

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

JOSIP MARIĆ

**IZRADA 3D IGRE SA 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 mogega 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

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 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, rujan, 2019. godine

Potpis
Josip Marić

Sažetak

Ovaj rad opisuje koncept razvoja 3D igre sa reaktivnom inteligencijom kod entiteta u Unity okruženju. Igra se sastoji od nekoliko razina sa 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...), te detaljnije objašnjenje izrade i ponašanja reaktivne inteligencije kod entiteta (neprijatelja), i 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 ...), 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.*

1. Uvod	1
2. Umjetna Inteligencija	2
2.1 Reaktivna Inteligencija	3
2.1.1 Teorija	3
2.1.2 Praksa	4
2.1.3 Reaktivne Tehnike u Razvoju Igara	5
2.1.4 Arhitektura	6
2.1.5 Komponenta	6
2.1.6 Organizacija	7
2.1.7 Dekompozicija	8
2.1.8 Odlučivanje	9
3. Unity3D	10
3.1 Unity Uređivač	11
3.2 Scene	14
3.3 Objekti Igre i Komponente	14
3.4 Montaže	15
4. Izrada Igre	16
4.1 Opis Igre	16
4.1.1 Cilj Igre	16
4.1.2 Izbornik Igre	17
4.1.3 Pogled na Razinu Igre	18
4.2 Organizacija	21
4.2.1 Organizacija Scena	21
4.2.2 Organizacija Objekata Igre i Komponenti	22

4.3	Struktura i Komunikacija	23
4.4	Kontroleri	25
4.4.1	<i>HotAndCold Kontroler</i>	25
4.4.2	<i>Canvas Kontroler</i>	26
4.4.3	<i>DevTool Kontroler</i>	26
4.4.4	<i>Save Kontroler</i>	27
4.4.5	<i>Level Kontroler</i>	28
4.5	Upravljanje Bazom	29
4.6	Umjetna Inteligencija Entiteta	31
4.6.1	<i>Opis Ponašanja</i>	31
4.6.2	<i>Kretanje Entiteta</i>	32
4.6.3	<i>Izbjegavanje Prepreka</i>	33
4.7	Ponašanje Entiteta Kroz Petrijeve Mreže	35
4.7.1	<i>Osnovno Ponašanje</i>	36
4.7.2	<i>Ponašanje Prilikom Eliminacije Neprijateljeve Baze</i>	37
4.7.3	<i>Ponašanje Prilikom Eliminiranja Svih Baza</i>	38
5.	Zaključak	39
6.	Literatura	40
7.	Popis Slika	41

1. Uvod

Tema ovog završnog rada je izrada 3D igre u Unity-u sa 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, mada nije preporučljivo zbog visokih grafičkih zahtjeva. Igra se sastoji od nekoliko razina, svaka razina sa svojim pravilima igre. Pristup svim razinama je jednak, te pri uspješno prođenoj razini mijenja se boja komponente u zelenu i zapisuje se najbolji postignuti rezultat te je to vidljivo u izborniku za odabir razine. Time se postiže želja za ponovnom igrom iste razine. Pogled na razinu je iz ptičje perspektive, te kontrola čovječuljka je osnovna (WASD) gdje svaka tipka ima kretanje u svojem smjeru te kombinacija dopušta kretanje ukoso. Također igrač se može kretati i ispod zemlje ali prijelaz je moguće ostvariti jedino na određenim mjestima. Igrač treba navigirati 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 detaljno 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

Svi smo do sada upoznati sa napredkom umjetne inteligencije u svim dijelovima društva, te je ostavio dojam na ljude da smo na pragu generalne umjetne inteligencije [2]. Naime uz pomoć strojnog učenja roboti su u mogućnosti razumijevanja verbalnih komandi, razlikovanja slika, pilotiranje automobila te čak imitiranje govora, kreiranje literature i skladanje glazbe. Problem takve inteligencije je njezina nepredvidivost i ne transparentnost što je čini nepraktičnom u dizajniranju video igara te se više preferiraju umjetne inteligencije prvog i drugog tipa.

Tipovi umjetne inteligencije:

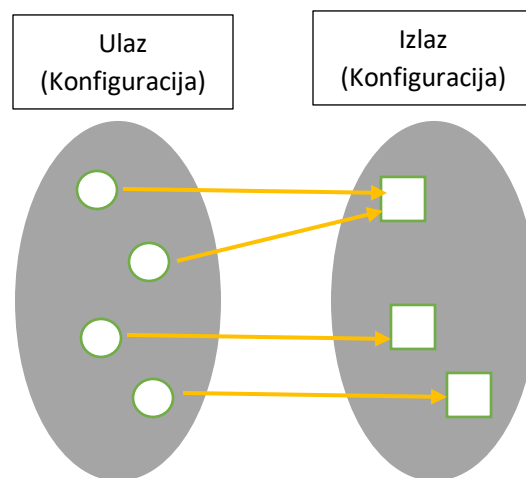
- **Reaktivni Strojevi (prvi tip)**
 - Najjednostavniji sustavi umjetne inteligencije su čisto reaktivni. Nemaju mogućnost formiranja memorije ili korištenja prijašnjih iskustva. Mada nemaju mogućnost učenja svejedno su sposobni te je tako IBM-om „Deep Blue“ algoritam pobijedio međunarodnog majstora šaha, Garry Kasparov u kasnim 1990s.
- **Ograničena Memorija (drugi tip)**
 - Takvi sustavi imaju mogućnost sadržavanja memorije kao reference u budućim odlukama te se koristi u pametnim automobilima (identificiraju i prate ostala vozila prilikom promjene brzine i smjera).
- **Teorija Uma (treći tip)**
 - Trenutno nedostižno, takvi strojevi će biti u mogućnosti razumijevanja okoline oko sebe na način da će shvaćati da i ostali entiteti koje očitavaju imaju mogućnost razmišljanja.
- **Samosvijest (četvrti tip)**
 - Ovo je nadogradnja na prijašnji tip Teorije uma, te razlika je u tome da će stroj biti svjestan samog sebe, odnosno bit će u mogućnosti opravdati svoje akcije.

2.1 Reaktivna Inteligencija

Reaktivna Inteligencija se već dugo koristila u igrama poput Doom-a, Super Mario, pa čak i u robotima poput Deep Blue. Te je zbog tog globalnog iskustva i kvalitetno istraženog područja proizašla teorija za Reaktivnu Inteligenciju [1].

2.1.1 Teorija

Svaka AI komponenta za isti input ima isti 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 jedna tip konfiguracije.



Slika 1 Determinističko Mapiranje

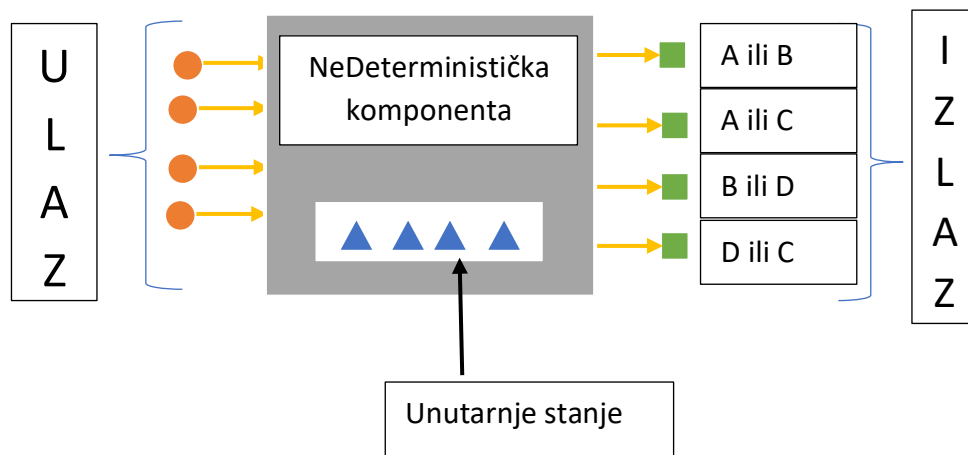
Iz navedenog, takozvane više-na-jedan funkcije zaključujemo da jedan ulaz nikad ne može rezultirati više od jednog izlaza. Pošto sustave gledamo iz perspektive komponenata i svaki izlaz je predvidljiv, takve komponente nazivamo determinističkim.

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 dolazimo do problema ukoliko je prihvatimo riječ za riječ. Naime po teoriji reaktivna komponenta je samo funkcija, nema definiranog stanja ili memorije. Alternativno determinističko mapiranje ne bi bilo moguće.

Pošto je riječ o interpretaciji, ukoliko unutarnje stanje komponente protumačimo kad još jedan od ulaza vraćamo se ponovno na reaktivnu konfiguraciju (ulaz->izlaz).



Slika 2 Ne Deterministička Komponenta



Slika 3 Deterministička Komponenta

2.1.3 *Reaktivne Tehnike u Razvoju Igara*

Reaktivne AI tehnike su bitan dio igara još od njihovog početka.

Prednosti Reaktivnih tehnika:

- U potpunosti deterministički
- Omogućava optimizaciju koda i strukture podataka
- Otklanjanje grešaka je olakšano zbog preciznijeg utvrđivanja točnog razloga
- Vrijeme izvođenja radnje je pretežno jednak

Najuspješnije reaktivne tehnike:

- Skripte kao mali programi koji za dane parametre proizvedu rezultat. Koristi se čist programski jezik za rješavanje problema(Python, Lua).
- Sustavi temeljeni na zbirci pravila „ako...onda“ koja se koriste za transformaciju varijabla. Namijenjeni većim problemima, pružaju fleksibilan i modularan pristup uz vrlo mali razvojni trošak. Loši kod niže razine kontrole.
- Strojevi sa 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 veš mašine, klima uređaja, automatike...

Cilj arhitekture je definiranje veza između komponenata.

2.1.5 Komponenta

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 veš mašine, klima uređaja, automatike...

Cilj arhitekture je definiranje veza između komponenata.

