# Primjena neizrazite logike u stvaranju nepredvidljivih grafika prilagodljivih igraču

**Novosel, Ivan**

**Undergraduate thesis / Završni rad**

**2024**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:211:575836

*Rights / Prava:* Attribution 3.0 Unported/Imenovanje 3.0

*Download date / Datum preuzimanja:* **2024-07-28**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

*Repository / Repozitorij:*

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ORGANIZACIJE I INFORMATIKE**
**VARAŽDIN**

**Ivan Novosel**

# PRIMJENA NEIZRAZITE LOGIKE U STVARANJU NEPREDVIDLJIVIH GRAFIKA PRILAGODLJIVIH IGRAČU

**ZAVRŠNI RAD**

**Varaždin, 2024.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Ivan Novosel**

**Matični broj: 0016153678**

**Studij: Informacijski i poslovni sustavi**

# PRIMJENA NEIZRAZITE LOGIKE U STVARANJU NEPREDVIDLJIVIH GRAFIKA PRILAGODLJIVIH IGRAČU

**ZAVRŠNI RAD**

**Mentor :**

Doc. dr. sc. Bogdan Okreša Đurić

**Varaždin, srpanj 2024.**

*Ivan Novosel*

**Izjava o izvornosti**

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

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

---

**Sažetak**

U ovom radu istražuje se primjena neizrazite logike u računalnoj grafici čime se ostvaruje nepredvidljivost vizualnog doživljaja igre. Vizualni doživljaj igre u ovom radu ne uključuje samo utjecaj neizrazite logike na grafiku igre već i mehanike igre vezane uz nju. Neizrazita logika kao alat u području umjetne inteligencije ostvaruje svoju korist dajući dojam nepredvidljivo određenih parametara, no uz korištenje pseudonasumično generiranih brojeva vrijednosti koje manipuliraju računalnom grafikom postaju iznimno nepredvidljive. U teorijskom dijelu rada opisat će se što je to neizrazita logika, koje su njene primjene u umjetnoj inteligenciji te kako se može primijeniti u računalnoj grafici. U praktičnom dijelu rada izrađena je igra u kojoj igrač može manipulirati neizrazitom logikom njene računalne grafike, no teško može pretpostaviti kako ona djeluje. Zahvaljujući nasumičnosti, grafika koju igrač mijenja dobiva realističnu dimenziju. Cilj primjene neizrazite logike je ostvarivanje realističnosti računalne grafike koja često dolazi do izražaja jedino kroz kvalitetu računalne grafike. Ovim pristupom želi se omogućiti i onim manje grafički intenzivnim igrama da ostvare realističnost vizualnog dojma.

**Ključne riječi**: umjetna inteligencija, neizrazita logika, pseudonasumični generator brojeva, videoigre, računalna grafika, realizam u grafici.

# Sadržaj

# 1. Uvod

Umjetna inteligencija predstavlja tehnologiju koja, poput telefona, postaje nerazdvojiv dio iskustva modernog čovjeka. Aplikacije koje ona sama pruža u kontekstu računalne znanosti postaju nezamislivo raznovrsne te već postoje različiti alati koji generiraju slike visoke razlučivosti i kvalitete u kojima je teško prepoznati je li ih napravio čovjek ili nije. Iako tim putem generativna umjetna inteligencija stvara svoju podlogu u području računalnih igara, ona nije jedini koncept koji je moguće primijeniti kako bi određena igra bila unaprijeđena.

Neizrazita logika je koncept koji se ne mora nužno primjenjivati samo u problemima računalne grafike, već se koristi i u različitim industrijskim područjima poput mehanike i autoindustrije. Mendel [1] ističe kako za mnoge probleme postoje dva različita oblika znanja o problemu: objektivno znanje, koje se koristi u formulacijama inženjerskih problema i subjektivno znanje, koje predstavlja jezične informacije koje je obično nemoguće kvantificirati koristeći tradicionalnu matematiku (npr. pravila, stručne informacije, zahtjevi dizajna). Smatra kako se subjektivno znanje obično ignorira na početku inženjerskog dizajna; ali se često koristi za procjenu takvog dizajna. On vjeruje da bi se obje vrste znanja trebale i mogu koristiti za rješavanje stvarnih problema. Objektno znanje ulaznih varijabli se kroz neizrazitu logiku (*eng. Fuzzy Logic*) može pretvoriti u objektno znanje izlaznih varijabli na temelju pravila koje predstavljaju subjektivno znanje.

Primjena neizrazite logike može se predstaviti na primjeru sustava klimatiziranog hlađenja. Parametar koji je varijabilan predstavlja temperatura sobe. Takav parametar je nepredvidljiv za svaku osobu; osobe koje su prilagodljivije hladnoći možda će smatrati kako je ugodna temperatura jednaka 20°C, dok će druga smatrati kako je ona preniska. Neizrazita pravila i funkcije pripadnosti, temelj neizrazite logike, predstavljaju subjektivno znanje, poslovnu logiku problema klime. Kako bi izlazna vrijednost bila dobivena poput brzine ventilatora potrebno je ulaznu vrijednost poput temperature analizirati u kontekstu određenog modela neizrazite logike te primijeniti funkcije pripadnosti i neizrazita pravila koje vode do konačnog rezultata.

Problemi realizma grafika u računalnim videoigrama i danas ostaju među najbitnijim elementima dojma igre [2]. Igre koje uspijevaju usavršiti realizam su među najpopularnijima u industriji zabave zahvaljujući interakciji i dojmu koje one pružaju, a time dovode i do uspjeha unatoč zasićenom tržištu.

Realizam se smatra čestom komponentom različitih visoko-kvalitetnih igara koje zahtijevaju intenzivne računalne resurse. Jednostavnije igre, koje se temelje na različitim apstrakcijama, često nemaju stil kojim se ostvaruje impresivna grafika. Spajanjem koncepata neizrazite logike i računalnih grafika unutar videoigara grafika koja se čini apstraktnom može poprimiti realističnu dimenziju zahvaljujući primjeni ljudskih pravila unutar same neizrazite logike nad vizualnim elementima. Način na koji se dobiva prividan osjećaj realističnosti grafike u praktičnom primjeru ovoga rada uključuje množenje izlaza neizrazite logike sa nasumičnim brojem čime se dobiva prividno stvarna, nepredvidljiva grafika. Vrijednosti koje se mijenjaju brojem koji je dobiven uključuju učestalost prikazivanja čestica te brzinu instanciranja objekata samih efekata, a ponekad i njihovu veličinu.

# 2. Metode i tehnike rada

Istraživanje teme neizrazite logike se temelji na pretraživanju znanstvenih članaka i web stranica koje objašnjavaju koncepte koji su potrebni za pretvorbu ulaznih vrijednosti u izlazne na temelju neizrazitih pravila i skupova. Kako bi se prikazalo kako neizrazita logika djeluje, provedeno je izračunavanje različitih vrijednosti na temelju primjera sustava za hlađenje. Ova izračunavanja služe kao osnova za primjenu neizrazite logike u samom praktičnom dijelu rada koji predstavlja videoigra izrađena u programu Godot, alatu otvorenog koda koji je u usponu zbog svojeg jednostavnog načina korištenja te mogućnosti promjene koda.

Pri izradi efekata je korišten alat za uređivanje rasterskih grafika Piskel, a za ubacivanje zvučnih efekata u samu igru je korišten alat LMMS, aplikacija za miješanje zvukova te online alat za snimanje mikrofona. Programski kod unutar Godot pogona igre se temelji na Godot skriptnom jeziku GDScript, izvornom programskom jeziku za aplikaciju Godot.

# 3. Povijest tehnologije i razvoj umjetne inteligencije

Računalna grafika svoju svrhu ostvaruje iz želje za vizualnim prikazom radnji koje se događaju u računalu. Iako današnji pojam grafike često vodi prema razmišljanju o tome kako prikazati nešto na ravnom monitoru oni nisu bili prvi način prikazivanja sadržaja računala. Moglo bi se tvrditi da se prvi računalni grafički sustav pojavio s prvim digitalnim računalima. MIT-ovo računalo Whirlwind imalo je CRT grafičke zaslone u kontrolnoj sobi (Slika 1). [3].

Prvi pravi primjeri računalne grafike su uključivali velike, teške monitore koji su bili široki i temeljili se na staklenim katodnim cijevima. Iako su za današnje standarde glomazni, u razdoblju eksplozije informatičkih uređaja 1990-ih imali su značajnu ulogu u proširivanju računalne tehnologije. Ukupna svjetska jedinična potrošnja velikih računalnih zaslona svih vrsta i za sve primjene (isključujući potrošačku TV) bila je u porastu još od 1995. Na osnovi jedinične potrošnje, primjene u poslovanju i obrazovanju predstavljaju 69,7% udjela krajem 20. stoljeća. [4].

Razvoj računala je sa sobom vodio i razvoj umjetne inteligencije, a time i različitih područja koja spadaju pod njen pojam. Konferencijom u Darthmouth-u te nastajanjem VLSI sklopova koji su mogli izvoditi mnogo paralelnih operacija uz nisku potrošnju energije, umjetna inteligencija je dobivala sve veću pozornost, no zbog razvoja stolnih računala koja su upravljala tržištem nije mogla doći do svog potpunog izražaja. Ipak, različiti događaji su dovodili do ponovnog uzdizanja njene popularnosti. "Jedan od najstarijih velikih izazova u informatici je stvaranje šahovskog računala na razini Svjetskog prvenstva. Kombinirajući prilagođene VLSI sklopove, namjenske masovno paralelne $\alpha\beta$ tražilice i razne nove algoritme pretraživanja, Deep Blue je dizajniran da bude takvo računalo." [5]. Unatoč velikim naporima Garija Kasparova, tadašnjeg svjetskog prvaka u šahu, Deep Blue se pokazao kao bolji igrač šaha zahvaljujući mogućnostima iznimno detaljnog predviđanja.

Jedan od problema koji umjetna inteligencija predstavlja je njena potreba za velikom računalnom snagom, no kako je vrijeme prolazilo tako su se razvijale i računalne komponente te algoritmi umjetne inteligencije koji su počeli prikazivati prave, vidljive aplikacije za nju u obliku razumijevanju slika i govora, samovozećih automobila te dizajna potpomognutog umjetnom inteligencijom [6]. Jedno od polja s kojim je umjetna inteligencija zasigurno usko povezana je područje računalnih igara. Još od rođenja ideje o umjetnoj inteligenciji, računalne igre su potpomogle napretku istraživanja umjetne inteligencije [6]. Igre ne postavljaju samo zanimljive i složene probleme za rješavanje umjetnom inteligencijom - npr. dobro igranje igre; već nude i platno za kreativnost i izraz koji doživljavaju korisnici (ljudi ili čak strojevi!) [6]. Iz tog razloga igre su vjerojatno rijetka domena gdje se znanost (rješavanje problema) susreće s umjetnošću i interakcijom: ovi sastojci učinili su igre jedinstvenom i omiljenom domenom za proučavanje AI [6].

