Izrada 3D igre s reaktivnom inteligencijom kod entiteta u Unity okruženju

Marić, Josip

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

JOSIP MARIĆ

IZRADA 3D IGRE S REAKTIVNOM INTELIGENCIJOM KOD ENTITETA U UNITY OKRUŽENJU

Završni rad

Pula, rujan, 2019. godine

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

JOSIP MARIĆ

IZRADA 3D IGRE SA REAKTIVNOM INTELIGENCIJOM KOD ENTITETA U UNITY OKRUŽENJU

Završni rad

JMBAG: 0303068646, redoviti student
Studijski smjer: Sveučilišni preddiplomski studij informatika
Kolegij: Napredne tehnike programiranja
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc. Tihomir Orehovački

Pula, rujan 2019. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisan, Josip Marić, kandidat za prvostupnika informatike 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 ni 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

Josip Marić

U Puli, rujan 2019. godine

IZJAVA

o korištenju autorskog djela

Ja, Josip Marić, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom:

„Izrada 3D igre sa reaktivnom inteligencijom kod entiteta u Unity okruženju“ koristi tako 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, rujan, 2019. godine

Potpis

Josip Marić

Sažetak

Ovaj rad opisuje koncept razvoja 3D igre s reaktivnom inteligencijom kod entiteta u Unity okruženju. Igra se sastoji od nekoliko razina s definiranim pravilima igre u kojoj se kontrola čovječuljka percipira iz ptičje perspektive. U radu se definira reaktivna inteligencija, ukratko se prolazi kroz osnove Unity programa, nakon čega se opisuje izrada igre (opis igre, organizacija komponenata, itd.). Detaljnije se pojašnjavaju izrade i ponašanja reaktivne inteligencije kod entiteta (neprijatelja) te se daje prikaz njegovog djelovanja kroz Petrijeve mreže.

Ključne riječi: *Reaktivna Inteligencija, Unity, 3D, igra, entitet, PC.*

Abstract

Bachelor thesis that describes the concept of game development with reactive intelligence in entities in the Unity environment. The game consists of several levels with defined rules of the game in which human control is perceived from a bird's eye view. The paper defines reactive intelligence, briefly goes through the basics of the Unity program, followed by the development of the game (description of the game, organization of components, etc.), and a more detailed explanation of the creation and behavior of reactive intelligence in entities (enemies), and a sequence of its operation through the Petri net.

Keywords: *Reactive Intelligence, Unity, 3D, game, entity, PC.*

Sadržaj

1. Uvod

Tema ovog završnog rada je izrada 3D igre u Unityju s reaktivnom inteligencijom kod entiteta. Zadatak je bio ostvariti dojam pametnog ponašanja kod računalno kontroliranog entiteta korištenjem reaktivnih tehnika. Igra je izrađena za PC, ali ju je moguće igrati na Android uređajima, iako nije preporučljivo zbog visokih grafičkih zahtjeva. Igra se sastoji od nekoliko razina, svaka razina sa svojim pravilima igre. Pristup svim razinama je jednak te se pri uspješno osvojenoj razini mijenja boja komponente u zelenu i zapisuje se najbolji postignuti rezultat, što postoje vidljivo i u izborniku za odabir razine. Time se postiže želja za ponovnom igrom iste razine. Pogled na razinu je iz ptičje perspektive, a kontrola čovječuljka je osnovna (WASD); svaka tipka ima kretanje u svojem smjeru te kombinacija tipki dopušta kretanje ukoso. Također, igrač se može kretati i ispod zemlje, ali prijelaz je moguće ostvariti jedino na određenim mjestima. Igrač se treba kretati mapom, tražeći sakrivene bodove, izbjegavati neprijatelje i eliminirati njihove baze. Pronalazak sakrivenih bodova se ostvaruje igrom toplo/hladno.

Ostali dijelovi ovog rada su osnove o Unity programu te detaljni prikaz o reaktivnoj inteligenciji, reaktivnim tehnikama, arhitekturi i razlici između teorije i prakse ideje reaktivne inteligencije. Također će se detaljno objasniti rad i ponašanje entiteta (neprijatelja) te organizaciju komponenti i objekata igre.

2. Umjetna inteligencija

Svijet je do sada upoznat s napretkom umjetne inteligencije u svim dijelovima društva te je na ljude ostavljen dojam da su na pragu generalne umjetne inteligencije [2].

Tipovi umjetne inteligencije [2]:

- reaktivni strojevi (prvi tip, eng. *reactive machines*)
    - Najjednostavniji sustavi umjetne inteligencije su čisto reaktivni. Nemaju mogućnost formiranja memorije ili korištenja prijašnjih iskustava. Iako nemaju mogućnost učenja, svejedno su sposobni te je tako IBM-ov „Deep Blue“ algoritam pobijedio međunarodnog majstora šaha, Garryja Kasparova u kasnim 1990-im godinama.
- ograničena memorija (drugi tip, eng. *limited memory*)
    - Takvi sustavi imaju mogućnost sadržavanja memorije kao reference u budućim odlukama te se koriste u pametnim automobilima (identificiraju i prate ostala vozila prilikom promjene brzine i smjera).
- teorija uma (treći tip, eng. *theory of mind*)
    - Trenutno nedostižni, takvi strojevi će moći razumijevati okolinu oko sebe tako da će shvaćati da i ostali entiteti koje očitavaju imaju mogućnost razmišljanja.
- samosvijest (četvrti tip, eng. *self-awareness*)
    - Ovo je nadogradnja na prijašnji tip teorije uma, a razlika je u tome da će stroj biti svjestan samog sebe, odnosno, moći će opravdati svoje akcije.

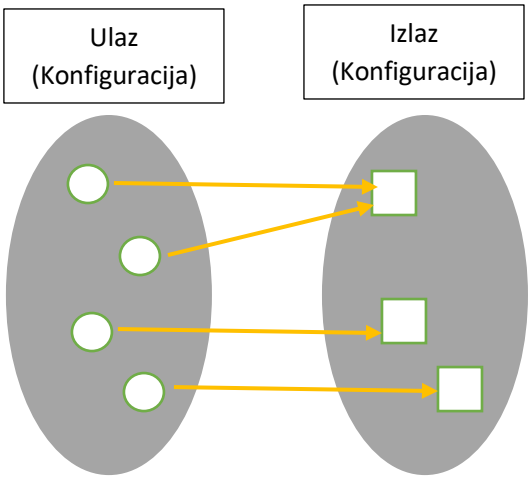
Naime, pomoću strojnog učenja roboti imaju mogućnost razumijevanja verbalnih komandi, razlikovanja slika, upravljanje automobila te čak imitiranja govora, kreiranja literature i skladanja glazbe. Problem takve inteligencije je njezina nepredvidivost i netransparentnost, što je čini nepraktičnom u dizajniranju video igara, zbog čega se više preferiraju umjetne inteligencije prvog i drugog tipa.

2.1 Reaktivna inteligencija

Reaktivna Inteligencija se već dugo koristila u igrama kao što su Doom i Super Mario, pa čak i u robotima poput Deep Blue. Upravo je zbog tog globalnog iskustva i kvalitetno istraženog područja proizašla teorija za Reaktivnu inteligenciju [1].

2.1.1 Teorija

Svaka AI komponenta za isti ulaz (eng. *input*) ima isti izlaz (eng. *output*), neovisno o ponašanju ili tehnici. Navedeno možemo proučavati kao matematičku funkciju; svaka vrijednost u domeni (ulaz) je pridružena točno jednoj vrijednosti u kodomeni (izlaz). Kombinaciju ulaza i izlaza možemo gledati kao jedan tip konfiguracije, odnosno deterministično mapiranje (eng. *deterministic mapping*, Slika 1).



Slika 1 Deterministično mapiranje [1]

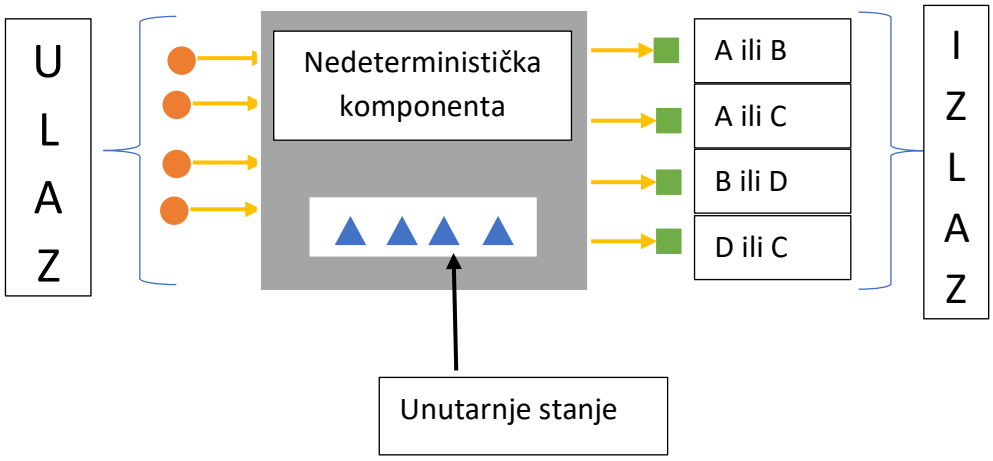
Iz navedenog, takozvane „više-na-jedan“ funkcije zaključuje se da jedan ulaz nikad ne može rezultirati s više od jednog izlaza. S obzirom na to da se na sustave promatra iz perspektive komponenata, svaki je izlaz predvidljiv, a takve se komponente nazivaju deterministične.

Drugim riječima, reaktivna komponenta će uvijek jednako reagirati u istoj situaciji. Takva predvidljivost je poželjnija kod programera i dizajnera igara.

2.1.2 Praksa

Navedenom definicijom dolazi se do problema ako ju se primjeni doslovno. Naime, po teoriji, reaktivna komponenta je samo funkcija i nema unutarnjeg stanja ili memorije. Alternativno determinističko mapiranje ne bi bilo moguće. Primjer nedeterminističke komponente (Slika 2).

Kako je riječ o interpretaciji, ako se unutarnje stanje komponente protumači kao još jedan od ulaza, komponenta se ponovno može promatrati kao reaktivna konfiguracija (ulaz->izlaz, Slika 3).



Slika 2 Nedeterministička komponenta [1]

Slika 3 Deterministička komponenta [1]

2.1.3 Reaktivne tehnike u razvoju igara

Reaktivne AI tehnike (eng. *Reactive AI Techniques*), ponekad zvane refleksivne tehnike, bitan su dio igara još od njihovog začetka.

Prednosti Reaktivnih tehnika su sljedeće [1]:

- u potpunosti determinističke,
- omogućavaju optimizaciju koda i strukture podataka,
- otklanjanje grešaka je olakšano zbog preciznijeg utvrđivanja točnog razloga i
- vrijeme izvođenja radnje je pretežno jednako.

Najuspješnije reaktivne tehnike [1]:

- Skripte kao mali programi koji za dane parametre proizvedu rezultat. Koristi se čist programski jezik za rješenje problema (Python, Lua).
- Sustavi temeljeni na zbirki pravila *„ako…onda“* koja se koriste za transformaciju varijabla. Namijenjeni većim problemima, pružaju fleksibilan i modularan pristup uz vrlo mali razvojni trošak. Lošiji kod niže razine kontrole.
- Strojevi s konačnim stanjem mogu se shvatiti kao pravila definirana za ograničeni broj situacija, opisujući što dalje činiti u svakom slučaju. Najčešće korišten u razvoju slijeda animacija (tranzicija).

2.1.4 Arhitektura

Umjetna inteligencija u igrama je najčešće dizajnirana slaganjem malih komponenata. Te komponente međusobno „komuniciraju“ kako bi zajednički izvršile određen zadatak. Arhitektura je ništa drugo nego ugniježđen set takvih komponenata.

