SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mario Štahan

IZRADA IGRE NA POTEZE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mario Štahan

Matični broj: 42143/13–R

Studij: Informacijski sustavi

IZRADA IGRE NA POTEZE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Dr. sc. Mladen Konecki

Varaždin, rujan 2018.

Mario Štahan

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

Igra na poteze, isto poznata kao strategija na poteze, je žanr video igre gdje dva ili više igrača naizmjenično izvršavaju skup strateško isplaniranih akcija bez vremenskog pritiska ili fokusa na vrijeme u svrhu pobjede nad drugim igraču. Igra je izrađena na Unity platformi, koja omogućava izradu ne samo video igara, nego i drugih vrsta medijskog sadržaja. Unity platforma podržava C# i javascript programske jezike, i mnoge igre su napravljene u njoj, uključujući i igre kao što su Hearthstone, 7 Days to die i Plague Inc. Svaka igra ima skup mehanika koje omogućuju pravilan rad igre. Nekoliko primjera mehanike su traženje putanje, generiranje karte igre i ponašanje umjetne inteligencije.

Ključne riječi: Unity, računalna igra, Strategija na poteze, A* algoritam, traženje putanje, umjetna inteligencija, generiranje karte.

Sadržaj

1. Uvod

Strateške igre na poteze su jednom bili jedna od popularnijih žanrova u svijetu PC-a, ali kroz godine su polagano prešle u drugi plan pokraj većih žanrova kao što su pucačine prvog lica ili akcijskih orijentiranih igara. Uz nekoliko iznimaka kao što su Civilization igre i Fire Emblem, trostruke A strateške igre su nepostojeće. S time rečeno, strategije na poteze sadrže zanimljive mehanike koje se dan danas relevantne u svim žanrovima video igara.

Kroz ovaj rad, što uključuje i izradu strateške igre na poteze na Unity platformi, ću proći kroz nekoliko mehanika, opisati ih i u praktičnom dijelu prikazati.

2. Metodike i tehnike rada

Izrada ovog projektnog rada je bilo podijeljena u dvije faze: faza izrade praktičnog prototipa igre i faza pisanja teoretskog dijela ovog rada.

Faza izrade praktičnog rada se sastoji od učenja korištenja alata Unity pomoću raznih youtube vodiča i tekstualnih vodiča. Usto su se istraživali različiti koncepti za samu igru kao npr. oblik polja za kartu, algoritmi za kreiranje putanja, ponašanje umjetne inteligencije i sl. Nakon što su se odabrali koncepti, ti isti koncepti su se primijenili na samu igru.

Faza pisanja teoretskog dijela se odvija nakon završene faze izrade igre. U ovoj fazi se prolazi kroz kod igre, gleda se koji dijelovi su igre su najvažniji i taj kod se pretvara u pseudokod za daljnju upotrebu. Pretvorenom pseudokodu se dodaje opis i kontekst. Tada se po potrebi koriste izvori nađeni na internetu vezani uz teoretske dijelove kao što su A* algoritam. Pri završetku opisa pseudokoda, dolazi opis samog alata, vrste žanra i opis igre. Na kraju se pismeni rad pravilno formatira.

Alati korišteni u ovom radu su: Visual Studio 2017, Unity Engine, Microsoft Office Word, Photoshop za grafičke elemente igre.

3. Unity

Unity je jedna od popularnijih platforma za kreiranje raznih medijskih sadržaja s velikim brojem mogućih upotreba. Unity se koristi u automotivnoj i transportnoj industriji, za kreiranje filmova, serija i video igara, za obrazovanje unutar VR-a itd[1].

Unity kao game engine podržava razne vrste igara: 2D, 3D, AR(augmentiranja realnost), VR(virtualna realnost) koje se mogu distribuirati na čak 27 različitih platforma koje uključuju PC, Mac, Linux, Playstation 4, Xbox One, Android, iOS itd. Unity podržava programiranje u C# programskom jeziku i Javascript programskom jeziku[2].

Unity svoj početak nalazi u godini 2006. gdje je prva verzija mogla distribuirati samo za Mac OS X računala. Tek od verzije 1.1. moglo se izvesti igra za Windows platformu. Od 2006. godine do danas je osvojilo nekolicinu nagrada od raznih organizacija i publikacija, uključujući Wall Street Journal, Computer Entertainment Developers Conference (CEDEC) i Game Developers Conference (GDC)[3].

Poznatije igre koje su izrađene u Unity-u[4]:

- Battlestar Galactica Online
- Plague Inc
- 7 Days to Die
- Shadowrun returns
- Dragon Quest VIII
- Hearthstone
- Angry Birds 2

4. Žanr strategija na poteze

Strateške računalne igre su žanr igara čiji je fokus na igračeve vještine razmišljanja i planiranja pomoću kojih mogu potencijalno ostvariti pobjedu u igri. U većini slučajeva podosta igara ovog žanra uključuju i ekonomske aspekte u igri kao i mogućnost istraživanja, bilo tehnologija ili novih nepoznatih prostora. Strategije se mogu kategorizirati u četiri skupine ovisno o tome ako su strategije na poteze ili strategije u realnom vremenu i ako se igra fokusira na strategiju ili taktiku[5].

Tablica 01: Prikaz podžanrova strateških igara s primjerima

	Strategija na poteze	**Strategija u realnom vremenu**
Fokus na strategiju	Civilization	Starcraft 2
Fokus na taktiku	XCOM: Enemy Unknown	Commandos 2: Men of Courage

(Izvor: Wikipedia, 2018.)

