kako nebi došlo do rušanja igre.

Slika 19: Metoda GetAvailableMoves

```
12 references
public virtual List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
{
    List<Vector2Int> r = new List<Vector2Int>();

    //Test for code (overwritten)
    r.Add(new Vector2Int(6, 3));
    r.Add(new Vector2Int(3, 5));
    r.Add(new Vector2Int(2, 3));
    r.Add(new Vector2Int(2, 4));
    r.Add(new Vector2Int(3, 4));

    return r;
}
```

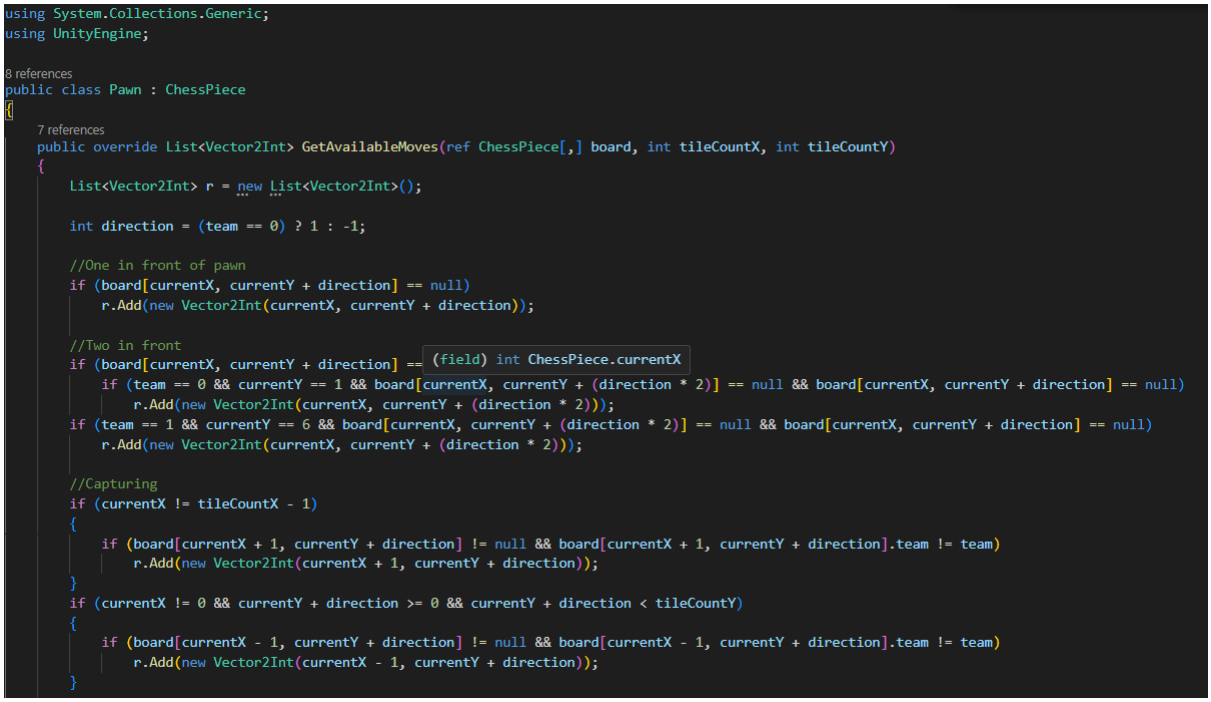
Izvor: Prikaz autora iz aplikacije Visual Studio Code

U skripti ChessPiece dodana je metoda GetAvailableMoves koja vraća popis dostupnih poteza za šahovsku figuru.
Metoda je označena kao virtualna, što znači da je namijenjena nadjačavanju izvedenih klasa. Potrebna je referenca 2D niza ChessPiece objekata (koji predstavljaju šahovsku ploču) i dva cijela broja za dimenzije ploče.
Metoda se poziva u Bord.cs skripti u metodama Update te MoveTo.
Sa skriptom svake figure ćemo nadjačati (override) tu metodu, te na taj način svakoj figuri zadati poteze.

4.1 Izrada koda za kretanje figure pijuna (Pawn)

Prilikom izrade koda za kretanje pijuna, stavio sam mu samo osnovna pravila kretanja radi lakšeg testiranja i dodavanja dodatnog koda kasnije. Pijun se mogao kretati naprijed za jedno mjesto, a ako se nije pomakao sa početne pozicije onda za 2 mjesta. Također pijun samo može jesti dijagonalno.

Slika 20: Prikaz koda klase Pawn

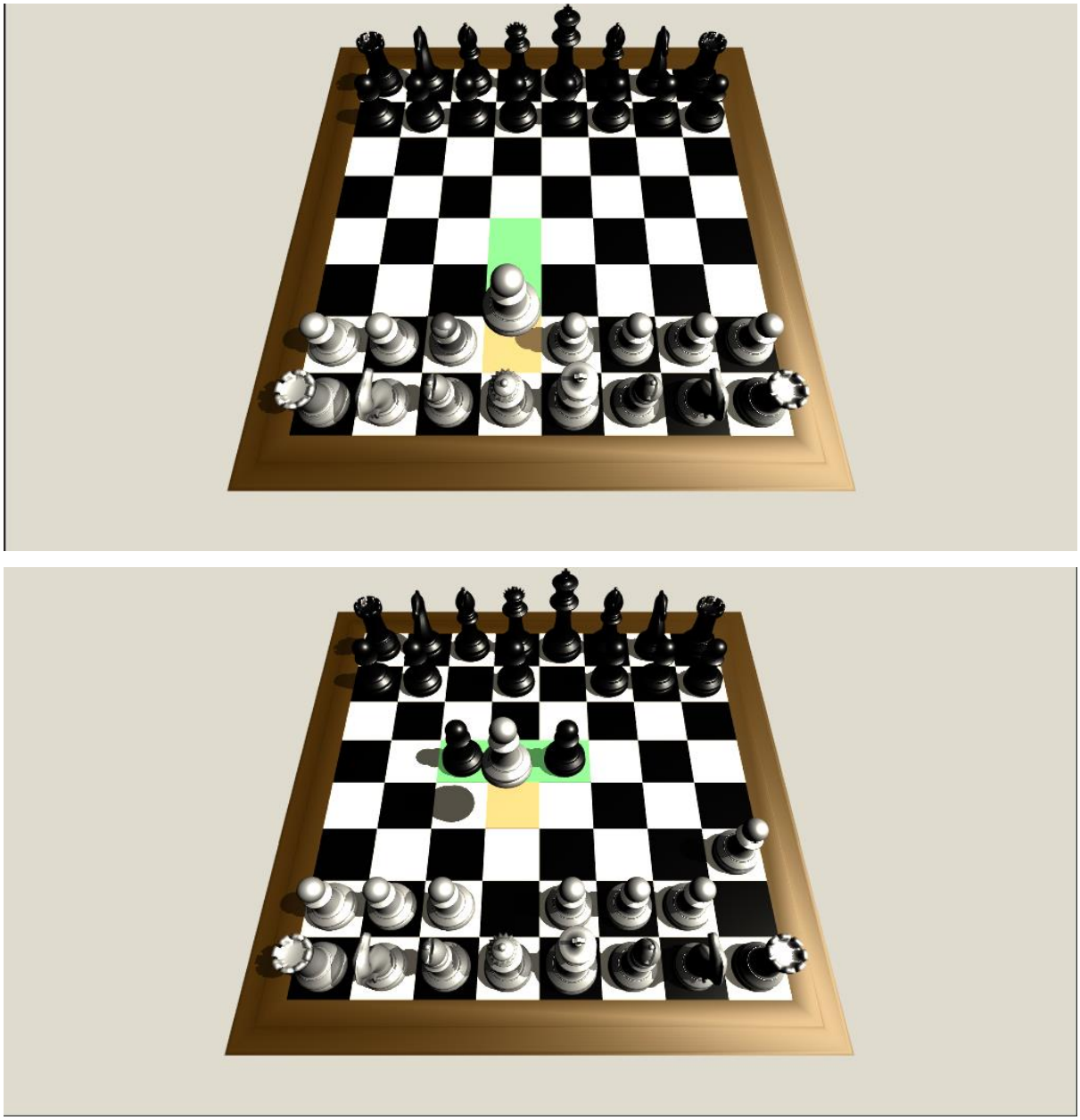


```
using System.Collections.Generic;
using UnityEngine;

8 references
public class Pawn : ChessPiece
{
    7 references
    public override List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        int direction = (team == 0) ? 1 : -1;

        //One in front of pawn
        if (board[currentX, currentY + direction] == null)
            r.Add(new Vector2Int(currentX, currentY + direction));

        //Two in front
        if (board[currentX, currentY + direction] ==  (field) int ChessPiece.currentX
            if (team == 0 && currentY == 1 && board[currentX, currentY + (direction * 2)] == null && board[currentX, currentY + direction] == null)
                r.Add(new Vector2Int(currentX, currentY + (direction * 2)));
        if (team == 1 && currentY == 6 && board[currentX, currentY + (direction * 2)] == null && board[currentX, currentY + direction] == null)
            r.Add(new Vector2Int(currentX, currentY + (direction * 2)));

        //Capturing
        if (currentX != tileCountX - 1)
        {
            if (board[currentX + 1, currentY + direction] != null && board[currentX + 1, currentY + direction].team != team)
                r.Add(new Vector2Int(currentX + 1, currentY + direction));
        }
        if (currentX != 0 && currentY + direction >= 0 && currentY + direction < tileCountY)
        {
            if (board[currentX - 1, currentY + direction] != null && board[currentX - 1, currentY + direction].team != team)
                r.Add(new Vector2Int(currentX - 1, currentY + direction));
        }
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Kod pruža logiku za određivanje dostupnih poteza figure pješaka u šahovskoj igri. Pješaci imaju jedinstvena pravila kretanja u šahu. Smjer kretanja pješaka određuje njegov tim; ako je u timu 0, kreće se prema gore (smjer postavljen na 1), a ako je u timu 1, kreće se prema dolje (smjer postavljen na -1). Pješak se može pomaknuti za jedno polje naprijed ako je prostor neposredno ispred prazan. Ako se nije pomaknuo sa svoje početne pozicije, ima mogućnost pomaknuti se dva polja naprijed, ali samo ako su oba polja ispred prazna. Osim toga, pješaci mogu uhvatiti protivničke figure dijagonalno. Kod provjerava oba dijagonalna smjera (lijevo i desno), a ako postoji protivnička figura u bilo kojem smjeru, ta pozicija postaje dostupan potez. Ova logika uzima u obzir granice ploče i osigurava da pješak ne izađe izvan granica ploče. Rezultat je lista, r, koja bilježi sve važeće poteze na temelju trenutnog položaja pješaka i stanja ploče.

Slika 21, 22: Dozvoljeni potezi figure Pawn



Izvor: Prikaz autora iz aplikacije Unity

Na slikama možemo vidjeti kako nam u igri izgledaju potezi pijuna kada se igra pokrene.

4.2 Izrada koda za kretanje figure kule (Rook)

Prilikom pisanja koda za kretanje figure kule potrebno je definirati logiku kretanja za figuru kule u partiji šaha. Kula se može kretati okomito (gore i dolje) ili vodoravno (lijevo i desno) po šahovskoj ploči, te ne može preskakati figure.

Slika 23: Prikaz koda klase Rook

```
using System.Collections.Generic;
using UnityEngine;