Da bi arhitektura bila reaktivna mora ovisno o ulazu proizvesti izlaz deterministički. Često se koristi u sustavima gdje je vrijeme reakcije prioritet, poput perilice za rublje, klima uređaja, automatike, itd.

Cilj arhitekture je definiranje veza između komponenata.

2.1.5 *Komponenta*

Ideju komponente se može zamišljati kao crnu kutiju s ulazima i izlazima, u kojoj se ulaz transformira na „nepoznat“ način. Također, komponenta može biti kombinacija manjih sustava kod kojih se međusobna komunikacija izvršava pomoću ulaza/izlaza.

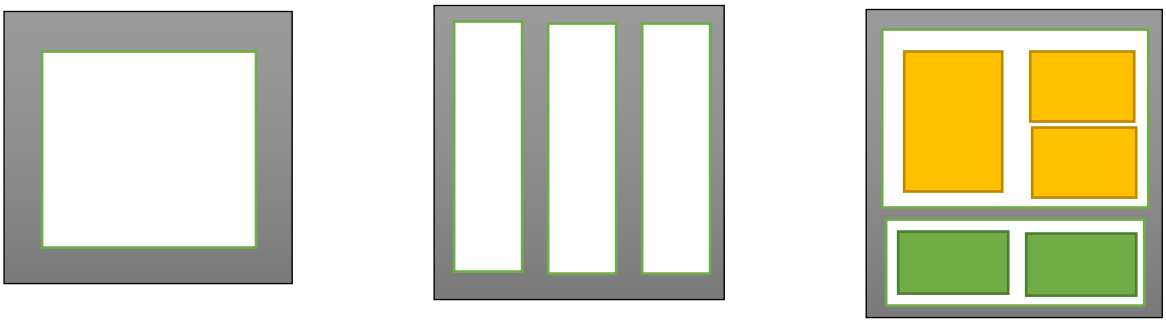
Takvim dizajnom se dolazi do modularnosti umjetne inteligencije, odnosno mogu se koristiti ugniježđene komponente kao jedan sustav.

2.1.6 *Organizacija*

O odabiru organizacije u arhitekturi odlučuje se ovisno o kompleksnosti. Postoji nekoliko načina slaganja komponenti u jedan sustav (Slika 4).

Tri takva tipa su [1]:

- monolitne arhitekture (eng. *monolithic architectures*) uključuju samo jednu komponentu,
- izravnate arhitekture (eng. *flat architectures*) imaju paralelno mnogo komponenti i
- hijerarhijski modeli (eng. *hierarchical models*) imaju komponente ugniježđene unutar drugih.



Slika 4 Od lijeva na desno primjeri arhitektura | monolitna | izravnata | hijerarhijska [1]

Za jednostavne probleme je dovoljno korištenje monolitne arhitekture, dok se teži problemi lakše rješavaju pomoću izravnate arhitekture. Uz Hijerarhijski model moguće je riješiti bilo koji problem tako da se rastavi na dijelove.

Primjerice, dizajniranjem nekog entiteta s umjetnom inteligencijom nad kojim je zahtjev da lovi, patrolira, bježi (prvi set) te zahtjev da u isto vrijeme trči i gađa (drugi set) dolazi do toga da prvi set ovisi o drugom setu. Takva arhitektura se naziva *tri razine hijerarhije*.

2.1.7 Dekompozicija

Da bi se uspješno odabrala arhitektura, prvo se mora postaviti pitanje kako rastaviti zadan problem. Ideja je rastaviti problem ovisno o određenim kriterijima.

Neki od tipova rastavljanja su [1]:

- strukturno rastavljanje (eng. *structural decomposition*)
    - Rastavljanje rješenja ovisno o funkciji svake komponente (npr. trčati, gađati).
- bihevioralna dekompozicija (eng. *behavioral decomposition*)
    - Bazirana na karakterističnim aktivnostima sustava. Za razliku od strukturne, ima procedure umjesto čistih funkcija.
- ciljno rastavljanje (eng. *goal decomposition*)
    - Koristi se ciljem sustava (onim što se želi postići) te na njemu bazira rastavljanje problema.

Također, postoje hibridne dekompozicije kod kojih, primjerice, prvotni problem može biti rastavljen kao ponašanje, a svako je ponašanje rastavljeno na određene funkcije što dozvoljava korištenje više kriterija u odnosu na čiste dekompozicije.

2.1.8 Odlučivanje

Uzevši određenu arhitekturu, potrebno je odabrati način komunikacije između ugniježđenih komponenti ovisno o zadanom problemu.

Poznata su 4 tipa interpretacije izlaza [1]:

- neovisan zbroj (eng. *independent sum*)
    - Povezuje svaku komponentu s različitim izlazima kako ne bi došlo do sukoba.
- kombinacija (eng. *combination*)
    - Omogućuje kombiniranje izlaza različitih komponenata kako bi se dobio konačni rezultat.
- suzbijanje (eng. *suppression*)
    - Neke komponente dobivaju prednost nad drugima.
- sekvencijalno odlučivanje (eng. *sequential*)
    - Prati kako se s vremenom izmjenjuju ishodi različitih komponenti.

Postoje različiti načini dekompozicije problema za izgradnju arhitekture, čime se pojednostavljuje razvoj umjetne inteligencije i njezinih komponenata.

3. Unity3D

U ovom poglavlju će se ukratko opisati osnovne dijelove Unity3D razvojne platforme za kreiranje 3D i 2D igara i drugih interaktivnih sadržaja na više platformi [3][4].

Prednosti Unity-a [3]:

- brzo prototipiranje igre,
- velik odabir izdavačkih platformi (Slika 5),

Slika 5 Platforme podržane od strane Unity [3]

- jedan kod za sve platforme, postoji mogućnost definiranja određenog koda za određenu platformu,
- jednostavan uvoz multimedijskog sadržaja i brza integracija (primjerice dodavanje materijala na model),
- vrlo aktivna zajednica,
- moguća integracija cijelog tima u razvojnu platformu.

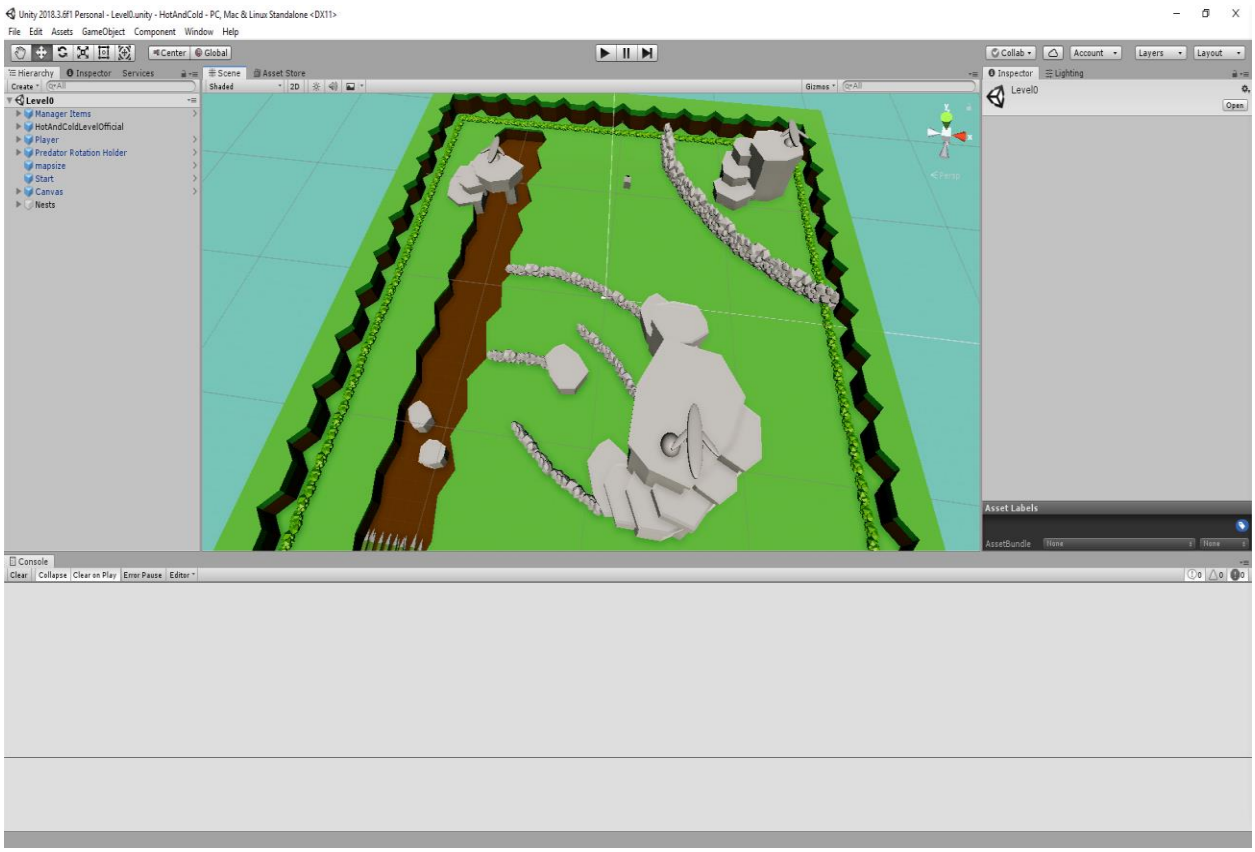
Nedostaci Unityja [3]:

- web platforma WebGL još uvijek nije zrela,
- Unity Webplayer je isključen po osnovnoj konfiguraciji u Chromu,
- u mobilnim/tablet uređajima troše više energije od izvornih aplikacija,
- potrebna timska struktura za pravilno korištenje u većim timovima.

Ukratko, Unity okruženje pruža bilo kome cijeli sustav za poslovnu izgradnju visoko kvalitetnog sadržaja i mogućnost interakcije sa strankama i igračima.

3.1 Unity uređivač

Sadrži poglede i ostale opcije, primarni pogled je scena (eng. *scene*) gdje se igra vizualno uređuje (Slika 6) te pogled igre (eng. *game*) koji prikazuje igru na način koji bi igrač doživio (Slika 7). Za pokretanje igre koristimo gumb „pokreni“, gumb „pauza“ za trenutno stopiranje igre, te gumb „sljedeći“ koji tada pokreće igru okvir po okvir (eng. *frame by frame*). U tijeku rada igre moguće je raditi promjene na sceni, ali te promjene neće biti spremljene.

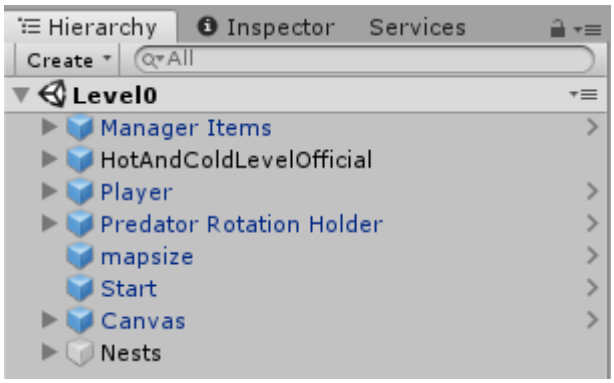


Slika 6 Pogled na Unity uređivač i pogled scena

Slika 7 Pogled igra

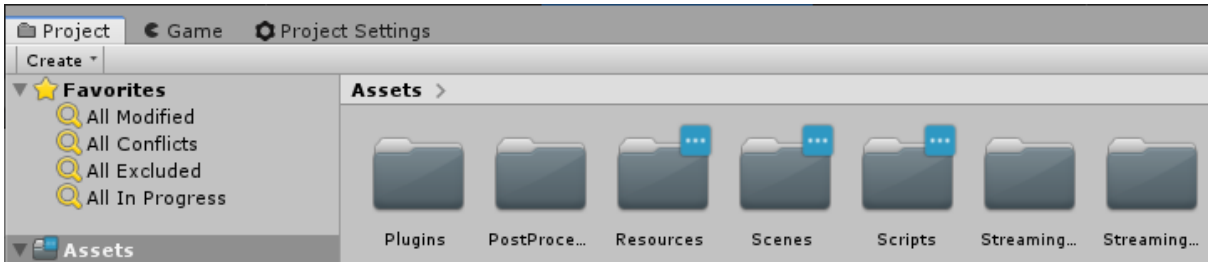
Sekundarni pogledi poput hijerarhije (eng. *hierarchy*, Slika 8), projekta (eng. *project*, Slika 9) i inspektora (eng. *inspector*, Slika 10) služe za podešavanje igre.