Komponenta

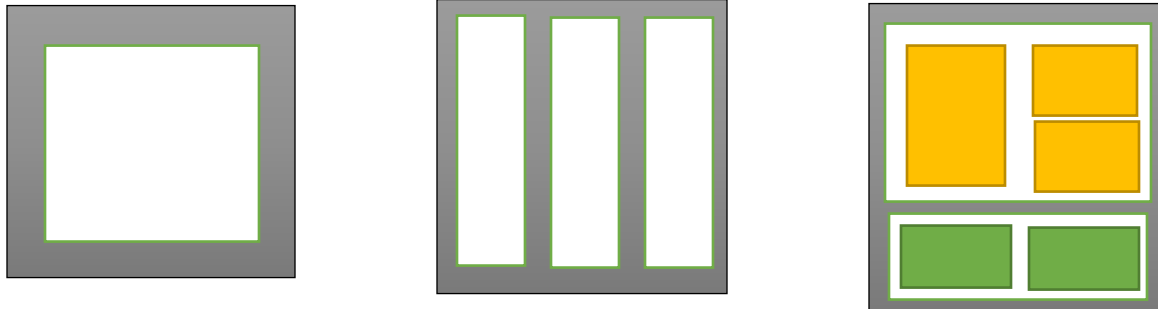
Ideju komponente bi trebali zamišljati kao crnu kutiju, sa ulazima i izlazima u kojoj se ulaz transformira na „nepoznat“ način. Također komponenta može biti kombinacija manjih sustava kod kojih se međusobna komunikacija izvršava uz pomoć ulaza/izlaza.

Takvim dizajnom dolazimo do modularnosti umjetne inteligencije, odnosno možemo koristiti ugniježdene komponente kao jedan sustav.

2.1.6 Organizacija

Postoji nekoliko načina slaganja komponenti u jedan sustav.

Tri takva tipa su:



Slika 4 Od Lijeva na Desno Primjeri arhitektura | Monolitna | Izravната | Hijerarhijska

- Monolitne arhitekture uključuju samo jednu komponentu
- Izravната arhitekture imaju paralelno mnogo komponenti
- Hijerarhijski modeli imaju komponente ugniježdene unutar drugih.

O odabiru organizacije u arhitekturi odlučujemo ovisno o kompleksnosti. Za jednostavne probleme je dovoljno korištenje monolitne arhitekture, dok teži problemi se lakše rješavaju uz pomoć izravната arhitekture. Uz Hijerarhijski model moguće je riješiti bilo koji problem na način da se rastavi na dijelove.

Primjerice dizajniranjem nekog entiteta sa umjetnom inteligencijom nad kojim želimo da lovi, patrolira, bježi (prvi set), te želimo da u isto vrijeme trči i gađa (drugi set) odnosno prvi set bi ovisio o drugom setu. Takvu arhitekturu nazivamo tri-razine hijerarhije.

2.1.7 Dekompozicija

Da bi uspješno odabrali arhitekturu, prvo si moramo postaviti pitanje kako rastaviti dan problem. Ideja je rastaviti problem ovisno o određenim kriterijima.

Neki od tipova rastavljanja su:

- Strukturno rastavljanje
 - Rastavljamo rješenje ovisno o funkciji svake komponente(npr. trčati, gađati)
- Bihevioralna dekompozicija
 - Bazirana na karakterističnim aktivnostima sustava. Za razliku od strukturne, imamo procedure umjesto čistih funkcija.
- Ciljno rastavljanje
 - Koristi se ciljem sustava te na njemu bazira rastavljanje problema.

Također imamo hibridne dekompozicije, primjerice prvotni problem može biti rastavljen kao ponašanje, te svako ponašanje rastavljeno na određene funkcije. To nam dozvoljava korištenje više kriterija u odnosu na čiste dekompozicije.

2.1.8 Odlučivanje

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

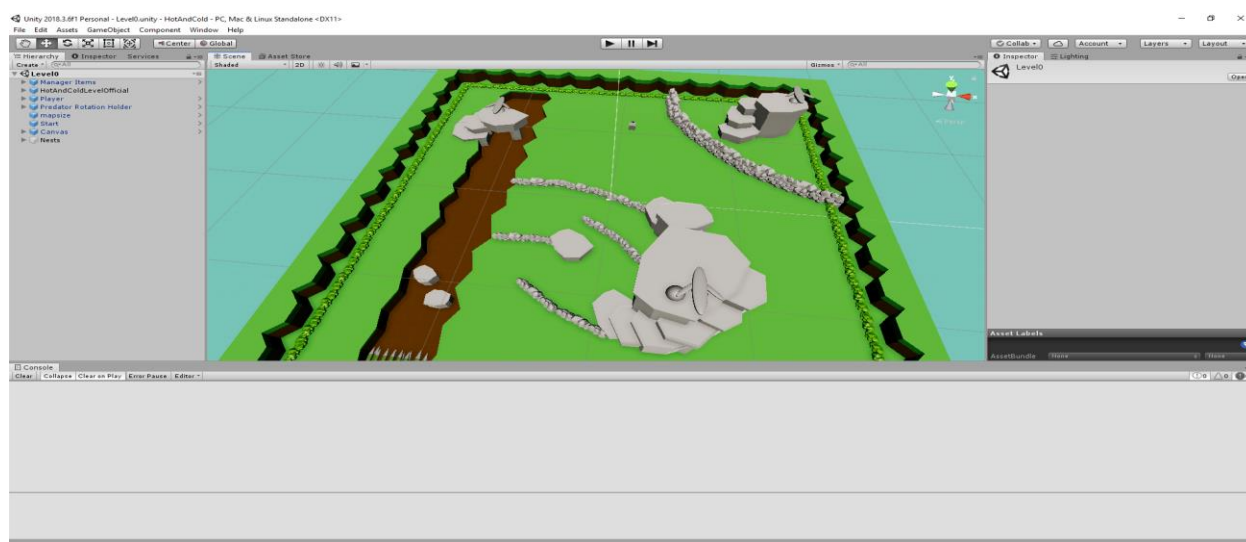
Poznata su 4 tipa interpretacije izlaza:

- Neovisan zbroj
 - Povezuje svaku komponentu s različitim izlazima kako ne bi došlo do sukoba
- Kombinacija
 - Omogućuje kombiniranje izlaza različitih komponenata kako bi se dobio konačni rezultat
- Suzbijanje
 - Neke komponente dobivaju prednost nad drugima
- Sekvencijalno odlučivanje
 - Prati kako se s vremenom izmjenjuju ishodi različitih komponenti

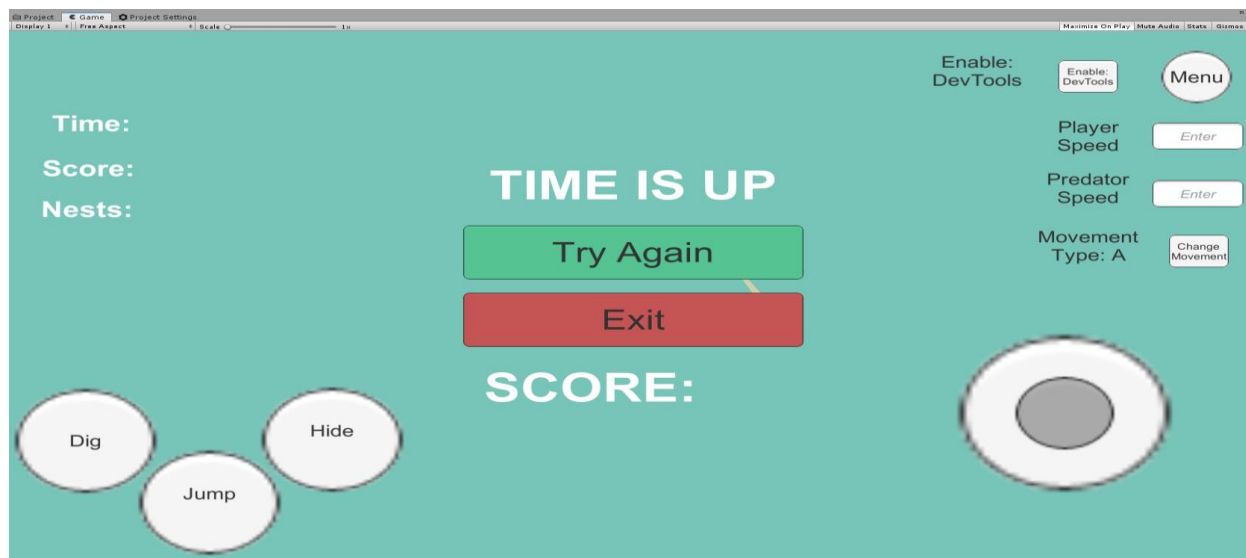
3.1 Unity Uređivač

Sadrži poglede i ostale opcije, primarni pogled je scena gdje se igra vizualno uređuje, te pogled Igra koji prikazuje igru na način koji bi igrač doživio. Za pokretanje igre koristimo gumb pokreni, gumb pauza za trenutno stopiranje igre te gumb sljedeći koji tada pokreće igru okvir po okvir.

U toku rada igre u mogućnosti smo raditi promjene na sceni ali te promjene neće biti spremljene.



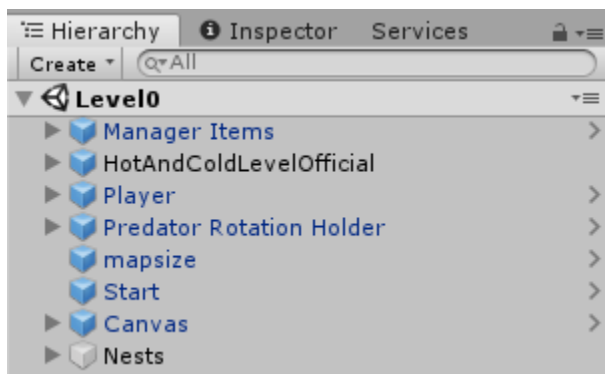
Slika 7 Pogled na Unity Uređivač i Pogled Scena



Slika 8 Pogled Igra

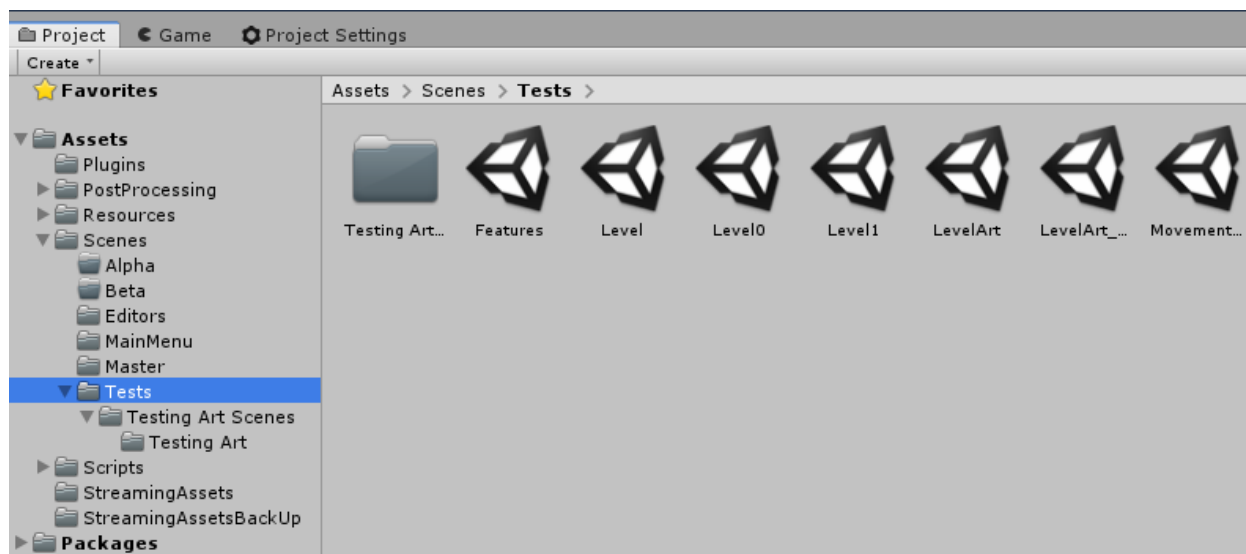
Sekundarni pogledi Hijerarhija, Projekt i Inspektor nam služe za podešavanje igre.