0 references
public class Rook : ChessPiece
{
    7 references
    public override List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        //Up & Down
        for (int i = currentY + 1; i<=7; i++)
        {
            if(board[currentX, i] == null)
                r.Add(new Vector2Int(currentX,i));
            if(board[currentX, i] != null)
            {
                if(board[currentX, i].team != team)
                    r.Add(new Vector2Int(currentX,i));

                break;
            }
        }
        for (int i = currentY - 1; i>=0; i--)
        {
            if(board[currentX, i] == null)
                r.Add(new Vector2Int(currentX,i));
            if(board[currentX, i] != null)
            {
                if(board[currentX, i].team != team)
                    r.Add(new Vector2Int(currentX,i));

                break;
            }
        }

        //Left & Right
        for (int i = currentX + 1; i<=7; i++)
        {
            if(board[i, currentY] == null)
                r.Add(new Vector2Int(i, currentY));
            if(board[i, currentY] != null)
            {
                if(board[i, currentY].team != team)
                    r.Add(new Vector2Int(i, currentY));

                break;
            }
        }
        for (int i = currentX - 1; i>=0; i--)
        {
            if(board[i, currentY] == null)
                r.Add(new Vector2Int(i, currentY));
            if(board[i, currentY] != null)
            {
                if(board[i, currentY].team != team)
                    r.Add(new Vector2Int(i, currentY));

                break;
            }
        }

        return r;
    }
}
```

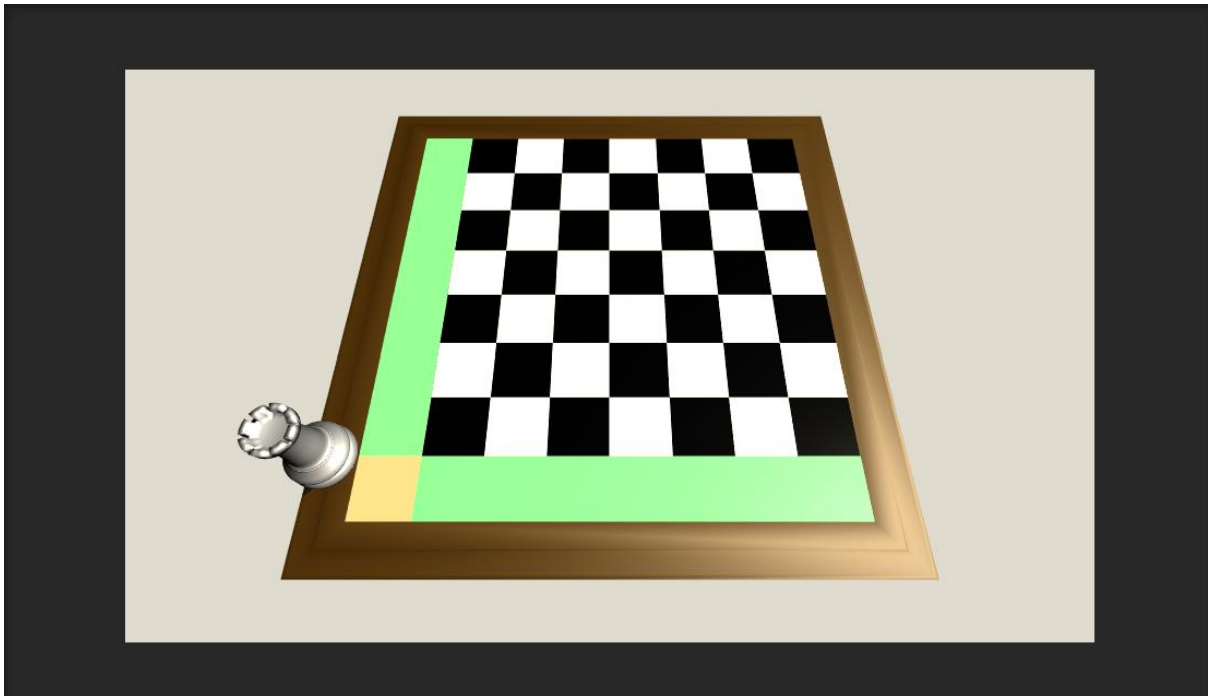
Izvor: Prikaz autora iz aplikacije Visual Studio Code

Metoda GetAvailableMoves izračunava sve moguće poteze kule s njegove trenutne pozicije. Koristi četiri petlje za provjeru svakog smjera: gore, dolje, lijevo i desno. U svakom smjeru, top nastavlja skenirati dok ne naiđe na zauzeto polje ili dok ne dođe do ruba ploče.

Ako naiđe na polje koju zauzima protivnička figura, ta se pozicija dodaje njegovim dostupnim potezima, a skeniranje u tom smjeru se zaustavlja. Ako naiđe na polje koje zauzima figura vlastitog tima, skeniranje u tom smjeru se zaustavlja bez dodavanja te pozicije dostupnim potezima.

Ako je polje prazno, pozicija se dodaje dostupnim potezima i skeniranje se nastavlja. Metoda u konačnici vraća popis, r, koji sadrži sve valjane poteze za top na temelju njegove trenutne pozicije i stanja ploče.

Slika 24: Dozvoljeni potezi figure Rook



Izvor: Prikaz autora iz aplikacije Unity

4.3 Izrada koda za kretanje figure skakača (Knight)

Prilikom pisanja koda za kretanje figure skakača potrebno je definirati logiku kretanja skakača. U šahu je skakač jedinstvena i svestrana figura poznata po potezu kretanja u obliku slova "L", dva polja u jednom smjeru (vodoravno ili okomito) nakon kojih slijedi jedno polje okomito na taj smjer. Ovaj pokret omogućuje skakaču da preskoči druge figure, što ga čini jedinom figurom koja to može učiniti.

Slika 25: Prikaz koda klase Knight

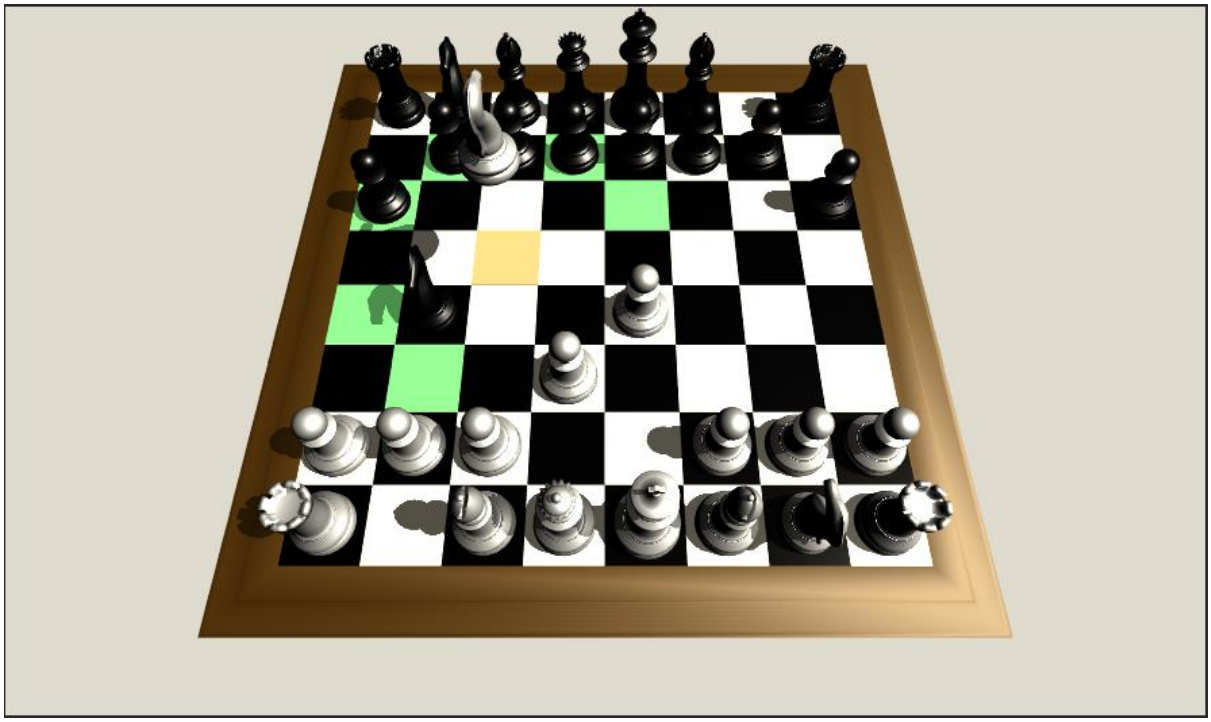
```
using System.Collections.Generic;
using UnityEngine;

0 references
public class Knight : ChessPiece
{
    7 references
    public override List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        // Define all possible relative moves for a knight
        Vector2Int[] possibleMoves =
        {
            new Vector2Int(-1, 2),
            new Vector2Int(1, 2),
            new Vector2Int(-1, -2),
            new Vector2Int(1, -2),
            new Vector2Int(2, 1),
            new Vector2Int(2, -1),
            new Vector2Int(-2, 1),
            new Vector2Int(-2, -1)
        };

        foreach (Vector2Int move in possibleMoves)
        {
            int newX = currentX + move.x;
            int newY = currentY + move.y;

            if (newX >= 0 && newX < tileCountX && newY >= 0 && newY < tileCountY)
            {
                if (board[newX, newY] == null || board[newX, newY].team != team)
                {
                    r.Add(new Vector2Int(newX, newY));
                }
            }
        }

        return r;
    }
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Unutar ove klase, nadjačana metoda nazvana GetAvailableMoves dizajnirana je za određivanje svih potencijalnih poteza skakača na šahovskoj ploči. Mogući relativni

potezi skakača, koji obuhvaćaju tipične L-oblike (kombinacije dvaju polja u jednom smjeru i jednog polja u okomitom smjeru), definirani su kao niz Vector2Int nazvan possibleMoves. Metoda zatim ponavlja te potencijalne poteze. Za svaki mogući potez izračunava novu poziciju skakača na ploči. Ako je izračunata pozicija unutar granica šahovske ploče, a odredišno polje je ili prazna ili zauzeta protivnikovom figurom, potez se smatra ispravnim i dodaje se na listu r. Konačno, metoda vraća popis r koji sadrži sve važeće poteze za skakača.

Slika 26: Dozvoljeni potezi figure skakača



Izvor: Prikaz autora iz aplikacije Unity

4.4 Izrada koda za kretanje figure lovca (Bishop)

Izrada koda za kretanje figure lovca, slična je načinu na koji pijun može uhvatiti protivničke figure dijagonalno. Pošto se lovac može samo dijagonalno kretati po ploči, obje koordinate x i y se moraju istovremeno mijenjati jednaku vrijednost.