Hijerarhija sadrži sve objekte igre na trenutnoj sceni te sve buduće razine koje se mogu kreirati tijekom rada igre. Također, može sadržati više scena tijekom izvršavanja igre.



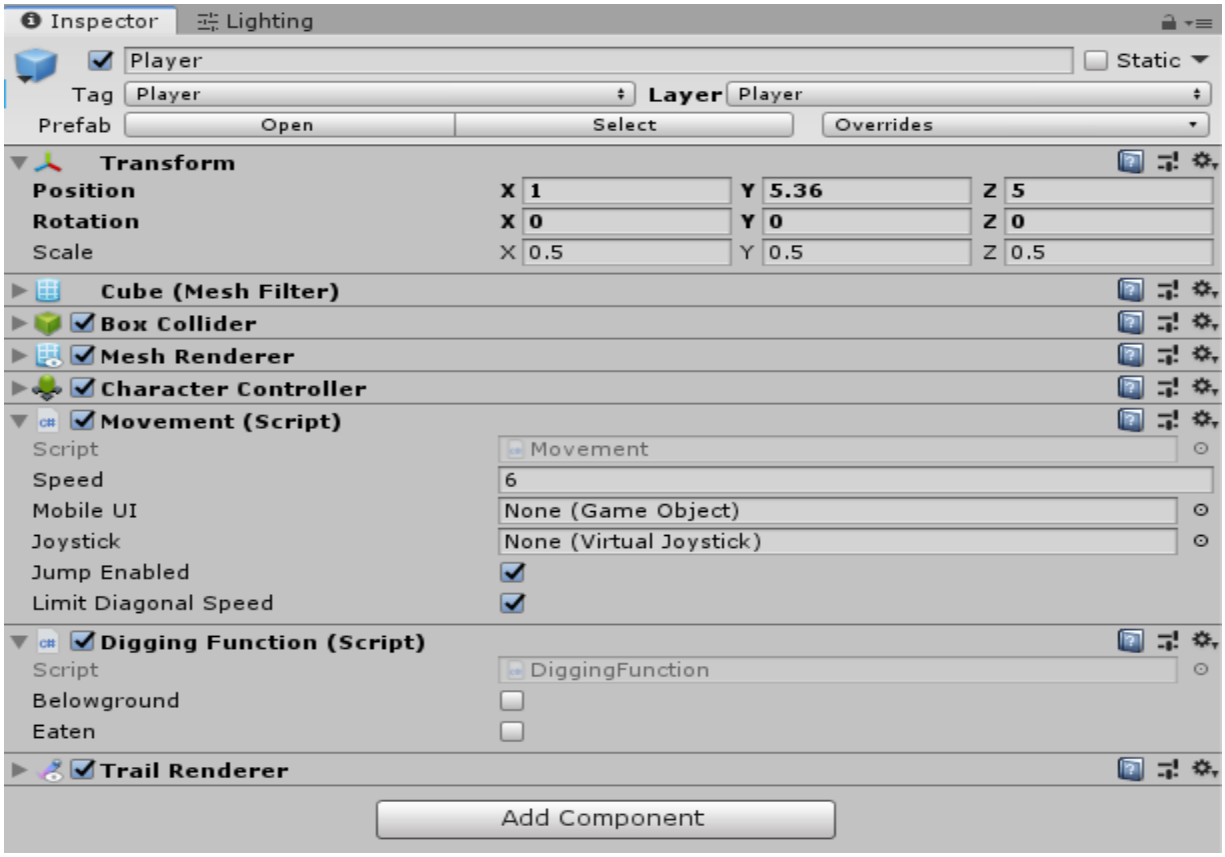
Slika 8 Pogled hijerarhija

Projekt sadrži sve multimedijske dijelove igre, uključujući skripte i datoteke. Također, sadrži scene i montaže (eng. *prefabs*). Unity radi tako da pri osnovnoj konfiguraciji direktan pristup datotekama kroz sve platforme imamo jedino u *„Resources“* mapi.



Slika 9 Pogled projekt

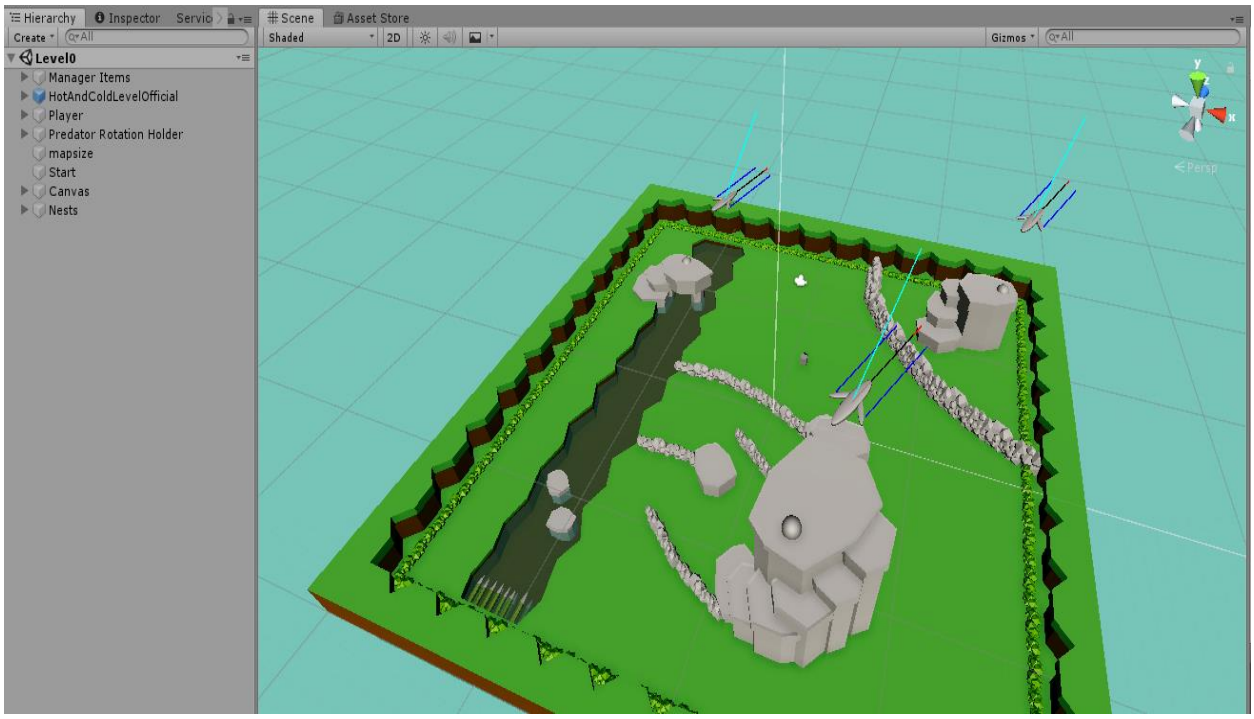
Inspektor se koristi za pregled i podešavanje komponenti objekata igre i ostalih postavki. Omogućuje dodavanje novih komponenti, kopiranja komponenti ili samo njihovih vrijednosti te ponovno postavljanje na prvobitno stanje komponente.



Slika 10 Pogled inspektor

3.2 Scene

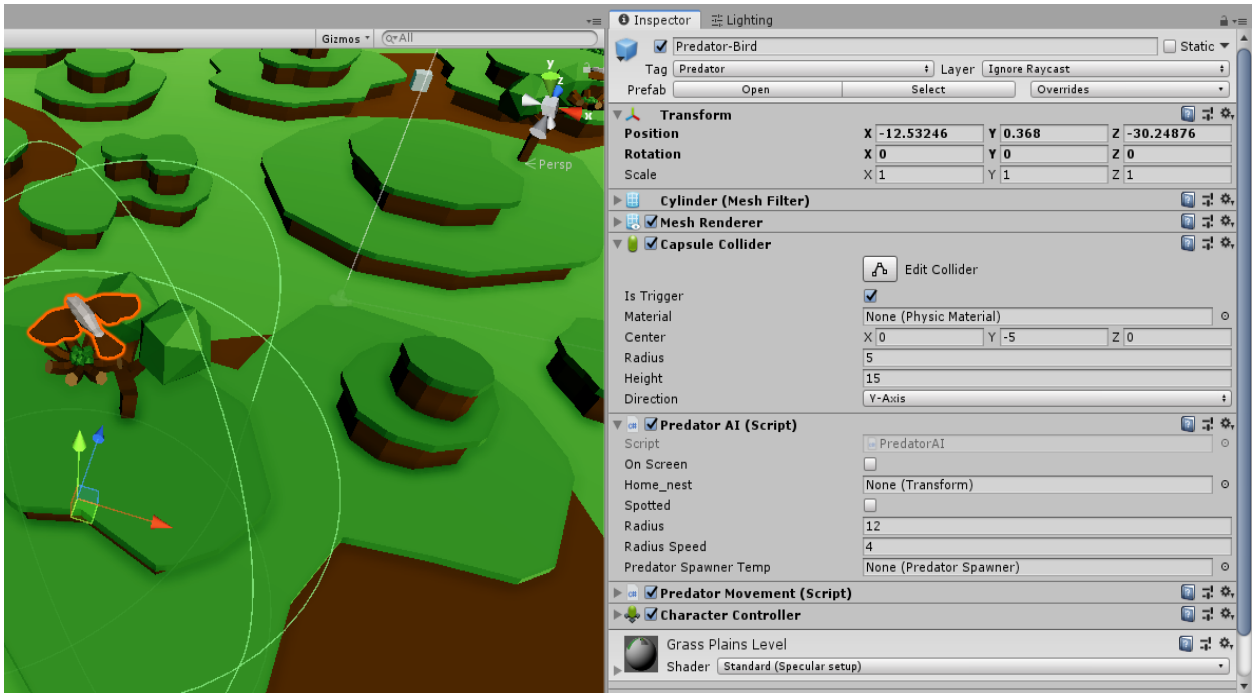
Sadrže okruženje i izbornik igre (Slika 11). Svaka scena je zamišljena kao jedna razina igre, a u svakoj od njih se slaže, dizajnira i postavlja objekte. Scena također sadrži objekte koji nemaju vizualnu reprezentaciju, ali su bitni zbog organizacije ili logike u igri.



Slika 11 Primjer scene

3.3 Objekti igre i komponente

Objekti igre sadrže komponente (Slika 12). Komponente mogu biti skripte, coliders[1], renderers[2], itd. Svaki objekt igre dolazi s transform[3] komponentom ili s rect-tranform[4] komponentom u slučaju UI objekata. Svi uključeni objekti se mogu pronaći u hijerarhijskom pogledu te su poslagani u hijerarhijskoj strukturi, ali u smislu veze otac-dijete.