Strategije na poteze su podžanr strateških igara koji za razliku od strategija u realnom vremenu ne stavljaju u prvi plan vremensku komponentu strategija, nego se fokusira na strateško i taktičko promišljanje i kreiranje plana za ostvarivanje pobjede. Igra se odvija izmjeničnim izvođenjem poteza između igrača i protivnika. Veliki broj društvenih igara kao što su Šah, Rizik, Monopoli i sl. su bazirane na ovoj mehanici[5].

Primjeri igara podžarna strategija na poteze[6]:

- Battle for Wesnoth
- Civilization
- Worms
- Heroes of Might and Magic
- Space Rangers

5. Opis igre

Igra kreirana za ovaj rad je strateška igra na poteze za dva igrača(plavi i crveni igrač) ili za jednog igrača koji igra protiv računala, tj. umjetne inteligencije. Igra je smještena u ratu između dva kralja gdje cilj svakog kralja je eliminacija drugog kralja.

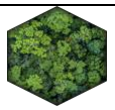
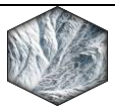
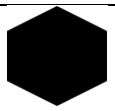
Svaka strana ima kontrolu nad dvije vrste jedinica: kralj i ratnici. Svaki igrač započinje s kraljem koji omogućuje stvaranja druge jedinice: ratnici. Obje jedinice imaju svoju karakteristiku kao što su jačina napada, obrana, šansa za obranu protiv napada, broj koraka tj. domet jedinice i život tj. koliko štete može primiti(Oznaka je HP).

Tablica 02: Prikaz jedinica u igri

Naziv	Karakteristike	Izgled
Kralj	HP: 30, Napad: 8-10 Obrana: 3, Blok šansa: 25% Domet: 5 koraka	 Slika 1: Kralj (Izvor: Konecki M., 2018)
Ratnik	HP: 10, Napad: 4-6 Obrana: 1, Blok šansa: 5% Domet: 3 koraka	 Slika 2: Ratnik (Izvor: Konecki M., 2018)

Jedinica se kreću na karti neodređene veličine(za svrhe ove igre veličina je 15x16) gdje jedna jedinica veličine je heksagonalno polje. Plavi igrač započinje u donjem lijevom kutu a crveni igrač započinje u gornjem desnom kutu. Svako polje na karti ima svoje specifikacije ovisne o tipu pod kojim je to polje definirano. Igra završava kada jedan od kraljeva je eliminiran.

Tablica 03: Prikaz tipova polja

Ime polja	Opis polja	Cijena	Izgled
Livada	Najosnovnije polje bez posebnih karakteristika.	1 korak	 Slika 3: Livada (Izvor: Konecki M., 2018)
Močvara	Ulaskom u polje jedinica gubi 1 HP.	1 korak	 Slika 4: Močvara (Izvor: Konecki M., 2018)
Planina	Ulazak na ovo polje košta više koraka.	3 koraka	 Slika 5: Planine (Izvor: Konecki M., 2018)
Selo	Ako se na početku poteza jedinica nalazi na ovom polju, dobiva se 1 zlatnik i jedinica se liječi za 2 HP-a.	1 korak	 Slika 6: Selo (Izvor: Konecki M., 2018)
Zidine	Na ovim poljima se stvaraju ratnici. Ratnik košta 1 zlatnik.	1 korak	 Slika 7: Zidine (Izvor: Konecki M., 2018)
Grad	Ako je kralj na ovom polju, onda igrač može kreirati ratnike u zidinama.	1 korak	 Slika 8: Grad (Izvor: Konecki M., 2018)
Void	Jedinica ne može ući u polje.	∞	 Slika 9: Void (Izvor: Konecki M., 2018)

Slika 10: Početno stanje igre

6. Mehanike igre

Mehanika igre se ostvaruju pomoću kombinacije komponenata koje dolaze s Unity alatom i skripti koje kreator igre napiše. Te skripte, u slučaju c#, su klase koje se vežu uz neki objekt na sceni i koje se onda koristeći razne okidače izvršavaju. Prije nego što opišemo same mehanike, treba opisati nekolicine manjih klasa pomoću kojih određeni dijelovi skripte ne bi bili ostvarivi. Te klase opisuju tip polja, red čekanja s prioritetom i čvorove koji čine graf karte za kreiranje putanja.

6.1. Klasa TipPolja

Iako je deklarirana kao klasa, TipPolja se koristi kao struktura podataka čija svrha je opis polja. Deklarirana je kao klasa jer je jednostavnije kreirati nove objekte od te klase unutar GUI-a Unity alata nego zasebno kodirati čitanje iz datoteke za te objekte ili hard-kodirati zasebno svaki objekt. Usto, Unity ima razne dodatne opcije koje se mogu aktivirati unutar koda koje omogućuju dodatne GUI opcije kao npr. serijalizacija za određenu klasu koja se koristi za ovu klasu. TipPolja sadrži:

- string imePolja koja sadrži ime polja
- GameObject imenom izgled
- Sprite klasa imenom sprite
- Color klasa imenom boja, zadana boja je bijela
- float varijabla imenom cijenaUlaska, zadana vrijednost je 1

GameObject izgled sadrži objekt sa svim potrebnim komponentama koje omogućuju izvršavanje pojedinih funkcija unutar igre. Sprite je objekt koji sadrži grafičku reprezentaciju polja. Klasa Color sadrži informacije o boji polja. Zadnja varijabla sadrži cijenu ulaska u taj tip polja.

6.2. Klasa Cvor

Objekti klase Cvor se koriste u strukturi grafa za kreiranje putanje od točka A do točke B. Klasa sadrži tri tipa podataka i jedne metode:

- lista objekta klase Cvor imenom susjedi
- Vector2Int objekt imenom pozicija
- Vector3 objekt imenom realnaPozicija
- metoda koja vraća float vrijednost imenom Distanca

Lista susjedi sadrži susjedne čvorove trenutnog čvora, pozicija sadrži koordinate unutar grafa i realnaPozicija sadrži realnu poziciju Cvor objekta u sceni unutar Unity alata. Metoda distanca prima kao ulaz Cvor objekt, a vraća „fizičku“ udaljenost između dva čvorova unutar grafa kao float vrijednost.