Slika 27: Prikaz koda klase Bishop

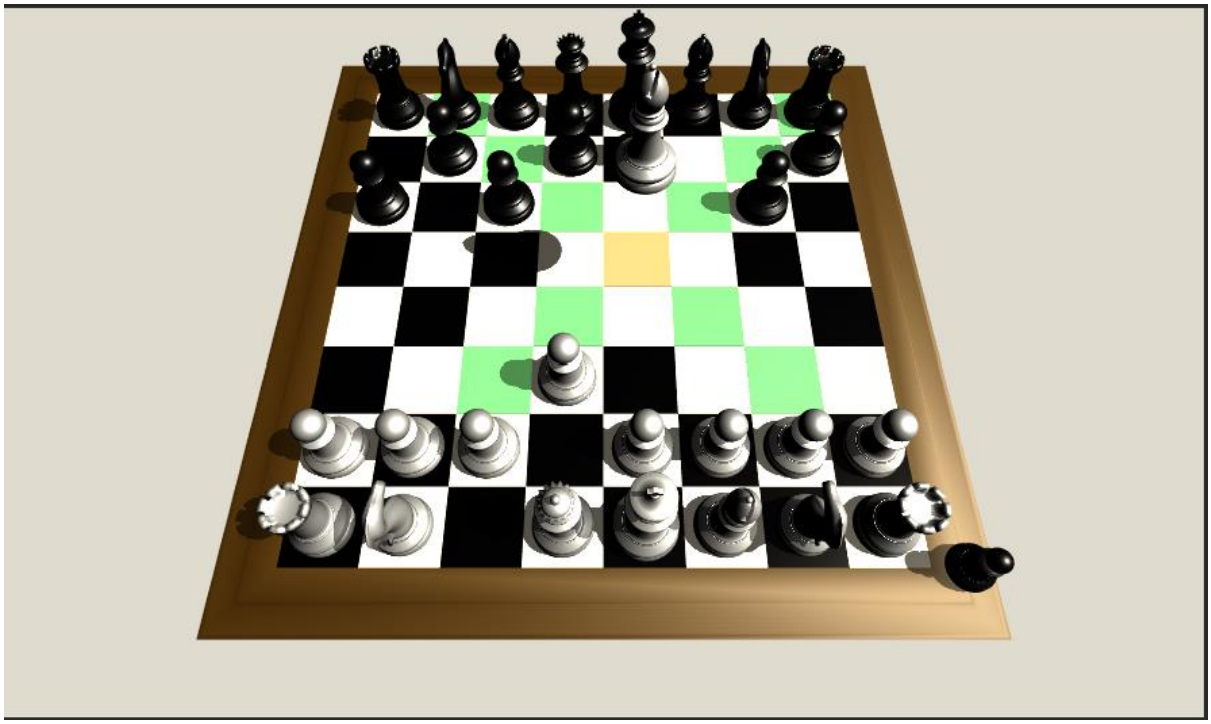
```
using System.Collections.Generic;
using UnityEngine;

0 references
public class Bishop : ChessPiece
{
    7 references
    public override List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        // Define all possible diagonal directions for a bishop: top-right, top-left, bottom-right, bottom-left
        Vector2Int[] directions =
        {
            new Vector2Int(1, 1),
            new Vector2Int(-1, 1),
            new Vector2Int(1, -1),
            new Vector2Int(-1, -1)
        };

        foreach (Vector2Int direction in directions)
        {
            int x = currentX + direction.x;
            int y = currentY + direction.y;

            while (x >= 0 && x < tileCountX && y >= 0 && y < tileCountY)
            {
                if (board[x, y] == null)
                {
                    r.Add(new Vector2Int(x, y));
                }
                else
                {
                    if (board[x, y].team != team)
                    {
                        r.Add(new Vector2Int(x, y));
                    }
                    break; // Stop checking further in this direction when a piece is encountered
                }

                x += direction.x;
                y += direction.y;
            }
        }

        return r;
    }
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Unutar ove klase također je nadjačana metoda GetAvailableMoves kako bi odredila sve moguće poteze koje lovac može učiniti na šahovskoj ploči. Lovčevo kretanje ograničeno je na dijagonalne staze. Koristeći skup unaprijed definiranih dijagonalnih smjerova: gore desno (1, 1), gore lijevo(-1, 1), dolje desno(1, -1), dolje lijevo(-1, -1). Kod ponavlja svaki smjer i kontinuirano provjerava polja u tom smjeru sve dok ne naiđe na protivničku figuru ili dok ne dođe do ruba ploče. Ako figura na koju naiđe

pripada suprotnom timu, njezina se pozicija dodaje na popis dostupnih poteza. Metoda zatim vraća potpuni popis valjanih poteza za lovca.

Slika 28: Dozvoljeni potezi figure lovca



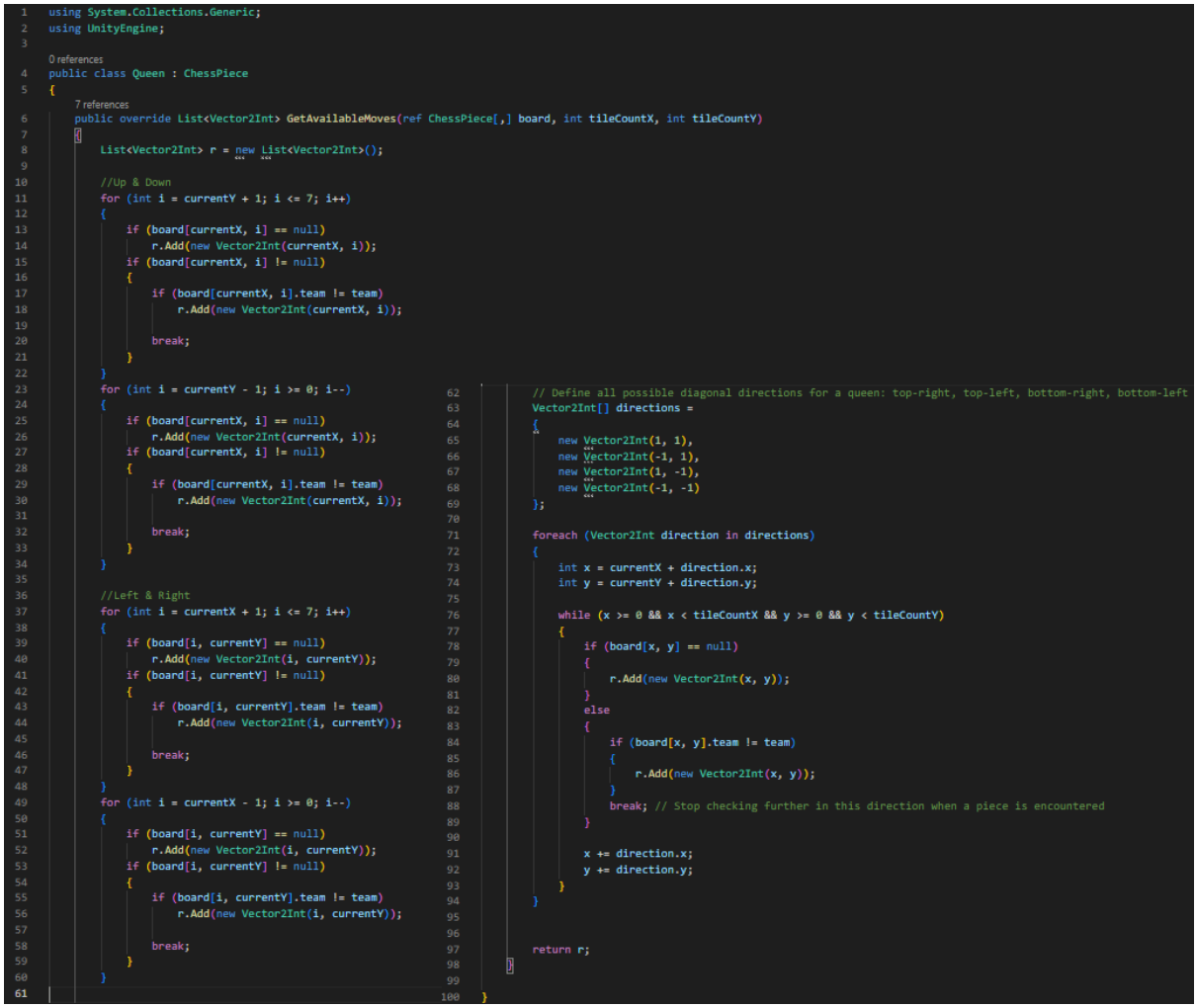
Izvor: Prikaz autora iz aplikacije Unity

4.5 Izrada koda za kretanje figure kraljice (Queen)

Kraljica je najsvestranija figura u šahu. Može se kretati dijagonalno, vodoravno ili okomito za bilo koji broj polja sve dok mu je put slobodan. To čini damu moćnom figurom za kontrolu ploče i napad na protivničke figure.

Zbog načina kretanja figure kraljice sam prilikom izrade koda za kretanje kraljice, iskoristio već postojeće kodove načina kretanja od figura kule i lovca.

Slika 29: Prikaz koda klase Queen

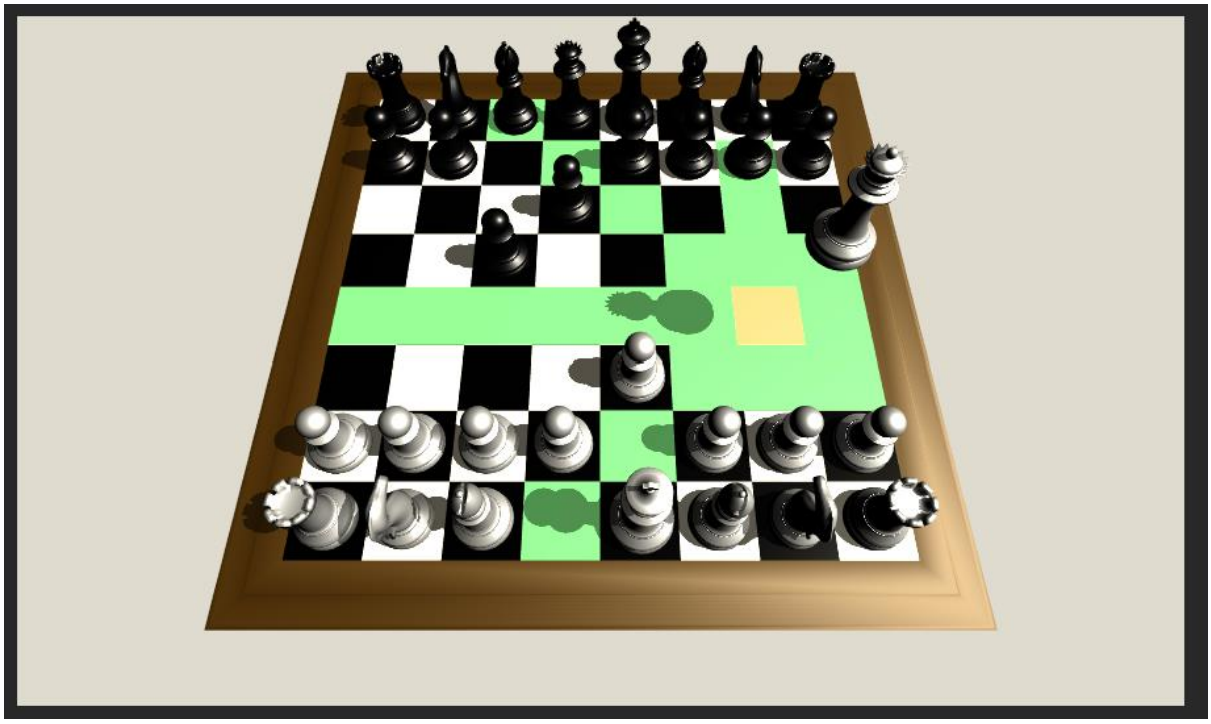


```
using System.Collections.Generic;
using UnityEngine;

0 references
public class Queen : ChessPiece
{
    7 references
    public override List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        //Up & Down
        for (int i = currentY + 1; i <= 7; i++)
        {
            if (board[currentX, i] == null)
                r.Add(new Vector2Int(currentX, i));
            if (board[currentX, i] != null)
            {
                if (board[currentX, i].team != team)
                    r.Add(new Vector2Int(currentX, i));

                break;
            }
        }
        for (int i = currentY - 1; i >= 0; i--)
        {
            if (board[currentX, i] == null)
                r.Add(new Vector2Int(currentX, i));
            if (board[currentX, i] != null)
            {
                if (board[currentX, i].team != team)
                    r.Add(new Vector2Int(currentX, i));

                break;
            }
        }

        //Left & Right
        for (int i = currentX + 1; i <= 7; i++)
        {
            if (board[i, currentY] == null)
                r.Add(new Vector2Int(i, currentY));
            if (board[i, currentY] != null)
            {
                if (board[i, currentY].team != team)
                    r.Add(new Vector2Int(i, currentY));

                break;
            }
        }
        for (int i = currentX - 1; i >= 0; i--)
        {
            if (board[i, currentY] == null)
                r.Add(new Vector2Int(i, currentY));
            if (board[i, currentY] != null)
            {
                if (board[i, currentY].team != team)
                    r.Add(new Vector2Int(i, currentY));

                break;
            }
        }

        // Define all possible diagonal directions for a queen: top-right, top-left, bottom-right, bottom-left
        Vector2Int[] directions =
        {
            new Vector2Int(1, 1),
            new Vector2Int(-1, 1),
            new Vector2Int(1, -1),
            new Vector2Int(-1, -1)
        };

        foreach (Vector2Int direction in directions)
        {
            int x = currentX + direction.x;
            int y = currentY + direction.y;

            while (x >= 0 && x < tileCountX && y >= 0 && y < tileCountY)
            {
                if (board[x, y] == null)
                {
                    r.Add(new Vector2Int(x, y));
                }
                else
                {
                    if (board[x, y].team != team)
                    {
                        r.Add(new Vector2Int(x, y));
                    }
                    break; // Stop checking further in this direction when a piece is encountered
                }

                x += direction.x;
                y += direction.y;
            }
        }


        return r;
    }
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Pošto je kod kombinacija kodova iz klasa Rook i Bishop se ponaša identično kao u klasi Rook za vodoravne i okomite poteze, te u klasi Bishop za sve diagonalne poteze. Tako dodaje sve moguće dostupne poteze na listu mogućih poteza.

Slika30: Dozvoljeni potezi figure kraljice



Izvor: Prikaz autora iz aplikacije Unity

4.6 Izrada koda za kretanje figure kralja (King)

U šahu je kralj vitalna figura s jedinstvenom sposobnošću kretanja. Može se pomaknuti za jedno polje u bilo kojem smjeru: vodoravno, okomito ili dijagonalno. Ova sposobnost omogućuje kralju da bude prilagodljiv i svestran na šahovskoj ploči, pri čemu je svaki potez kritičan za ishod partije.