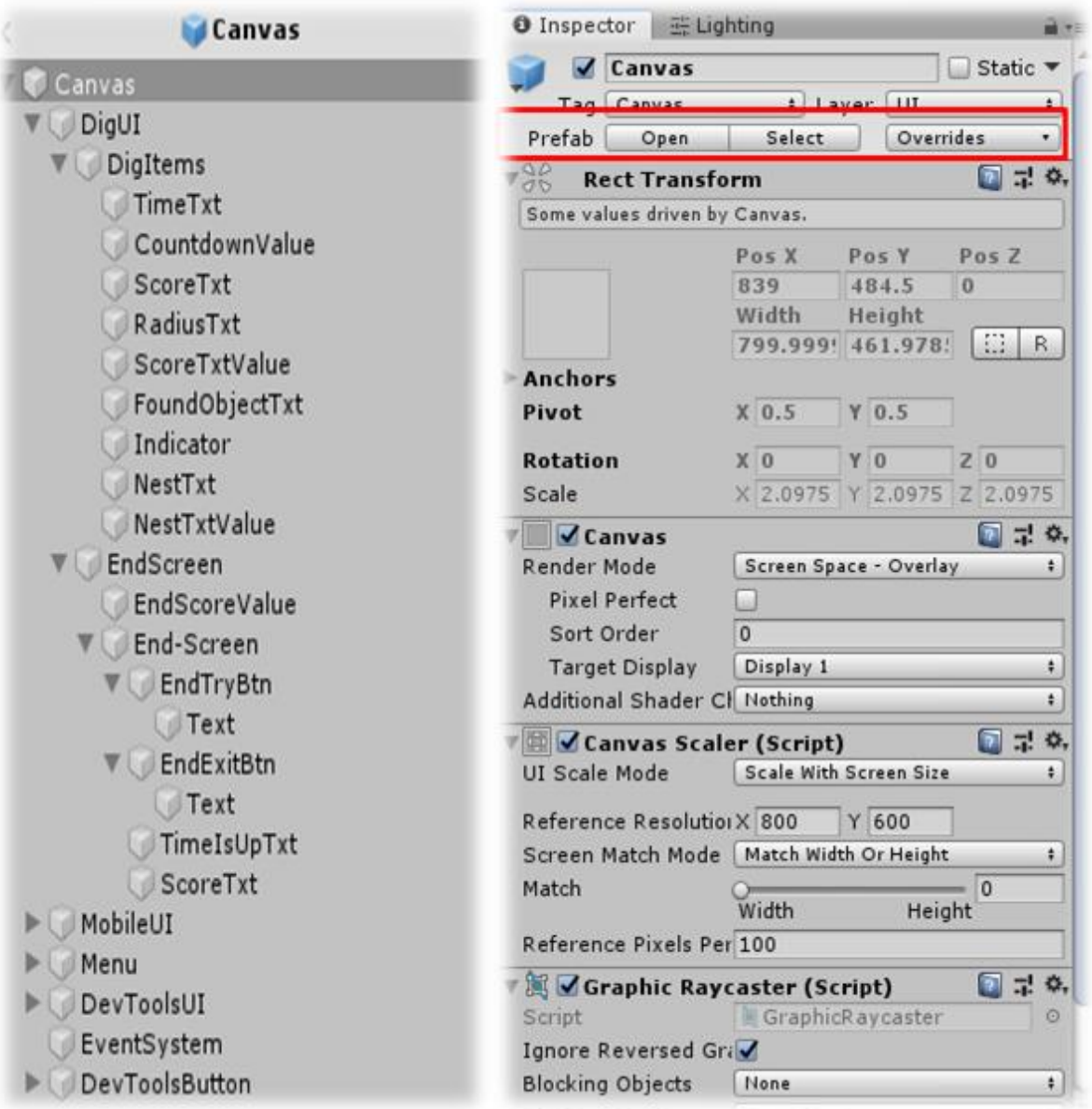


Slika 12 Objekt ptica i njezine komponente

[1] hrv. Kolizija, definiraju oblik za korištenje kod fizičkih sudara [4]
[2] hrv. Renderer, uzima geometriju modela i vizualizira ju ovisno o transform komponenti [4]
[3] hrv. Transformacija, sadrži poziciju, rotaciju i skalu svakog objekta u igri [4]
[4] 2D nacrt transform komponente [4]

3.4 Montaže

Montaže (eng. *prefabs*) su unaprijed konfigurirani objekti igre koji se mogu koristiti kao predlošci u sceni (Slika 13). Mogu se kreirati ili klonirati u sceni ili tijekom igre, a promjene nad jednom montažom se mogu prenijeti na sve njezine inačice.



Slika 13 Montaža u hijerarhiji | Montaža u inspektoru

4. Izrada igre

U prvom dijelu ovog poglavlja ukratko će se opisati rad i cilj igre, organizaciju objekata igre i komponenti, njihovu međusobnu komunikaciju i spremanje/učitavanje iz json baze podataka. U sljedećem poglavlju opširnije će se opisati rad umjetne inteligencije entiteta te primjer izvođenja kroz Petrijeve mreže.

4.1 Opis igre

Nakon pokretanja igre iz operacijskog sustava, igraču se prikazuje početni zaslon igre (Slika 14). Na početnom zaslonu ima odabir započni, zasluge i izlaz. Nakon klika na započni, igraču se prikazuje novi pogled na izbornik za odabir razine koje se generiraju iz json datoteke (Slika 15). Također, tu se nalazi i gumb resetiraj (vraća na početne vrijednosti sve razine, odnosno json datoteke), i gumb natrag (povratak na početni zaslon). Naposljetku, odabirom razine zatvara se izbornik za odabir razine i počinje učitavanje odabrane razine.

Nakon učitavanja, igrač se nađe u kontroli čovječuljka (simbolizira krticu, Slika 16) nad kojim se može kretati po mapi, skakati, kopati, odnosno ići ispod ili iznad zemlje, očitavati lokaciju boda na temelju toplo/hladno (Slika 19) te taj bod iskopati (Slika 20). Potrebno je izbjegavati neprijatelje (simbolizira sokola, Slika 17), a u slučaju da krenu u napad, igrač može jedino pobjeći ili sakriti se ispod zemlje (Slika 18). Igrač ima i mogućnost eliminiranja neprijateljeve baze (simbolizira gnijezdo) čime će se lakše kretati u njezinoj okolini, ali će susjednu bazu biti teže srušiti (detaljnije u drugom poglavlju). Kao i u većini igara, igrač može pauzirati (Slika 21).

4.1.1 Cilj igre

Moguće je definirati cilj između nekoliko ciljeva igre, a on ovisi o postavkama razine. Naime, zbog složenosti, uzaludno je definirati sve moguće kombinacije cilja. Umjesto toga, definirane su samo osnovne koje se naknadno mogu kombinirati, a čime se dobije cilj igre.

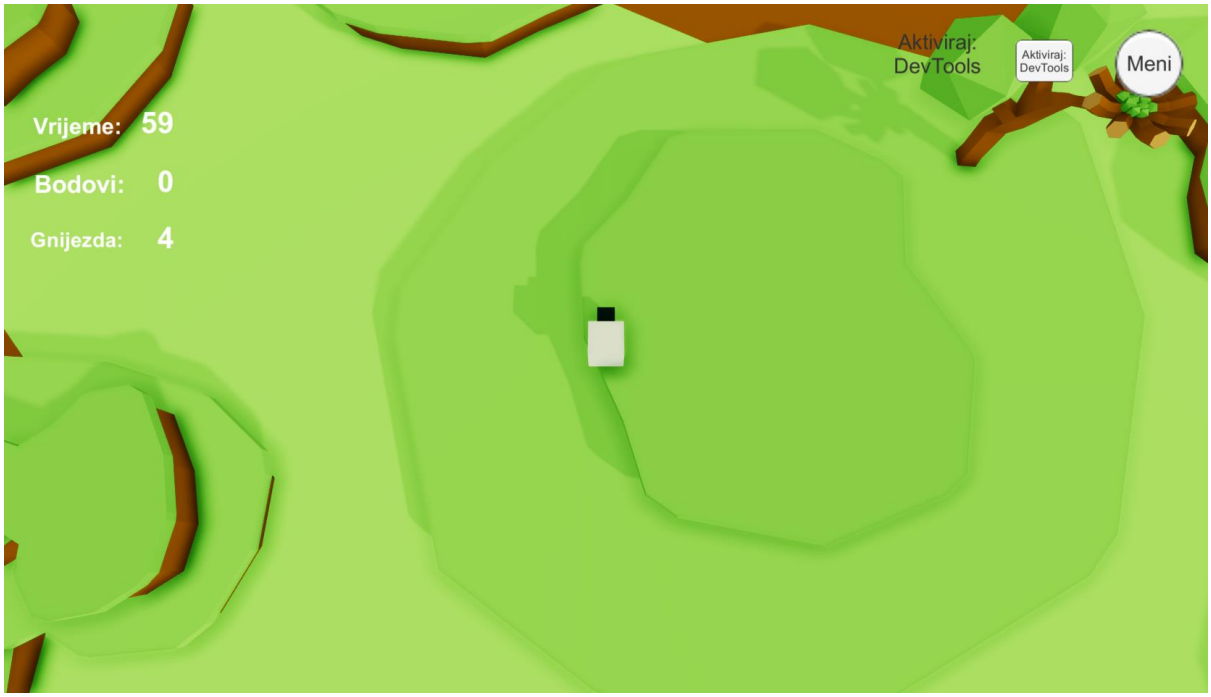
Osnovne mogućnosti cilja:

- skupi x bodova
  - Aktivira se kada je vrijednost broja uvjeta jednaka broju bodova.
- odbrojavanje od x vremena
  - Zaustavlja se kada je x jednak određeni broj (primjerice nula).
- štoperica
  - Nakon određenog isteka vremena aktivira se uvjet.
- eliminiranje neprijateljeve baze
  - Kada je vrijednost broja uvjeta jednaka broju gnijezda na mapi, aktivira se uvjet.
- ništa
  - Koristi se kada primjerice ne želimo imati bonus (ili određeni uvjet)

Te iste mogućnosti cilja mogu se koristiti za aktivaciju definiranog bonusa što uključuje povećanje vremena, brzinu kretanja čovječuljka i brzinu kopanja čovječuljka (Slika 22).

Slika 14 Početni zaslon igre

Slika 15 Izbornik za odabir razine (zeleno označuje osvojenu razinu)



Slika 16 Pogled na razinu | U desnom kutu (vrijeme | broj bodova | broj gnijezda) | U lijevom kutu (aktivacija DevTools-a | izbornik)

Slika 17 Napad ptice i njeno gnijezdo

Slika 18 Ptica iščekuje igrača na površini | Igrač je ispod zemlje

Slika 19 Indikator udaljenosti od boda (dolje) | Indikatori smjera iz kojeg dolazi ptica (ostalo)

Slika 20 Osvojen bod | Broj iskopa do boda

Slika 21 Pauzirana igra | Aktiviran DevTools (omogućava mijenjanje brzine igraču, neprijatelju i mijenjanje tipa kontrole igrača)

Slika 22 Aktiviran bonus nakon svaka 2 skupljena boda

4.2 Organizacija

U ovom poglavlju će se opisati korištena organizacija scena koja olakšava prelazak iz jednog prikaza u drugi, čime dolazi do uštede na aktivnoj memoriji. Naposljetku, objasnit će se organizacija objekata i njihovih komponenti.

4.2.1 Organizacija scena

Sadrži redoslijed izvršavanja scena kojim se može manipulirati (Slika 23). Početna scena na pokretanju same igre je uvijek master te ona sadrži sve podatke za izvršavanje i pozivanje ostalih scena.