6.3. Klasa RedSPrioritetom

Red čekanja s prioritetom(eng. *Priority Queue*) je apstraktna vrsta strukture podataka slična običnom redu čekanja, s razlikom u tome kako raspoređuju stavke unutar reda. Dok običan red čekanja koristi princip prvi unutra, prvi van (eng. *first-in, first out*), red čekanja s prioritetom raspoređuje svoje stavke prema prioritetu koji je vezan uz stavku, od najbližeg prioriteta nuli do najdaljeg prioriteta nuli. Drugim riječima, prva stavka koja izlazi iz reda čekanja je uvijek ona s najbližim prioritetom.[7]

Implementacija reda čekanja u ovom radu je napravljena preko klase koja je specifično napisana za rad s čvorovima koji se nalaze u grafu i/ili u putanji jedinice. Klasa se sastoji od:

- niza čvorova neodređene veličine imenom cvorovi
- niza float vrijednosti neodređene veličine imena prioriteti
- integer koji sadrži broj elemenata u red čekanja imena brElemenata
- metoda Dodaj koja stavlja stavku u red čekanja
- metoda Preuzmi koja vadi prvi element u redu čekanja
- metoda NijePrazan koja nakon provjere ako je red prazan vraća bool vrijednost false, tj. ako red sadrži barem jednu stavku, vraća bool vrijednost true

Treba napomenuti da pri deklariranju reda čekanja treba odrediti veličinu samih nizova cvorovi i prioriteti.

Metoda Dodaj, koja prima objekt Cvor i float vrijednost prioritet, dodaje novi čvor u red čekanja. U pseudokodu, metoda izgleda ovako:

```
if brElemenata == 0:
      cvorovi[0] = noviCvor;
      priroteti[0] = prioritet;
      brElemenata++;
      return;
for i = 0; i < brElemenata; i++:
      if prioriteti[i] > prioritet:
            for j = brElemenata; j > i; j++:
                  prioriteti[j] = prioriteti[j-1];
                  cvorovi[j] = cvorovi[j-1];
            prioriteti[i] = prioritet;
            cvorovi[i] = noviCvor;
            brElemenata++;
            return;
prioriteti[brElemenata] = prioritet;
cvorovi[brElemenata] = noviCvor;
brElemenata++;
```

Metoda počinje s provjerom ako red čekanja ima elemenata. Ako nema, tada samo dodaje na početak nizova cvorovi i prioriteti cvor koji se dodaje u taj red čekanja, povećava se brElemenata za 1 i završava ovu metodu.

Ako već postoje čvorovi u redu čekanja, tada se prolazi kroz sve čvorovi sve dok se nađe čvor čiji prioritet je veći od onog koji se dodaje. Kada nađe takvog čvora tada miče sve čvorovi od tog čvora s većim prioritetom, uključujući i taj čvor, za jedno mjesto nazad. Na kraju stavlja dodani čvor i njegov prioritet na mjesto u nizu gdje je prvi čvor s većim prioritetom prije bio i izlazi iz metode. U slučaju da ne naiđe na čvor čiji prioritet je veći od dodanog, tada se samo na kraju niza dodaje čvor.

Metoda preuzmi izbacuje van prvi čvor u redu čekanja. To ostvaruje tako da prvi čvor u nizu stavi privremeno stavi sa strane, pomakne sve čvorove u nizu za jedan naprijed, smanji brElemenata za 1 i postavlja čvor kojeg smo privremeno stavili sa strane kao izlazni rezultat metode. Ta metodu u pseudokodu izgleda ovako:

```
if brElemenata == 0: return null;
Cvor outputCvor = cvorovi[0];
for i = 0; i<brElemenata; i++:
      cvorovi[i] = cvorovi[i+1];
      prioriteti[i] = prioriteti[i+1];
brElemenata--;
return outputCvor;
```

6.4. Mehanika generiranje karte

Mehanika generiranje karte je prva mehanika koja se izvršava pri inicijalizaciji igre. Ta mehanika omogućuje čitanje i kreiranje karte s potrebnim grafom za rad s putanjama i postavlja scenu za početak igre. Ta mehanika se može podijeliti u 4 koraka:

1. Učitavanje karte
2. Prikazivanje karte
3. Generiranje grafa
4. Postavljanje jedinica

Učitavanje karte je prvi korak generiranja karte. U ovom koraku se iz tekstualne datoteke definira koliko je velika karta i na kojoj poziciji je kakva vrsta polja. Jedino što se ne iščitava iz datoteke je pozicija grada i njegovih zidina, koje su uvijek u donjem lijevom kutu i u gornjem desnom kutu, i sva polja koja nisu posebno specificirana u datoteci su definirana kao livade.