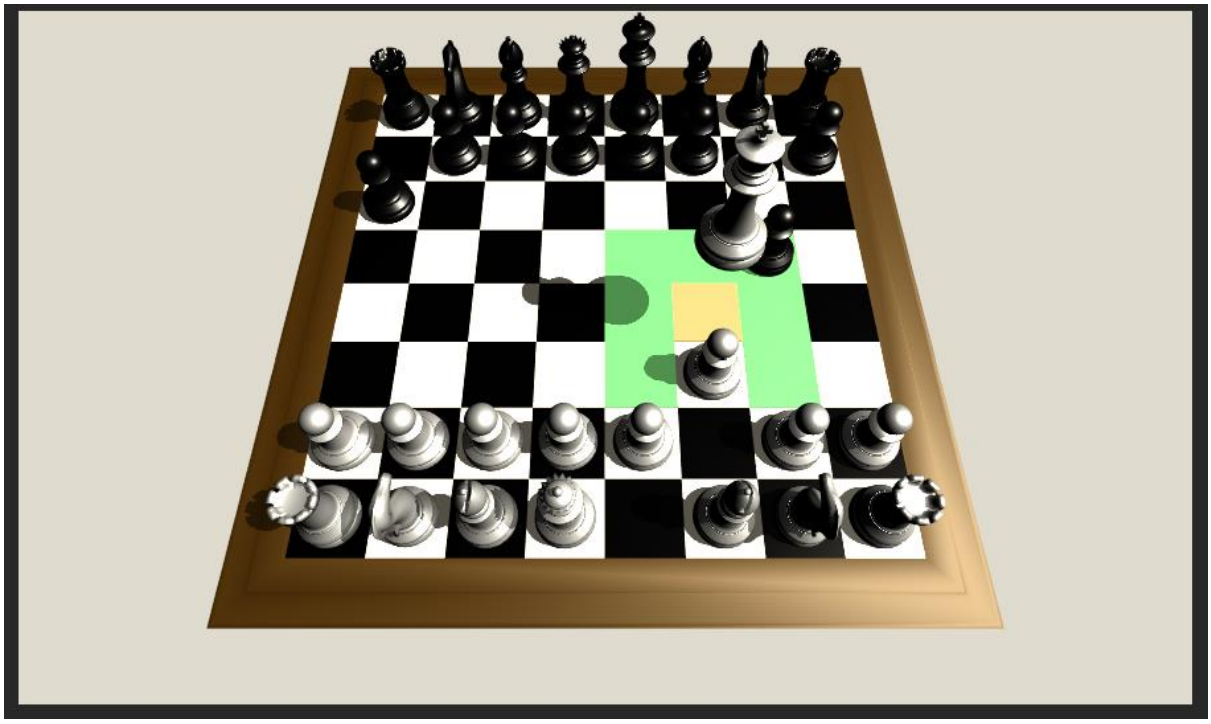
Slika 31: Prikaz koda klase King

```
using System.Collections.Generic;
using UnityEngine;

2 references
public class King : ChessPiece
{
    7 references
    public override List<Vector2Int> GetAvailableMoves(ref ChessPiece[,] board, int tileCountX, int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        int[] directions = { -1, 0, 1 };

        foreach (int i in directions)
        {
            foreach (int j in directions)
            {
                if (i == 0 && j == 0)
                    continue; // Skip the current position

                int newX = currentX + i;
                int newY = currentY + j;

                if (newX >= 0 && newX < tileCountX && newY >= 0 && newY < tileCountY)
                {
                    if (board[newX, newY] == null || board[newX, newY].team != team)
                        r.Add(new Vector2Int(newX, newY));
                }
            }
        }
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Kod identificira i pohranjuje položaje određenih susjednih polja u popis r tipa Vector2Int. Koristeći dvije ugniježđene foreach petlje, kod ispituje svih osam susjednih strana oko dane točke (currentX, currentY). Preskače središnje polje (gdje su i i j 0), provjerava je li svako susjedno polje unutar definiranih granica i provjerava je li polje prazno ili je zauzeto drugim timom. Ako su ti uvjeti ispunjeni, pozicija se dodaje na listu mogućih poteza.

Slika 32: Dozvoljeni potezi figure kralja



Izvor: Prikaz autora iz aplikacije Unity

4.7 Druga verzija 3D igre „šah“

Nakon nekoliko ponavljanja razmišljanja i testiranja, dovršio sam razvoj druge verzije 3D igre šaha. Svaka šahovska figura ima svoju logiku poteza u zasebnim klasama, čime se osigurava da se mogu kretati samo unutar dopuštenog prostora. Iako je to korak naprijed, nije konačni rezultat i ima prostora za poboljšanje točnosti i usavršavanje ukupnog iskustva igrača.

5. Izrada logike posebnih poteza, te UI za pobjedu

Za treću i konačnu verziju, koja se temelji na radu početne verzije, gdje je primarni fokus bio na uspostavljanju logike šahovske ploče i postavljanju bitnih skripti u Unityju, te se temelji na drugoj verziji, gdje su pojedinačni setovi poteza za svaku figuru implementirani, fokus se pomaknuo na uvođenje posebnih poteza.

To uključuje uvođenje poteza rokade, en-passant te izradu logike za šah i šah mat te također izradu korisničkog sučelja za pobjednički zaslon. Sve temeljne skripte dobro

su uspostavljene u mapi "scripts", s pojedinačnim skriptama smještenim u podmapu ChessPieces, postavljajući čvrstu osnovu za integraciju ovih naprednih značajki i poboljšanja.

5.1 En passant logika

Pregledavajući proces razvoja, vratio sam se u skriptu za pješaka kako bih integrirao dodatni kod prilagođen za potez „en passant“. Ovo poboljšanje osigurava da je logika kretanja pješaka sveobuhvatna i u skladu s tradicionalnim šahovskim pravilima. Istodobno su napravljene izmjene na funkciji MoveTo u skripti board.cs. Usavršavanje ove funkcije povećava sposobnost igre za precizno upravljanje i provjeru valjanosti raznolikog niza šahovskih pokreta i scenarija.

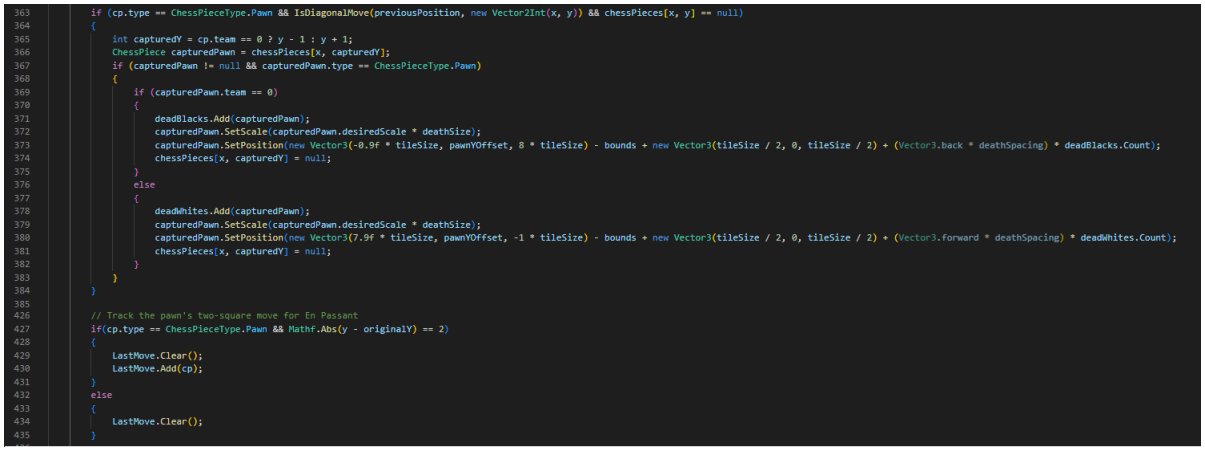
Slika 33: Prikaz dodanog koda u klasu Pawn

```
// En Passant Left
if (currentX != 0 && board[currentX - 1, currentY] != null && board[currentX - 1, currentY].GetType() == typeof(Pawn) && board[currentX - 1, currentY].team != team)
{
    if (board[currentX - 1, currentY].team == 0 && currentY == 3 && Board.LastMove.Contains(board[currentX - 1, currentY]))
        r.Add(new Vector2Int(currentX - 1, currentY + direction));
    else if (board[currentX - 1, currentY].team == 1 && currentY == 4 && Board.LastMove.Contains(board[currentX - 1, currentY]))
        r.Add(new Vector2Int(currentX - 1, currentY + direction));
}
// En Passant Right
if (currentX != tileCountX - 1 && board[currentX + 1, currentY] != null && board[currentX + 1, currentY].GetType() == typeof(Pawn) && board[currentX + 1, currentY].team != team)
{
    if (board[currentX + 1, currentY].team == 0 && currentY == 3 && Board.LastMove.Contains(board[currentX + 1, currentY]))
        r.Add(new Vector2Int(currentX + 1, currentY + direction));
    else if (board[currentX + 1, currentY].team == 1 && currentY == 4 && Board.LastMove.Contains(board[currentX + 1, currentY]))
        r.Add(new Vector2Int(currentX + 1, currentY + direction));
}

return r;
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Kod provjerava je li susjedni pješak protivničke ekipe prisutan i ispunjava li uvjete za hvatanje u prolazu. Ova se podobnost utvrđuje provjerom zadnjeg poteza pješaka. Ako je protivnički pješak upravo napravio potez za dva polja naprijed (kao što je zabilježeno u Board.LastMove), on postaje važeći kandidat za en passant. Kada se dogodi takav scenarij, pozicija za potez u prolazu se izračunava i dodaje na listu r, koja sadrži sve dostupne poteze za pješaka. Na ovaj način, kada igrač odluči napraviti potez, opcija en passant dostupna je ako su ispunjeni uvjeti.