Glavna domena primjene AI u računalnim igrama je povijesno bila igranje igara, no ona uključuje i mogućnosti generiranja sadržaja te modeliranja ne-igrivih likova unutar same igre [6]. Iako je generiranje sadržaja postala jedna od glavnih tema nedavnim napretkom umjetne inteligencije, primarna domena modeliranja igrača te analiziranja toga kako bi umjetna inte-

ligencija mogla biti implementirana u strukturu računalne igre je povijesno dovela do velikih razvoja u toj domeni. Prema [6], uloga igranja umjetne inteligencije u računalnim igrama može biti podijeljena u četiri kategorije: s ciljem da pobijedi u ulozi igrača, s ciljem da pobijedi u ulozi ne-igrivog lika, s ciljem da pruži iskustvo u ulozi igrača te s ciljem da pruži iskustvo u ulozi ne-igrivog igrača.

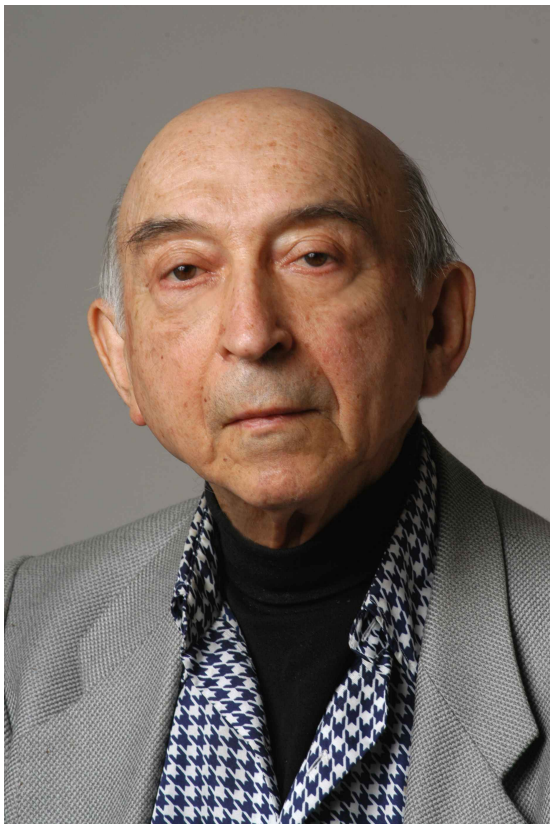
Ako je umjetnoj inteligenciji cilj da pobijedi u ulozi igrača, umjetna inteligencija predstavlja igrača u nekoj igri kojemu je cilj poraziti ostale igrače, tj. pobijediti [6]. U slučaju uloge s ciljem da pobijedi u ulozi ne-igrivog lika ona predstavlja likove koji pružaju izazov samom igraču poput automobila u trkaćim igrama [6]. Zanimljiva primjena umjetne inteligencije uključuje domenu gdje je cilj pružiti iskustvo u ulozi igrača te se njena najčešća primjena povezuje umjetnom inteligencijom koja predstavlja čovjekolikog agenta [6]. Ovakav agent se koristi kako bi dizajner računalne igre mogao prepoznati što igrač može raditi u njegovoj igri. Namještanjem agenta na brzine reakcije igrača, radnje koje bi igrač radio i slično dobivena je simulaciju igrača, a time i testni podaci na kojima se može temeljiti procjena iskustva igre. Posljednja od kategorija je uloga pružanja iskustva u ulozi ne-igrivog igrača koja je i među najčešćima u igrama. U računalnim igrama česti cilj umjetne inteligencije nije da pobijedi igrača, već da pruži iskustvo igraču s realističnim likovima unutar igre [6]. Ovakva umjetna inteligencija se koristi u računalnim igrama u različitim situacijama; likovi koji pružaju pomoć igraču, formiraju dio zagonetke ili pričaju određenu priču [6].

Algoritmi koji se koriste kako bi ostvarili ciljeve temelje se na različitim metodama: algoritmi stabla pretraživanja, klasično i duboko poticano učenje, učenje s nadzorom te himerični igrači [6]. Mnogi od tih algoritama, koji su uključeni u različite vrste igara poput strateških i trkaćih koriste neizrazitu logiku kako bi ostvarili svoje ciljeve [6]. Neizrazita logika predstavlja jedan od načina na koji programer može modelirati politiku ponašanja određenog modela umjetne inteligencije te stvoriti uvjete kroz koje ona djeluje [6]. Prema [7], neizrazita logika može biti korisna za AI igru u nekoliko aspekata. Između ostalih upotreba, može se koristiti za donošenje odluka NPC-a kao što je odabir predmeta ili oružja, za kontrolu kretanja jedinica slično onome što se događa s kontrolnim sustavima, omogućavajući AI protivniku da procijeniti prijetnje i klasifikaciju, na primjer rangiranjem igrača i NPC-ova u smislu zdravlja ili moći koristeći neizrazite varijable [7]. Zbog lingvističke prirode neizrazite logike, formuliranje njezinih pravila mogu napraviti stručnjaci u domeni polja, a neizraziti sustav se zatim može koristiti za emuliranje razmišljanja stručnjaka [7].

Garrido [8] objašnjava kako se neizrazita logika temeljila na razmišljanjima o tvrdnjama koje su djelomično istinite. Rješenje o problemima standardne logike vezane uz paradokse pripadnosti određene tvrdnje više različitih skupova koje ona stvara zbog toga što je svaka tvrdnja potpuno istinita ili lažna nastaje u obliku prijedloga Jana Lukasiewicza 1920-te [8] koji osmišljava trostruku logiku, gdje određena tvrdnja može biti istinita, lažna ili djelomično istinita. Djelomično istinita tvrdnja sadrži pripadnu vrijednost od 0,5.

Istraživanja su vodila različite znanstvenike do novih ideja, no za bitan doprinos začetku neizrazite logike zasigurno je bio zaslužan kvantni fizičar i njemački filozof Max Black koji je analizirao problem modeliranja neodređenosti [8]. Ideju koju je on htio predstaviti je korištenje

načina izrade profila ili modela kojim bi se mogla analizirati dvosmislenost značenja određene riječi ili simbola. Napredak u pokušavanju modeliranja neodređenosti Lukasiewicza i Blacka su doveli i do konačnog objavljivanja znanstvenog rada Loftija A. Zadeha, matematičara koji se smatra ocem neizrazite logike, u kojem je objašnjen koncept "Čupavog algoritma" [8].



Slika 1: Lofti A. Zadeh (Izvor: [9]

Zapad isprva nije prihvaćao njegove ideje, smatrajući da nisu postojale prave aplikacije čupavih algoritama. Ipak, različiti su inovativni znanstvenici iz Japana, Južne Koreje te Kine i Indije pokušali iskoristiti novoistraženu logiku u pravim sustavima. Zadehova zamisao je bila da kreira formalan način za prikazati nepreciznost ljudskog razmišljanja [8]. 1971. izdaje "Kvantitativnu neizrazitu semantiku" u kojoj objašnjava sve formalne elemente korištene i danas u aplikacijama neizrazite logike [8].

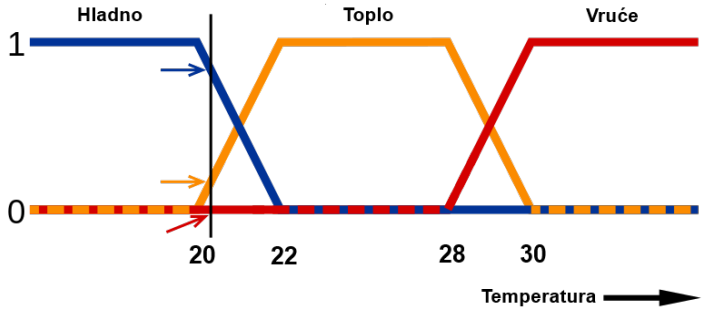
Kako bi se shvatio koncept neizrazite logike, potrebno je potpuno promijeniti mišljenje o tome što tvrdnje mogu biti. Istina i laž ne postaju činjenice već postotak točnosti koju određena tvrdnja ima [10]. Zahvaljujući neizrazitoj logici moguće je definirati pravila koja nemaju jednostavno objašnjenje u programskim sustavima, no bilo koja osoba bi mogla reći kako ta pravila djeluju.

# 4. Mehanizam neizrazite logike

Kako bi neizrazita logika mogla biti pravilno primijenjena, potrebno je shvatiti zašto se ona koristi te koji su koraci potrebni u njenoj upotrebi. Neizrazita logika predstavlja alternativu klasičnoj logici; kada ljudi žele protumačiti određenu tvrdnju, isto značenje može imati različitu vrijednost za različite ljude, dok klasična logika želi apsolutno pretpostaviti jednoznačan izlaz za određeni ulaz. U središtu razlike između klasične i neizrazite logike je nešto što je Aristotel nazvao pravilo isključene sredine [10]. Dok u klasičnoj logici određeni objekt može ili pripadati ili ne pripadati određenom skupu, npr. broj pet u potpunosti pripada neparnim brojevima dok u potpunosti ne pripada parnim brojevima, u neizrazitoj logici pravilo isključene sredine se djelomično krši [10]. Neizraziti skupovi stvaraju djelomične kontradikcije zato što izrazi mogu pripadati u više skupova u isto vrijeme, tj. ako imamo određenu temperaturu zraka, ona može biti 20 posto hladna, ali u isto vrijeme i 80 posto ne-hladna [10]. Ljudski mozak može obrazložiti neizrazite tvrdnje ili stavove koji uključuju nesigurnost ili prosudbu vrijednosti: "Zrak je hladan," ili "Ova brzina je velika," ili "Ona je mlada." [10]. Za razliku od računala, ljudi imaju zdrav razum koji im omogućuje protumačiti tvrdnje u svijetu gdje su stvari djelomično točne [10]. U već spomenutom primjeru sustava hlađenja, različite temperature mogu imati različito značenje za dva različita čovjeka. Neizrazita logika otvara mogućnost pretvaranja različitih stavova na jednoznačan način, ovisno o pravilima i postavkama neizrazite logike postavljene za taj sustav.