Scene su podijeljene na:

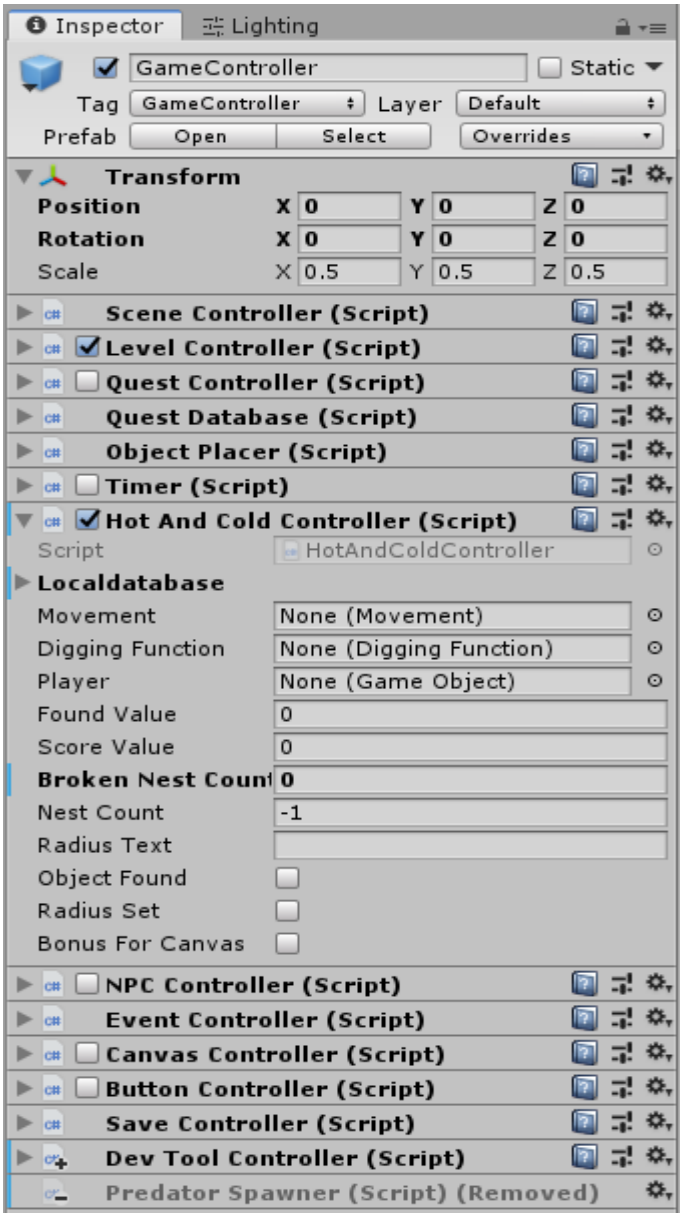
- master
  - Sadrži objekt igre s komponentama kontrole svih dijelova igre.
- izbornik
  - Sastoji se od glavnog izbornika i izbornika odabira razine i služi kao tranzicija između master scene i scena razine.
- razine
  - Sadrže sve ovisno o određenoj razini i njihov naziv se koristi kao identifikacija u json bazi.

Slika 23 Opcije izdanje igre, odabrane scene za izgradnju

4.2.2 *Organizacija objekata igre i komponenti*

Fokus je bio na korištenju centralnog kontrolnog dizajna što omogućava kontrolerima jednostavniju pristupačnost svim podacima. Time su sve komponente kontrole sadržane unutar GameContoller objekta igre (Slika 24). S kontrolerom direktno može komunicirati jedino drugi kontroler i kontroler ima pristup svim ostalim objektima unutar igre (kroz sve scene).

Ostale komponente logike su vezane uz pametne entitete, odnosno igračkog čovječuljka, neprijatelja i neprijateljeve baze. Čovječuljak osim Unity komponenti sadrži skriptu za kretanje i kopanje, a njegovo dijete „Director“ sadrži skriptu za očitavanje sakrivenog boda.



Slika 24 Pogled na GameController objekt igre, njegove komponente

4.3 Struktura i komunikacija

Svi kontroleri su MonoBehavior [5] , ali i nasljeđuju GetControllers i ScriptControllers te ControllersTemplate (Slika 25). GetControllers sadrži zaštićena svojstva (eng. *protected properties*) koja služe za međusobni pristup kontrolerima. Također, imamo interface[6] kako bi resetirali i pokrenuli svaku skriptu neovisno o Unity ugrađenom sustavu.

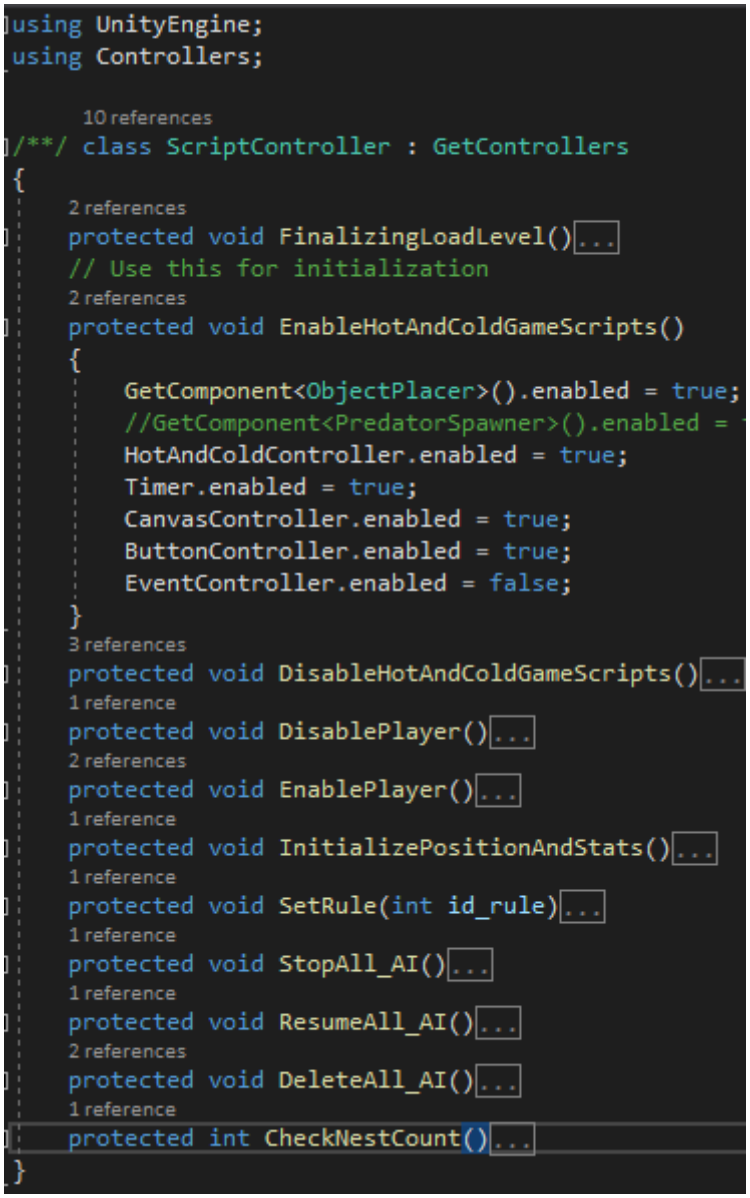
Napomena: Neke od komponenti nisu implementirane

```
using UnityEngine;
namespace Controllers
{
    1 reference
    class GetControllers : MonoBehaviour
    {
        43 references
        protected HotAndColdController HotAndColdController { get { return GetComponent<HotAndColdCor
        16 references
        protected CanvasController CanvasController { get {return GetComponent<CanvasController>(); }
        13 references
        protected Timer Timer { get { return GetComponent<Timer>(); }}
        13 references
        protected LevelController LevelController { get { return GetComponent<LevelController>(); }}
        0 references
        protected NPCController NPCController { get { return GetComponent<NPCController>(); }}
        0 references
        protected QuestController QuestController { get { return GetComponent<QuestController>(); }}
        2 references
        protected EventController EventController { get { return GetComponent<EventController>(); }}
        2 references
        protected SaveController SaveController { get { return GetComponent<SaveController>(); }}
        4 references
        protected ButtonController ButtonController { get { return GetComponent<ButtonController>();
        2 references
        protected DevToolController DevToolController { get { return GetComponent<DevToolController>(
    }

    6 references
    interface ControllersTemplate
    {
        12 references
        void StartScript();
        14 references
        void Reset();
    }
}
```

Slika 25 GetControllers i ControllersTemplate

[5] Osnovna klasa Unity-a iz koje proizlazi svaka Unity skripta.
[6] Ugovor između entiteta (klasa) koji zahtjeva da se definirane metode implementiraju.

Sljedeći je ScriptController (Slika 26) kojeg nasljeđuju svi kontroleri, a koji nasljeđuje GetController. On sadrži osnovne metode koje se koriste od strane više od jednog kontrolera ili metode koje izvršavaju rad nad nekim dijelom unutar jednog ili više kontrolera.



Slika 26 ScriptController | Prikazana je metoda koja aktivira sve vezano uz HotAndCold kontroler pri učitavanju razine

4.4 Kontroleri

Služe kao izvršni odlučitelji tijeka igre. Sadrže svu logiku za izvršavanje, učitavanje, spremanje, prikazivanje i očitavanje promjena. Postoje jednostavni koji uglavnom sadrže metode za aktiviranje, odnosno prikazivanje određenih UI elemenata poput Canvas kontrolera ili poput DevTool kontrolera (omogućuje mijenjanje ponašanja određenih elemenata igre). Naknadno će se detaljnije objasniti neke od ostalih kontrolera.

4.4.1 Save kontroler

Služi za spremanje statusa razine u json bazu, a koristi se Unity JSON Serialization za učitavanje. Sama metoda se poziva pri završetku razine te se u varijablu *„alldata“* dodaju svi podaci razina, uključujući nove informacije o trenutno prijeđenoj razini. Sadržaj stare json datoteke se briše i na njegovo mjesto se upisuje *„alldata“* s određenim tekstualnim kosturom (Slika 27).

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

3 references
class SaveController : SaveDatabase {
    string alldata;
    2 references
    public void SaveLevelChanges(List<LevelsList> data)
    {
        foreach (LevelsList element in data)
        {
            alldata = alldata + JsonUtility.ToJson(element);
            if (data.IndexOf(element) != data.Count-1)
            {
                alldata = alldata + ",";
            }
        }
        JsonFileReader.ClearResource("/StreamingAssets/Levels.json");
        JsonFileReader.ToJsonAsResource("/StreamingAssets/Levels.json",
            "{"+'"'+"Level" + '"' + ":[" + alldata + "]" + "}");
        alldata = null;
    }
}
```

Slika 27 SaveController | Sprema podatke u obliku json sadržaja

4.4.2 HotAndCold kontroler

Upravlja samom *toplo/hladno* igrom. Prvotno dohvaća potrebne objekte i komponente sa scene i pravila igre iz json baze. Nakon toga postavlja vrijednosti varijabla te postavlja težinu iskopa sakrivenog boda i inicijalizira objekt bod na sceni. Poslije inicijalizacije vrti update metodu u kojoj čeka input igrača, odnosno očitavanje udaljenosti od boda ili kopanje kada pronađe bod (Slika 28). CalculateStopTime se poziva kao simulacija zastoja kopanja pri očitavanju udaljenosti što bi se u daljnjem razvitku igre zamijenilo animacijom te naposljetku ima provjeru u slučaju da se bonus aktivirao.

```
public void StartScript()...
14 references
public void Reset()...
//------------------- Update Void ----------------//
0 references
void Update()
{
    if ((Input.GetButtonDown("Scan") || buttonpressed) &&
        (movement.enabled || stopWhileDigging == 0) &&
        !diggingFunction.belowground && movement.OnGround())
    {
        DetermineRadius();
        if (StartDigging())
        {
            dighardness -= diggingspeed;
            diggingObject = true;
            if (dighardness <= -1)
                StopDigging();
        }
        stopWhileDigging = Time.fixedTime;
    }
    if (CalculateStopTime(stopWhileDigging))
    {
        stopWhileDigging = 0;
        if (!diggingObject)
        {
            movement.enabled = true;

        }

    }
    if (upgrade == true)//Bonus settings
    {
        Timer.timerLength += localdatabase[0].Bonus[0].bonustime;
        movement.values["speed"] *= localdatabase[0].Bonus[0].movementspeed;
        diggingspeed *= localdatabase[0].Bonus[0].diggingspeed;
        upgrade = false;
        BonusForCanvas = true;
    }
    calculateNests();
}
```

Slika 28 Isječak HotAndCold kontroler | Prikazana je update metoda

Sadržane su i metode za provjeru statusa razine (Slika 29):

- provjeri je li ispunjen uvjet za pobjedu
    - Pamti se i ako igrač preživi do kraja razine sprema se rezultat.
- provjeri je li ispunjen uvjet za kraj razine
    - Kada se ispuni ovaj uvjet završava razina. Može biti jednak s uvjetom za pobjedu.
- provjeri je li ispunjen uvjet za Bonus
    - Svakog puta kada se ispuni uvjet bonus se aktivira.