Pseudokod prvog koraka počinje s učitavanjem tekstualne datoteke u string varijablu i micanjem formatiranja koji se koriste za lakše pisanje i čitanje informacija od strane čovjeka.

```
string korigirano = ucitajTextFile(karta.txt);
korigirano = textKarta.text.Replace("\n", string.Empty);
korigirano = korigirano.Replace("\r", string.Empty);
korigirano = korigirano.Replace("\t", string.Empty);
korigirano = korigirano.Replace(" ", string.Empty);
string podaci = korigirano.Split(';');
```

Kranji string imenom korigirana se onda odvaja u niz koji sadrži podatke pomoću separatora točka-zarez(;). Prvi član u nizu je veličina karte, a ostali članovi su tip polja za određeno polje na karti.

```
string velMapa = podaci[0].Split('|');
velicinaX = int.Parse(velMapa[0]);
velicinaY = int.Parse(velMapa[1]);
poljaKarte = new int[velicinaX, velicinaY];
```

Nakon što se izvuče veličina karte, definira se dvodimenzionalno polje veličine karte koja sadrži za svaku koordinatu koja vrsta polja mora biti.

```
for x = 0; x < velicinaX; x++:
      for y = 0; y < velicinaY; y++:
                poljaKarte[x, y] = 0;

poljaKarte[0, 0] = 4;
poljaKarte[velicinaX - 1, velicinaY - 1] = 4;
poljaKarte[0, 1] = 5;
poljaKarte[1, 0] = 5;
poljaKarte[velicinaX - 1, velicinaY - 2] = 5;
poljaKarte[velicinaX - 2, velicinaY - 1] = 5;
```

To dvodimenzionalno polje se privremeno puni indeksom polja za livade. Pri završetku te petlje, definira se u gornjem desnom i donjem lijevom kutu baze za oba igrača.

```
foreach linija in podaci:
      if linija == podaci[0]: continue;
      if linija == podaci[podaci.Length - 1]: break;

      string polje = linija.Split('|');
      int x = int.Parse(polje[0]);
      int y = int.Parse(polje[1]);
      int vrsta = int.Parse(polje[2]);

      poljaKarte[x, y] = vrsta;
```

Na kraju se prolazi kroz ostatak podataka izvučene iz tekstualne datoteke i definiraju se sva polja na karti.

Drugi korak u generiranju karte je prikazivanje same karte na scenu igru. Ovdje treba napomenuti varijablu pomak, koja je definirana kao razlomak veličine karte na Y osi s 4 pomnoženo s -1. Taj pomak služi za točno pozicioniranje heksagonalnih polja po Y osi na scenariju. Ako se taj pomak ne uračuna u pozicioniranje polja, tada će razmak između redova karte biti prevelika. Taj pomak nakon svakog prolaska reda se smanjuje za jednu četvrtinu da se održi taj manjak razmaka.

Polja se postavljaju u scenu po redovima i svaki drugi red mora biti pomaknut za 0.5 na osi x da se izbjegne preklapanje redova polja. U pseudokodu izgleda to ovako:

```
float pomak = -velicinaY / 4f;
for y = 0; y < velicinaY; y++:
      for x = 0; x < velicinaX; x++:
            Polje trenutnoPolje = new trenutnoPolje();
            TipPolja tp= tipoviPolja[poljaKarte[x, y]];
            Vector3 novaRealPozicija;
            trenutnoPolje.sprite = tp.sprite;
            trenutnoPolje.boja = tp.boja;
            trenutnoPolje.pozicija = new Vector2Int(x, y);

            if (y % 2 == 0)
                  novaRealPozicija = new Vector3(x, y + pomak, 2);
                  else:
                  novaRealPozicija = new Vector3(x + 0.5f, y + pomak, 2);
                  trenutnoPolje.realnaPozicija = novaRealPozicija;
            trenutnoPolje.roditelj = this;
            prikaziPolje(trenutnoPolje);
      pomak -= 1f / 4f;
```

Unutar druge petlje se definira izgled polja prema tipu polja, gdje u sceni se to polje nalazi, gdje se u koordinatnom sustavu karte određeno polje nalazi i za veću preglednost u hijerarhiji scene se postavlja karta kao roditelj svakog polja.

Treći korak za generiranje karte je generiranje grafa za rad s putanjama. Graf se sastoji od dvodimenzionalnog niza čvorova iste veličine kao i sama karta. Još jedna sličnost s prijašnjim koracima je pomak koji se računa za realnu poziciju čvora unutar scene igre.

Ovaj korak se može podijeliti u dva dijela: deklariranje novih čvorova i definiranje susjeda za svaki čvor. Ta dva dijela su odvojena jer inače bi za trenutnog čvora vrijedilo da su neki ili čak svi susjedi null vrijednosti, jer čvorovi koji slijede poslije trenutnog nisu deklarirani.

Prvi dio osim deklaracije čvora postavlja poziciju na karti i poziciju u sceni. Taj dio pseudokoda izgleda ovako:

```
graf = new Cvor[velicinaX, velicinaY];
float pomak = -velicinaY / 4f;
for y = 0; y < velicinaY; y++:
      for x = 0; x < velicinaX; x++:
            graf[x, y] = new Cvor();
            graf[x, y].pozicija = new Vector2Int(x, y);

            if y % 2 == 0:
                  graf[x, y].realnaPozicija = new Vector3(x, y + pomak, 2);
                  else:
                  graf[x, y].realnaPozicija = new Vector3(x + 0.5f, y +
            pomak, 2);
            pomak -= 1f / 4f;
```

Drugi dio definira susjede za svaki čvor u grafu. Treba napomenuti da pozicija susjeda nije isti u svakom redu, npr. ako je čvor u neparnom redu, tada su susjedni čvorovi u redovima pokraj ovog za jedan iza po osi x, dok za čvorove u parnom redu susjedi su na osi x za jedan naprijed. Ta situacija je nastala jer redovi nisu raspoređeni točno jedan iznad drugog, nego je svaki parni red pomaknut prema desno da se redovi ne poklapaju jedan s drugim.