Svaki sustav neizrazite logike u sebi sadrži četiri komponente: ulaznu vrijednost, neizrazite skupove koji sa sobom sadrže i funkcije pripadnosti, neizrazita pravila koja definiraju značenje vrijednosti pripadnosti funkcijama pripadnosti te izlaznu vrijednost koja se računa na temelju neizrazitih pravila [11].

Način na koji se može iskoristiti neizrazita logika je objašnjen na primjeru sustava hlađenja. Ulaz koji je u takav sustav unešen predstavlja trenutna temperatura sobe, a funkcijama pripadnosti se računa koliko određena temperatura pripada određenom neizrazitom skupu.




Slika 2: Prikaz neizrazite logike u sustavu klima uređaja (Prema [12])

Na slici je prikazan dijagram pripadnosti vrijednosti temperature različitim neizrazitim skupovima. Neizraziti skupovi predstavljaju ugođaj koji korisnik može imati, u ovom slučaju "Hladno", "Toplo" i "Vruće". Vrijednost pripadnosti svakom skupu se prikazuje decimalnim brojem od 0 do 1, a računa na temelju funkcija pripadnosti. Funkcije pripadnosti predstavljaju mehanizam koji pretvara ulaznu vrijednost u vrijednost pripadnosti svakom od neizrazitih sku-

pova. [13] Ovisno o postavkama samoga korisnika, temperature hladno, toplo i vruće mogu predstavljati različite temperature za svakoga korisnika, a time i kakvu pripadnost imaju prema svakom neizrazitom skupu. Bitno je naglasiti da zbroj pripadnosti svim neizrazitim skupovima određene ulazne vrijednosti ne mora nužno biti jednaka 1, no vrijednosti pripadnosti se nalaze u rasponu od 0 do 1, gdje 0 znači potpunu nepripadnost skupu, a 1 znači potpunu pripadnost.

Tablica 1: Prikaz težinske vrijednosti neizrazitim skupovima na temelju temperature (Prema [12])

| Temperatura | Hladno | Toplo | Vruće |
|---|---|---|---|
| 20°C | 1.0 | 0.0 | 0.0 |
| 21°C | 0.5 | 0.5 | 0.0 |
| 22°C | 0.0 | 1.0 | 0.0 |
| 28°C | 0.0 | 1.0 | 0.0 |
| 29°C | 0.0 | 0.5 | 0.5 |
| 30°C | 0.0 | 0.0 | 1.0 |

U tablici je prikazano koliku pripadnost na temelju različitih temperatura ulazna vrijednost ima neizrazitim skupovima. Ako se temperatura nalazi na prijelazu vrijednosti, ona djelomično pripada u više različitih skupova. U ovom slučaju temperatura 21°C ima vrijednost pripadnosti neizrazitom skupu "Hladno" u vrijednosti od 0.5, te pripadnosti "Toplo" u vrijednosti od 0.5. Ta pripadnost skupu "Hladno" smanjuje se kako se temperatura povećava, dok se pripadnost skupu "Toplo" povećava. Pri većim temperaturama, pripadnost skupu "Toplo" smanjuje se dok se skupu "Vruće" povećava. Funkcije pripadnosti koje pretvaraju vrijednost temperature u vrijednost pripadnosti se mogu opisati na sljedeći način:

**Funkcija pripadnosti za "Hladno":**

$$\mu_{\text{Hladno}}(x) = \begin{cases} 1 & : x \leq 20 \\ \frac{22-x}{2} & : 20 < x \leq 22 \\ 0 & : x > 22 \end{cases}$$

**Funkcija pripadnosti za "Toplo":**

$$\mu_{\text{Toplo}}(x) = \begin{cases} 0 & : x \leq 20 \\ \frac{x-20}{2} & : 20 < x \leq 22 \\ 1 & : 22 < x \leq 28 \\ \frac{30-x}{2} & : 28 < x \leq 30 \\ 0 & : x > 30 \end{cases}$$

**Funkcija pripadnosti za "Vruće":**

$$\mu_{\text{Vruće}}(x) = \begin{cases} 0 & : x < 28 \\ \frac{x-28}{2} & : 28 \leq x \leq 30 \\ 1 & : x > 30 \end{cases}$$

Pripadnost svakom neizrazitom skupu ipak nije dovoljna kako bi se ulazne vrijednosti mogle pretvoriti u izlazne vrijednosti koje se mogu ponovno iskoristiti. Neizrazita pravila nemaju algoritam kojim se može definirati kakav bi izlaz dala svaka od pripadnosti upravo zato što je neizrazita logika ovisna o tome kako ju korisnik definira. Ipak, kako bi se jednostavno definirala pravila, korisnik može definirati izlazne vrijednosti u slučaju kada je vrijednost pripadnosti maksimalno istinita za određeni neizraziti skup. U ovom slučaju je pretpostavljeno sljedeće, pod uvjetom da izlazna vrijednost koju želimo označava brzinu okretanja ventilatora unutar klimatskog uređaja:

- Ako je pripadnost "Hladno" tada je brzina ventilatora 30%
- Ako je pripadnost "Toplo" tada je brzina ventilatora 55%
- Ako je pripadnost "Vruće" tada je brzina ventilatora 80%

Ovako definirana neizrazita pravila tada mogu pretvoriti ulaznu vrijednost temperature sobe u realne vrijednosti brzine okretanja ventilatora čime dobivamo konkretne naredbe za sami uređaj. Izračunavanje izlazne vrijednosti se temelji na sljedećoj formuli:

$$\text{Izlazna vrijednost} = \frac{\sum_{i=1}^{n} x_i \cdot \mu(x_i)}{\sum_{i=1}^{n} \mu(x_i)}$$

Gdje se svaka od vrijednosti pripadnosti množi sa izlaznom vrijednošću kada je ta težina maksimalno istinita podijeljeno sa zbrojem svih vrijednosti pripadnosti. U slučaju kada je temperatura 21°C, vrijednost pripadnosti skupu "Toplo" iznosi 0.5, a vrijednost pripadnosti skupu "Hladno" također iznosi 0.5 pa je dobiveni rezultat:

$$\frac{(30\% * 0.5) + (55\% * 0.5)}{0.5 + 0.5} = 42.5\%$$

Kada je vrijednost pripadnosti u potpunosti u određenom skupu, tada će ona iznositi onu vrijednost koja je definirana u pravilima. U slučaju pripadnosti skupu "Hladno", maksimalno istinita vrijednost na temelju spomenutih neizrazitih pravila iznosi 30% te se ona množila sa 0.5, svojom vrijednošću pripadnosti tom skupu. Ako je pripadnost 1 skupu "Vruće":

$$\frac{80\% * 1}{1} = 80\%$$

Temperatura se mijenja temeljem modela klimatskog uređaja koji je već prikazan te se zahvaljujući primjeni neizrazitih skupova i neizrazitih pravila ulazna vrijednost pretvorila u izlaznu na temelju ljudske logike, a ne računalne. Iako je ovo jednostavan model koji uključuje

samo tri neizrazita skupa, određeni modeli mogu sadržati i više skupova. Time se stvaraju kompleksni sustavi koji imaju mnogo nepredvidljivih varijabli kako bi se na jednostavan način određena ulazna vrijednost pretvorila u izlaznu. Na temelju različitih temperatura dobivamo različite brzine ventilatora:

Tablica 2: Prikaz brzine ventilatora za različite temperature

| Temperatura | Brzina ventilatora |
|---|---|
| 20°C | 30% |
| 21°C | 42.5% |
| 22°C | 55% |
| 28°C | 55% |
| 29°C | 67,5% |
| 30°C | 80% |

# 5. Primjena neizrazite logike u računalnoj grafici na primjeru računalne igre

Računalna igra je praktični dio ovog završnoga rada koji predstavlja prikaz implementacije neizrazite logike na primjeru grafike računalne igre. U ovom poglavlju su objašnjene same funkcionalnosti videoigre, kako sama videoigra funkcionira te na koji način je neizrazita logika implementirana kako bi poboljšala vizualni dojam same videoigre.

## 5.1. Opis igre Flow Magweather

Igra "Flow Magweather" je 2D videoigra žanra shoot'em up. Igrač predstavlja farmera koji želi zaštititi svoje imanje od ljigavaca, neprijatelja koji ga žele poraziti. Igra traje beskonačno dugo, a cilj igrača je preživjeti što duže i poraziti što više ljigavaca. Kako bi igrač mogao poraziti svoje neprijatelje on posjeduje dvije vrste oružja: pištolj koji nanosi štetu ljigavcima te uređaj za kontrolu vremena koji omogućava igraču da upravlja vremenom oko sebe, stvarajući vizualne efekte na svom ekranu. Svaki od efekata ima različit utjecaj na različitog neprijatelja te igrač mora pametno manevrirati svojim mogućnostima kako bi uspio proći svaku razinu. Sama igra je kreirana u platformi Godot, aplikaciji otvorenog koda koja služi kreiranju videoigara na jednostavan i brz način.

## 5.2. Glavne funkcionalnosti igre

### 5.2.1. Početni zaslon

Početni zaslon videoigre je mjesto na kojem igrač započinje samu igru. Na njoj se nalazi početni prikaz mape na kojoj igrač može igrati sa gumbima za početak igre te izlaz iz igre. Osim što igrač može pritisnuti gumb za izlaz iz igre, također se može pritisnuti i tipka "Escape" na tipkovnici. Kada igra započne, igrač se stvara unutar svijeta te započinje sama igra. Prozor igre se može proširiti ili smanjiti, no funkcionalnost same igre ostaje ista.


Slika 3: Početni zaslon videoigre

Na zaslonu su prikazane i sve kontrole u samoj igri kako bi igrač znao što može raditi u njoj. Resursi igre preuzeti sa itch.io korišteni u kreiranju svijeta u kojem se igrač nalazi te tekstovima za UI su "Pixel Art Top Down - Basic by Cainos", "Basic Tileskup and Asskup Pack 32x32 Pixels by schawrnhild" te "Pixel Art GUI Elements by Mounir Tohami".

### 5.2.2. Glavna igra


Slika 4: Pokrenuta igra

Unutar same glavne igre nalazi se igrač u sredini koji upravlja pištoljem. U donjem desnom kutu nalazi se uređaj za kontrolu vremena. Kako bi igrač upravljao uređajem mora poraziti neprijatelje, čime se puni indikator energije. Energija se može dijeliti između četiri različite postavke vremena: Vatreno, Munjevito, Kišovito i Vjetrovito. Igrač mora pametno manipulirati energijom ovisno o situaciji u kojoj se nalazi. Na temelju uređaja u igri se događaju različiti vremenski efekti čije vrijednosti kontrolira neizrazita logika.