Slika 34: Prikaz dodanog koda u klasu Board u funkciju MoveTo



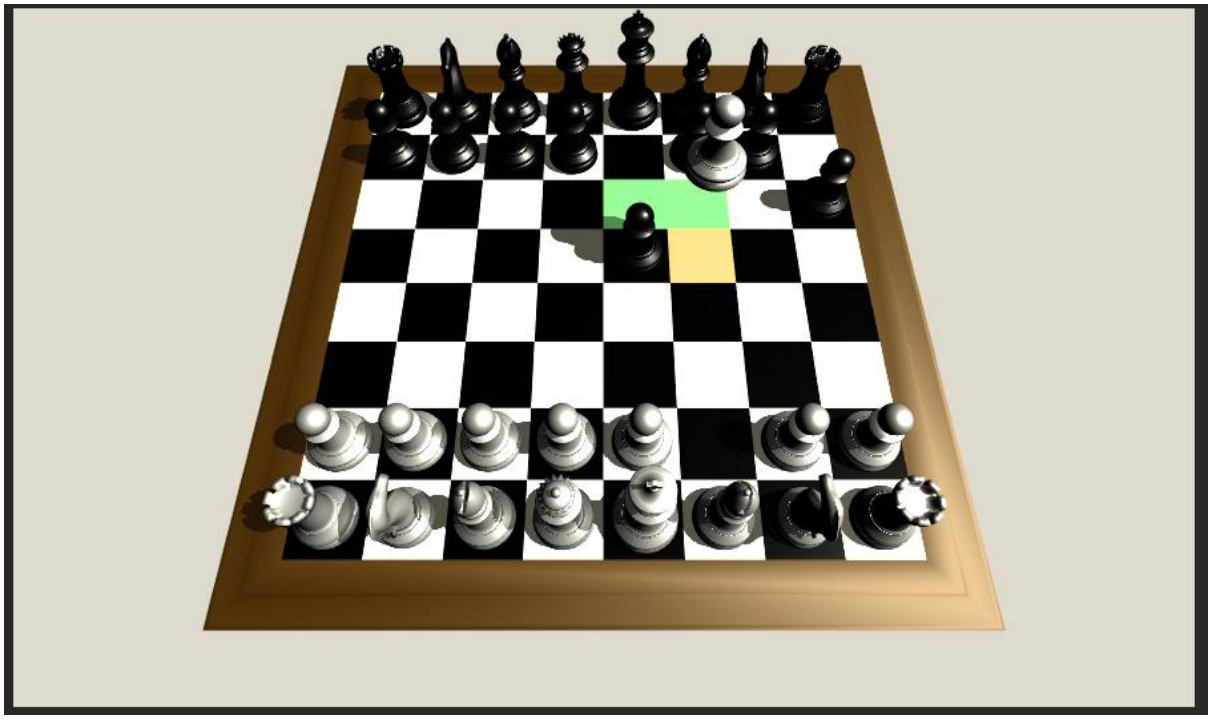
```
363         if (cp.type == ChessPieceType.Pawn && IsDiagonalMove(previousPosition, new Vector2Int(x, y)) && chessPieces[x, y] == null)
364         {
365             int capturedY = cp.team == 0 ? y - 1 : y + 1;
366             ChessPiece capturedPawn = chessPieces[x, capturedY];
367             if (capturedPawn != null && capturedPawn.type == ChessPieceType.Pawn)
368             {
369                 if (capturedPawn.team == 0)
370                 {
371                     deadBlacks.Add(capturedPawn);
372                     capturedPawn.SetScale(capturedPawn.desiredScale * deathSize);
373                     capturedPawn.SetPosition(new Vector3(-0.9f * tileSize, pawnYOffset, 8 * tileSize) - bounds + new Vector3(tileSize / 2, 0, tileSize / 2) + (Vector3.back * deathSpacing) * deadBlacks.Count);
374                     chessPieces[x, capturedY] = null;
375                 }
376                 else
377                 {
378                     deadWhites.Add(capturedPawn);
379                     capturedPawn.SetScale(capturedPawn.desiredScale * deathSize);
380                     capturedPawn.SetPosition(new Vector3(7.9f * tileSize, pawnYOffset, -1 * tileSize) - bounds + new Vector3(tileSize / 2, 0, tileSize / 2) + (Vector3.forward * deathSpacing) * deadWhites.Count);
381                     chessPieces[x, capturedY] = null;
382                 }
383             }
384         }
385 
426         // Track the pawn's two-square move for En Passant
427         if(cp.type == ChessPieceType.Pawn && Mathf.Abs(y - originalY) == 2)
428         {
429             LastMove.Clear();
430             LastMove.Add(cp);
431         }
432         else
433         {
434             LastMove.Clear();
435         }
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Kod provjerava ide li pješak dijagonalno na prazno polje, što je uvjet za en passant potez. Ako je ovaj uvjet ispunjen, izračunava poziciju protivničkog pješaka kojeg treba uhvatiti. Nakon hvatanja, kod prilagođava vizualni prikaz uhvaćenog pješaka i uklanja ga sa šahovske ploče, stavljajući ga među uhvaćene figure.

Dodani kod također prati pješake koji su se pomaknuli dva polja naprijed, što je neophodan uvjet za omogućavanje en-passant poteza u sljedećem potezu. Ako pješak napravi takav potez, dodaje se na popis LastMove. U suprotnom se popis briše kako bi se poništilo praćenje potencijalnih prolaznih uvjeta.

Slika 35: Prikaz en passant poteza u igri

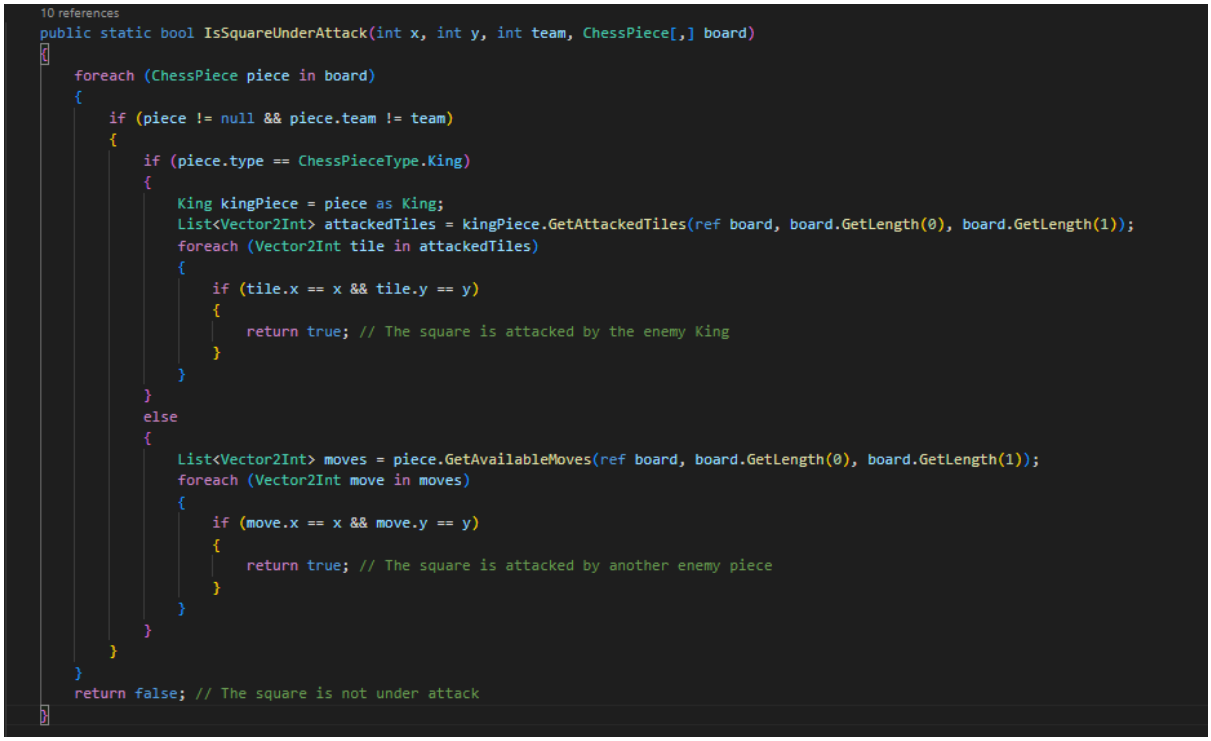


Izvor: Prikaz autora iz aplikacije Unity

5.2 Logika za detekciju napadnutih polja, rokadu, te za detekciju šaha.

Kako bih implementirao logiku rokade, detekcije šaha i napadnutih polja, ponovno sam pregledao skriptu King kako bih integrirao logiku potrebnu za rokadu. Prilagodbe su napravljene kako bi se osiguralo da se kralj i topovi prije toga nisu pomaknuli i da polja kojima kralj prolazi nisu napadnuta, čime se sprječava rokada putem šaha. Štoviše, u skripti Board, izmjene su učinjene u funkciji MoveTo kako bi se uzela u obzir jedinstvena priroda ovog poteza. Također sam ažurirao skriptu ChessPiece, uvodeći nove funkcije za provjeru je li kralj u šahu te koja su polja trenutno napadnuta. Ove su funkcije vitalne u određivanju zakonitosti poteza, osobito kada je kralj pod prijetnjom. Uključio sam još jednu funkciju koja vraća samo poteze koji mogu blokirati napad na kralja. Na kraju, unutar funkcije Update u skripti Board.cs postavljena su ograničenja kako bi se osiguralo da kada je kralj u šahu, igraču je ograničeno koristiti samo poteze koji blokiraju prijetnju ili pomicati kralja, održavajući integritet igre i pridržavajući se utvrđena pravila šaha.

Slika 36: Prikaz funkcije IsSquareUnderAttack

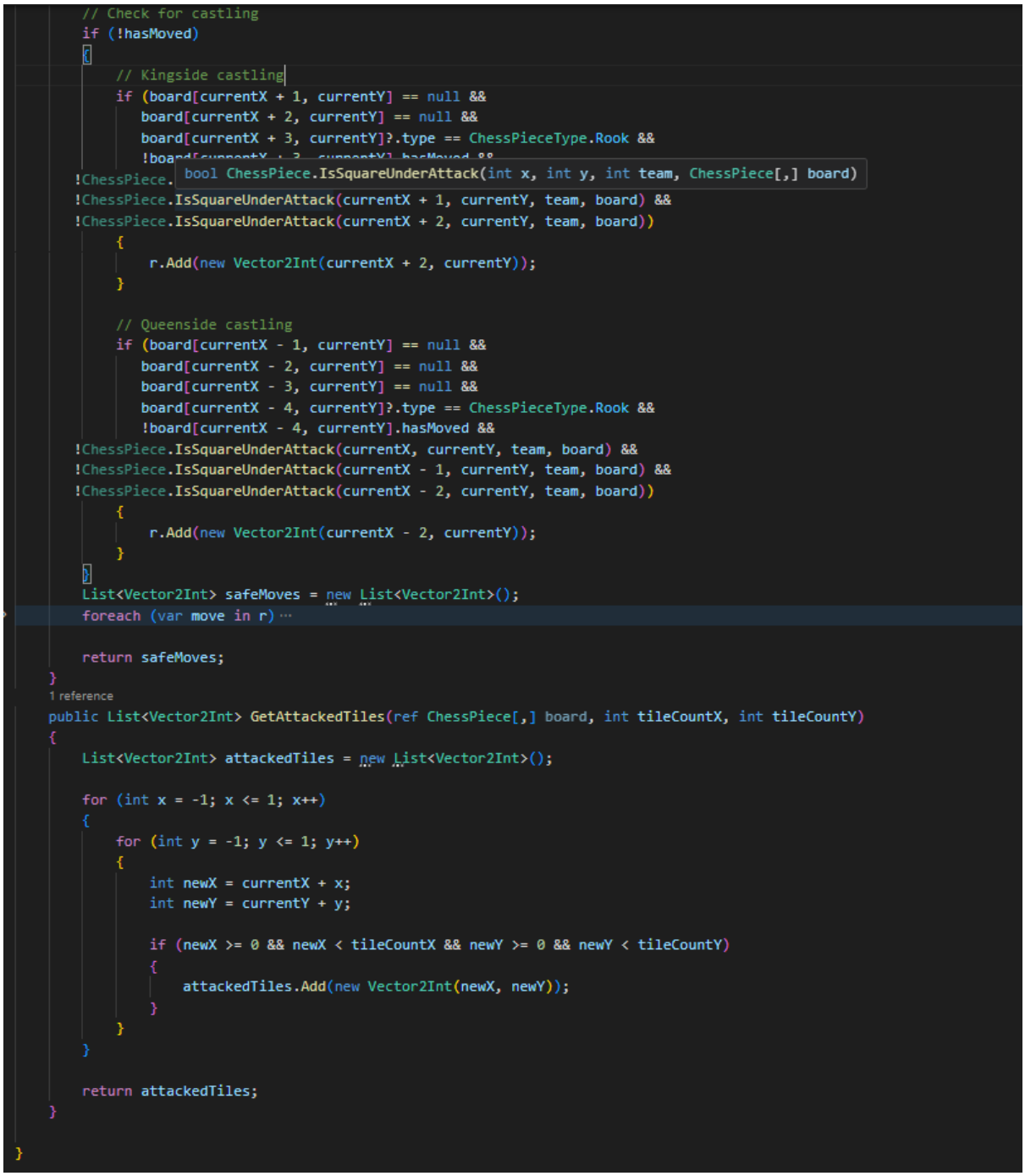


```
10 references
public static bool IsSquareUnderAttack(int x, int y, int team, ChessPiece[,] board)
{
    foreach (ChessPiece piece in board)
    {
        if (piece != null && piece.team != team)
        {
            if (piece.type == ChessPieceType.King)
            {
                King kingPiece = piece as King;
                List<Vector2Int> attackedTiles = kingPiece.GetAttackedTiles(ref board, board.GetLength(0), board.GetLength(1));
                foreach (Vector2Int tile in attackedTiles)
                {
                    if (tile.x == x && tile.y == y)
                    {
                        return true; // The square is attacked by the enemy King
                    }
                }
            }
            else
            {
                List<Vector2Int> moves = piece.GetAvailableMoves(ref board, board.GetLength(0), board.GetLength(1));
                foreach (Vector2Int move in moves)
                {
                    if (move.x == x && move.y == y)
                    {
                        return true; // The square is attacked by another enemy piece
                    }
                }
            }
        }
    }
    return false; // The square is not under attack
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Ovaj kod definira funkciju, IsSquareUnderAttack, koja određuje je li određeno polje na šahovskoj ploči napadnuto od strane protivničkog tima. Funkcija uzima x i y koordinate polja, broj tima i trenutno stanje šahovske ploče kao parametre. Iterira kroz svaku figuru na ploči, provjerava je li figura iz protivničkog tima, a zatim procjenjuje nalazi li se polje određeno x i y koordinatama unutar dostupnih poteza figure ili napadnutih polja. Funkcija vraća true ako je polje pod napadom bilo koje neprijateljske figure, a false u suprotnom.