```
public bool SeeIfConditionMetWin()
{
    return CheckCondition(localdatabase[0].Win[0].condition, localdatabase[0].Win[0].objective);
}
1 reference
public bool SeeIfConditionMetEndGame()
{
    return CheckCondition(localdatabase[0].EndGame[0].condition, localdatabase[0].EndGame[0].obje
}
1 reference
public bool SeeIfConditionMetBonusCondition()
{
    return CheckCondition(localdatabase[0].BonusCondition[0].condition, localdatabase[0].BonusCon
}
//-------------------- CheckCondition bool ----------------//
3 references
private bool CheckCondition(string x, string y)
{
    switch (x)
    {
        case "time":
            if (Timer.timerLength <= int.Parse(y))
                return true;
            break;
        case "found"://Use only for BonusCondition
            if ((foundValue % int.Parse(y)) == 0)
                return true;
            break;
        case "foundonce":
            if (foundValue >= int.Parse(y))
                return true;
            break;
        case "breaknest":
            if ((float)brokenNestCounter / nestCount * 100 <= int.Parse(y))
                return true;
            break;
        case "empty":
        default:
            break;
    }
    return false;
```

Slika 29: Prikazan je isječak HotAndCold kontrolera | Sve provjere prolaze kroz switch

4.4.3 *Level kontroler*

Upravlja svim aspektima razine (Slika 30). Od izbornika na razini, do samog učitavanja razine i pokretanja pa čak i praćenja promjene statusa razine i spremanja tog statusa u json bazu. Pri spremanju statusa rekordno stanje se sprema ovisno o pravilima razine.

```
class LevelController : LevelDatabase
{
    private Vector3 test;
    private int id_Rule = 0;
    private int id_Level = 0;
    private int id_Map = 0;
    1 reference
    public void Run(int id_rule, int id_level, int id_map)...
    0 references
    public void RunStoryMode(GameObject objectRead, GameObject Player)...
    0 references
    void Update()//need to edit(clean up)...
    1 reference
    void OnEnable()...

    1 reference
    void OnDisable()...

    2 references
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)...
    2 references
    private void Load()...
    1 reference
    private void SetButtons()...
    1 reference
    private void StartLevel()...
    1 reference
    public void RestartLevel()...
    3 references
    public void ExitLevel()...
    3 references
    public void OpenMenu()...
    2 references
    public void CloseMenu()...
    6 references
    public bool IfEndGame()...
    1 reference
    public SaveController getSaveController()...
}
```

Slika 30 Isječak level kontroler | Sadrži sve metode vezane uz učitavanje, izlazak i spremanje razine

4.5 Upravljanje bazom

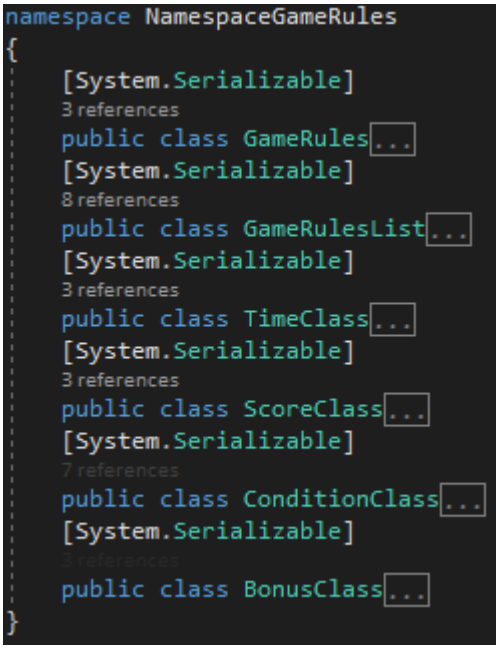
Prvo napomena, ne koristi se baza u standardnom smislu, već su to json datoteke (Slika 31).

Slika 31 Json datoteke

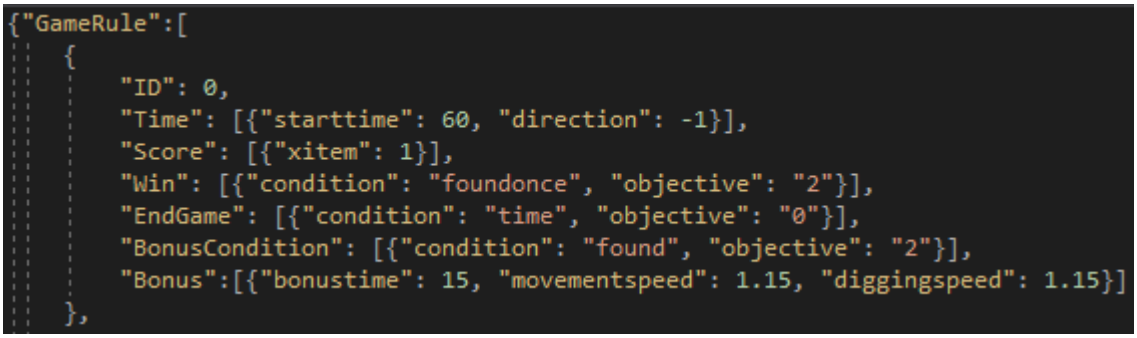
Datoteka GameRules.json sadrži sva pravila igre (Slika 34). Sva pravila se učitavaju pri pokretanju igre, spremaju se u privatnu varijablu te njima pristupamo pomoću identifikacijskog broja (Slika 32, Slika 33).

```
class GameRulesDatabase : ScriptController
{
    private List<GameRulesList> database = new List<GameRulesList>();
    private GameRules jsonlist;
    0 references
    void Awake()//To Be Edited Once GameRulesEditor Introduced
    {
        string itemData = JsonFileReader.LoadJsonAsResource("/StreamingAssets/GameRules.json");
        jsonlist = JsonUtility.FromJson<GameRules>(itemData);
        for (int i = 0; i < jsonlist.GameRule.Count; i++)
        {
            database.Add(jsonlist.GameRule[i]);
        }
    }
    1 reference
    public GameRulesList FetchRulesByID(int id)
    {
        for (int i = 0; i < database.Count; i++)
        {
            if (database[i].ID == id)
            {
                return database[i];
            }
        }
        return null;
    }
}
```

Slika 32 Game Rules Database | Učitava i omogućava pristup pravilima igre



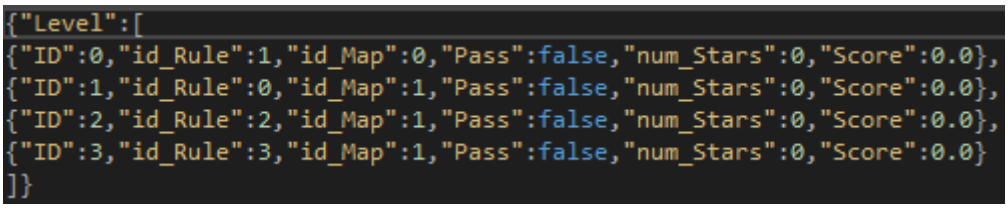
Slika 33 Definira strukturu podataka u json datoteci



Slika 34 Primjer jednog pravilnika igre u json-u

Sljedeće su Levels i LevelsSave. LevelsSave nam je samo backup Levelsa te se on poziva klikom na „reset“ u izborniku za odabir razine. Levels sadrži status svih razina, koja pravila igra ta razina koristi te koju vizualnu mapu će učitati. Nakon toga, imamo „Pass“ koji bilježi je li razina uspješno osvojena i „Score“, odnosno najbolji rezultat ostvaren ovisno o pravilima igre (Slika 35).

Napomena: „num_Stars“ nije implementiran.



Slika 35 Primjer kako su definirane razine u json-u

4.6 Umjetna inteligencija entiteta

Zahtjev entiteta je da postigne inteligentan dojam i izazov za igrača uz dodatan zahtjev da mora pouzdano prelaziti iz jednog ponašanja u drugo, te ako se i dogodi neočekivan ishod, da može uz određene blokade uspostaviti definiran režim rada.

Simuliranje neprijatelja (simbolizira sokola), njegova ponašanja:

- patroliranje baze (simbolizira gnijezdo)
    - Neprijatelj na određenoj visini i udaljenosti kruži oko baze.
- napadanje
    - Detekcijom igrača kreće u napad, a visina leta ovisni o udaljenosti od igrača.
- prizemljenje
    - Pri nestanku igrača, ako je udaljenost između neprijatelja i igrača bila manja od određene, neprijatelj slijeće na zadnje poznato igračevo mjesto gdje će čekati na neodređeno vrijeme.
- reorganizaciju
    - Neprijatelj kreće prema određenoj bazi gdje ponovno prelazi na patroliranje
- obnavljanje baze (simbolizira gnijezdo)
    - Ako neprijatelj ne patrolira oko početne baze, kreće prema njoj i obnavlja je

4.6.1 Opis ponašanja

Pri učitavanju razine neprijatelj se nalazi na bazi i započinje spiralno kružiti dok ne uspostavi određenu orbitu na definiranoj visini oko baze. Ako igrač s čovječuljkom uđe u neprijateljevu koliziju, neprijatelj ga primjećuje i započinje napad. Pri napadu, na ekranu će se pokazati strelica u smjeru od kojeg neprijatelj napada, ali samo ako neprijatelj nije vidljiv na ekranu. Pri spuštanju neprijatelja na igrača, on ubrza dok se ne spusti do određene visine te će, ako se igrač do tada nije sakrio, vjerojatno biti eliminiran. Ako se igrač uspio sakriti i neprijatelj je na nižoj visini, tada će se neprijatelj prizemljiti i čekati neko vrijeme u slučaju da se igrač pojavi. U suprotnom, neprijatelj odustaje od potjere. Oba slučaja rezultiraju reorganizacijom (vraćanje u bazu) te patroliranjem baze.

Za igrača je također moguće da eliminira bazu neprijatelja, pri čemu će se svaka buduća reorganizacija dogoditi u susjednoj bazi (ili centralnoj bazi ako su sve baze eliminirane) na neodređeno vrijeme nakon kojega slijedi obnavljanje prvobitne baze.

4.6.2 Kretanje entiteta

Neprijatelj se kreće tako da ima definiranu najnižu moguću visinu, koja se mijenja ovisno o ponašanju. Za razliku od standardnog načina pomicanja objekata, neprijatelj se konstantno kreće u jednolikom pravcu određenom brzinom, a skretanje se vrši tako da se kalkulira normala ovisno o odredištu te se neprijatelj rotira određenom brzinom kroz svaki okvir dok nije okrenut u smjeru tog pravca. Time se dobiva uglađeno kretanje (Slika 36).

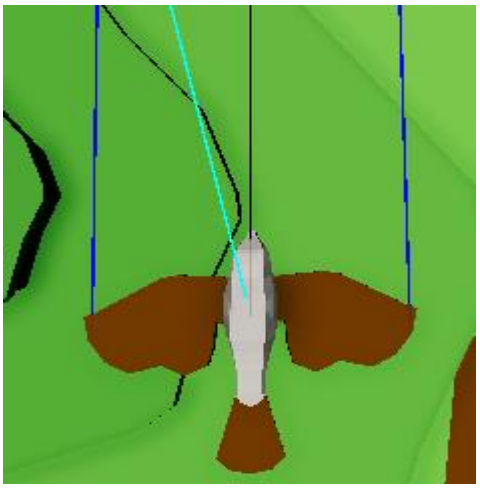
```
protected void Move(Vector3 destination)
{
    destination = new Vector3(destination.x, thisMinHeight, destination.z);
    Debug.DrawRay(transform.position, destination - transform.position, Color.magenta);
    if (!destinationSet || (destinationSet && avoid == 2 && pathFound))
    {
        diff = destination - transform.position;
        destinationSet = true;
    }
    controller.Move(transform.forward * Time.deltaTime * speed);
    Vector3 newDir = Vector3.RotateTowards(transform.forward, diff.normalized,
        Time.deltaTime * rotationSpeed, 0f);
    transform.rotation = Quaternion.LookRotation(newDir);
    if (!transform.rotation.Equals(Quaternion.LookRotation(newDir)) || Stuck())
    {
        offsetdistance += 0.3f;
    }