U gornjem lijevom kutu nalaze se indikatori za životne bodove, broj poraženih ljigavaca koji predstavlja bodove koje igrač ostvaruje, trenutni val ljigavaca koji dolazi na igrača te ukupan broj preostalih ljigavaca u određenom valu neprijatelja. Sa svakim novim valom povećava se broj neprijatelja, a time i opasnost koju oni predstavljaju za igrača. Igrač mora oprezno manevrirati kroz svijet kako ga neprijatelji ne bi okružili i porazili. Osim broja neprijatelja sa svakim valom se povećava i brzina stvaranja vremena neprijatelja čime se dobiva vrlo dinamično iskustvo igre.

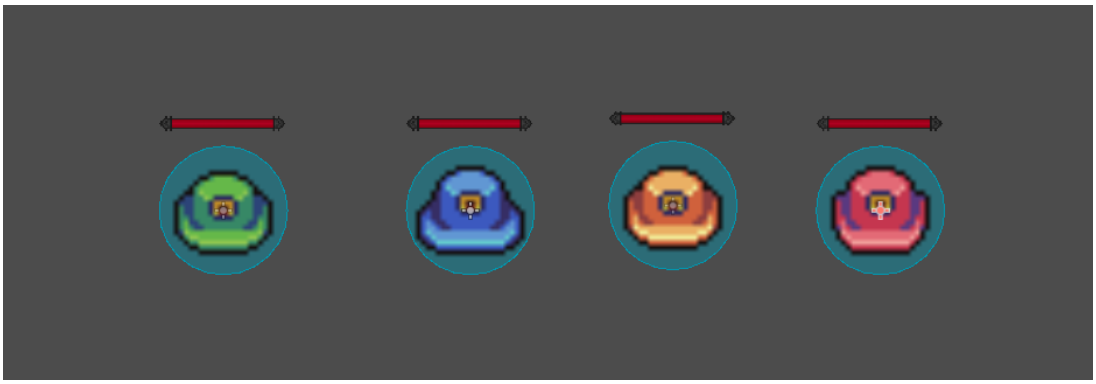
### 5.2.3. Način djelovanja Godot sučelja

Kod se temelji na GDScript skriptnom jeziku samog Godot pogona igre. Način na koji su elementi igre kreirani u Godot-u je sličan objektno orijentiranom programiranju. Elementi Godot sučelja se zovu scene koje same predstavljaju sučelje za kreiranje čvorova, sadržajnih elemenata u igri. Unutar jedne scene može se kreirati shema za objekte koji će biti instancirani u druge scene koje se sve povezuju na glavnu scenu. Glavna scena mora biti određena te ona predstavlja ulaznu točku u samu videoigru. Ako se naprave promjene na shemi jedne scene

ona utječe na instance objekata te scene čime je vrlo jednostavno modularno manipulirati različitim vrijednostima i parametrima unutar skripti samih elemenata.

Resursi igre korišteni za igrača te životne trake su "Tiny Farmer by TMPelay", "itty bitty gun pack by marximus" te "Pixel Health Bar Asskup Pack 2 by adwitr".

### 5.2.4. Mehanika ljigavaca



Slika 5: Prikaz ljigavaca

Ljigavci su glavni neprijatelji igrača. Postoje četiri vrste ljigavaca: Vodeni (plavi), Vatreni (crveni), Munjeviti (narančasti) te Vjetroviti (zeleni). Svaki od ovih ljigavaca ima različit način interakcije sa svijetom u igri, no svima je cilj poraziti igrača. Sljedeće je naveden kod samih ljigavaca na primjeru skripte Vodenog ljigavca:

```
extends Area2D

const DEFAULT_SPEED = 60
@export var health = 100
var damage_flag = 0
var can_attack = 1
var damage = 20
var speed = DEFAULT_SPEED
var player = null
var can_push = true
var dead = false

func lowerHealth(DAMAGE):
        if dead:
                return
        health -= DAMAGE
        $HealthBar.value -= DAMAGE
        if health <= 0:
                dead = true
                SignalBus.enemy_died.emit()
                queue_free()

func _ready():
        SignalBus.connect("player_spawned", _on_player_spawned)
        if get_tree().root.has_node("player"):
```

```
                player = get_tree().root.get_node("player")
        $AnimatedSprite2D.play("Walk")

# Called every frame. 'delta' is the elapsed time since the previous frame.
func _physics_process(delta):
        if is_instance_valid(player):
                var direction = (player.global_position - global_position).
                    normalized()
                global_position += direction * speed * delta
        if damage_flag == 1 and can_attack == 1:
                print("Damaging")
                $AttackTimer.start()
                can_attack = 0
                SignalBus.take_damage.emit(damage)

func _on_player_spawned(Player_From_Signal):
        player = Player_From_Signal

func _on_area_entered(area):
        if area is Bullet:
                lowerHealth(25)
        if area.is_in_group("Fire"):
                lowerHealth(50)
        if area.is_in_group("Rain"):
                speed = 100
                $SlowBackTimer.start()
        if area.is_in_group("Thunder"):
                speed = 0
                $StunBackTimer.start()
        if area.is_in_group("Tornado"):
                if can_push == true:
                        can_push = false
                        speed = -100
                        $PushBackTimer.start()
                        $CanPushTimer.start()

func _on_body_entered(body):
        if body is CharacterBody2D:
                print("Damage␣flag␣=␣1")
                damage_flag = 1


func _on_body_exited(body):
        if body is CharacterBody2D:
                print("Damage␣flag␣=␣0")
                damage_flag = 0


func _on_attack_timer_timeout():
        can_attack = 1


func _on_slow_back_timer_timeout():
```

```
        speed = DEFAULT_SPEED


func _on_stun_back_timer_timeout():
        speed = DEFAULT_SPEED


func _on_push_back_timer_timeout():
        speed = DEFAULT_SPEED


func _on_can_push_timer_timeout():
        can_push = true
```

Sam kod je kod svih ljigavaca približno jednak, no razlikuje se u interakciji sa vremenskim efektima u igri. Uvodna točka pokretanja same skripte je funkcija ready() koja predstavlja trenutak kada se objekt doda kao dijete u stvarnu scenu. Objekt je moguće instancirati, no tada ostaje zapamćen u memoriji i smatra se dijelom scene samo u njegovim atributima. To znači da se atributi samog objekta mogu mijenjati, no pokretanje skripte započinje tek kada se on doda u scenu. Metoda ready() postavlja sve varijable koje su joj potrebne kako bi objekt uspješno radio.

Funkcija physics_process() je ta koja dovodi do stvaranja kretanja samih neprijatelja i različitih dijelova funkcionalnosti koje ljigavci imaju. Ova funkcija provjerava gdje se igrač nalazi te ovisno o tome pomiče ljigavca prema njemu, a osim toga i upravlja štetom koju ljigavac nanosi igraču.

Ostale funkcije se pozivaju na temelju signala koje sam objekt ljigavca prima iz svog okoliša. Signali su posebna vrsta mehanizma u Godot pogonu igre. Pomoću signala, određeni čvor se obavještava da se određena radnja dogodila. U slučaju kada čvor naleti na čvor tipa Područje (*eng. Area*) provjerava se na koju vrstu područja je objekt naletio i izvršavaju se radnje na temelju toga. Područja predstavljaju različite vremenske efekte koji se nalaze u igri te utječu na samoga neprijatelja.

Resursi igre korišteni za ljigavce se nalaze u itch.io paketu "Aesthetic Enemies: Slimes (Free!) by Phantom Cooper".

### 5.2.5. Mehanika stvaranja neprijatelja

Kako bi ljigavci mogli napasti samoga igrača, potrebno je implementirati mehaniku kojom će se i stvarati. Način na koji je ona implementirana je putem valova neprijatelja koji se stvaraju u svim smjerovima oko igrača. Kod za valove stvaranja neprijatelja je sljedeći:

```
func get_viewport_bounds():
        var viewport_size : Vector2
        viewport_size = get_viewport().size

        var camera_pos = $Camera2D.global_position
        var top_left = camera_pos - (viewport_size / 2)
```

```
        return Rect2(top_left, viewport_size)

func get_random_spawn_position(viewport_bounds):
        var spawn_position = Vector2()
        var side = randi() % 4

        match side:
                0:
                        spawn_position.x = randf_range(viewport_bounds.position.x -
                            SPAWN_DISTANCE, viewport_bounds.position.x +
                            viewport_bounds.size.x + SPAWN_DISTANCE)
                        spawn_position.y = viewport_bounds.position.y - randf_range
                            (0, SPAWN_DISTANCE)
                1:
                        spawn_position.x = viewport_bounds.position.x +
                            viewport_bounds.size.x + randf_range(0, SPAWN_DISTANCE)
                        spawn_position.y = randf_range(viewport_bounds.position.y -
                            SPAWN_DISTANCE, viewport_bounds.position.y +
                            viewport_bounds.size.y + SPAWN_DISTANCE)
                2:
                        spawn_position.x = randf_range(viewport_bounds.position.x -
                            SPAWN_DISTANCE, viewport_bounds.position.x +
                            viewport_bounds.size.x + SPAWN_DISTANCE)
                        spawn_position.y = viewport_bounds.position.y +
                            viewport_bounds.size.y + randf_range(0, SPAWN_DISTANCE)
                3:
                        spawn_position.x = viewport_bounds.position.x - randf_range
                            (0, SPAWN_DISTANCE)
                        spawn_position.y = randf_range(viewport_bounds.position.y -
                            SPAWN_DISTANCE, viewport_bounds.position.y +
                            viewport_bounds.size.y + SPAWN_DISTANCE)

        return spawn_position

func _on_wave_ended():
        $NewWaveTimer.start()

func _on_enemy_died():
        Total_Enemies -= 1
        Kill_Count += 1
        if Kill_Counter != null and Enemies_Remaining_Counter != null:
                Kill_Counter.text = str(Kill_Count)
                Enemies_Remaining_Counter.text = str(Total_Enemies)
        if (Kill_Count % 40 == 0 && Kill_Count <= 400):
                SignalBus.fill_energy.emit()
        if Total_Enemies <= 0:
                Current_Wave += 1
                SignalBus.wave_ended.emit()

func Start_Wave(Wave_Number : int):

        print("Started␣wave␣", Wave_Number)
        if Wave_Counter != null:
```

```
                Wave_Counter.text = str(Wave_Number)
        var EnemyCounter : int
        EnemyCounter = roundi(20*Wave_Number**1.2)
        Total_Enemies = EnemyCounter
        if Enemies_Remaining_Counter != null:
                Enemies_Remaining_Counter.text = str(Total_Enemies)

        for n in EnemyCounter:
                await get_tree().create_timer(0.1**Wave_Number).timeout
                var Slime_Inst
                var pick_slime = randi() % 4
                match pick_slime:
                        0: Slime_Inst = Blue_Slime.instantiate()
                        1: Slime_Inst = Red_Slime.instantiate()
                        2: Slime_Inst = Yellow_Slime.instantiate()
                        3: Slime_Inst = Green_Slime.instantiate()
                Slime_Inst.global_position = get_random_spawn_position(
                    get_viewport_bounds())
                add_child(Slime_Inst)
                SignalBus.player_spawned.emit(Player_Instance)

func _on_new_wave_timer_timeout():
        Start_Wave(Current_Wave)
```