Slika 37: Dodatni kod u klasi King.cs



Izvor: Prikaz autora iz aplikacije Visual Studio Code

Kod provjerava uvjete za rokadu i za obje strane. Da bi došlo do rokade, ni kralj ni odabrani top ne smiju se prethodno pomaknuti, a sva mjesta između njih moraju biti prazna. Osim toga, niti jedno od polja koje kralj prelazi ili zauzima ne smije biti

napadnuto. Ako su svi uvjeti zadovoljeni, rokada se dodaje dostupnim potezima za kralja.

Drugo, u kod sam uveo funkciju, GetAttackedTiles, posebno napisanu za kralja. Ova funkcija generira popis polja koje kralj može napasti, uzimajući u obzir kraljevu jedinstvenu sposobnost kretanja u šahu. Ispituje svako susjedno polje, i ako je polje unutar okvira šahovske ploče, dodaje se na popis napadnutih polja. Uključivanje ove funkcije sprječava potencijalan stack overflow izbjegavajući rekurziju s prethodno spomenutom funkcijom IsSquareUnderAttack, osiguravajući da program radi učinkovito.

Slika 38: Dodani kod u klasi Board u funkciji MoveTo

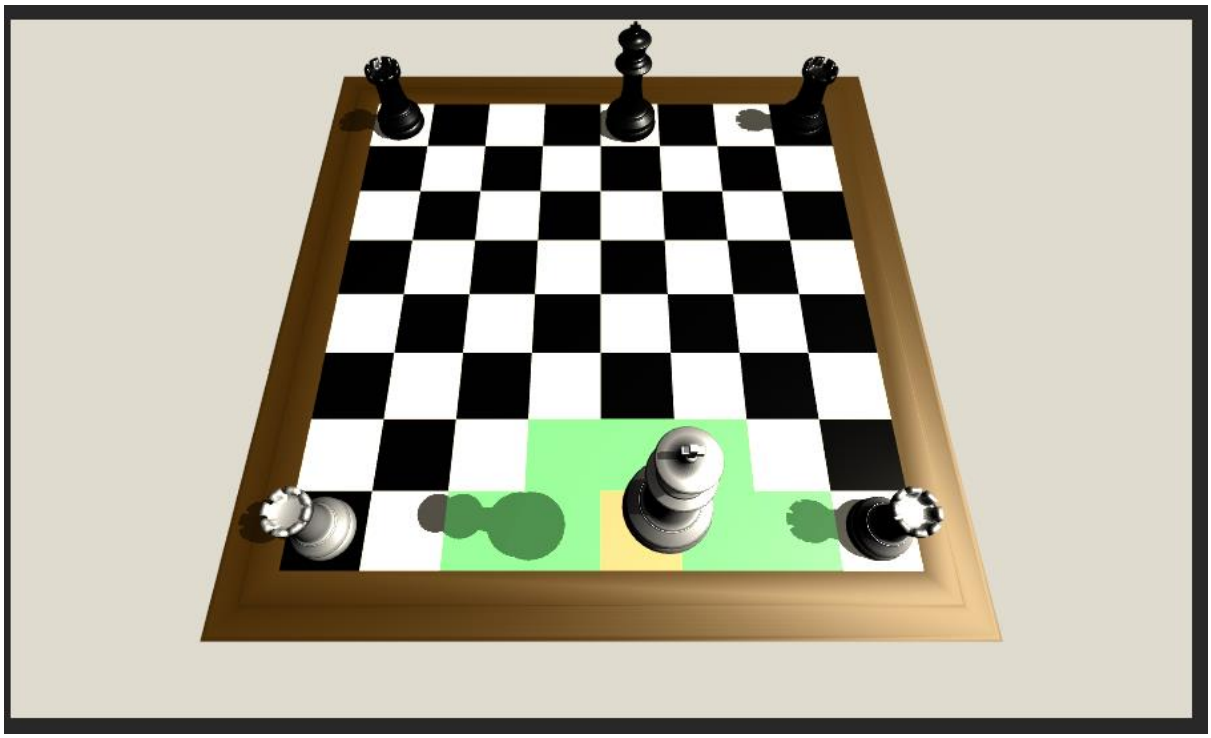
```
//castling
if (cp.type == ChessPieceType.King && (x - cp.currentX == 2 || x - cp.currentX == -2))
{
    int direction = (x - cp.currentX) > 0 ? 1 : -1;
    ChessPiece rook;
    if (direction == 1) // Kingside
        rook = chessPieces[cp.currentX + 3, y];
    else // Queenside
        rook = chessPieces[cp.currentX - 4, y];

    // Move the rook
    chessPieces[cp.currentX + direction, y] = rook;
    if (direction == 1)
        chessPieces[cp.currentX + 3, y] = null;
    else
        chessPieces[cp.currentX - 4, y] = null;

    PositionSinglePiece(cp.currentX + direction, y);
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Ovaj isječak koda upravlja kretanjem topa tijekom rokade u šahu. Kada kralj napravi rokadu, što je označeno pomicanjem dvaju polja po x osi, odgovarajući top se mora premjestiti. Kod izračunava smjer rokade, procjenjujući je li kretanje kralja u desnu stranu 1, ili u lijevu stranu -1. Nakon toga pomiče top na ispravan položaj pokraj kralja i u skladu s tim ažurira šahovsku ploču.

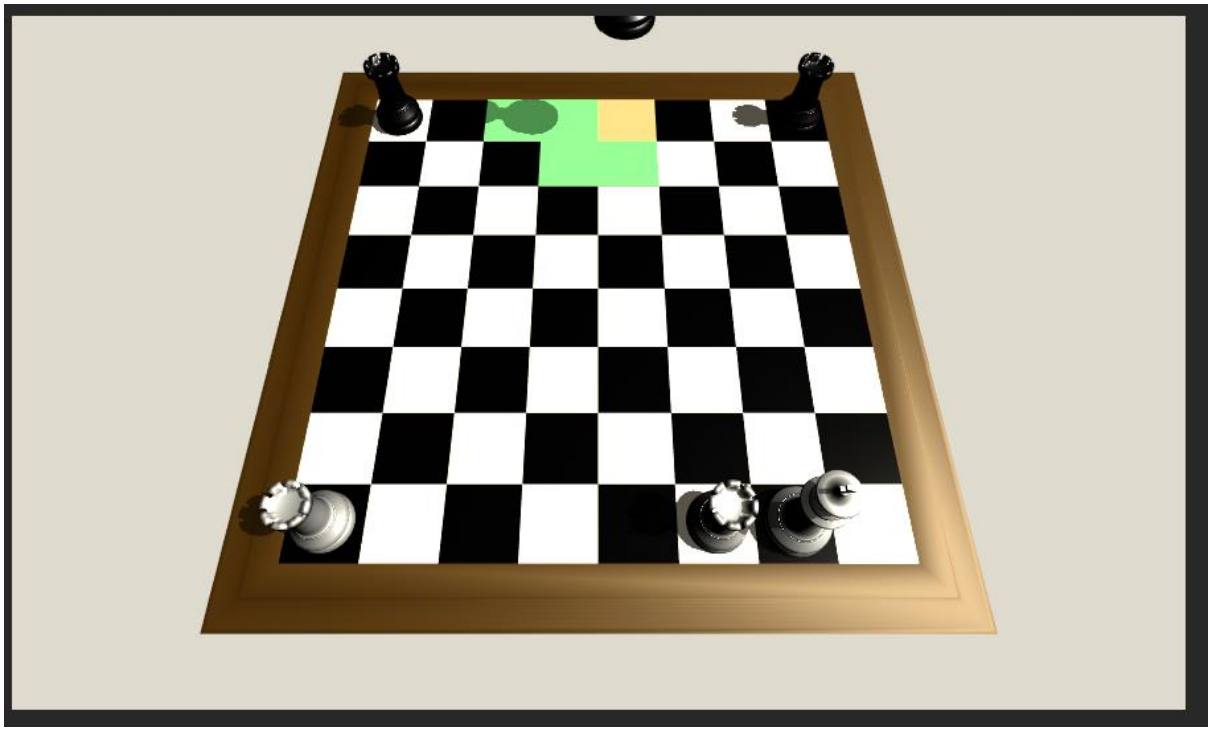
Implementiranjem ovih kodova dobili smo sljedeće rezultate (kod za generiranje određenih figura je komentiran radi lakšeg testiranja):

Slika 39: Mogući potezi figure kralja nakon dodatnog koda



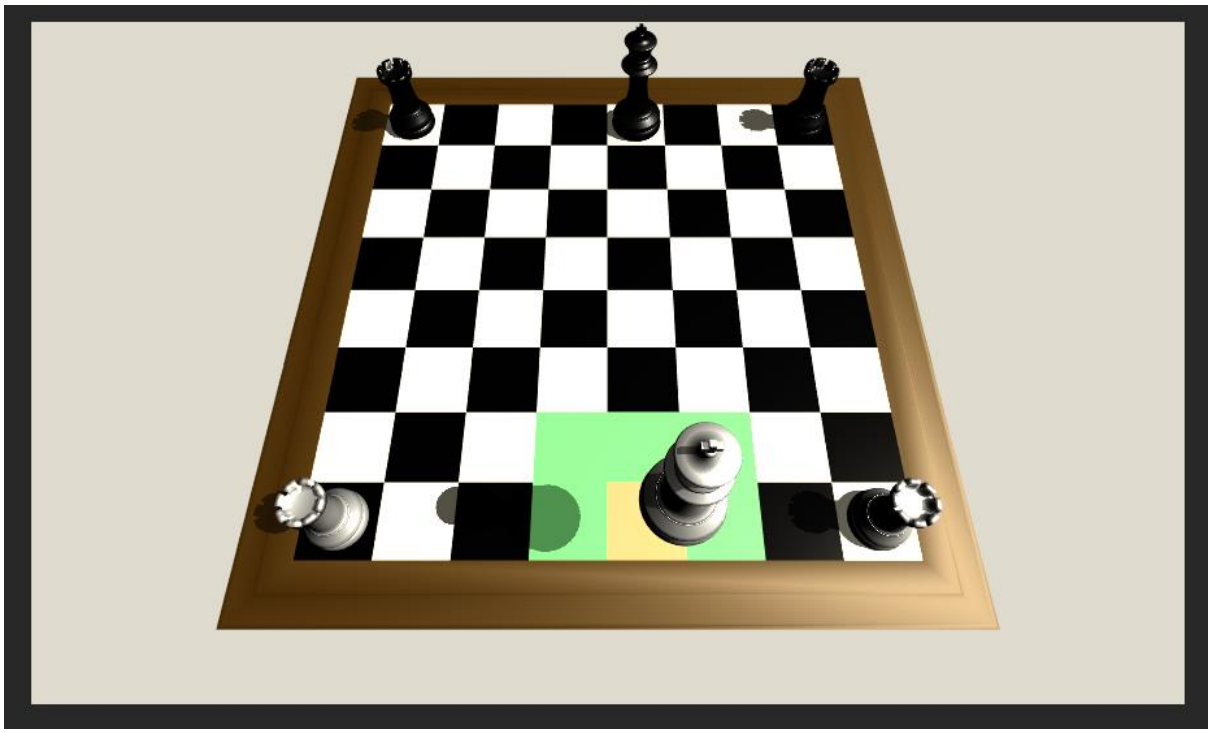
Izvor: Prikaz autora iz aplikacije Unity

Slika 40: Mogući potezi figure kralja ako su polja napadnuta



Izvor: Prikaz autora iz aplikacije Unity

Slika 41: Mogući potezi figure kralja ako su se kralj ili kula pomaknuli



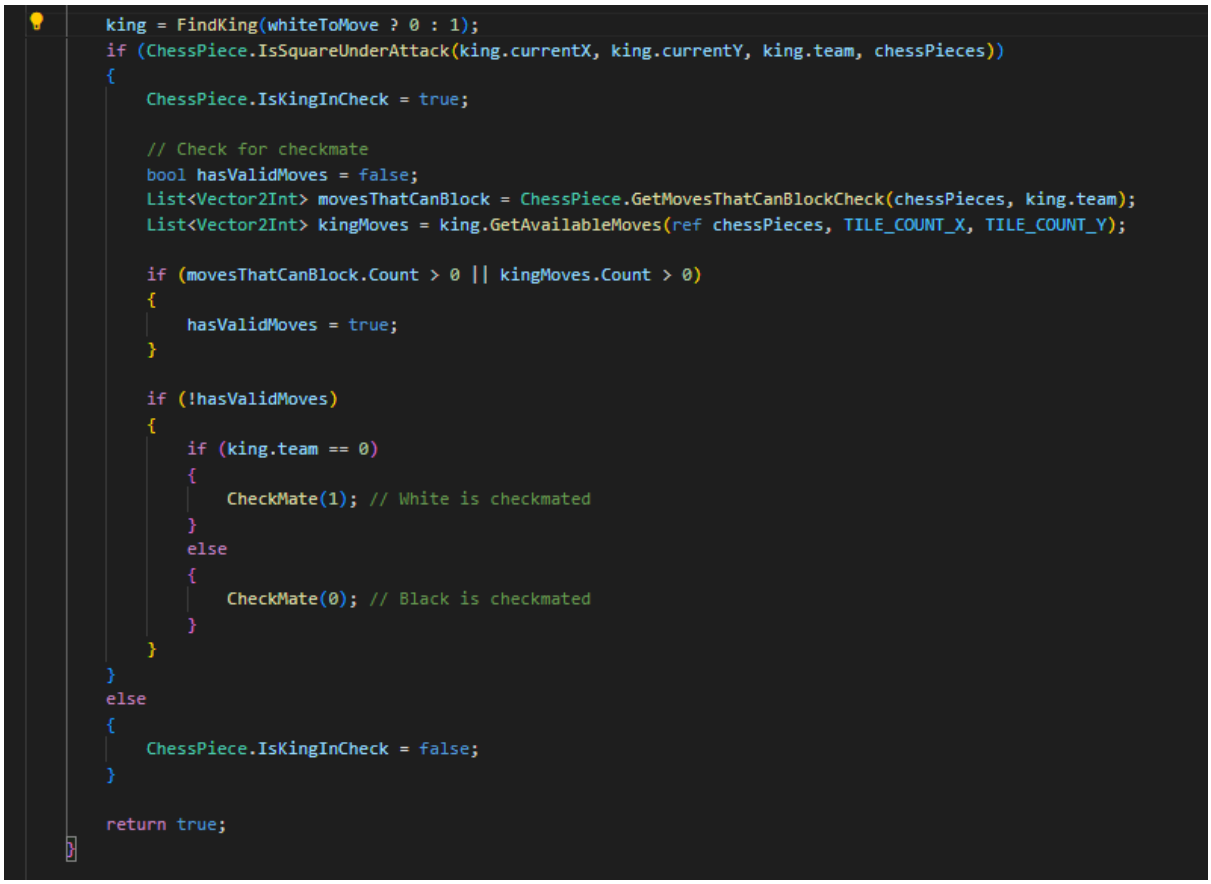
Izvor: Prikaz autora iz aplikacije Unity

5.3 Šah i šahmat logika

Kako bih poboljšao logiku igre, usredotočio sam se na implementaciju detekcije šaha i filtriranje dostupnih poteza kako bih identificirao one koji bi mogli blokirati napad na kralja.

Osim otkrivanja šaha i filtriranja poteza, implementirao sam šah-mat logiku da identificiram situacije u kojima kralj nema polja za bijeg i nijedna dostupna figura ne može blokirati šah.

Slika 42:Prikaz koda funkcije MoveTo za detekciju šaha

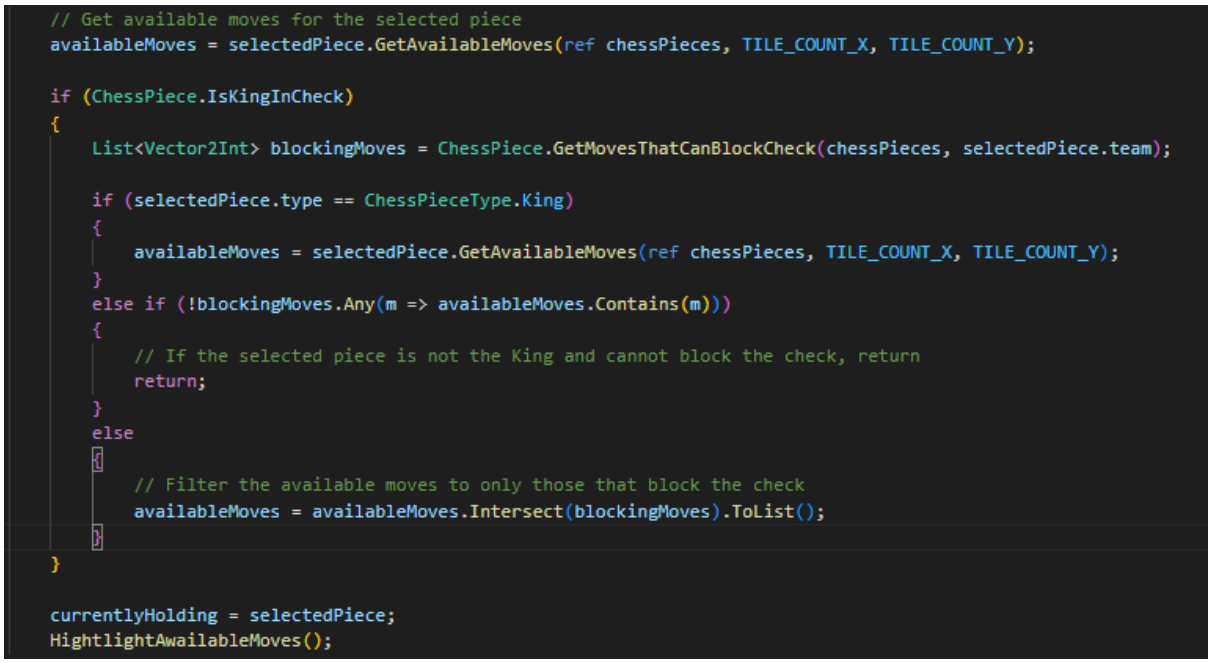