Svaki čvor ima najmanje 2 susjeda, a najviše 6 susjeda:

```
for x = 0; x < velicinaX; x++:
      for y = 0; y < velicinaY; y++:
            if x > 0: graf[x, y].susjedi.Add(graf[x - 1, y]);
            if x < velicinaX – 1: graf[x, y].susjedi.Add(graf[x + 1, y]);

            if y % 2 == 1:
                  if y > 0: graf[x, y].susjedi.Add(graf[x, y - 1]);
                  if y > 0 && x < velicinaX – 1:
                        graf[x,y].susjedi.Add(graf[x +1, y - 1]);
                  if y < velicinaY – 1:
                        graf[x,y].susjedi.Add(graf[x, y + 1]);
                  if y < velicinaY - 1 && x < velicinaX – 1:
                        graf[x, y].susjedi.Add(graf[x + 1, y + 1]);
            else
                  if y > 0: graf[x, y].susjedi.Add(graf[x, y - 1]);
                  if y > 0 && x > 0:
                        graf[x, y].susjedi.Add(graf[x - 1, y - 1]);
                  if y < velicinaY – 1:
                        graf[x, y].susjedi.Add(graf[x, y + 1]);
                  if y < velicinaY - 1 && x > 0:
                        graf[x, y].susjedi.Add(graf[x - 1, y + 1]);
```

Četvrti i zadnji korak u generiranju karte je pozicioniranje jedinica za početak igre. Pozicioniraju se dvije jedinice, i to kraljevi, jedan za svakog igrač. Za plavog igrača, kralj se pozicionira u donjem lijevom kutu, gdje mu je i baza, a za crvenog igrača, kralj se pozicionira u gornjem desnom kutu, gdje je njegova baza. Nakon pozicioniranja, proces generacije karte završava i igra može započeti.

6.5. Mehanika poteza

Mehanika poteza se sastoji od jedne klase koja prati nekolicinu informacija kao što su resursi igrača, čiji je potez, ako je opcija za AI igrač uključena i sl. Usto prati izmjene poteza i daje kontrolu igraču kojem je trenutni potez i prati ako je jedan od kraljeva eliminiran.

Jedno od važnijih funkcija za koju je zadužena klasa je izmjena poteza, tj. prebacivanje na sljedeći potez:

```
if (zauzet) return;

potez++;

if potez % 2 == 1 && AIDostupan:
      onesposobiGumbSljedeciPotez();
else:
      osposobiGumbSljedeciPotez();
```

Zauzet je bool varijabla koja je aktivna, tj. istinita ako se neka jedinica kreće po karti. Ako je zauzet aktivan, tada se promjena poteza ne izvršava. Sljedeće što provjerava je ako je uključen AI igrač i ako je njegov potez i pri zadovoljavanju tih dva uvjeta se onesposobi gumb koji služi kao okidač za ovu funkciju.

```
bool plavi = false;
Jedinica sveJedinice;
Jedinica protivnickeJedinice;
ocistiInfotext();
deselektirajSveJedinice();
if (potez % 2 == 0)
      sveJedinice = preuzmiJedinice("Plavi");
      protivnickeJedinice = preuzmiJedinice("Crveni");
      plavi = true;
else
      sveJedinice = preuzmiJedinice("Crveni");
      protivnickeJedinice = preuzmiJedinice("Plavi");

foreach jedinica in protivnickeJedinice:
      jedinica.ocistiPutanju();
foreach jedinica in sveJedinice:
      if cordSela.Contains(jedinica.pozicija) && plavi:
                  plaviZlato++;
                  if jedinica.HP < jedinica.maxHP && (jedinica.maxHP -
jedinica.HP) > 1:
                  jedinica.HP += 2;
                  elif jedinica.HP < jedinica.maxHP && (jedinica.maxHP -
jedinica.HP) == 1:
                  jedinica.HP += 1;
            if cordSela.Contains(trenutna.pozicija) && !plavi:
                crveniZlato++;
                if (jedinica.HP < jedinica.maxHP && (jedinica.maxHP -
jedinica.HP) > 1)
            jedinica.HP += 2;
                elif (jedinica.HP < jedinica.maxHP && (jedinica.maxHP -
jedinica.HP) == 1)
            jedinica.HP += 1;

            jedinica.osvjezi();

            if sveJedinice[sveJedinice.Length - 1] == jedinica:
                  if plavi: jedinica.deselect("Crveni");
                  else: jedinica.deselect("Plavi");

if AIDostupan && potez % 2 == 1: AIOperacije.potez();
```

Nakon inicijalizacije potrebnih varijabli, ovisno o tome čiji je sljedeći potez preuzimanje svih jedinica i čišćenje preostalih putanja koje su ostale spremljene kod jedinica drugog igrača, deselektira jedinice i očisti sve informacije na ekranu vezane uz jedinicu, funkcija prolazi kroz sve jedinice, provjerava ako su u selu i ako jesu poveća resurs za 1 i liječi jedinicu za 2, osvježava jedinicu da se može ponovno kretati po karti. Pri kraju, ako je AI igrač aktivan, tada se pokreće mehanika AI igrača za izvođenje svojeg poteza.

6.6. Mehanika izvođenja poteza za AI igrača

Mehanika izvođenja poteza za AI igrača se sastoji od 3 glavne akcije koje može igrač izvršiti:

- Prebroji jedinice
- Kreiraj jedinice
- Strateški rasporedi jedinice

Prebroji jedinice samo stavlja referencu na sve svoje jedinice koje su na karti za daljnje korištenje. Ta akcija se uvijek prvo izvršava.

Kreiraj jedinice se na početku ili na kraju poteza, ovisno o tome ako je kralj jedinca jedinice ili AI igrač ima više jedinica u kontroli. Provjerava ako je kralj u bazi i ima resurse za kreaciju novih jedinica i naravno, kreira te jedinice ako zadovoljava ta dva uvjeta.