Način stvaranja ljigavaca djeluje na temelju samog prozora na monitoru igrača. Neprijatelji se stvaraju izvan prozora kako igrač ne bi mogao predvidjeti od kuda dolaze. Svaki val započinje računanjem broja neprijatelja koji se treba stvoriti. Broj neprijatelja se temelji na formuli:

$$BrojNeprijatelja = 20 * Wave\_Number^{1.2}$$

Gdje je Wave_Number trenutna razina vala. Brzinom kojom se neprijatelji stvaraju temelji se na formuli:

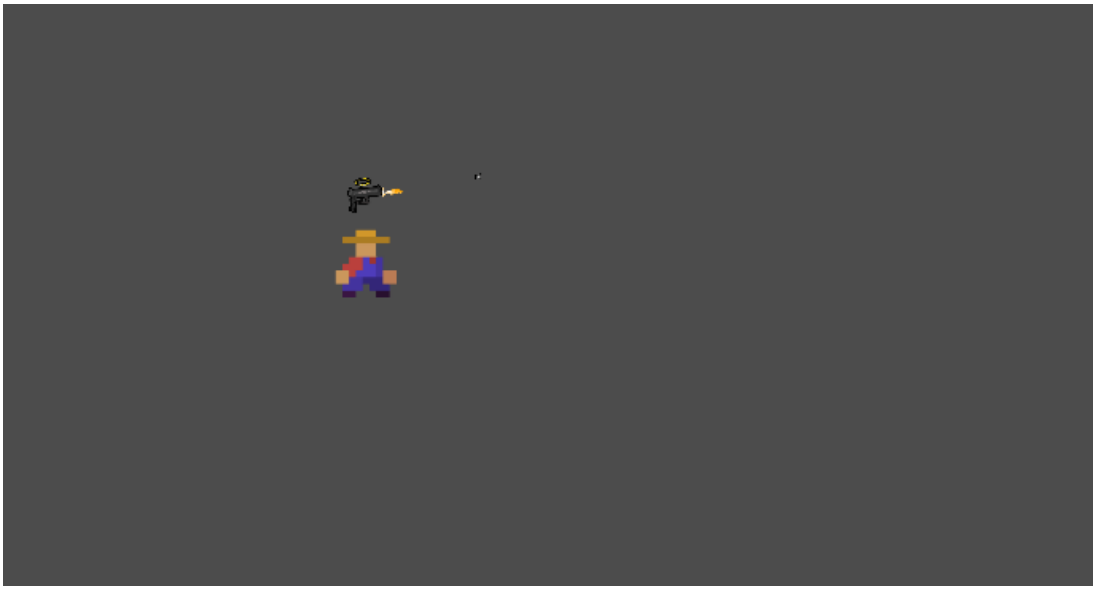
$$BrzinaStvaranja = 0.1^{Wave\_Number}$$

Svakim novim valom, brzina stvaranja neprijatelja se povećava. Pozicija za stvaranje neprijatelja izvan prozora se računa na temelju konstante SPAWN_DISTANCE koja stvara određeni razmak koji neprijatelji mogu biti odmaknuti od igrača, čime se dobivaju različite točke stvaranja neprijatelja, a time i dodatna nepredvidljivost uz već nasumične brojeve koji se koriste pri odabiru točke stvaranja.

### 5.2.6. Mehanika pištolja

Jedna od glavnih dijelova mehanizama same igre je funkcija pištolja kojeg ima igrač. Pištolj može ispucavati metke te se okreće u onom smjeru koji igrač pokazuje pokazivačem miša. Brzina ispucavanja metaka je u intervalima od jedne sekunde, a sami metak može proći kroz neograničeno neprijatelja te nestaje nakon pet sekundi kako se objekt samoga metka ne

bi mogao ograničeno stvarati.



Slika 6: Prikaz mehanike pucanja

Sljedeće je naveden prikaz koda pištolja kao i metka:

**Prikaz koda pištolja**

```
extends Node2D

var bullet_speed = 100
var can_shoot = true
@export var Bullet : PackedScene

# Called when the node enters the scene tree for the first time.
func _ready():
	pass # Replace with function body.

func makeEffect():
	if $GunBody/BulletShoot.is_playing():
		$GunBody/BulletShoot.stop()
		$GunBody/BulletShoot.play("explosion")
		$GunBody/ShootTimer.start()
		can_shoot = false
	else:
		$GunBody/BulletShoot.play("explosion")
		$GunBody/ShootTimer.start()
		can_shoot = false

func sendBullet():
	var Bullet_Instance = Bullet.instantiate()
	Bullet_Instance.global_position = $BulletShoot.global_position
	get_parent().get_parent().add_child(Bullet_Instance)

func Shoot():
	makeEffect()
	sendBullet()
```

```
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
        if Input.is_action_pressed("shoot") and can_shoot == true:
                Shoot()
        var mouse_pos = get_global_mouse_position()
        look_at(mouse_pos)


func _on_shoot_timer_timeout():
        can_shoot = true


func _on_bullet_shoot_animation_looped():
        $GunBody/BulletShoot.stop()
```

Pištolj djeluje na temelju process() funkcije koja djeluje slično kao physics_process() funkcija. Ova funkcija se poziva svaku sličicu igre koja se događa te se na pritisak akcije "Shoot", što predstavlja lijevi klik miša, izrvšava funkcija Shoot() koja izvršava stvaranje efekta pucanja iz pištolja te stvaranja instance objekta metka. Pištolj je moguće opaliti svaku sekundu.

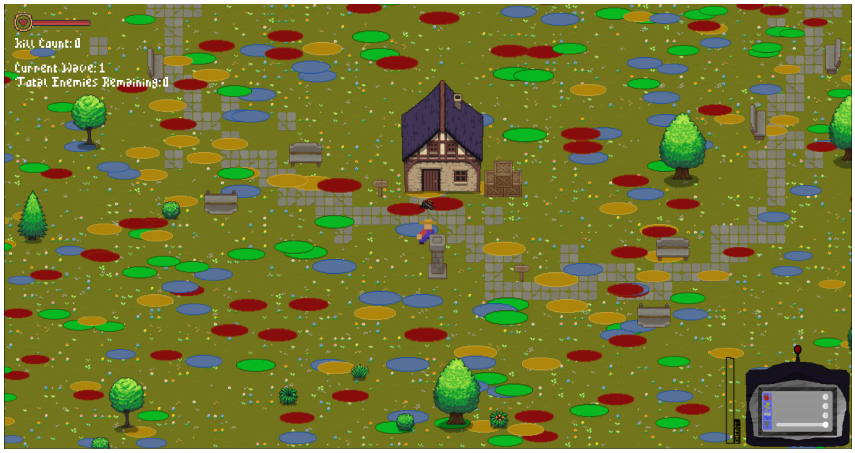
**Prikaz koda metka**

```
extends Area2D

class_name Bullet

const BULLET_SPEED = 400
var mouse_position
var direction : Vector2

# Called when the node enters the scene tree for the first time.
func _ready():
        mouse_position = get_global_mouse_position()
        direction = (mouse_position - position).normalized()
        rotation_degrees = rad_to_deg(position.angle_to_point(mouse_position))
        $DeleteTimer.start()

# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
        position += direction * BULLET_SPEED * delta


func _on_delete_timer_timeout():
        queue_free()
```

Sam metak je programiran kako bi putovao u onom smjeru gdje je pištolj okrenut, što označava pozicija miša. Nakon izračuna kuta okrenutosti prema mišu i njegove pozicije, metak se instancira te se kreće u jednom smjeru kroz process() funkciju. Također se i pokreće objekt vrste Timer koji nakon intervala od 5 sekundi pokreće funkciju queue_free() za brisanje samog objekta metka iz igre. Ova funkcionalnost je jedan od glavnih načina na koji se igrač može boriti protiv neprijatelja, no na raspolaganju ima i uređaj za stvaranje vremenskih efekata.

### 5.2.7. Mehanika stvaranja vremenskih efekata

Na slici je prikazana sama mehanika vremenskih efekata. Kako bi se jednostavnije mogao vidjeti način na koji su efekti rasprostranjeni kroz igru, postavljeni su u obliku jednostavnih krugova različitih boja: Vatreni krug (crvena), Vodeni krug (plava), Munjeviti krug (žuta) te Vjetroviti krug (zelena).