    Debug.DrawRay(transform.position, diff, Color.red);

}
```

Slika 36 Move metoda iz Predator Movement | Algoritam koji transformira (pokreće) neprijatelja

4.6.3 Izbjegavanje prepreka

Detekcija prepreka radi tako da se za svaki okvir ispituje ako je zraka (prikazano plavom na (Slika 37), eng. *raycast*) pogodila neki objekt (u slučaju igre registriraju se jedino objekti s osnovnim slojem (eng. *default layer*, Slika 38)).

Ako se dogodi detekcija, poziva se metoda za pronalazak sigurne rute (Slika 39). Metoda radi tako da u isto vrijeme pronalazi rutu na lijevo i desno pomoću lijeve i desne zrake. Za svaki okvir se dodaju 5 stupnja dok ne postignemo da zraka ne pogađa ni jedan objekt. Ako se ruta ne pronađe, a obje zrake su prešle 90 stupnjeva, tada se jednom testira ako zraka pogađa objekt ravno iznad neprijatelja (ukoso naprijed) te se onda šalje neprijatelju da uradi salto unazad, dok će u suprotnom ići iznad objekta.



Slika 37 Pogled iz scene na neprijatelja sa vizualiziranim zraka

```
protected bool AvoidanceRaycast()
{
    RaycastHit obstacle;
    Debug.DrawRay(transform.position + transform.right * 1,
        transform.TransformDirection(forward), Color.blue);
    Debug.DrawRay(transform.position - transform.right * 1,
        transform.TransformDirection(forward), Color.blue);
    if (Physics.Raycast(transform.position, transform.TransformDirection(forward),
        out obstacle, raySize) ||
        Physics.Raycast(transform.position + transform.right * 1,
        transform.TransformDirection(forward), out obstacle, raySize) ||
        Physics.Raycast(transform.position - transform.right * 1,
        transform.TransformDirection(forward), out obstacle, raySize))
    {
        return true;
    }
    return false;
}
```

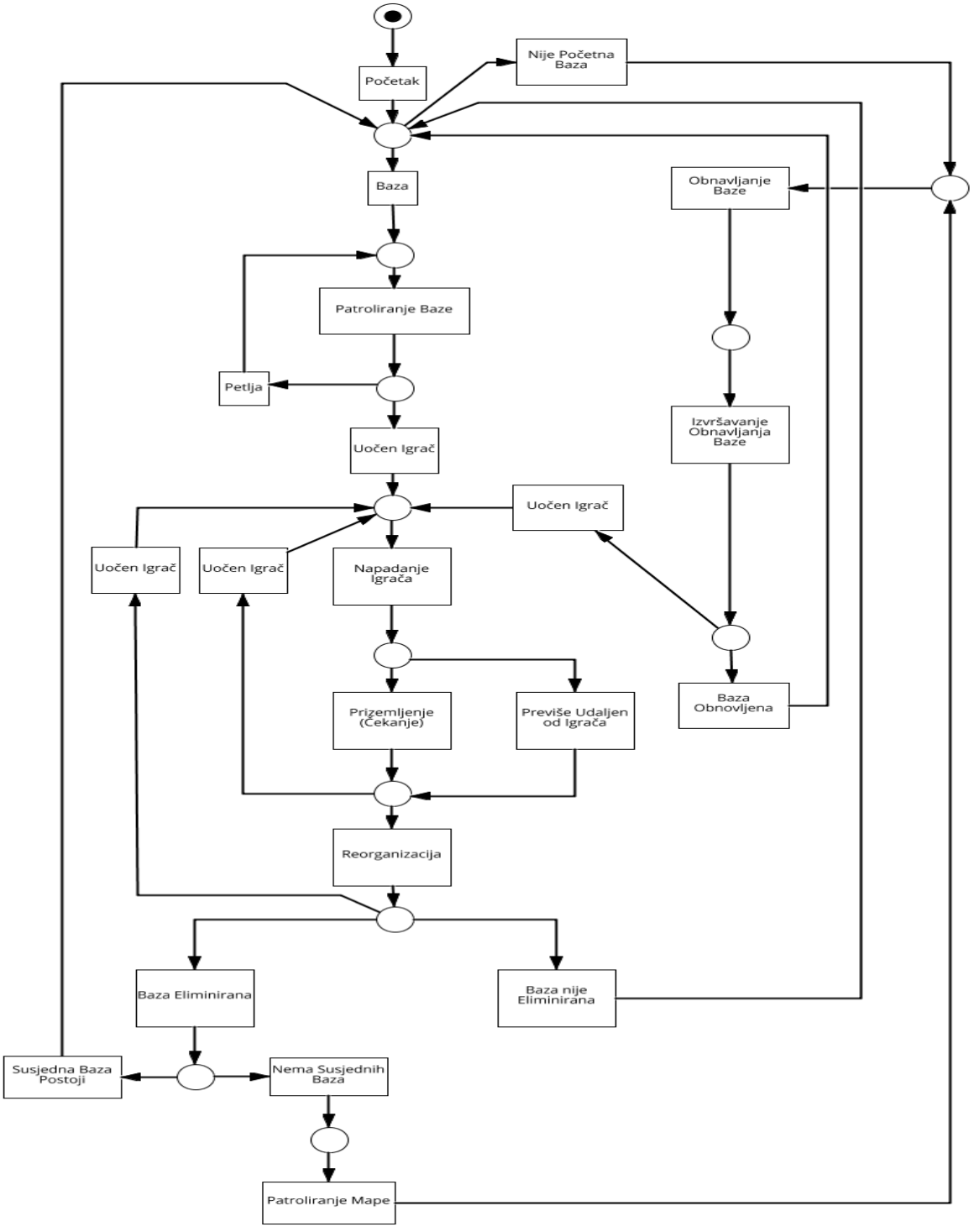
Slika 38 Metoda očitavanja prepreke

```
if (angleRight > 90 || angleLeft < -90)
{
    if (Physics.Raycast(transform.position, transform.TransformDirection(forward) + tra
    {
        direction = transform.TransformPoint(-forward / 2) + transform.up;
        pathFound = true;
        destinationSet = false;
        //Debug.Log("BackFlip");
        //BackFlip
    }
    else
    {
        direction = transform.TransformPoint(forward + transform.up * (raySize + 3));
        pathFound = true;
        destinationSet = false;
        //Debug.Log("Go Above");
        //Go Above
    }
}
else
{
    if (Physics.Raycast(transform.position, testAroundRight, out pathRight, raySize) ||
    else
    {
        Finalize(testAroundRight, 1);
        //Debug.Log("Right working");
    }