Zadnja akcija koju može AI igrač izvršiti je strateško pozicioniranje jedinica. Kako pozicionira jedinice ovisi o tome ako je kralj sam ili ima jedinice i što je AI igraču važnije: napasti neprijatelja ili zauzeti selo. AI igrač odlučuje što je važnije pomoću „utility theory“ [8].

Utility theory (eng. *Teorija korisnosti*) je sistem ocjenjivanja određenih akcija gdje se onda odabere akcija čija ocjena je najbolja. Ocjenjivanje akcija ovisi o kontekstu same igre i koriste se različite formule za određivanje te ocjene. Kako se odabire koja je ocjena najbolja isto ovisi o kontekstu igre, gdje najmanja ocjena je možda najbolja ili najveća ocjena je najbolja.

U slučaju ove igre, za određivanje cilja za svaku pojedinu jedinicu pod kontrolom AI igrača, uzima se najmanja udaljenost od mogućeg cilja pomnoženo s statičkom vrijednošću za taj cilj, npr. za selo statička vrijednost je 0.75, dok za neprijateljsku jedinicu je 1. Drugim riječima, veću važnost stavlja na zauzimanju sela nego na napad na neprijatelja.

Sljedeći pseudokod prikazuje implementaciju određivanje cilja za određenu jedinicu:

```
Vector2Int odrediste = new Vector2Int(0, 0);
float distanca = karta.velicinaX * karta.velicinaY + 1;
float novaDistanca = 0;
Jedinica[] igracJedinice = preuzmiIgracJedinice();
Polje[] poljaKarte = preuzmiPoljaKarte();

foreach polje in poljaKarte:
      if tipPolja(polje.pozicija) == 6:
                  if !polje.provjeriZauzeto():
                        novaDistanca = racunajDistancu(polje.pozicija,
trenutnaPozicija);
                              if novaDistanca*tezinaSelo < distanca:
                                    distanca = novaDistanca;
                                    odrediste =poklje.pozicija;

foreach jedinica in igracJedinice:
            novaDistanca = racunajDistancu(jedinica.pozicija,
trenutnaPozicija);
            if distanca > novaDistanca * tezinaNeprijatelj:
                  distanca = novaDistanca;
                  odrediste = postavkeJedinice.pozicija;

return odrediste;
```

Ova funkcija očekuje za ulaz trenutnu lokaciju jedinice na karti, a izbacuje lokacija cilja. Funkcija prolazi kroz sva sela i kroz sve neprijateljske jedinice, i ako nađe cilj čija ocjena je manja od trenutne, koja je na početku površina karte, tada se ta koordinata zamjenjuje s prijašnjim boljim ciljem.

Samo pozicioniranje jedinica se izvršava preko funkcije, čiji pseudokod je prikazan ispod:

```
If AIJedinice.Count() == 1: kreirajRatnike();
bool cekajKralja = false;
SistemPoteza sljedeciPotez = preuzmiSistemPoteza();
```

U slučaju da je samo jedna jedinica na karti(i podrazumijeva se da je ta jedinica kralj), tada AI igrač prolazi kroz akciju kreiranja ratnike. U većini slučajeva, ako je kralj sam na karti, velika je vjerojatnost da je kralj već u bazi i u mogućnosti za kreiranje novih ratnika. Objekt sljedeciPotez je referenca na dio igre koji upravlja potezima, i koristi se za provjeravanje ako se neka jedinica kreće po karti.

```
foreach jedinica in AIJedinice:
            if jedinica.name == "Kralj" && AIJedinice.Count() > 1:
                  Vector2Int baza = preuzmiLokacijuBaze();

                  if (jedinica.pozicija == baza) continue;

                  cekajKralja = true;
                  jedinica.pomakniJedinicu(baza);

                  while cekajKralja:
                              if sljedeciPotez.zauzet: cekajSekundu();
                              else: cekajKralja = false;
                  continue;
```

Petlja prolazi kroz sve jedinice u kontroli AI igrača strateški ih miče. Prikazani blok instrukcija unutar if uvjeta se izvršava kada je selektiran kralj i AI igrač ima više jedinica. Navedeni blok instrukcija postavlja lokaciju baze kao cilj kralju ako kralj nije u bazi.

```
      if provjeriPoziciju(jedinica.pozicija): continue;

            Vector2Int odrediste = odaberiMetu(jedinica.pozicija);
            jedinica.pomakniJedinicu(odrediste);



            while sljedeciPotez.zauzet: cekajSekundu();

kreirajRatnike();
```

Ovaj dio koda provjera ako je jedinica u selu, i preskače tu jedinicu ako je uvjet istinit. Ako nije, tada odabere strateški cilj i počinje se micati prema cilju. Pri kraju, kada su se sve jedinice pomakle, ako ima slobodnog mjesta kod zidina baze, tada kreira nove ratnike.

Kraj poteza se ne aktiviran sve dok navedene 3 akcije nisu do kraja izvršene.

6.7. Mehanika kretanja jedinica

Mehanika kretanja jedinice izvršava se pri odabiru cilja do kojeg selektirana jedinica treba stići. Ova mehanika je najvažniji dio igre, jer omogućava izvršavanje strateških aspekata od strane igrača. Ta mehanika se sastoji od dva dijela glavna dijela: kreiranje putanje i kretanje jedinice po karti.

Nakon provjere ako se već neka jedinica kreće po karti, pri čemu se ne izvršava ova mehanika ako se već neka druga jedinica kreće, počinje s procesom stvaranje putanje po kojoj će se jedinica na karti kretati. Putanja se generira pomoću modificiranog A* (čitaj: A zvijezda; eng. *A star*) algoritma.

A* algoritam je računalni algoritam korišten za pretraživanje najkraće putanje između dva čvora unutar grafa. A* algoritam je baziran na Dijkstrinom algoritmu s razlikom da A* star algoritam koristi heuristički pristup što kao rezultat ubrzava izvođenje algoritma nasuprot Dijkstrinom algoritmu[9].