Slika 7: Prikaz mehanike stvaranja vremenskih efekata

Svaki od navedenih krugova drugačije utječu na same ljigavce, no svi se upravljaju kroz uređaj u donjem desnom kutu ekrana. Kod za uređaj je naveden ispod:

```
func update_slider_values(slider, value):
        Sum_value = Fire_value + Thunder_value + Rain_value + Tornado_value
        if Sum_value > energy:
                var max_value_for_slider = energy - (Sum_value - value)
                if value > max_value_for_slider:
                        slider.value = max_value_for_slider
        current_energy = energy - Sum_value
        $AbilityButtonLayer/Gadget/EnergyBar.value = current_energy

func _on_fire_value_changed(value):
        Fire_value = value
        update_slider_values($AbilityButtonLayer/Gadget/Fire, value)
        Fire_value = $AbilityButtonLayer/Gadget/Fire.value
        SignalBus.changed_weather.emit(0, Fire_value)

func _on_thunder_value_changed(value):
        Thunder_value = value
        update_slider_values($AbilityButtonLayer/Gadget/Thunder, value)
        Thunder_value = $AbilityButtonLayer/Gadget/Thunder.value
        SignalBus.changed_weather.emit(1, Thunder_value)

func _on_rain_value_changed(value):
        Rain_value = value
        update_slider_values($AbilityButtonLayer/Gadget/Rain, value)
        Rain_value = $AbilityButtonLayer/Gadget/Rain.value
        SignalBus.changed_weather.emit(2, Rain_value)
```

```
func _on_tornado_value_changed(value):
        Tornado_value = value
        update_slider_values($AbilityButtonLayer/Gadget/Tornado, value)
        Tornado_value = $AbilityButtonLayer/Gadget/Tornado.value
        SignalBus.changed_weather.emit(3, Tornado_value)
```

Postoje četiri funkcije s nastavkom _on_value_changed() koje se pozivaju svaki put kada igrač promijeni vrijednost klizača na samom uređaju. Sami klizači ne mogu izaći van obujma energije koju igrač trenutno ima. Time upravlja funkcija update_slider_values() na temelju trenutne vrijednosti određenog klizača koja je spremljena u samoj skripti igrača gdje se nalazi kod za uređaj.

Resursi igre korišteni za efekte stvaranja vremena se nalaze u paketima "Pixel Art - Lightning Strike by JekGG", "Particle FX by RagnaPixel" te "Whirlwind - Arne16" preuzet sa stranice OpenArtGame.org.

Navedene mehanike zajedno stvaraju igru koja ima dubinu u svojim mehanizmima. Ipak, dobra igra nije samo temeljena na svojim mehanizmima, već i grafičkom dojmu koji ona ima. Potrebno je implementirati mehanizam koji će dati dubinu samoj grafici. Iako postoje različiti mehanizmi koji to uspijevaju, ovdje je odabran onaj koji uključuje primjenu neizrazite logike.

## 5.3. Grafika igre na temelju neizrazite logike

Kako bi se uspješno implementirala neizrazita logika u određenu mehaniku grafike, potrebno je proći kroz iste korake koji su bili ključni za izračunavanje primjera s hlađenjem. Mehanika koja u sebi ima implementiranu neizrazitu logiku je mehanika stvaranja vremenskih efekata. Kako bi se moglo vidjeti zašto je bilo potrebno implementirati neizrazitu logiku potrebno je shvatiti cilj problema koji se predstavio: kako realistično prikazati vremenske efekte u slučaju njihove veličine.

### 5.3.1. Stvaranje i mijenjanje grafika na temelju postavki vremena

Metode kojima se dojam grafike može promijeniti u samoj igri uključuje stvaranje novih grafika i mijenjanje postojećih grafika na temelju određene vrijednosti. Na temelju četiri postavke vremena: "Vruće", "Munjevito", "Kišovito" i "Vjetrovito" određene grafike daju povratnu informaciju igraču o tome što se događa, a time i poboljšavaju ugođaj same igre. Izlazne vrijednosti na temelju neizrazite logike se u određenim slučajevima ne množe sa nasumičnim vrijednostima kako bi se grafika mijenjala konzistentno.

Glavni elementi koji su uključeni u stvaranju i mijenjanju grafika su Shader datoteke koje predstavljaju kod za manipuliranje određene teksture te funkcijski kod koji pomoću neizrazite logike mijenja vrijednosti parametara Shader datoteka. U određenim slučajevima nije potrebno koristiti Shader datoteke već je dovoljno promijeniti atribut određenog kontrolnog čvora, elementa UI sučelja u Godotu.

U svim slučajevima stvaranja i mijenjanja grafike ulazna vrijednost označava "value", trenutnu vrijednost koja se prosljeđuje funkciji _on_changed_weather() pri promjeni određene vrijednosti na uređaju za promjenu vremena unutar igre. Kod za navedenu funkciju je sljedeći:

```
@onready var lake = get_node("Map/WaterTexture")
@onready var fog = get_node("Fog/ParallaxLayer/ColorRect")

func _on_changed_weather(Weather, value):
        var shader_mat = lake.material as ShaderMaterial
        var fog_mat = fog.material as ShaderMaterial
        var rain_mat = $Overlay/RainOverlay.material as ShaderMaterial
        print(rain_mat.get_shader_parameter("rain_amount"))
        match Weather:
                0:
                        Fire_Intensity = value
                        var heatCalculation = heat_calculate(value)
                        $Overlay/HotOverlay.color = Color(155.0/255.0, 63.0/255.0,
                            33.0/255.0, heatCalculation/255.0)
                1:
                        Thunder_Intensity = value
                        var darkCalculation = dark_calculate(value)
                        $Overlay/DarkOverlay.color = Color(22.0/255.0, 22.0/255.0,
                            22.0/255.0, darkCalculation/255.0)
                2:
                        Rain_Intensity = value
                        var distortionCalculation = distortion_calculate(value)
                        shader_mat.skup_shader_parameter("distortion_strength",
                            distortionCalculation)

                        var rainTransparencyCalculation =
                            rain_transparency_calculate(value)
                        var rainAmountCalculation = rain_amount_calculate(value)

                        rain_mat.skup_shader_parameter("rain_color", Color
                            (108.0/255.0, 164.0/255.0, 246.0/255.0,
                            rainTransparencyCalculation/255.0))
                        rain_mat.skup_shader_parameter("rain_amount",
                            rainAmountCalculation)
                3:
                        Tornado_Intensity = value
                        var scrollCalculation = scroll_calculate(value)
                        var fogCalculation = fog_calculate(value)
                        shader_mat.skup_shader_parameter("scroll", Vector2(
                            scrollCalculation, scrollCalculation))
                        fog_mat.skup_shader_parameter("density", fogCalculation)
                        fog_mat.skup_shader_parameter("speed", Vector2(
                            fogCalculation, fogCalculation))

                        var rainSlantCalculation = rain_slant_calculate(value)
                        rain_mat.skup_shader_parameter("slant", rainSlantCalculation
                            )
```

Na temelju vrste postavke vremena koja se promijeni također se prosljeđuje parametar "Weather" koji određuje koja vrsta grafike će se mijenjati na temelju "value". Funkcije sa nastavkom _calculate() predstavljaju funkcije za izračun konačne vrijednosti na temelju neizrazite logike.

Svaka od funkcija za kalkulaciju vrijednosti ima svoja vlastita pravila, no ipak im je zajedničko korištenje funkcija pripadnosti kojima se izračunava pripadnost neizrazitim skupovima "LOW", "MEDIUM" i "HIGH" koji predstavljaju intenzitet vremena. Programski kod funkcija pripadnosti za vrijeme je sljedeći:

```
const LOW = 0.1
const UNDER_MEDIUM = 0.4
const HIGH_MEDIUM = 0.7
const HIGH = 1

func fuzzy_low_weather(value):
        if value <= LOW:
                return 1.0
        elif value > LOW and value <= UNDER_MEDIUM:
                return (UNDER_MEDIUM - value) / (UNDER_MEDIUM - LOW)
        else:
                return 0.0

func fuzzy_medium_weather(value):
        if value < LOW:
                return 0.0
        elif value >= LOW and value <= UNDER_MEDIUM:
                return (value - LOW) / (UNDER_MEDIUM - LOW)
        elif value > UNDER_MEDIUM and value < HIGH_MEDIUM:
                return 1.0
        elif value >= HIGH_MEDIUM and value <= HIGH:
                return (HIGH - value) / (HIGH - UNDER_MEDIUM)
        else:
                return 0.0

func fuzzy_high_weather(value):
        if value < HIGH_MEDIUM:
                return 0.0
        elif value >= HIGH_MEDIUM and value <= HIGH:
                return (value - HIGH_MEDIUM) / (HIGH - HIGH_MEDIUM)
        else:
                return 1.0
```

Funkcije fuzzy_low_weather(), fuzzy_medium_weather() i fuzzy_high_weather() predstavljaju funkcije za izračun vrijednosti pripadnosti skupovima LOW, MEDIUM i HIGH. Konstante LOW, UNDER_MEDIUM, HIGH_MEDIUM i HIGH predstavljaju točke prelamanja svakih od funkcija neizrazitih skupova. Kako se vrijednost pripadnosti jedne funkcije smanjuje tako se vrijednost pripadnosti druge funkcije povećava. Na temelju vrijednosti "Value" koja se prosljeđuje tim funkcijama navedena je sljedeća tablica prikaza vrijednosti pripadnosti svakoj od neizrazitih skupova:

Tablica 3: Prikaz težinske vrijednosti neizrazitim skupovima na temelju "Value"

| Value | LOW | MEDIUM | HIGH |
|---|---|---|---|
| 0.1 | 1.0 | 0.0 | 0.0 |
| 0.25 | 0.5 | 0.5 | 0.0 |
| 0.4 | 0.0 | 1.0 | 0.0 |
| 0.7 | 0.0 | 1.0 | 0.0 |
| 0.85 | 0.0 | 0.5 | 0.5 |
| 1 | 0.0 | 0.0 | 1.0 |

(Izrađena na temelju izračuna vrijednosti)

Svaka od vrijednosti pripadnosti utječe na povratnu vrijednost koju će vratiti funkcija kalkulacije određene grafike za izračun konačne, izlazne vrijednosti. Kodovi za izračun svih izlaznih vrijednosti su sljedeći:

```
func rain_slant_calculate(value):
        var rain_slant_multiplier = fuzzy_low_weather(value) * 0 +
            fuzzy_medium_weather(value) * -0.5 + fuzzy_high_weather(value) * -1
        return rain_slant_multiplier

func rain_transparency_calculate(value):
        var rain_transparency_multiplier = fuzzy_low_weather(value) * 0 +
            fuzzy_medium_weather(value) * 100 + fuzzy_high_weather(value) * 255
        return rain_transparency_multiplier

func rain_amount_calculate(value):
        var rain_amount_multiplier = fuzzy_low_weather(value) * 100 +
            fuzzy_medium_weather(value) * 200 + fuzzy_high_weather(value) * 500
        return rain_amount_multiplier

func dark_calculate(value):
        var dark_multiplier = fuzzy_low_weather(value) * 1 + fuzzy_medium_weather(
            value) * 75 + fuzzy_high_weather(value) * 150
        return dark_multiplier

func heat_calculate(value):
        var heat_multiplier = fuzzy_low_weather(value) * 1 + fuzzy_medium_weather(
            value) * 50 + fuzzy_high_weather(value) * 75
        return heat_multiplier

func distortion_calculate(value):
        var distortion_multiplier = fuzzy_low_weather(value) * 0.1 +
            fuzzy_medium_weather(value) * 0.4 + fuzzy_high_weather(value) * 1
        return distortion_multiplier + randf_range(0, 0.1)

func scroll_calculate(value):
        var scroll_multiplier = fuzzy_low_weather(value) * 0.05 +
            fuzzy_medium_weather(value) * 1 + fuzzy_high_weather(value) * 3
        return scroll_multiplier + randf_range(0, 0.3)

func fog_calculate(value):
```

```
var fog_multiplier = fuzzy_low_weather(value) * 0 + fuzzy_medium_weather(
    value) * 0.4 + fuzzy_high_weather(value) * 1
return fog_multiplier
```

Vrijednost value koja se prosljeđuje svakoj od funkcija kalkulacija se koristi za izračun vrijednosti pripadnosti svakoj od funkcija pripadnosti. Nakon toga, na temelju grafike za koju se računaju određene vrijednosti, vrijednosti pripadnosti se množe sa pravom, brojčanom vrijednosti koja će predstavljati izlaznu vrijednost. Prave, brojčane vrijednosti su direktna primjena neizrazitih pravila te su određene za slučajeve kada bi vrijednost pripadnosti svake funkcije pripadnosti bila potpuna istina, odnosno 1.0.