Hijerarhija sadrži sve objekte igre na trenutnoj sceni, te sve buduće instance koje se mogu kreirati tokom rada igre.



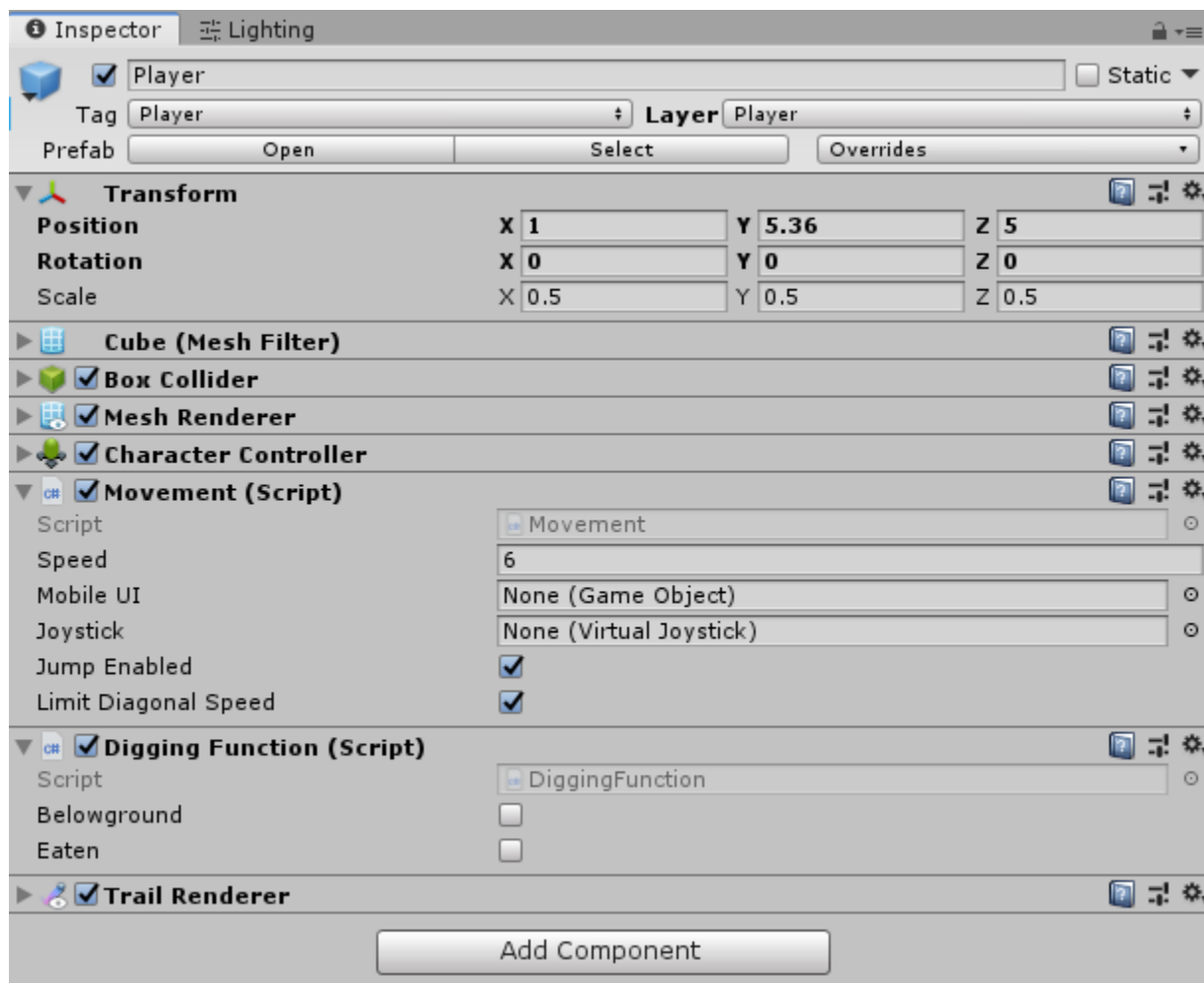
Slika 9 Pogled Hijerarhija

Projekt sadrži sve multimedijske dijelove igre, uključujući skripte i datoteke.



Slika 10 Pogled Projekt

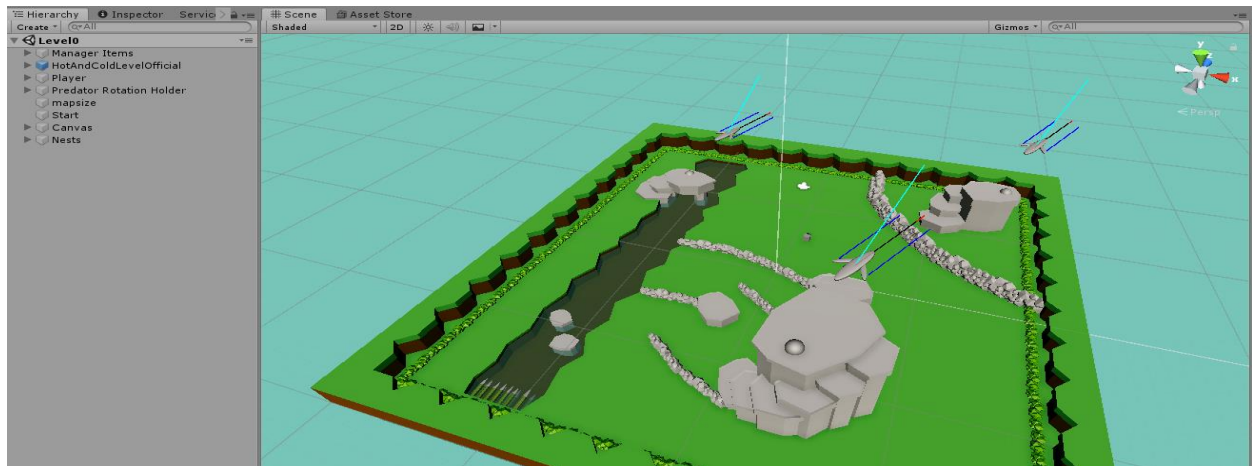
Inspektor se koristi za pregled i podešavanje komponenti objekta igre i ostalih postavka. Omogućuje dodavanje novih komponenti, kopiranja komponenti ili samo njihovih vrijednosti ili ponovno postavljanje na prvobitno stanje komponente.



Slika 11 Pogled Inspektor

3.2 Scene

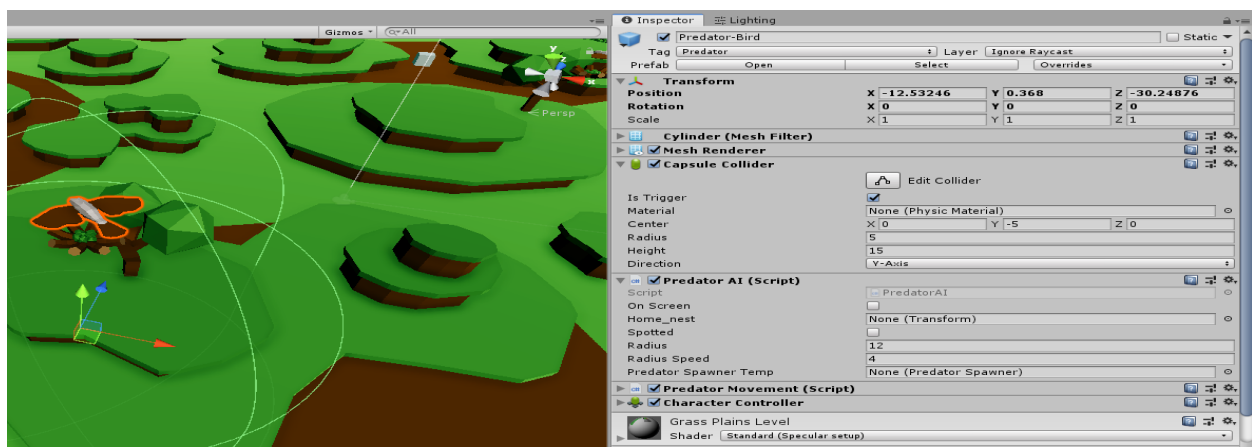
Sadrže okruženje i izbornik igre. Svaka scena je kao jedna razina igre, te u njima slažemo, dizajniramo i postavljamo objekte. Scena također sadrži objekte koji nemaju vizualnu reprezentaciju ali su bitni zbog organizacije ili logike u igri.