```
king = FindKing(whiteToMove ? 0 : 1);
if (ChessPiece.IsSquareUnderAttack(king.currentX, king.currentY, king.team, chessPieces))
{
    ChessPiece.IsKingInCheck = true;

    // Check for checkmate
    bool hasValidMoves = false;
    List<Vector2Int> movesThatCanBlock = ChessPiece.GetMovesThatCanBlockCheck(chessPieces, king.team);
    List<Vector2Int> kingMoves = king.GetAvailableMoves(ref chessPieces, TILE_COUNT_X, TILE_COUNT_Y);

    if (movesThatCanBlock.Count > 0 || kingMoves.Count > 0)
    {
        hasValidMoves = true;
    }

    if (!hasValidMoves)
    {
        if (king.team == 0)
        {
            CheckMate(1); // White is checkmated
        }
        else
        {
            CheckMate(0); // Black is checkmated
        }
    }
}
else
{
    ChessPiece.IsKingInCheck = false;
}

return true;
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Ovaj se kod nalazi na kraju funkcije MoveTo. Nakon što se figura pomaknula, provjerava je li kralj protivničke ekipe napadnut. Ako je kralj u šahu, tada izračunava ima li valjanih poteza za blokiranje šaha ili se kralj može pomaknuti na sigurno polje. Ako nema takvih poteza, poziva funkciju CheckMate za odgovarajući tim.

Slika 43:Prikaz koda funkcije Update za forsiranje određenih poteza



```
// Get available moves for the selected piece
availableMoves = selectedPiece.GetAvailableMoves(ref chessPieces, TILE_COUNT_X, TILE_COUNT_Y);

if (ChessPiece.IsKingInCheck)
{
    List<Vector2Int> blockingMoves = ChessPiece.GetMovesThatCanBlockCheck(chessPieces, selectedPiece.team);

    if (selectedPiece.type == ChessPieceType.King)
    {
        availableMoves = selectedPiece.GetAvailableMoves(ref chessPieces, TILE_COUNT_X, TILE_COUNT_Y);
    }
    else if (!blockingMoves.Any(m => availableMoves.Contains(m)))
    {
        // If the selected piece is not the King and cannot block the check, return
        return;
    }
    else
    {
        // Filter the available moves to only those that block the check
        availableMoves = availableMoves.Intersect(blockingMoves).ToList();
    }
}

currentlyHolding = selectedPiece;
HightlightAwailableMoves();
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Implementirao sam dodatnu logiku unutar funkcije Update za scenarije u kojima je kralj u šahu. Uveo sam uvjetnu naredbu koja provjerava je li kralj trenutno pod šahom. Ako je ovaj uvjet ispunjen, generira se lista potencijalnih poteza koji mogu blokirati šah. Nakon dobivanja ove liste, kod procjenjuje je li odabrana figura jedna od figura koje mogu blokirati šah. Ako odabrana figura nema poteza koji mogu blokirati šah, automatski se poništava odabir.

Ova prilagodba osigurava da kada je kralj u opasnosti, igrač je dužan odigrati poteze koji se izravno odnose na šah (potezi koji blokiraju šah ili miču kralja iz šaha). Posljedično, ovo pojačava usklađenost igre sa službenim šahovskim pravilima i povećava i razinu izazova i angažiranost iskustva igranja.

5.4 Kreiranje UI-a za pobjedu

Za izradu jednostavnog korisničkog sučelja za pobjednički zaslon u Unityju započeo sam stvaranjem platna (Canvas). Unutar ovog platna ugradio sam sliku te ju nazvao VictoryScreen, prilagodio njezinu boju na crnu i postavio neprozirnost na 200. Nakon toga sam unutar nje ugradio dva gumba, označena kao 'Restart' i 'Exit'. Nakon toga, ugradio sam tekst koji pokazuje je li crni ili bijeli tim pobjedio. Kako bih organizirao te elemente, stvorio sam komponentu Vertical Layout Group, dodijelio razmak od 15 i poravnao elemente u Middle Center. U Unityju, pod karticom Hijerarhija, unutar platna u VictoryScreen, preuredio sam elemente sljedećim redoslijedom: WhiteWin, BlackWin, Restart, Exit. WhiteWin i BlackWin služe kao tekstualne indikacije, pojavljuju se na temelju pobjedničkog tima, dok gumbi olakšavaju izlazak iz igre ili pokretanje nove.

Slika 44: Prikaz kreiranog UI

Izvor: Prikaz autora iz aplikacije Unity

Zatim je samo trebalo dodati funkcije u mom kodu kako bi se UI pojavio samo kada dođe do šah mata.

Slika 45: Prikaz dodanog koda za prikaz UI

```
2 references
private void CheckMate(int team)
{
    Victory(team);
}
1 reference
private void Victory(int winningTeam)
{
    victoryScreen.SetActive(true);
    victoryScreen.transform.GetChild(winningTeam).gameObject.SetActive(true);
}
0 references
public void ResetButton()
{
    //UI
    victoryScreen.transform.GetChild(0).gameObject.SetActive(false);
    victoryScreen.transform.GetChild(1).gameObject.SetActive(false);
    victoryScreen.SetActive(false);

    //Field reset
    currentlyHolding = null;
    availableMoves = new List<Vector2Int>();

    //Clean up the board
    for (int x = 0; x < TILE_COUNT_X; x++)
        for (int y = 0; y < TILE_COUNT_Y; y++)
        {
            if (chessPieces[x, y] != null)
                Destroy(chessPieces[x, y].gameObject);

            chessPieces[x, y] = null;
        }
    //Clean captured
    for (int i = 0; i < deadWhites.Count; i++)
        Destroy(deadWhites[i].gameObject);
    for (int i = 0; i < deadBlacks.Count; i++)
        Destroy(deadBlacks[i].gameObject);

    deadWhites.Clear();
    deadBlacks.Clear();