Stvaranje grafike uključuje primjenu četiri različite promjene parametara Shadera ili atributa različitih kontrolnih čvorova na temelju trenutne vrijednosti vremena u uređaju za upravljanje vremenom. Prva vrsta promjene parametara se temelji na postavki "Vruće". Povećavanjem postavke se stvara crvenkasti sjaj u samom ekranu koji označava da se povećava temperatura te time stvara više efekata vatre.


Slika 8: Promjena boje ekrana na temelju postavke "Vruće"

Druga vrsta promjene parametara se temelji na postavki "Munjevito". Kada je vrijeme munjevito u stvarnome svijetu, stvaraju se tamni oblaci te je iz tog razloga odabran crni kvadrat koji mijenja prozirnost kako se povećava postavka.


Slika 9: Promjena boje ekrana na temelju postavke "Munjevito"

Treća vrsta promjene parametara se temelji na postavki "Kišovito". Kako bi se simulirala kiša iskorištena je posebna vrsta Shader teksture na temelju paketa "Rain and Snow Shader with Parallax Effect for Godot by Steampunkdemon" preuzet sa itch.io stranice. Kod za ovaj Shader je sljedeći:

```
// Rain and Snow shader by Brian Smith (steampunkdemon.itch.io)
// MIT licence

shader_type canvas_item;

uniform float rain_amount = 200.0;
uniform float near_rain_length : hint_range(0.01, 1.0) = 0.2;
uniform float far_rain_length : hint_range(0.01, 1.0) = 0.1;
uniform float near_rain_width : hint_range(0.1, 1.0) = 1.0;
uniform float far_rain_width : hint_range(0.1, 1.0) = 0.5;
uniform float near_rain_transparency : hint_range(0.1, 1.0) = 1.0;
uniform float far_rain_transparency : hint_range(0.1, 1.0) = 0.5;
// Replace the below reference to source_color with hint_color if you are using a
    version of Godot before 4.0.
uniform vec4 rain_color : source_color = vec4(0.8, 0.8, 0.8, 1.0);
uniform float base_rain_speed : hint_range(0.1, 1.0) = 0.5;
uniform float additional_rain_speed : hint_range(0.1, 1.0) = 0.5;
uniform float slant : hint_range(-1.0, 1.0) = 0.2;

void fragment() {
// To control the rainfall from your program comment out the below line and add a
    new uniform above as:
// uniform float time = 10000.0;
// Then update the time uniform from your _physics_process function by adding delta.
     You can then pause the rainfall by not changing the time uniform.
        float time = 10000.0 + TIME;

// Uncomment the following line if you are applying the shader to a TextureRect and
    using a version of Godot before 4.0.
//      COLOR = texture(TEXTURE,UV);

        vec2 uv = vec2(0.0);
        float remainder = mod(UV.x - UV.y * slant, 1.0 / rain_amount);
        uv.x = (UV.x - UV.y * slant) - remainder;
        float rn = fract(sin(uv.x * rain_amount));
        uv.y = fract((UV.y + rn));

// Blurred trail. Works well for rain:
//      COLOR = mix(COLOR, rain_color, smoothstep(1.0 - (far_rain_length + (
    near_rain_length - far_rain_length) * rn), 1.0, fract(uv.y - time * (
    base_rain_speed + additional_rain_speed * rn))) * (far_rain_transparency + (
    near_rain_transparency - far_rain_transparency) * rn) * step(remainder *
    rain_amount, far_rain_width + (near_rain_width - far_rain_width) * rn));

// No trail. Works well for snow:
        COLOR = mix(COLOR, rain_color, step(1.0 - (far_rain_length + (
            near_rain_length - far_rain_length) * rn), fract(uv.y - time * (
            base_rain_speed + additional_rain_speed * rn))) * (far_rain_transparency
```

```
            + (near_rain_transparency - far_rain_transparency) * rn) * step(
            remainder * rain_amount, far_rain_width + (near_rain_width -
            far_rain_width) * rn));
}
```

Različiti parametri kiše se zahvaljujući Shader datoteci mogu mijenjati dinamično u samoj igri. U slučaju postavke "Kišovito" sa većom vrijednosti stvara se više čestica kiše te se time dobije dojam kišovitog vremena. Ipak, kako kiša ne bi bila konstantno vidljiva također se mijenja i prozirnost boje samih čestica od potpuno nevidljive pri niskim vrijednostima do većih vrijednosti pri rastu vrijednosti postavke.

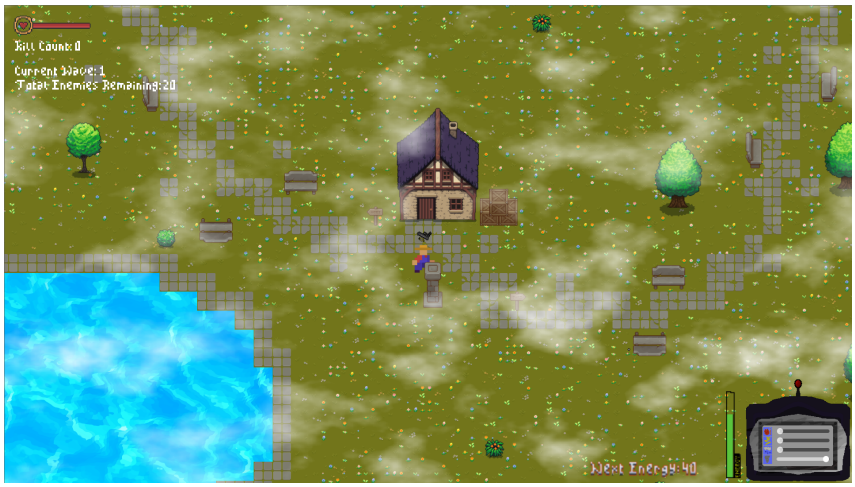

Slika 10: Promjena efekata ekrana na temelju postavke "Kišovito"

Posljednja vrsta promjene parametra se temelji na postavki "Vjetrovito". Vrsta efekta koji nastaje je gustoća magle i vjetra koji se kreće po ekranu. Kako bi se simulirao vjetar korištej je jednostavni shader za simulaciju magle koja se kreće postavljena u Parallax teksturi kako bi se ponavljaja kako se igrač kreće kroz svijet. Kod za shader magle je sljedeći:

```
shader_type canvas_item;
//render_mode unshaded; // optional

// Noise texture
uniform sampler2D noise_texture: repeat_enable, filter_nearest;
// Fog density
uniform float density: hint_range(0.0, 1.0) = 0;
// Fog speed
uniform vec2 speed = vec2(0.0, 0.0);


// Called for every pixel the material is visible on
void fragment() {
        // Make the fog slowly move
        vec2 uv = UV + speed * TIME;
        // Sample the noise texture
        float noise = texture(noise_texture, uv).r;
        // Convert the noise from the (0.0, 1.0) range to the (-1.0, 1.0) range
        // and clamp it between 0.0 and 1.0 again
        float fog = clamp(noise * 2.0 - 1.0, 0.0, 1.0);
```

```
        // Apply the fog effect
        COLOR.a *= fog * density;
}
```

Sa povećanjem postavke "Vjetrovito" stvara se brži, gušći izgled magle kroz cijeli ekran. Osim efekta magle, efekt kiše se također mijenja te sa većim vrijednostima kiša dobiva kut padanja čime dobivamo istinski realističan dojam vremena.

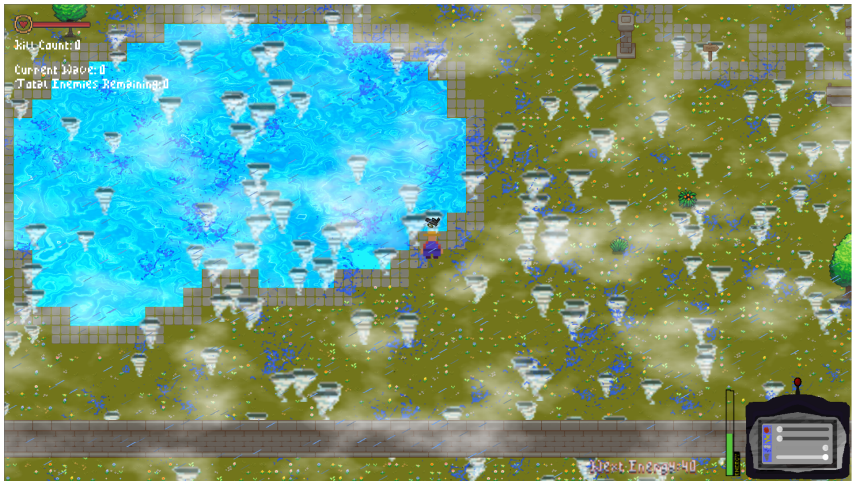



Slika 11: Promjena efekata ekrana na temelju postavke "Vjetrovito"