Funkcija za kreiranje putanje ima dostupno početnu pozicija, spremljena kao Vector2Int pocetak, odredišnu poziciju isto spremljenu kao Vector2Int, graf koji reprezentira kartu i bool vrijednost imena zaobilaziSve čija zadana vrijednost je false i koja služi kao okidač za drugačije računanje putanje. Taj okidač se aktivira specifično kod AI igrača ako se određena jedinica u prijašnjem potezu nije pomakla iz svoje početne pozicije. Opis točno što ta bool vrijednost znači u kontekstu kreiranja putanje je opisano u dijelu pseudokoda vezanoh uz računanje duljine između dva čvorova.

Pseudokod za kreiranje putanje algoritam se može podijeliti u tri dijela:

- Inicijalizacija
- A* algoritam
- Dodatne izmjene nakon izvršenja A* algoritma

Prvi dio kreiranja putanje je inicijalizacija, koja priprema sve potrebno za izvršenje A* algoritma. To uključuje inicijalizaciju reda čekanja s prioritetom, početni čvor i završni čvor, asocijativna lista koja sadrži cijene ulaska za čvor i vezana lista koja za određeni čvor sadrži referencu prijašnjeg čvora u putanji. To u pseudokodu izgleda ovako:

```
Cvor[] redCekanja = new PriorityQuery(Cvor);
Cvor[] putanja = new Cvor[];
Cvor start = graf[pocetak.x, pocetak.y];
Cvor cilj = graf[odrediste.x, odrediste.y];
Cvor trenutni = null;
Dictionary duljina = new Dictionary(Cvor, float);
Dictionary prijasnji = new Dictionary(Cvor, Cvor);
redCekanja.Add(start, 0);
foreach v in graf:
      duljina[v] = infinity;
      prijasnji[v] = null;
```

Na kraju se još u red čekanja dodaje početni čvor i postavlja se da svi čvorovi za sada imaju cijenu ulaska jednaku beskonačno i nemaju referencu na prijašnji čvor.

Sljedeći dio je sam A* algoritam. Ovaj dio se izvršava sve dok red čekanja nije prazan ili dok A* algoritam ne dođe do odredišnog čvora. Sljedeći korak u A* algoritmu je računanje cijene ulaska u susjedne čvorove trenutnog čvora. Cijena ulaska u čvor se računa tako da se trenutna ukupna cijena putovanja zbroji s cijenom ulaska u čvor. Cijena ulaska čvora je jednaka cijeni ulaska u polje čija pozicija je jednaka čvoru. U tu sumu se još dodaje vrijednost 0 ili beskonačno, ovisno o tome ako je to polje zauzima neprijateljska jedinice ili ne.

Iznimke toga su ako je čvor kojeg razmatra zapravo odredište, pri čemu tada ignorira neprijateljske jedinice i ako je bool vrijednost zaobilaziSve postavljena na true, gdje se tada uzimaju u obzir ne samo neprijateljske jedinice, nego i vlastite jedinice.

Ako je nova suma cijena je manja od dosadašnje sume cijena, tada se za taj čvor postavlja ta cijena, računa se prioritet, koji je zapravo suma nove cijene putovanja i „fizičke“ udaljenosti između trenutnog susjednog čvora i cilja, postavlja se taj susjedni čvor u red čekanja s tim izračunatim prioritetom i dodaje se trenutni čvor u vezani listu.

U pseudokodu, ovaj gore navedeni opis izgleda ovako:

```
while redCekanja.NotEmpty():
      trenutni = redCekanja.Get();
      if(trenutni == cilj) break;
      foreach v in trenutni.susjedi:
            if v == cilj:
                  novaDuljina = duljina[trenutni]  + cijenaKretanja(v.x,
            v.y);
            elif zaobilaziSve:
                  novaDuljina = duljina[trenutni] + cijenaKretanja(v.x,
            v.y) + zauzetoPolje(v.x,v.y,"Plavi") +
            zauzetoPolje(v.x,v.y,"Crveni");
            else:
                  novaDuljina = duljina[trenutni] + cijenaKretanja(v.x,
v.y) + zauzetoPolje(v.x,v.y, neprijateljskiTim);
            if novaDuljina < duljina[v]:
                  duljina[v] = novaDuljina;
                  float prioritet = novaDuljina + cilj.fizickaDistanca(v)
                  redCekanja.Add(v, prioritet);
                  prijasnji[v] = trenutni;
```

Treći dio u procesu kreiranje putanje je prebacivanje vezane liste u samu putanju, koja je sama po sebi obična lista, za potrebe ostatka igre i izmjene te putanje za specifične stanje koje se može pojaviti na karti:

```
if prijasnji[cilj] == null: return null;
trenutni = cilj;
while trenutni != null:
      putanja.Add(trenutni);
      trenutni = prijasnji[trenutni];

while brTrenPokreta < putanja.Count – 1: putanja.RemoveAt(0);

if putanja.Count - 1 < 1: return null;

putanja.Reverse();
```

Putanja se nakon prebacivanja s vezane liste preobraća, jer inače zbog vezane liste koja započinje od cilja, putanja za jedinicu bi započela kod cilja i kretala bi se prema početnoj poziciji. Usto se miču čvorovi putanje koji su u tom trenutku nedostižni jedinici.

Sljedeća provjera kod koje može doći do modifikacije putanje je u slučaju gdje je cilj zauzet neprijateljskom jedinicom, što znači da napada tu neprijateljsku jedinicu. Gleda se ako čvorovi, tj. polja koji prethodne odredišnom polju zauzimaju prijateljske jedinice i ako je istinito, traži susjedna polja preko kojeg bi onda napao neprijateljsku jedinicu.