Slika 12 Primjer Scene

3.3 Objekti Igre i Komponente

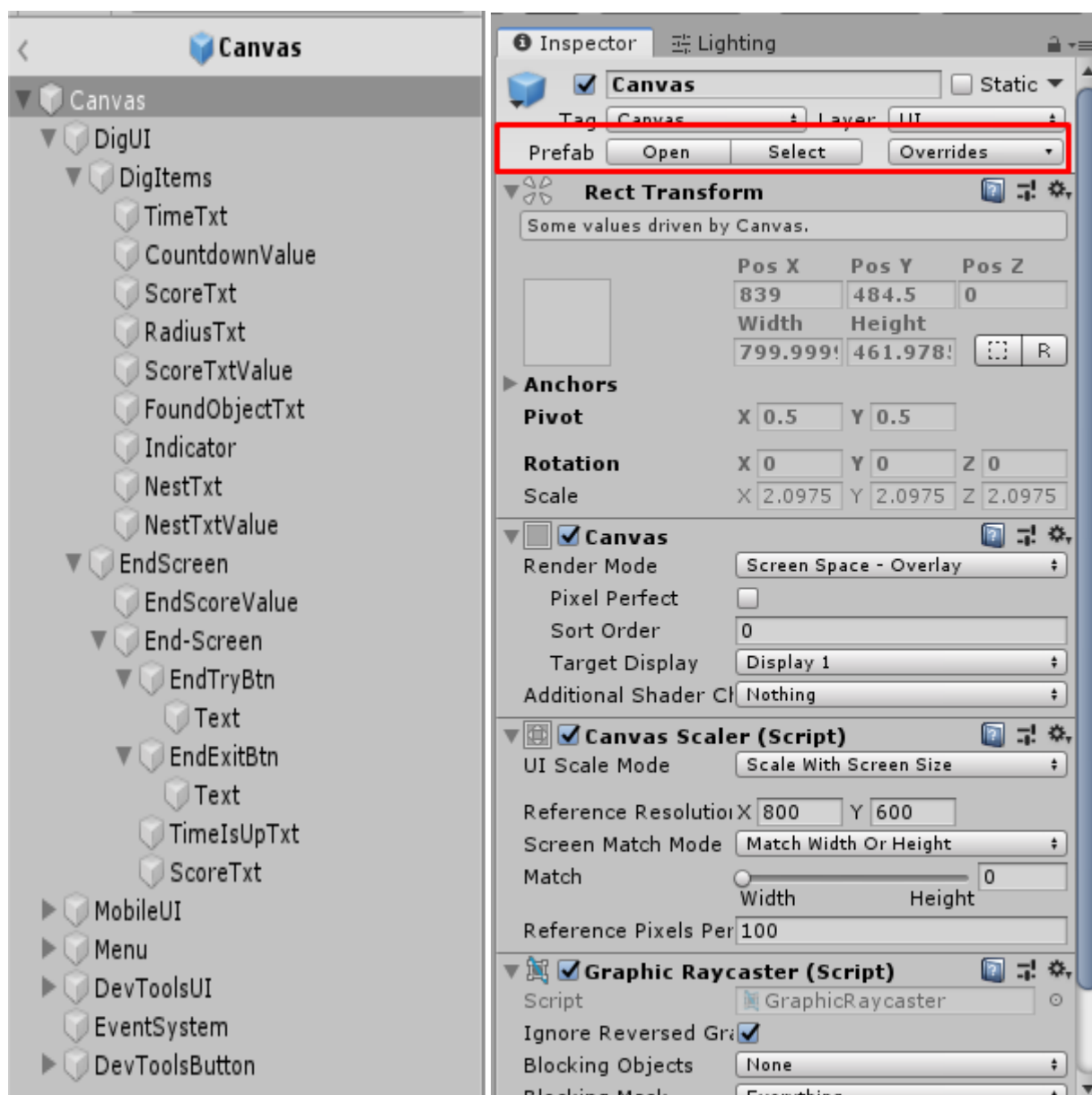
Objekti Igre sadrže komponente. Komponente mogu biti skripte, coliders, renderers... Svaki objekt igre dolazi sa transform komponentom ili sa rect tranform komponentom u slučaju UI objekata. Svi uključeni objekti se mogu pronaći u hijerarhijskog pogledu te su poslagani u hijerarhijskoj strukturi ali u smislu veze otac-dijete.



Slika 13 Objekt Ptica i Njegine Komponente

3.4 Montaže

Montaže su unaprijed konfigurirani Objekti igre koji se mogu koristiti kao predlošci u sceni. Mogu se instancirati ili klonirati u sceni ili tokom igre, te promjene nad jednom montažom se mogu prenijeti na sve njezine instance.



Slika 14 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

Igrač kontrolira čovječuljka (simbolizira krticu) nad kojim se može kretati po mapi, skakati, kopati, odnosno ići ispod ili iznad zemlje, očitavati lokaciju boda na temelju toplo/hladno te taj bod iskopati. Potrebno je izbjegavati neprijatelje (simbolizira sokola), a u slučaju da krenu u napad igrač može jedino pobjeći ili sakriti se ispod zemlje. Igrač ima i mogućnost eliminiranja neprijateljeve baze (simbolizira gnijezdo) čime će se lakše kretati u njenoj okolini, ali će susjedna baza biti teže za srušiti (detaljnije u drugom poglavlju).

4.1.1 Cilj Igre

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

Osnovne mogućnosti cilja:

- Skupi x bodova
- Odbrojavanje od x vremena
- Štoperica
- Eliminiranje neprijateljeve baze
- Ništa

Te iste mogućnosti cilja možemo koristiti za aktivaciju definiranog bonusa što uključuje povećanje vremena, brzinu kretanja čovječuljka i brzinu kopanja čovječuljka.

4.1.2 Izbornik Igre



Slika 15 Početni Zaslon Igre



Slika 16 Izbornik za Odabir Razine (Zeleno označuje osvojenu razinu)

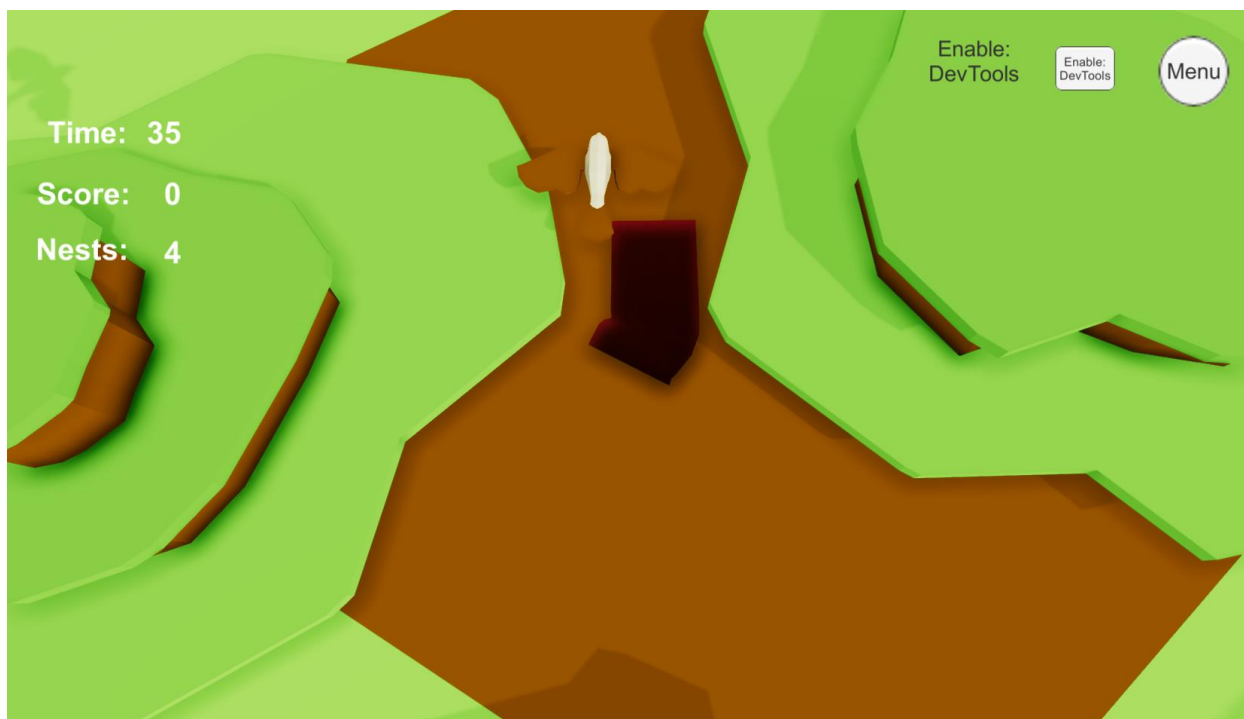
4.1.3 Pogled na Razinu Igre



Slika 17 Pogled Na Razinu | U desnom kutu (Vrijeme | Broj Bodova | Broj Gnijezda) | U lijevom kutu (Aktivacija DevTools-a | Izbornik)