    if (Physics.Raycast(transform.position, testAroundLeft, out pathLeft, raySize) || Pl
    else
    {
        Finalize(testAroundLeft, -1);
        //Debug.Log("Left working");
    }
}
```

Slika 39 Isječak metode koja pronalazi sigurnu rutu

4.7 Ponašanje entiteta kroz Petrijeve mreže

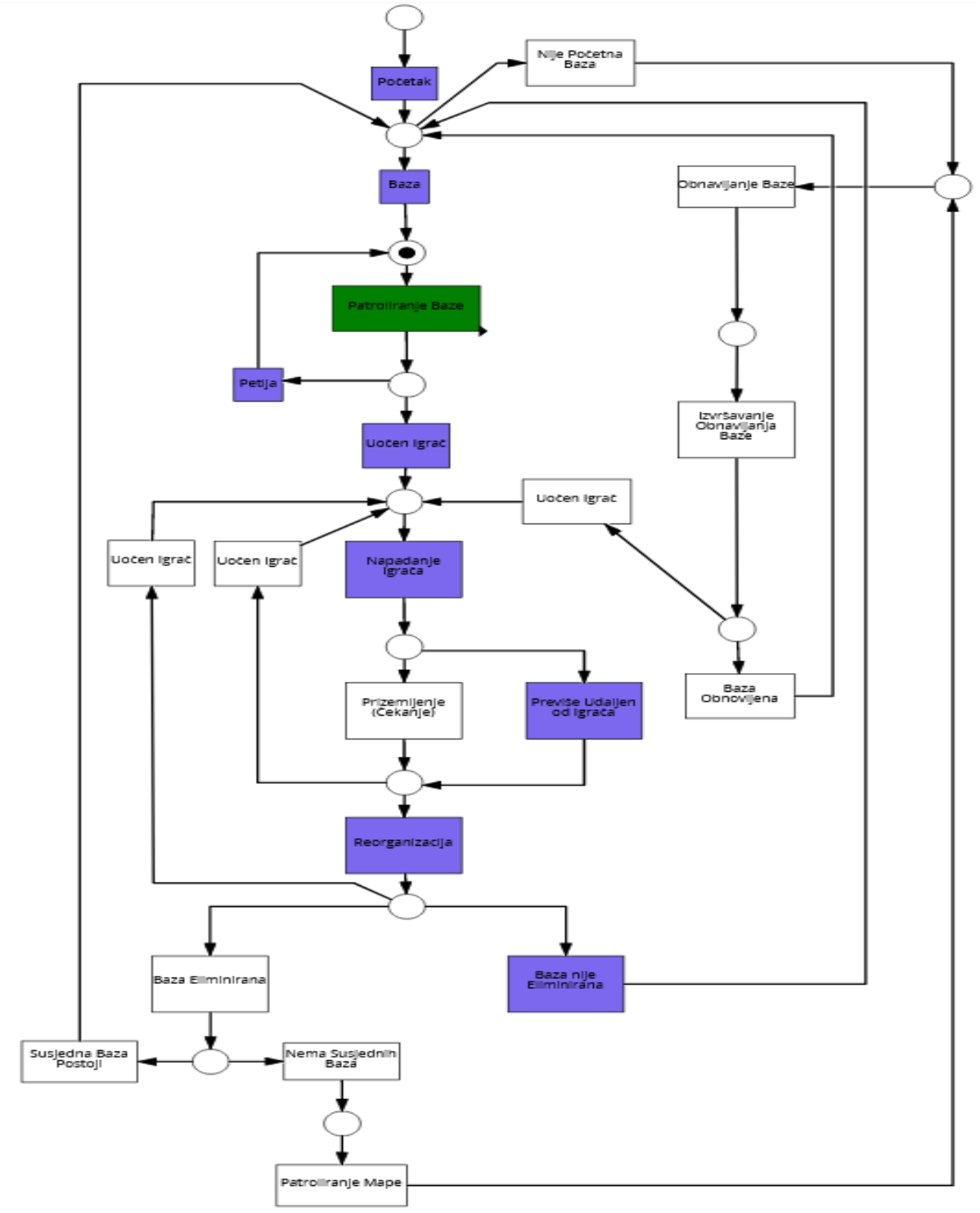
Petrijeve mreže koriste se za prikaz i modeliranje dinamičkih sustava u svrhu promatranja njihovih ponašanja u različitim okolnostima [5]. U tu svrhu promatrat će se slijed izvršavanja entiteta kroz tri scenarija kroz sljedeću mrežu (Slika 40).



Slika 40 Petrijeva mreža, ponašanja entiteta

4.7.1 Osnovno ponašanje

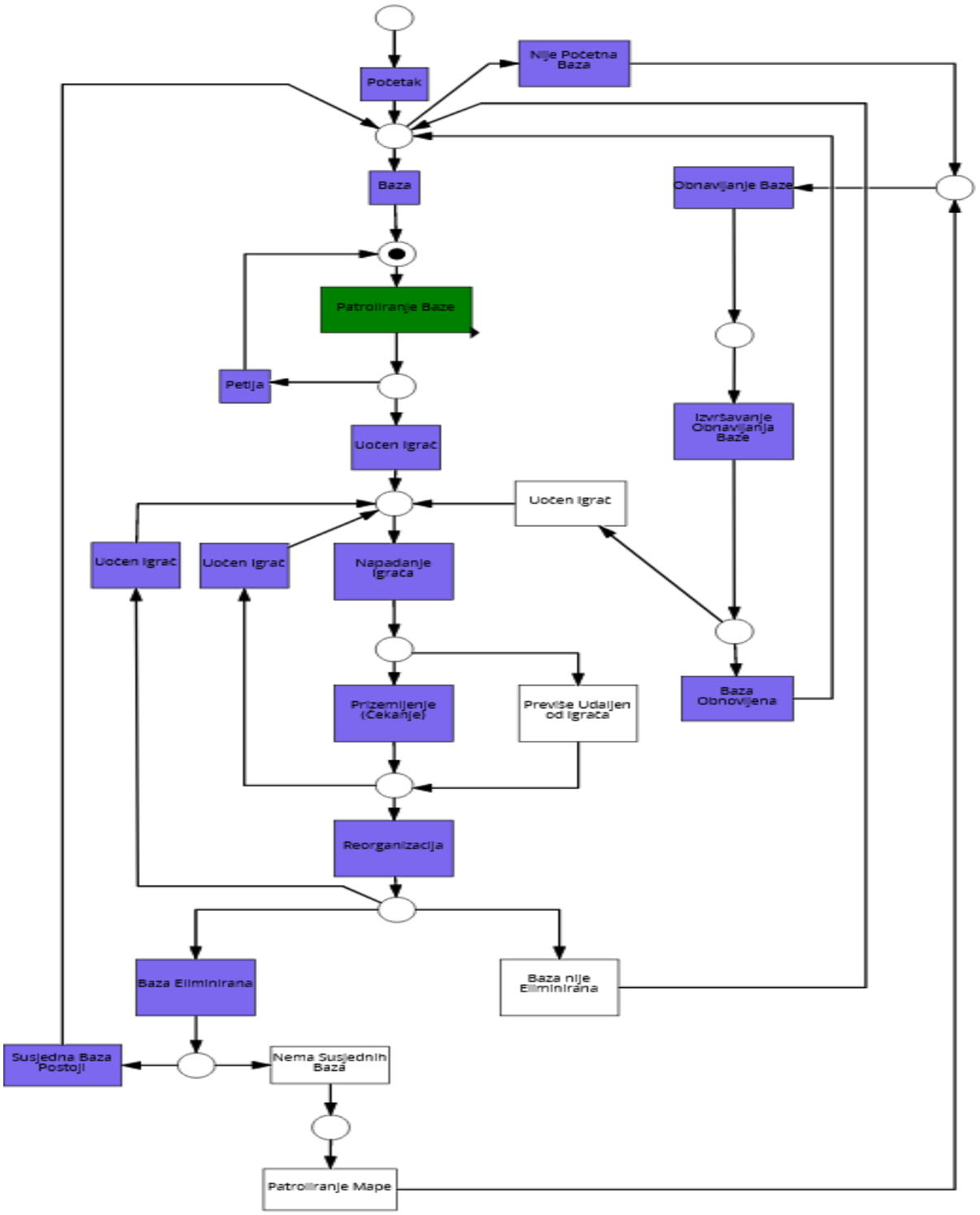
Prvi scenarij započinje patroliranjem oko baze dok se ne primijeti igračev čovječuljak, a naknadno kada je čovječuljak sakriven prelazimo na prizemljenje. Nakon neodređenog vremena neprijatelj odustaje i vraća se u bazu. Primjer tijeka događaja bez prizemljenja (Slika 41).



Slika 41 Petrijeva mreža osnovnog ponašanja entiteta

4.7.2 *Ponašanje prilikom eliminacije neprijateljeve baze*

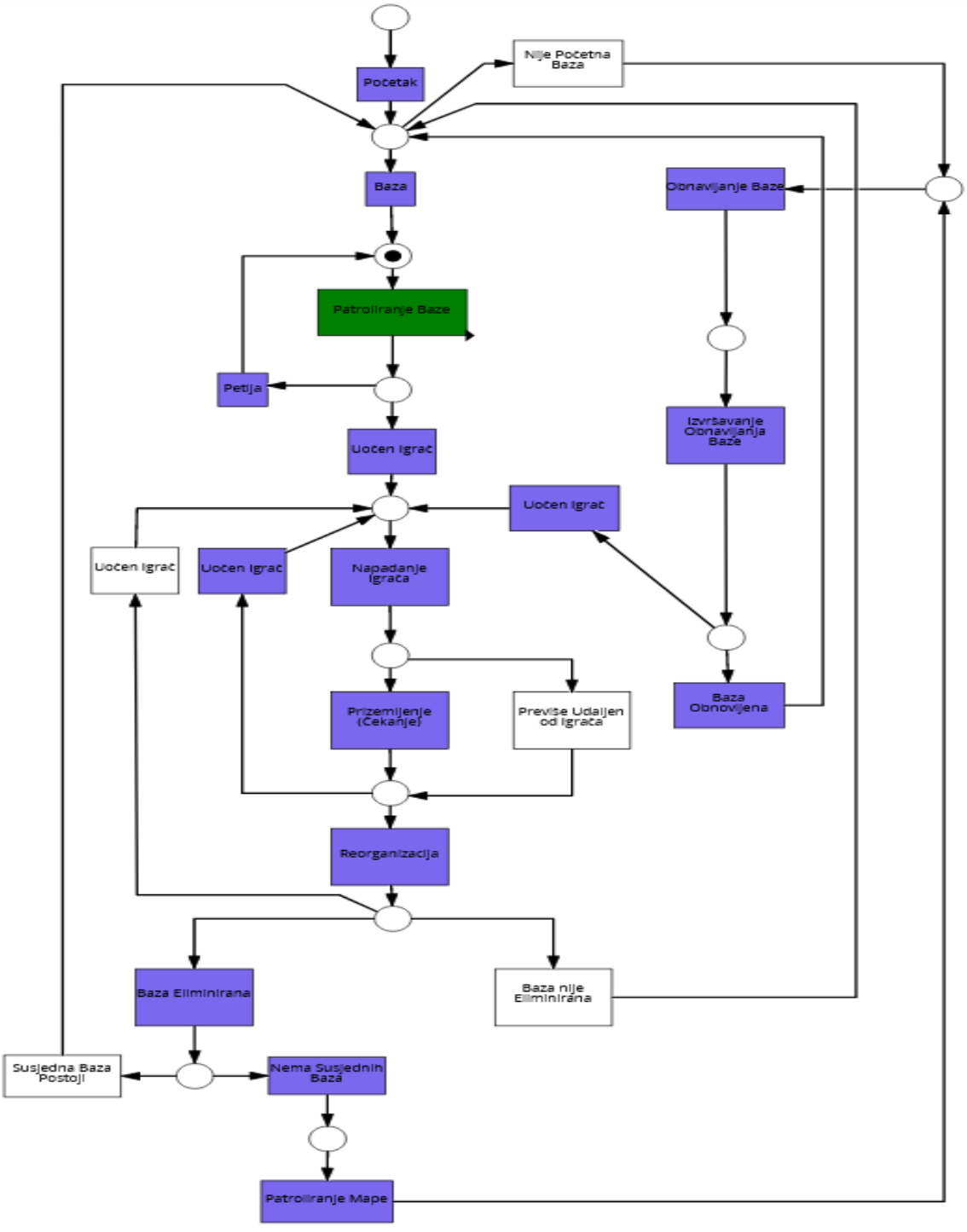
Drugi scenarij ima prijašnje osnovno ponašanje uz dodatne detekcije igrača od strane neprijatelja i uz dodatno prizemljenje. Zbog eliminacije neprijateljeve baze traži se najbliža sljedeća baza. Nakon nekog vremena kruženja oko te baze, neprijatelj kreće u obnavljanje primarne baze. Primjer tijeka događaja (Slika 42).



Slika 42 Petrijeva mreža, ponašanje prilikom eliminacije neprijateljeve baze

4.7.3 *Ponašanje prilikom eliminiranja svih baza*

Naposljetku, treći scenarij također ima osnovno ponašanje, ali razlika je u tome da su sve baze eliminirane te nakon neodređenog kruženja oko centralne baze neprijatelj kreće u obnavljanje primarne baze. Treba također napomenuti da će u rasponu od procesa izgradnje baze i završetka izgradnje baze neprijatelj napasti igrača ako ga detektira. Primjer tijeka događaja (Slika 43).



Slika 43 Petrijeva mreža, ponašanje prilikom eliminiranja svih baza

5. Zaključak

Reaktivna komponenta je deterministično mapiranje ulaza -> izlaza. Komponente s unutarnjom memorijom mogu se smatrati reaktivnima te je moguće izraziti nedeterminističke komponente (s unutarnjim stanjem) kao reaktivne korištenjem dodatnih ulaza. Također, komponente se tretiraju kao crne kutije koje je moguće slagati u skupine, a prednost je u tome što brzo reagiraju na vanjske zahtjeve ili na podražaje iz okoline.

Zaključuje se da je reaktivno ponašanje bolje prilagođeno igrama zbog svoje učinkovitosti, pouzdanosti i predvidljivosti.

Unity okruženje omogućuje brzu izradu kompleksnih igara, ima veliku podršku od strane razvojnog tima i zajednice, omogućuje integraciju cijelog tima u razvojnu platformu te ima velik izbor mogućnosti izdavanja na velik broj platformi.

Izrada umjetne inteligencije je bila izazovna i zanimljiva. Najteži dio je bio napraviti kvalitetno kretanje (letenje) te u to dodati izbjegavanje prepreka. Naime, korišten princip kretanja još uvijek nije savršen te sadrži određene blokade u slučaju da se ne uspije vratiti u prijašnje stanje. Postoje bolje alternative, ali koriste mnogo više zraka za detekciju, dok detekcija opisana u ovom radu u tome uspijeva sa samo tri, pri čemu se postiže ušteda na računalne resurse.

6. Literatura

[1] Alex J. Champandard, (21.11.2009.), AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors, https://flylib.com/books/en/2.71.1, 29.08.2019, Chapter 3. Reactive Approach

[2] Arend Hintze, (14.11.2016.), Understanding the four types of AI, from reactive robots to self-aware beings, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwiJnK2Dr8HkAhVFiYsKHT7JDBUQFjAAegQIABAC&url=https%3A%2F%2Ftechxplore.com%2Fpdf398333855.pdf&usg=AOvVaw3bZc3RkKtvd3Rw-Uvn0x0r, 29.08.2019

[3] Rafael Kuffner Dos Anjos, Unity3D, https://fenix.tecnico.ulisboa.pt/downloadFile/282093452038979/Unity%203D.pdf, 29.08.2019

[4] Unity Technologies, Unity User Manual, https://docs.unity3d.com/Manual/index.html, 29.08.2019

[5] Wikipedia, Petrijeve mreže, https://hr.wikipedia.org/wiki/Petrijeve_mre%C5%BEe

7. Popis slika