Zadnja izmjena koju radi je u slučaju da novo odredište jedinice nakon svih izmjena ima već prijateljsku jedinicu koja okupira to polje. Tada se jednostavno se miču čvorovi s putanje sve dok se ne dođe do prvog odredišnog čvora čiji ekvivalentno polje nije okupirano prijateljskom jedinicom.

Pretočeno u pseudokod izgleda ovako:

```
if putanja.Count - 1  > 1:
      Cvor v = putanja.last();
      Putanja.Remove(v);
      Cvor susjed = putanja.Last();
      float tempD = infinity;
      Cvor tempC = null;
      bool pogreskaCalPut = false;

      if(zauzetoPolje(susjed.pozicija, vlastitiTim) == infinity &&
zauzetoPolje(cilj, protivničkiTim) == infinity):
            foreach u in susjed.susjedi:
                        if  zaobiciJedinicu(u.pozicija) == 0 && tempD > u.
fizickaDistanca (v) && tipPolja(u.pozicija) != 3:
                              foreach w in u.susjedi:
                                    if(w == v):
                                          pogreskaCalPut=false;
                                          break;
                                    pogreskaCalPut = true;
                              if(pogreskaCalPut) continue;
                              tempD = u.fizickaDistanca(v);
                              tempC = u;

                              if u.fizickadistanca(Start) == 1:
                                    promjeniPut = true
                              else: promjeniPut = fasle;

            if promjeniPut:
                  putanja.Remove(susjed);
                  putanja.Insert(putanja.Count, tempC);
            elif tempC != null: putanja.Insert(putanja.Count, tempC);
```

```
putanja.Insert(putanja.Count, v);

while izracunajVrPutanje() > brTrenpokreta:
      putanja.Remove(putanja.last());
 if putanja.Count > 1:
      while zauzetoPoljeVTim(putanja.last().pozicija):
            putanja.Remove(putanja.last(());
            if putanja.Count < 1: break;
```

Drugi dio mehanike kretanja jedinica po karti je sam proces koji vizualno prikazuje kako jedinica prolazi kroz kartu do svojeg odredišta. To ostvaruje tako da uzima prvi čvor iz putanje, uzima njegove koordinate i pomakne jedinicu na te koordinate. To radi sve dok ne prođe kroz sve čvorove u putanji ili mu ponestane pokreta koje bi mogao potrošiti na ulazak u polje ili ako jedinica slučajno umre dok prolazi kroz močvaru ili ako napada neprijatelja.

Napad na neprijateljsku jedinicu se odvija kroz par manjih koraka. Pro provjerava ako je neprijatelj uspješno blokirao napad. Blokiranje napada radi na način da se uzme nasumična vrijednost od 0 do 100 i ako je ta vrijednost veća od blok vrijednosti neprijateljske jedinice, tada je napad uspješan i računa se šteta napravljena prema neprijateljskoj jedinici. Ako je uspješno blokirano, tada se ne računa napad. Šteta se računa tako da se uzmi nasumičan broj koji je u rasponu napada jedinice i taj se broj oduzima s vrijednošću obrane jedinice koja se brani od neprijatelja. U slučaju da jedinica eliminira neprijatelja, tada zauzima to polje koje je bilo prije okupirano, inače ostaje na onom polju od kojeg je započeo sam napad. Tokom napada, jedinica koja napada ne prima štetu.

7. Zaključak

Strategija na poteze upotrebljava iznimno pun broj zanimljiv mehanika. Mehanike koje se mogu naći i u drugim žanrovima, ako ne u cijelosti, onda parcijalno implementirane unutar neke druge mehanike. Od generiranje karta ili reljefa u 3D prostoru, do načina ponašanja umjetne inteligencije u igrama, dosta tih mehanika iz drugi žanrova se mogu povezati s onima unutar strategija na poteze.

Te sličnosti samo potvrđuju koliko je zapravo relevantan ovaj žanr i koliko se može naučiti od njega.

Popis literature

[1] Unity (bez dat.) *Unity3D* [Na Internetu]. Dostupno: https://unity3d.com/ [pristupano 10.09.2018.]

[2] “Unity (game engine),“ (bez dat.) u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/Unity_(game_engine) [Pristupano 09.09.2018.]

[3] J. Haas, “A History of the Unity Game Engine“ (bez dat.) Dostupno: https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf [Pristupano 09.09.2018.]

[4] “List of Unity games,“ (bez dat.) u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/List_of_Unity_games [Pristupano 09.09.2018.]

[5] “Strategy video game,“ (bez dat.) u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/Strategy_video_game [Pristupano 07.09.2018.]

[6] “List of turn-based strategy video games,“ (bez dat.) u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/List_of_turn-based_strategy_video_games [Pristupano 07.09.2018.]

[7] “Data Structure - Priority Queue,“ (bez dat.) u *Tutorials Point*. Dostupno: https://www.tutorialspoint.com/data_structures_algorithms/priority_queue.htm [Pristupano 04.09.2018.]

[8] D. Graham, “An Introduction to Utility Theory,“ (bez dat.) Dostupno: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf [Pristupano 04.09.2018.]

[9] “A* search algorithm,“ (bez dat.) u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/A*_search_algorithm [Pristupano 03.09.2018.]

Popis slika

Popis tablica

Prilog 1 – Prototip igre

Uz rad se prilaže prototip igre koji služi kao praktični dio ovog rada. Prototip igre je komprimiran u .rar datoteku i sadrži sve datoteke koje se koriste unutar igre. Taj unity projekt možete uvesti u Unity alat i koristeći Unity alat pokrenuti igru.