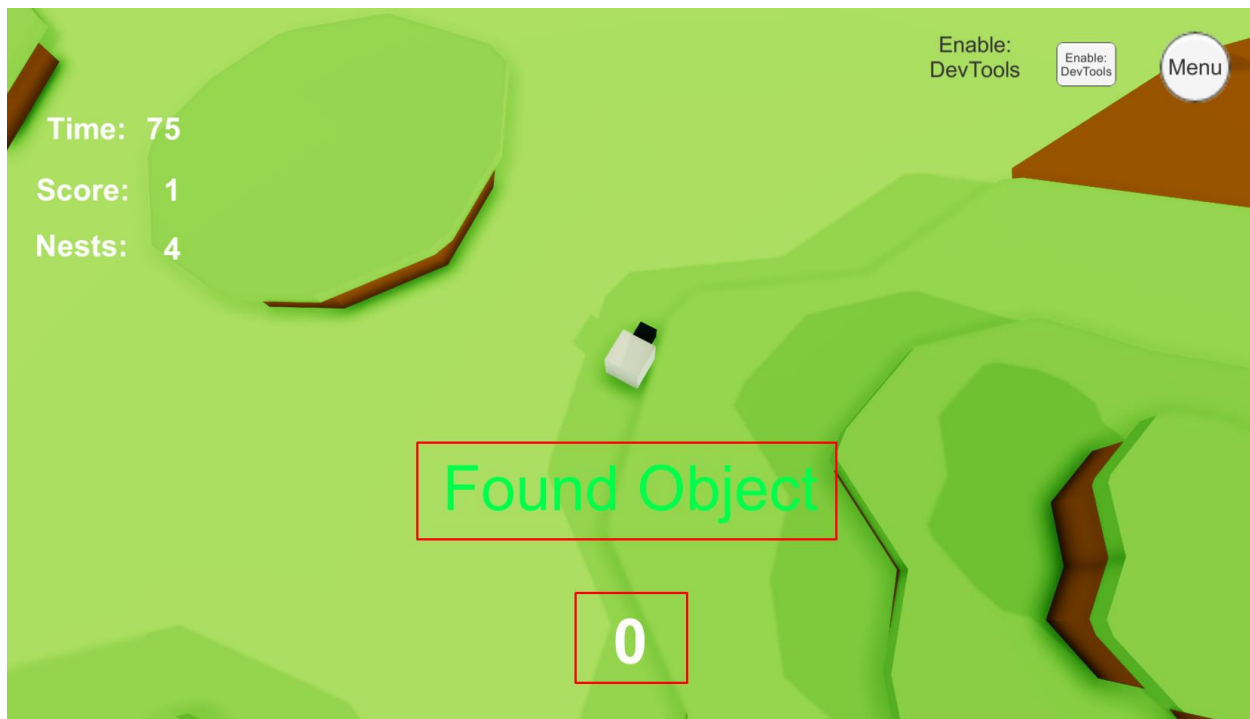
Slika 18 Napad Ptice i Njeno Gnijezdo



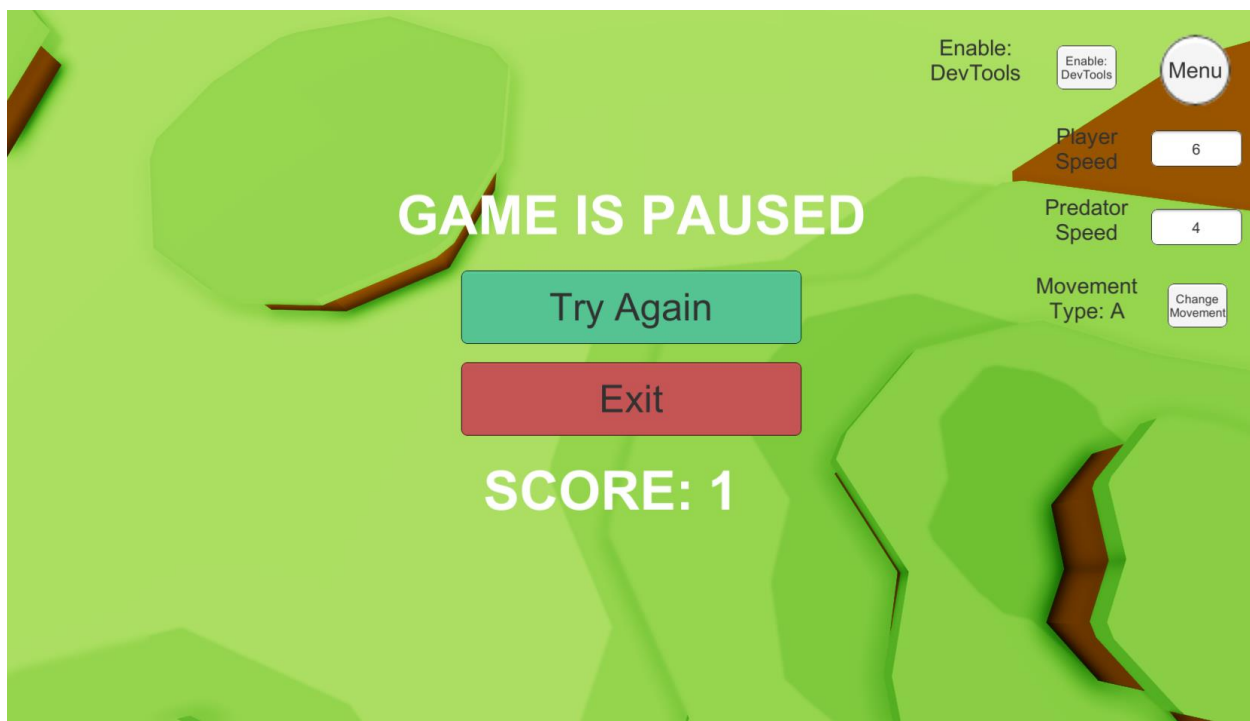
Slika 19 Ptica iščekuje Igrača na Površini | Igrač je ispod zemlje



Slika 20 Indikator Udaljenosti od Boda (Dolje) | Indikator Smjera iz Kojeg Dolazi Ptica (Sredina Ulijevo)



Slika 21 Pronađen Bod | Broj Iskopa do Boda



Slika 22 Pauzirana Igra | Aktiviran DevTools (Omogućava mijenjanje brzine Igraču, Neprijatelju i mjenjanje tipa kontrole igrača)

4.2 Organizacija

4.2.1 Organizacija Scena

Scene su podijeljene na:

- Master
 - Sadrži objekt igre sa komponentama kontrole svih dijelova igre
- Izbornik
 - Sastoji se od glavnog izbornika i izbornika odabira razine
 - Služi kao tranzicija između master scene i scena razine
- Razine
 - Sadrže sve ovisno o određenoj razini
 - Njihov naziv također služi kao identifikacija u json bazi



Slika 23 Opcije Izdranje 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 svi komponenti kontrole sadržani unutar GameController objekta igre. Sa 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, kopanje, a njegovo dijete „Director“ sadrži skriptu za očitavanje sakrivenog boda.

4.3 Struktura i Komunikacija

Svi kontroleri su MonoBehaviour, ali i nasljeđuju GetControllers i ScriptControllers, te ControllersTemplate. GetControllers sadrži zaštićena svojstva koja služe za međusobni pristup kontrolerima. Također imamo interface 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
    {
        41 references
        protected HotAndColdController HotAndColdController { get { return GetComponent<HotAndColdController>(); }}
        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 24 GetControllers i ControllersTemplate

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

```

using UnityEngine;
using Controllers;

10 references
/**/ class ScriptController : GetControllers
{
    2 references
    protected void FinalizingLoadLevel()...
    // Use this for initialization
    2 references
    protected void EnableHotAndColdGameScripts()
    {
        GetComponent<ObjectPlacer>().enabled = true;
        //GetComponent<PredatorSpawner>().enabled = true;
        HotAndColdController.enabled = true;
        Timer.enabled = true;
        CanvasController.enabled = true;
        ButtonController.enabled = true;
        EventController.enabled = false;
    }
    3 references
    protected void DisableHotAndColdGameScripts()...
    1 reference
    protected void DisablePlayer()...
    2 references
    protected void EnablePlayer()...
    1 reference
    protected void InitializePositionAndStats()...
    1 reference
    protected void SetRule(int id_rule)...
    1 reference
    protected void StopAll_AI()...
    1 reference
    protected void ResumeAll_AI()...
    2 references
    protected void DeleteAll_AI()...
    1 reference
    protected int CheckNestCount()...

```

Slika 25 ScriptController | Prikazana je metoda koja aktivira sve vezano uz HotAndCold Kontroler pri učitavanju razine

4.4 Kontroleri

4.4.1 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, postavlja težinu iskopa sakrivenog boda, i inicijalizira objekt bod na sceni. Poslije inicijalizacije vrći update metodu u kojoj čeka input igrača, odnosno očitavanje udaljenosti od boda ili kopanje kada pronade bod, CalculateStopTime koji simulira zastoj kopanja pri očitavanju udaljenosti i provjeru u slučaju da se bonus aktivirao.

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

Slika 26 HotAndCold Kontroler | Prikazana je update metoda

Također sadrži metode za provjeru statusa razine:

- Provjeri je li ispunjen uvjet za pobjedu
- Provjeri je li ispunjen uvjet za kraj razine
- Provjeri je li ispunjen uvjet za Bonus

```
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].objective);
}
1 reference
public bool SeeIfConditionMetBonusCondition()
{
    return CheckCondition(localdatabase[0].BonusCondition[0].condition, localdatabase[0].BonusCondition[0].objective);
}
//----- 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 27: Prikazan je HotAndCold Kontroler | Sve provjere prolaze kroz switch

4.4.2 Canvas Kontroler

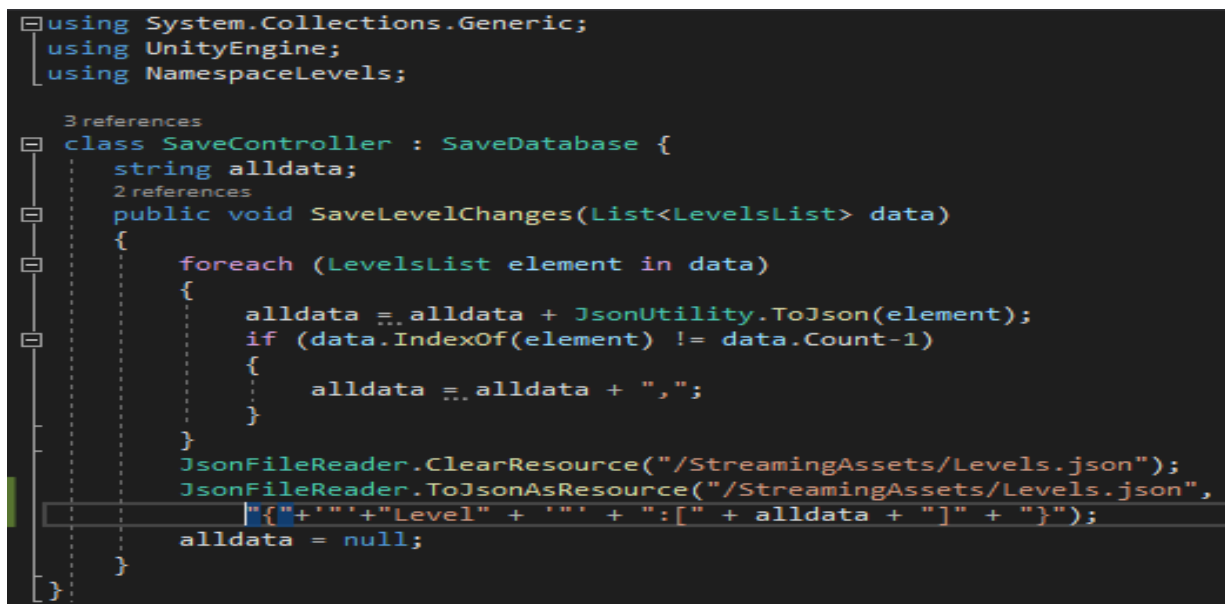
Osvježava i upravlja svim UI komponentama na učitanjoj razini.

4.4.3 DevTool Kontroler

Upravlja i reagira na promjene razvojnih alata na učitanjoj razini.

4.4.4 Save Kontroler

Služi za spremanje statusa razine u json bazu.



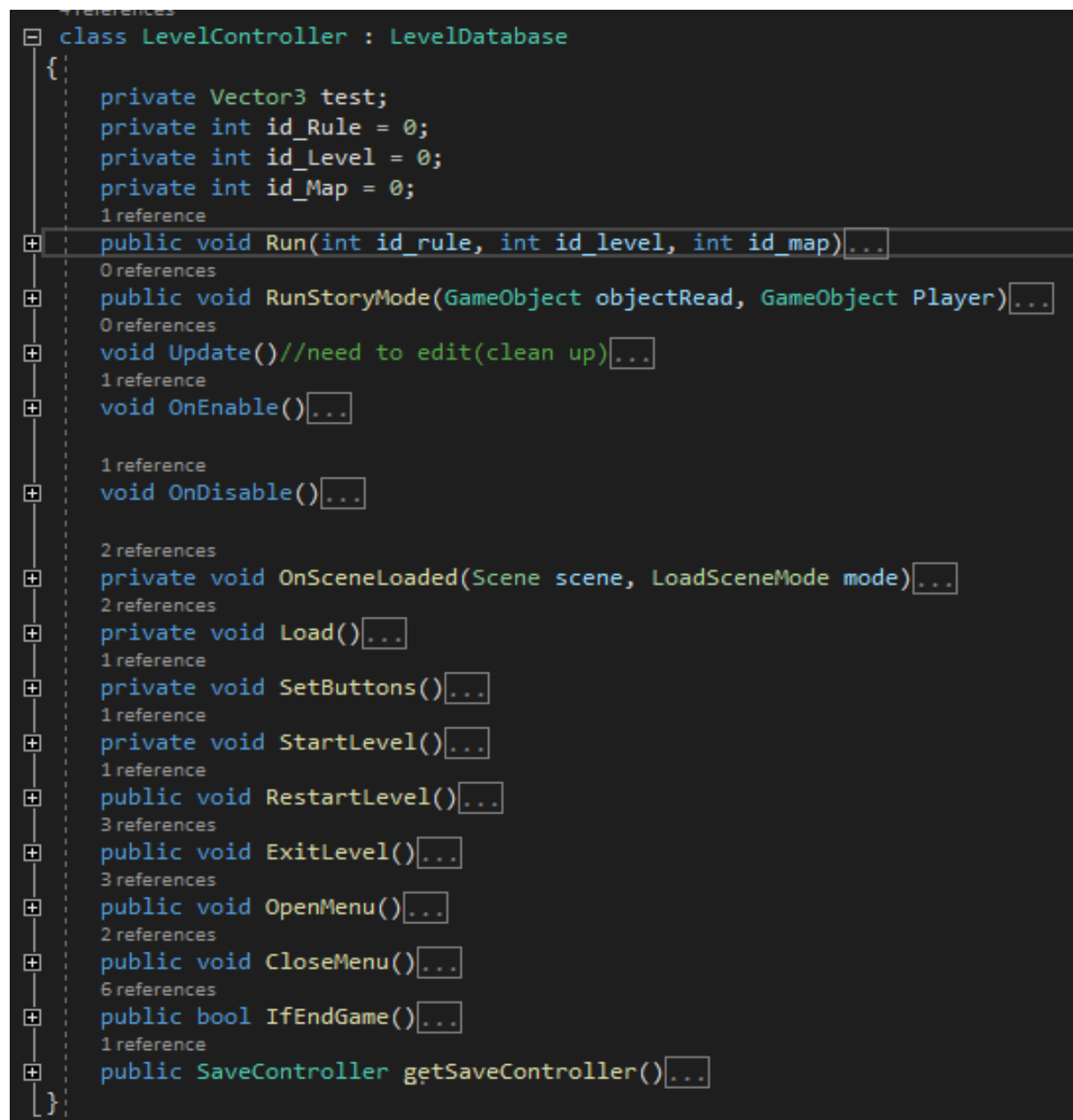
```
using System.Collections.Generic;
using UnityEngine;
using NamespaceLevels;

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

Slika 28: SaveController | Sprema podatke u obliku json sadržaja

4.4.5 Level Kontroler

Upravlja svim aspektima razine. 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 29 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 30 Json Datoteke

GameRules.json datoteka sadrži sva pravila igre. Sva pravila se učitavaju pri pokretanju igre, spremaju se privatnu varijablu te njima pristupamo pomoću identifikacijskog broja.

```
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;
}
```

```
namespace NamespaceGameRules
{
    [System.Serializable]
    3 references
    public class GameRules...
    [System.Serializable]
    8 references
    public class GameRulesList...
    [System.Serializable]
    3 references
    public class TimeClass...
    [System.Serializable]
    3 references
    public class ScoreClass...
    [System.Serializable]
    7 references
    public class ConditionClass...
    [System.Serializable]
    3 references
    public class BonusClass...
}

{"GameRule": [
{
    "ID": 0,
    "Time": [{"starttime": 60, "direction": -1}],
    "Score": [{"xitem": 1}],
    "Win": [{"condition": "foundonce", "objective": "2"}],
    "EndGame": [{"condition": "time", "objective": "0"}],
    "BonusCondition": [{"condition": "found", "objective": "2"}],
    "Bonus": [{"bonustime": 15, "movementspeed": 1.15, "diggingspeed": 1.15}]
},
]
```

Slika 31 Po Redu: Game Rules Database -> učitava i omogućava pristup Pravilima Igre | Definira strukturu podataka u json datoteci | Primjer jednog pravilnika igre u json-u

Sljedeće imamo Levels i LevelsSave. LevelsSave nam je samo backup Levels-a, te se on poziva pri 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.

Napomena: „num_Stars“ nije implementiran.

```
{ "Level": [
  { "ID": 0, "id_Rule": 1, "id_Map": 0, "Pass": false, "num_Stars": 0, "Score": 0.0 },
  { "ID": 1, "id_Rule": 0, "id_Map": 1, "Pass": false, "num_Stars": 0, "Score": 0.0 },
  { "ID": 2, "id_Rule": 2, "id_Map": 1, "Pass": false, "num_Stars": 0, "Score": 0.0 },
  { "ID": 3, "id_Rule": 3, "id_Map": 1, "Pass": true, "num_Stars": 0, "Score": 0.0 }
]
```

Slika 32 Primjer kako su definirane razine u Json-u

4.6 Umjetna Inteligencija Entiteta

Simuliramo neprijatelja (simbolizira sokola), njegovo ponašanje:

- Patroliranje baze (simbolizira gnijezdo)
- Napadanje
- Prizemljenje
- Reorganizaciju
- Obnavljanje baze (simbolizira gnijezdo)

4.6.1 Opis Ponašanja

Pri učitavanju razine neprijatelj se nalazi na bazi, Spiralno kruži oko njega dok ne uspostavi određenu orbitu na definiranoj visini oko baze. Ukoliko igrač sa 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 ukoliko se igrač do tada nije sakrio vjerojatno će 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 reorganizaciju (vraćanje u bazu), te patroliranje baze. Također je moguće za igrača da eliminira bazu neprijatelja, pri čemu će se svaka buduća reorganizacija dogoditi u susjednoj bazi (ili centralnoj bazi -> ukoliko su sve baze eliminirane) na neodređeno vrijeme nakon kojega slijedi obnavljanje prvobitne baze.

4.6.2 Kretanje Entiteta

Neprijatelj se kreće na način 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 na način da se kalkuliра normala ovisno o odredištu te se neprijatelj rotira određenom brzinom svaki okvir dok nije okrenut istom pravcu. Time se dobiva ugađeno kretanje.

```
1reference
protected void Move(Vector3 destination)
{
    destination = new Vector3(destination.x, thisMinHeight, destination.z);
    if (!destinationSet)
    {
        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;
    }

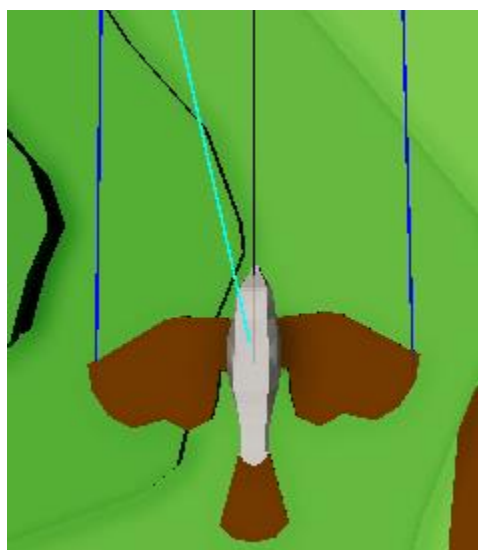
    Debug.DrawRay(transform.position, diff, Color.red);
}
```

Slika 33 Move Metoda iz Predator Movement | Algoritam koji transformira (pokreće ga) neprijatelja

4.6.3 Izbjegavanje Prepreka

Detekcija prepreka radi na način da se na svaki okvir ispituje ukoliko je zraka (prikazano plavom na slici) pogodila neki objekt (u slučaju igre registriraju se jedino objekti sa osnovnim slojem).

Ukoliko se desi detekcija poziva se metoda za pronalazak sigurne rute. Metoda radi na način da u isto vrijeme pronalazi rutu na lijevo i desno uz pomoć lijeve i desne zrake. Za svaki okvir se dodaju 5 stupnja dok ne postignemo da zraka ne pogađa niti jedan objekt. Ako se ruta ne pronade, a obe zrake su prešle 90 stupnjeva, tada se jednom testira ukoliko zraka pogađa objekt ravno iznad neprijatelja (ukoso naprijed) te se onda šalje neprijatelju da uradi salto unazad, u suprotnom će ići iznad objekta.