    GenerateAllPieces();
    PositionAllPieces();
    whiteToMove = true;
}
0 references
public void ExitButton()
{
    Application.Quit();
}
```

Izvor: Prikaz autora iz aplikacije Visual Studio Code

Funkcija CheckMate prihvaća identifikator tima kao parametar i poziva funkciju Victory. Unutar funkcije Victory aktivirao sam VictoryScreen i, ovisno o pobjedničkom timu, prikazao odgovarajuću poruku pobjede.

Uz to, uveo sam funkciju ResetButton za upravljanje korisničkim sučeljem. Ova funkcija deaktivira VictoryScreen, resetira polje i čisti ploču i uhvaćene figure. Nakon

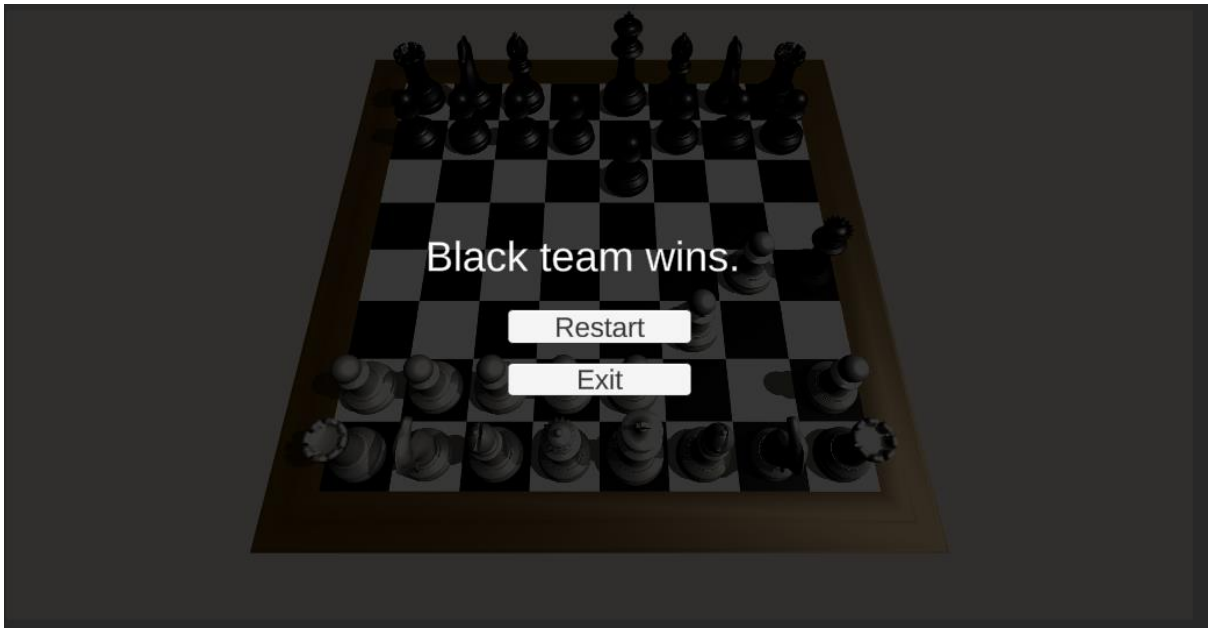
toga, ponovno inicijalizira i postavlja sve figure i postavlja bijeli tim da igra prvi, čime se kreira početak nove igre.

Na kraju, implementirao sam funkciju ExitButton koja, kada se pozove, prekida aplikaciju, dopuštajući igraču da izađe iz igre.

Slika 46: Pobjeda bijelog tima

Izvor: Prikaz autora iz aplikacije Unity

Slika 47: Pobjeda crnog tima

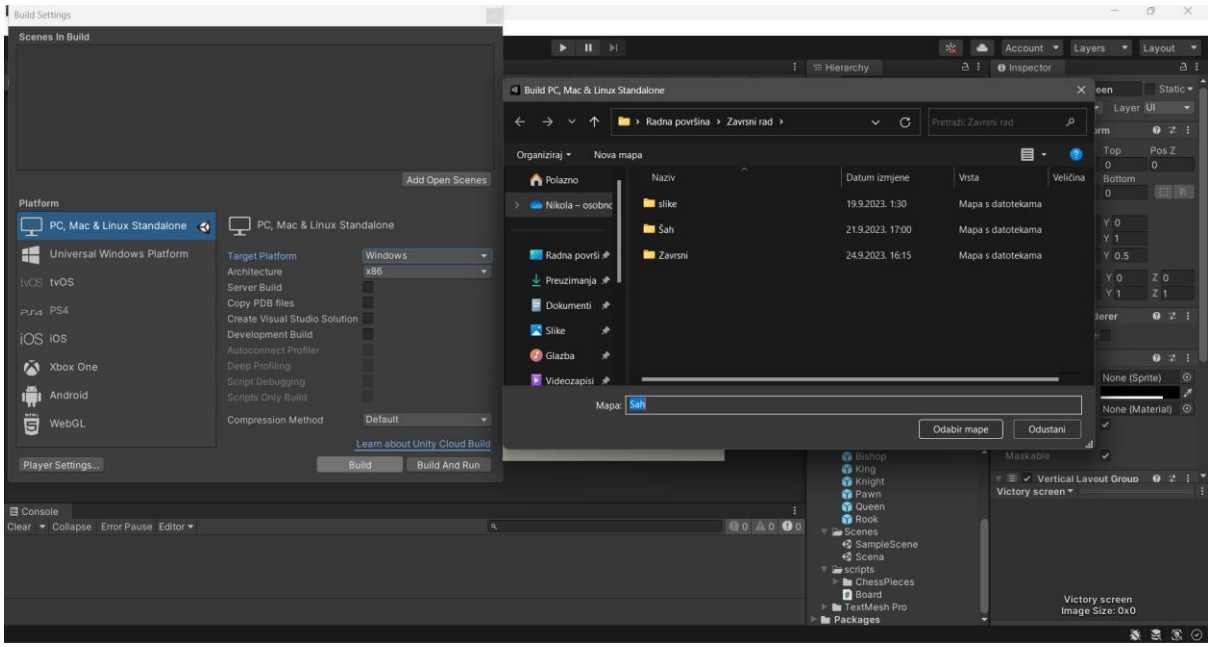


Izvor: Prikaz autora iz aplikacije Unity

5.5 Izgradnja aplikacije

Nakon što sam dovršio aspekte programiranja, nastavio sam s izgradnjom aplikacije u Unityju. Izrada aplikacije u Unityju može biti relativno jednostavan proces. Unity nudi korisničko sučelje i niz alata i značajki koje olakšavaju proces izgradnje.
Unity preuzima brigu o kompajliranju i pakiranju aplikacije, čineći proces pristupačnim, čak i za one koji su novi u razvoju igara.

Slika 48: Izgradnja aplikacije u Unityu



Izvor: Prikaz autora iz aplikacije Unity

Na kraju izrade projekta, kada smo spremni izgraditi aplikaciju, sve što trebamo napraviti je kliknuti File > Build settings > Build i odabrati novu praznu mapu.

6. Zaključak

Unity je odličan softver za sve koji žele naučiti i baviti se kreacijom 2D i 3D igrica, a također je vrlo korišten i tražen u industriji.

Njegovo korisničko sučelje, u kombinaciji s jakom zajednicom i opsežnom dokumentacijom, čini ga dostupnim programerima različitih razina vještina.
Unity Asset Store pruža mnoštvo resursa koji mogu značajno ubrzati razvoj i poboljšati kvalitetu krajnjeg proizvoda.
Njegova besplatna dostupnost otvara vrata raznolikom rasponu kreatora, dopuštajući im da transformiraju svoje vizije u stvarnost bez financijskih ograničenja.
C# je svestran i objektno orijentiran programski jezik koji je sastavni dio razvoja igara u Unityju.
Nudi mnoštvo biblioteka i okvira, omogućujući programerima pisanje učinkovitog, skalabilnog koda koji se može održavati.
Njegovo automatsko upravljanje memorijom i mogućnosti rukovanja pogreškama pridonose njegovoj popularnosti i pouzdanosti u zajednici razvijača igara.

U ovom radu demonstrirane su osnove Unityja i C#. Iskorištavanje besplatnih sredstava iz Unity Asset Storea omogućilo je brži proces razvoja igre. Šahovska igra je uspješno kreirana te je potpuno funkcionalna, iako s daljnjim iskustvom postoji potencijal za značajnu optimizaciju koda i same igre.

7. Popis literature

[1] Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. Dostupno na: https://unity.com/ (Pristupljeno: 20. 9. 2023).

[2] Churchville, F. (2021) What is User Interface (UI)? definition from searchapparchitecture, App Architecture. Dostupno na: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI (Pristupljeno: 20 9. 2023).

[3] What is User Interface (UI) design? (2023) The Interaction Design Foundation. Dostupno na: https://www.interaction-design.org/literature/topics/ui-design (Pristupljeno: 20. 9. 2023).

[4] The unity tutorial for complete beginners (2022) YouTube. Dostupno na: https://www.youtube.com/watch?v=XtQMytORBmM (Pristupljeno: 20. 9. 2023).

[5] Technologies, U. (no date) Welcome to the Unity scripting reference!, Unity. Dostupno na: https://docs.unity3d.com/ScriptReference/index.html (Pristupljeno: 21. 9. 2023).

[6] Technologies, U. Unity user manual 2022.3 (LTS), Unity. Dostupno na: https://docs.unity3d.com/Manual/index.html (Pristupljeno: 21. 9. 2023).

[7] Technologies, U. Unity's Asset Store, Unity. Dostupno na: https://docs.unity3d.com/Manual/AssetStore.html (Pristupljeno: 22. 9. 2023).

[8] Rules of chess (2023) Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Rules_of_chess (Pristupljeno: 22. 9. 2023).

[9] Where developers learn, share, & build careers (no date) Stack Overflow. Dostupno na: https://stackoverflow.com/ (Pristupljeno: 22. 9. 2023).

[10] C# tutorial (no date) C# Tutorial (C Sharp). Dostupno na: https://www.w3schools.com/cs/index.php (Pristupljeno: 23. 9. 2023).

8. Popis slika

9. Sažetak

Tema ovog završnog rada je razvoj 3D šahovske igre koristeći Unity okruženje te programski jezik C#. Fokus je na demonstraciji osnova Unityja i C#, a korištenjem besplatnih sredstava iz Unity Asset Storea omogućen je brži proces razvoja. U uvodnom dijelu rada, detaljno je predstavljen program Unity i programski jezik C#, uz osvrt na njihovu povijest i razvoj. Nakon toga, rad pruža uvide u različite primjene Unityja i C# u brojnim industrijama. Dodatno, rad obuhvaća detaljnu analizu korisničkog sučelja Unitya, razmatra različite vrste dostupnih licenci Unitya i istražuje mogućnosti koje nudi Unity Asset Store.

U praktičnom dijelu rada, demonstrirane su mogućnosti programa Unity i programskog jezika C# kroz konkretan primjer. Razvijena je 3D šahovska igra u tri različite verzije. U prvoj verziji, igra je osnovna i ne uključuje pravila. U drugoj verziji, implementirani su specifični potezi za svaku šahovsku figuru. U finalnoj, trećoj verziji, dodani su kompleksni potezi poput rokade i en passanta, kao i logika za šah, šah-mat i pobjednički korisnički interfejs (UI).

Ključne riječi: Unity okruženje, Programski jezik, C#, Korisničko sučelje, Verzije, Potezi, Šahovske figure, Logika, Šah, Šah-mat

10. Abstract

The topic of this final thesis is the development of a 3D chess game using the Unity environment and the C# programming language. The focus is on demonstrating the basics of Unity and C#, and the use of free assets from the Unity Asset Store enables a faster development process. In the introductory part of the paper, the Unity program and the C# programming language are presented in detail, with a review of their history and development. After that, the paper provides insights into various applications of Unity and C# in a number of industries. Additionally, the paper includes a detailed analysis of the Unity user interface, discusses the different types of Unity licenses available, and explores the capabilities offered by the Unity Asset Store.
In the practical part of the work, the capabilities of the Unity program and the C# programming language were demonstrated through a concrete example. A 3D chess game was developed in three different versions. In the first version, the game is basic and does not include rules. In the second version, specific moves are implemented for each chess piece. In the final, third version, complex moves such as castling and en passant were added, as well as logic for chess, checkmate and a winning user interface (UI).

Keywords: Unity environment, Programming language, C#, User interface, Versions, Moves, Chess pieces, Logic, Chess, Checkmate