Slika 34 Pogled iz Scene na neprijatelja sa vizualiziranim zraka

```
1 reference
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.TransformDirection(-forward), out obstacle, raySize))
    {
        return true;
    }
    return false;
}
```

Slika 35 Metoda Očitavanja Prepreke


```

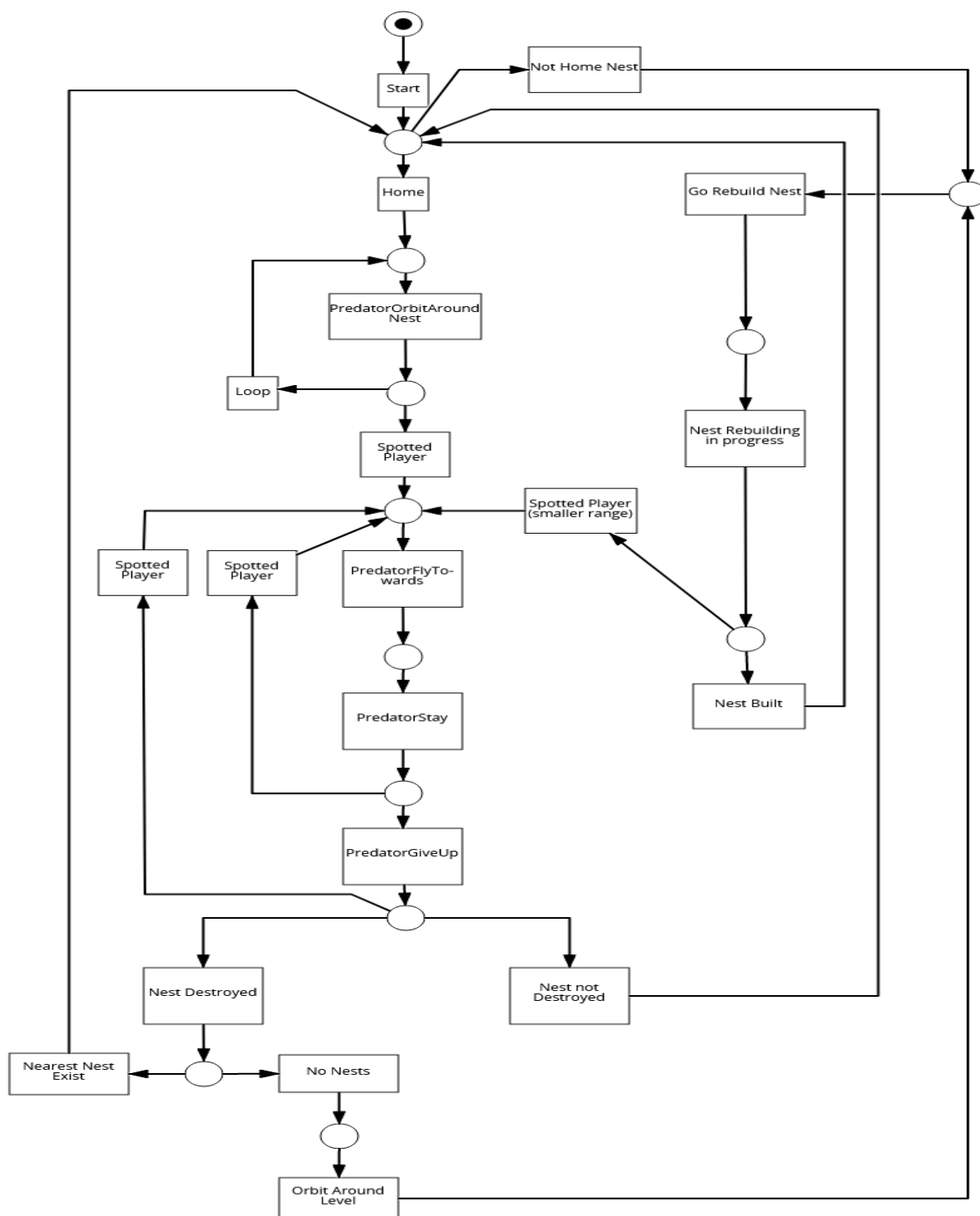
if (angleRight > 90 || angleLeft < -90)
{
    if (Physics.Raycast(transform.position, transform.TransformDirection(forward) + transform.up, out pathFound, raySize) || Physics.Raycast(transform.position, transform.TransformDirection(forward) - transform.up, out pathFound, raySize))
    {
        direction = 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) || Physics.Raycast(transform.position, testAroundLeft, out pathLeft, raySize))
    else
    {
        Finalize(testAroundRight, 1);
        //Debug.Log("Right working");
    }

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

```

Slika 36 Isječak Metoda koja pronalazi sigurnu rutu

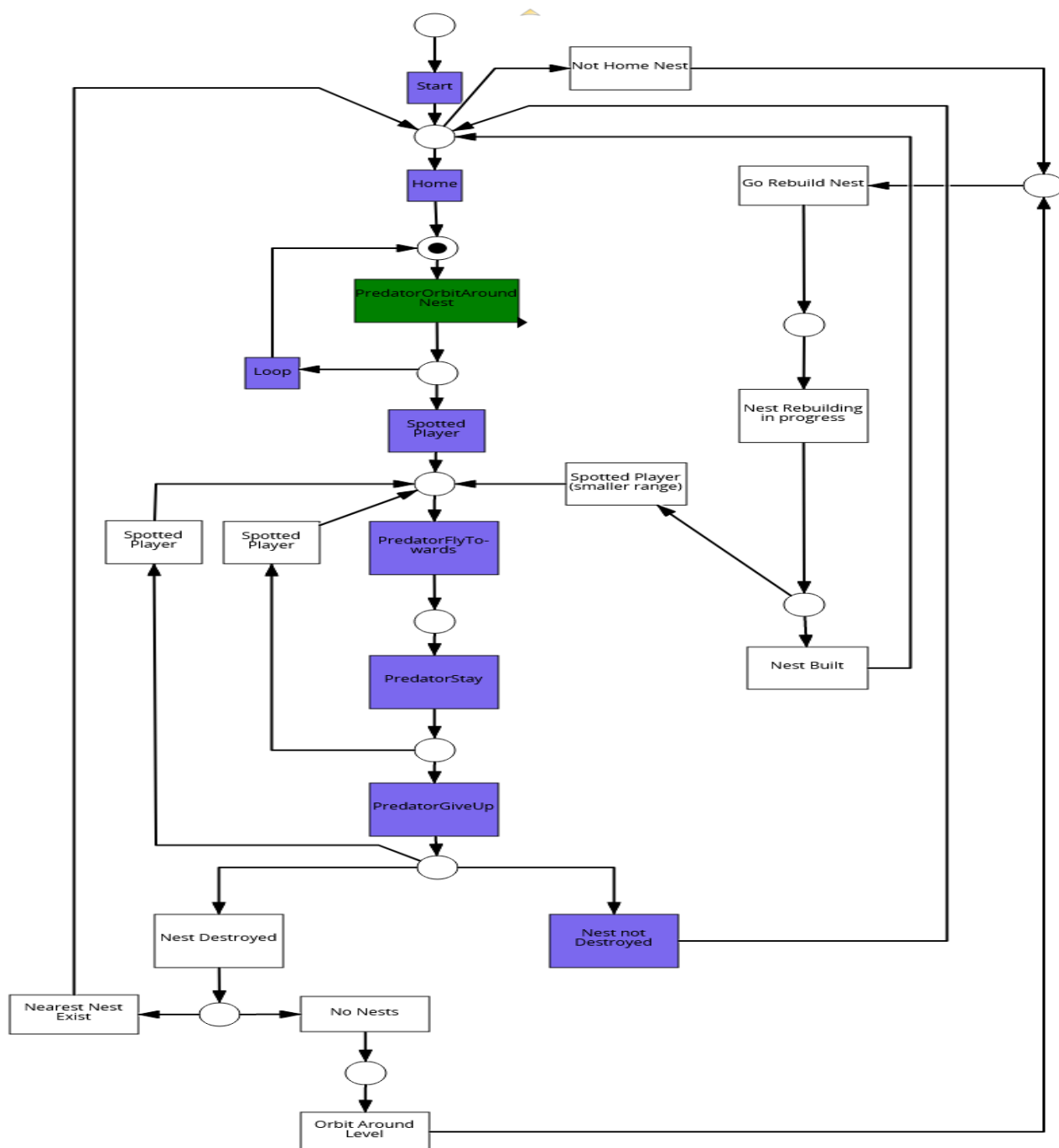
4.7 Ponašanje Entiteta Kroz Petrijeve Mreže



Slika 37 Petrijeva Mreža Ponašanja Entiteta

4.7.1 Osnovno Ponašanje

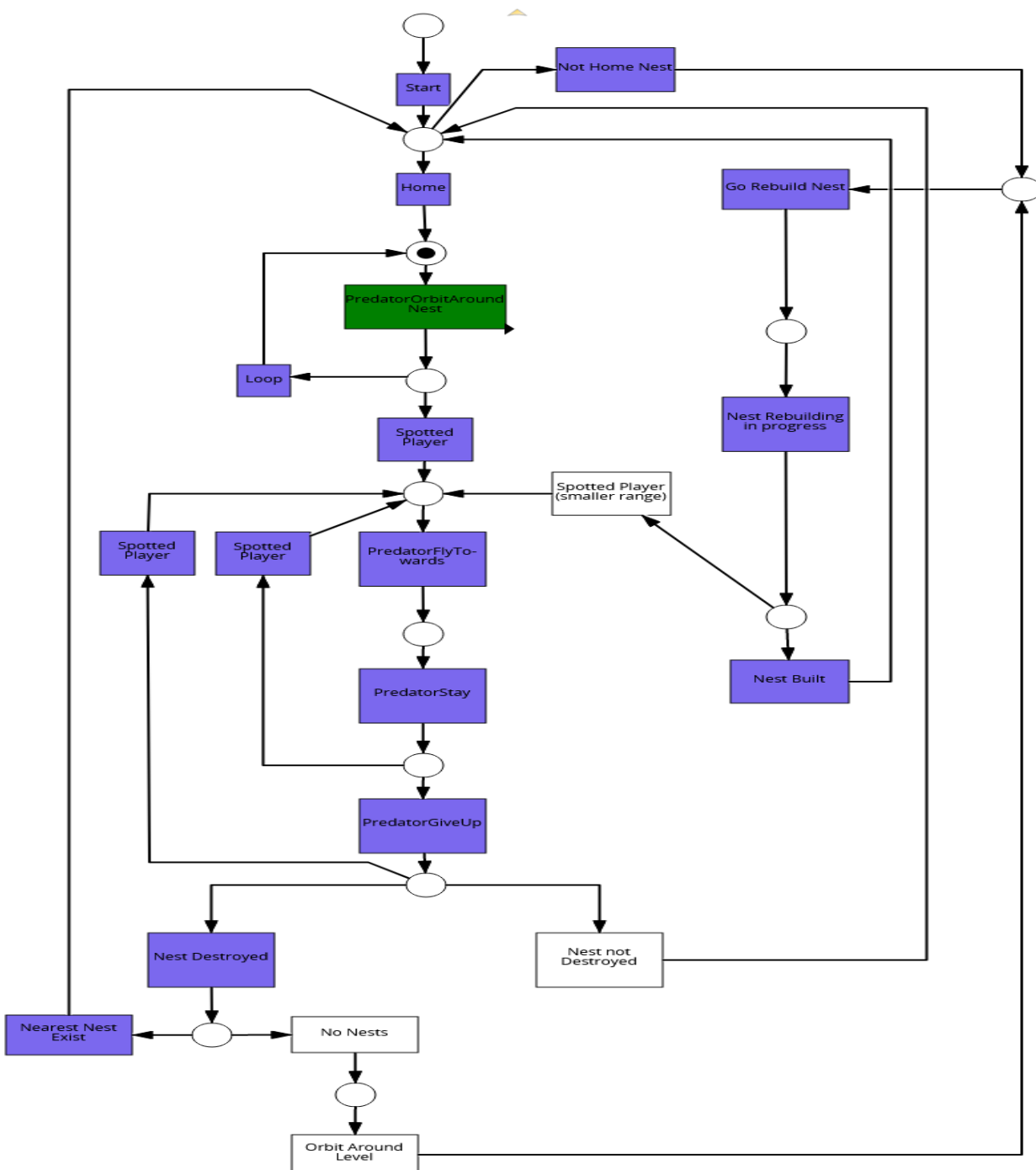
Imamo patroliranje oko baze dok se ne primijeti igračev čovječuljak, naknadno kada je čovječuljak sakriven prelazimo na prizemljenje. Nakon neodređenog vremena neprijatelj odustaje i vraća se natrag u bazu.



Slika 38 Petrijeva Mreža Osnovnog Ponašanja Entiteta

4.7.2 Ponašanje Prilikom Eliminacije Neprijateljeve Baze

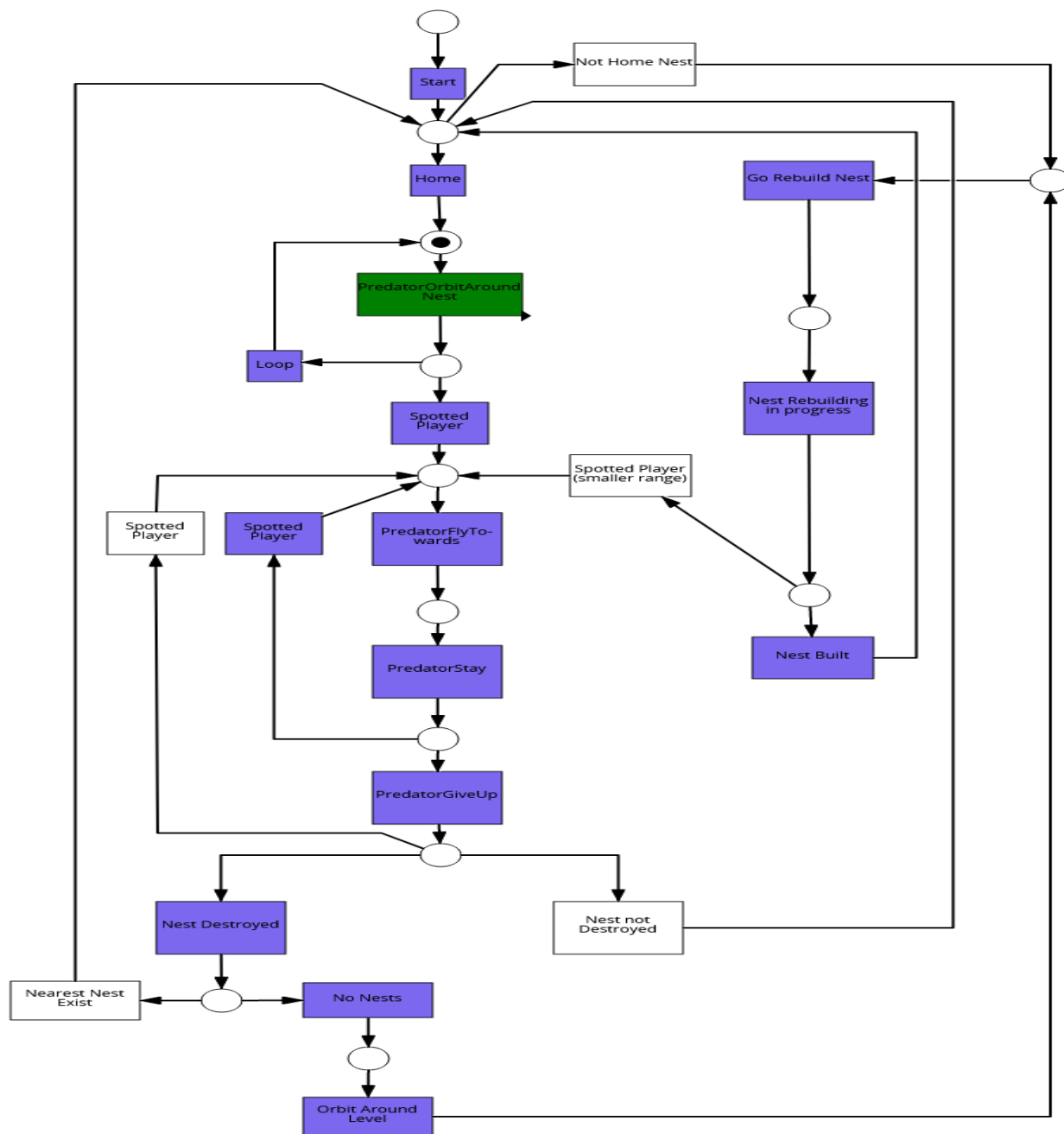
Imamo prijašnje osnovno ponašanje, uz dodatne detekcije igrača od strane neprijatelja, i zbog eliminacija neprijateljeve baze traži se najbliža sljedeća baza. Nakon nekog vremena kruženja oko te baze, neprijatelj kreće u obnavljanje primarne baze.



Slika 39 Petrijeva Mreža Ponašanje Prilikom Eliminacije Neprijateljeve Baze

4.7.3 Ponašanje Prilikom Eliminiranja Svih Baza

Imamo prijašnje 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. Također treba napomenuti da u rasponu od procesa izgradnje baze i završetka izgradnje baze, neprijatelj će napasti igrača ukoliko ga detektira.



Slika 40 Petrijeva Mreža Ponašanje Prilikom Eliminiranja Svih Baza

5. Zaključak

Reaktivna komponenta je determinističko mapiranje ulaza -> izlaza. Komponente sa unutarnjom memorijom mogu se smatrati reaktivnima, te moguće je izraziti ne determinističke komponente (sa unutarnjim stanjem) kao reaktivne korištenjem dodatnim ulaza. Također komponente tretiramo kao crne kutije koje možemo slagati u skupine, te prednost je u tome što brzo reagiraju na vanjske zahtjeve ili na podražaje iz okoline.

Zaključili smo 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, zanimljiva i frustrirajuća. Najteži dio je bio napraviti kvalitetno kretanje (letenje), te u to dodati izbjegavanje prepreka koje još uvijek nije savršeno te sadrži određene blokade u slučaju da se ne uspije vratiti u prijašnje stanje prije izbjegavanja. Postoje bolje alternative ali koriste mnogo više zraka za detekciju dok moja detekcija uspijeva u tome 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, <file:///C:/Users/Joso/AppData/Local/Temp/2016-11-ai-reactive-robots-self-aware.pdf>, 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

7. Popis Slika

Slika 1 Determinističko Mapiranje	3
Slika 2 Ne Deterministička Komponenta	4
Slika 3 Deterministička Komponenta.....	4
Slika 4 Od Lijeva na Desno Primjeri arhitektura Monolitna Izravnata Hijerarhijska ...	7
Slika 5 Platforme Podržane od Strane Unity [3]	10
Slika 6 Formati Podržani od Strane Unity [3].....	10
Slika 7 Pogled na Unity Uređivač i Pogled Scena	11
Slika 8 Pogled Igra.....	11
Slika 9 Pogled Hijerarhija	12
Slika 10 Pogled Projekt	12
Slika 11 Pogled Inspektor	13
Slika 12 Primjer Scene.....	14
Slika 13 Objekt Ptica i Njezine Komponente	14
Slika 14 Montaža u Hijerarhiji Montaža u Inspektoru	15
Slika 15 Početni Zaslona Igre	17
Slika 16 Izbornik za Odabir Razine (Zeleno označuje osvojenu razinu)	17
Slika 17 Pogled Na Razinu U desnom kutu (Vrijeme Broj Bodova Broj Gnijezda) U lijevom kutu (Aktivacija DevTools-a Izbornik)	18
Slika 18 Napad Ptice i Njeno Gnijezdo.....	18
Slika 19 Ptica Iščekuje Igrača na Površini Igrač je ispod zemlje.....	19
Slika 20 Indikator Udaljenosti od Boda (Dolje) Indikator Smjera iz Kojeg Dolazi Ptica (Sredina Ulijevo)	19
Slika 21 Pronađen Bod Broj Iskopa do Boda.....	20
Slika 22 Pauzirana Igra Aktiviran DevTools (Omogućava mijenjanje brzine Igraču, Neprijatelju i mjenjanje tipa kontrole igrača).....	20
Slika 23 Opcije Izdranje Igre, Odabrane Scene za Izgradnju	21
Slika 24 GetControllers i ControllersTemplate.....	23
Slika 25 ScriptController Prikazana je metoda koja aktivira sve vezano uz HotAndCold Kontroler pri učitavanju razine.....	24
Slika 26 HotAndCold Kontroler Prikazana je update metoda.....	25

Slika 27: Prikazan je HotAndCold Kontroler Sve provjere prolaze kroz switch	26
Slika 28: SaveController Sprema podatke u obliku json sadržaja	27
Slika 29 Level Kontroler Sadrži sve metode vezane uz učitavanje, izlazak i spremanje razine	28
Slika 30 Json Datoteke	29
Slika 31 Po Redu: Game Rules Database -> učitava i omogućava pristup Pravilima Igre Definira strukturu podataka u json datoteci Primjer jednog pravilnika igre u json-u ..	29
Slika 32 Primjer kako su definirane razine u Json-u	30
Slika 33 Move Metoda iz Predator Movement Algoritam koji transformira (pokreće ga) neprijatelja	32
Slika 34 Pogled iz Scene na neprijatelja sa vizualiziranim zraka	33
Slika 35 Metoda Očitavanja Prepreke	33
Slika 36 Isječak Metoda koja pronalazi sigurnu rutu	34
Slika 37 Petrijeva Mreža Ponašanja Entiteta	35
Slika 38 Petrijeva Mreža Osnovnog Ponašanja Entiteta	36
Slika 39 Petrijeva Mreža Ponašanje Prilikom Eliminacije Neprijateljeve Baze	37
Slika 40 Petrijeva Mreža Ponašanje Prilikom Eliminiranja Svih Baza	38