Osim stvaranja postojeće grafike, element mijenjanja grafike primijenjen u igri se temelji na prikazu jezera u donjem lijevom kutu početne pozicije stvaranja igrača. Kako ne bi došlo do trganja Shader tekstura povezanih uz stvaranje grafike, konačna vrijednost koja se dobiva na temelju neizrazitih pravila za takve funkcije nema množenje sa nasumičnim vrijednostima. Prvo množenje nasumičnih vrijednosti koje daje nepredvidljivost prikaza vode te time i realističnost grafike povezana je uz simulaciju grafike samoga jezera. Jezero se temelji na jednostavnoj visoko pikseliziranoj teksturi vode nad kojom je primijenjen Shader distorcije. Kod za shader jezera je sljedeći:

```
shader_type canvas_item;

// Noise texture to create wave distortion
uniform sampler2D noise : hint_default_white, repeat_enable;
// Speed and direction of wave movement
uniform vec2 scroll = vec2(0.05, 0.05);
// Strength of the distortion effect
uniform float distortion_strength : hint_range(0.0, 1.0) = 0.1;

void fragment() {
    // Sample the noise texture with scrolling offskup
    vec2 noise_uv = UV + scroll * TIME;
    vec4 noise_col = texture(noise, noise_uv);

    // Apply distortion to the UV coordinates
    vec2 distorted_uv = UV + distortion_strength * (noise_col.rg - 0.5);

    // Sample the base texture using the distorted UV coordinates
```

```
    vec4 base_col = texture(TEXTURE, distorted_uv);

    // skup the final color
    COLOR = base_col;
}
```




Slika 12: Promjena teksture jezera na temelju maksimalnih postavki "Kišovito" i "Vjetrovito"

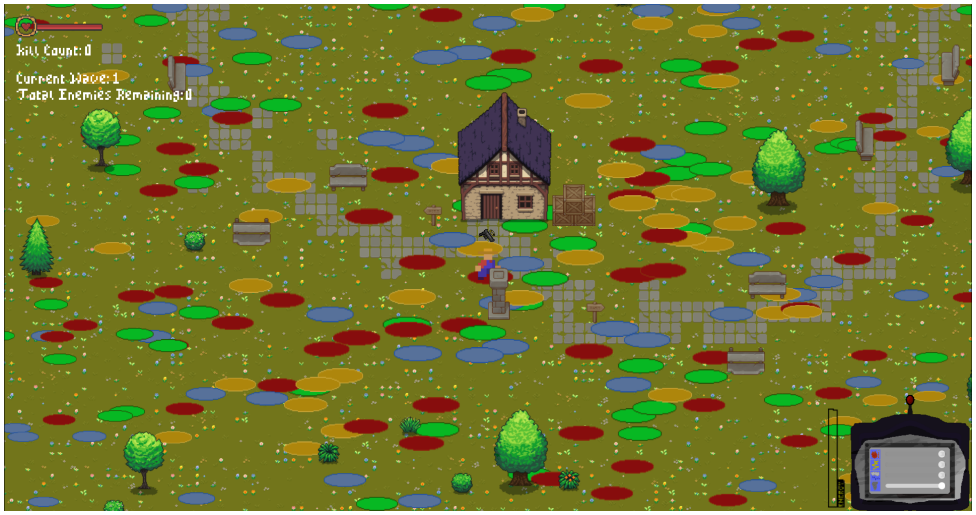
Promjena grafike jezera se temelji na parametrima scroll i distortion_strength. Scroll ubrzava vektore kretanja teksture jezera čime se dobiva efekt kretanja valova u samom jezeru, pogodno za postavku "Vjetrovito", dok distortion_strength predstavlja efekt distorcije same teksture čime se dobiva efekt gustoće vode pogodan za postavku "Kišovito". Na temelju jačih postavki povećava se distorcija i brzina kretanja teksture jezera.

### 5.3.2. Prikaz vremenskih efekata pomoću nasumičnosti i neizrazite logike


Slika 13: Stvaranje vremenskih efekata pomoću nasumičnosti

Korištenjem nasumičnosti, objekti vremenskih efekata također dobivaju potpuno nepredvidiv oblik koji izaziva zanimljiv grafički dojam, no također i vrlo nepraktičan. Igra postaje

zanimljiva, no teška zbog svoje nasumičnosti oblika te nema nikakve strategije koju sam igrač može primijeniti od same mehanike. Grafički izgled postaje maksimalno koristan, no igra postaje iznimno nepredvidljiva. Kako bi se poboljšao ovakav oblik prikazivanja efekata, potrebno je ubaciti uniformna pravila koja će definirati lako predvidljive mehanike igranja, poput smanjivanja veličine kruga na temelju udaljenosti od igrača.




Slika 14: Stvaranje vremenskih efekata pomoću neizrazite logike

Korištenjem neizrazite logike, dobivamo definiran oblik efekata koji možemo mijenjati na temelju neizrazitih pravila i neizrazitih skupova. Dobivamo vrlo praktičan izgled efekata: najveću površinu efekata dobivamo blizu igrača, dok se površina smanjuje što smo dalje od njega. Igrač može imati različite strategije zato što se pravila definiraju na temelju postavki vremena koje je programer odredio, tj. čovjek.

Kako bi se neizrazita logika uspješno implementirala u samu igru, potrebno je definirati sve komponente koje ona uključuje. Sljedeće je naveden kod za neizrazitu logiku:

```
const NEAR_DISTANCE = 200
const FAR_DISTANCE = 1200

func fuzzy_near(distance: float) -> float: #Vrati 1.0 ako je blizu
        if distance <= NEAR_DISTANCE:
                return 1.0
        elif distance <= FAR_DISTANCE:
                return (FAR_DISTANCE - distance) / (FAR_DISTANCE - NEAR_DISTANCE)
        else:
                return 0.0

func fuzzy_far(distance: float) -> float: # Vrati 1.0 ako je daleko
        if distance <= FAR_DISTANCE:
                return (distance - NEAR_DISTANCE) / (FAR_DISTANCE - NEAR_DISTANCE)
        else:
                return 1.0

func get_fuzzy_size(effect_position: Vector2, player_position: Vector2) -> float:
        var distance_to_player = player_position.distance_to(effect_position)
        var size_multiplier = fuzzy_near(distance_to_player) * 1.5 + fuzzy_far(
```

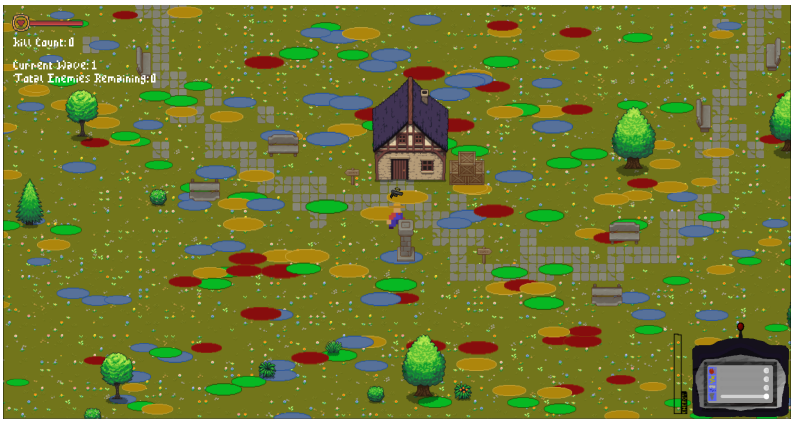
```
        distance_to_player) * 0.5 # Vrijednosti se mnoze
    return size_multiplier  + randf_range(-0.2, 0.2)
```

Ulazna vrijednost u sustavu neizrazite logike predstavlja udaljenost efekta koji će se stvoriti u svijetu od igrača spremljen u varijablu distance_to_player. Sustav ima dva neizrazita skupa kojima ta pozicija može pripadati: je li određena pozicija blizu igraču i je li određena pozicija daleko od igrača. Te pozicije su definirane kao 200 piksela za blizu te 1200 piksela za daleko. Funkcije pripadnosti predstavljaju funkcije fuzzy_near() i fuzzy_far(). Svaka od tih funkcija vraća vrijednost pripadnosti svom skupu te raste i pada ovisno o udaljenosti. fuzzy_near() vraća vrijednost pripadnosti 1.0 ako je efekt koji će se stvoriti 200 piksela ili bliže igraču, a fuzzy_far() vraća vrijednost pripadnosti 1.0 ako je efekt koji će se stvoriti 1200 piksela ili dalje od igrača te se te vrijednosti mogu mijenjati unutar samoga koda. Izlazna vrijednost koja je dobivena je promjena atributa "Scale", tj. koliko puta će objekt biti veći ili manji nego što je trebao biti. Kako bi se konačna vrijednost izračunala, potrebno je definirati neizrazita pravila koja definiraju svaku težinsku vrijednost kao određenu izlaznu. Neizrazita pravila koja vrijede za ovaj sustav su:

- Ako je pripadnost "Blizu" tada je veličina objekta 1.5 puta veća.
- Ako je pripadnost "Daleko" tada je veličina objekta 0.5 puta manja.

Unutar koda to znači da se povratne vrijednosti fuzzy_near() funkcije i fuzzy_far() funkcije množe sa svojim izlaznim veličinama definiranima u neizrazitim pravilima. Prema formuli, potrebno je i podijeliti cijeli izračun sa zbrojem težinskih vrijednosti, no kako je zbroj tih vrijednosti za ovaj primjer uvijek jednak 1 dijeljenje se može zanemariti.

Problem nastaje u tome što savršeno koncentrične kružnice koje nastaju oko igrača te se smanjuju prema rubovima ne čine igru zanimljivom, već vrlo predvidljivom i lakom za vješte igrače. Najbolji efekt se dobiva spajanjem dojma nasumičnosti i njene nepredvidljivosti sa ljudskim pravilima neizrazite logike.




Slika 15: Stvaranje vremenskih efekata pomoću neizrazite logike spojene sa nasumičnošću

Iako je efekt koncentričnih kružnica djelomično vidljiv, stvara se dojam nepredvidljivosti koji čine nasumični brojevi. Igra je dovoljno zanimljiva da nije jednoznačno određena, no ne toliko da igrač ne može imati strategiju, planove kako dobiti što više bodova i napredovati u igri.

Sama pravila programer može u bilo kojem trenutku promijeniti ako želi dodatno razgranati način na koji neizrazita logika djeluje te se time dobiva dubina ovisno o tome što programer želi. Ako bi programer želio da sustav ovisi ne samo o poziciji efekta od igrača, već i poziciji efekta od neprijatelja kako bi ga zatvorio u krug samih efekata, potrebno je dodati dodatne funkcije pripadnosti koje definiraju poziciju samoga efekta na temelju nasumično odabrane pozicije. Ipak, pretvorbu iz pozicije u poziciju bi bilo moguće ostvariti i kroz različite vektorske formule za izračunavanje pozicije koje su implementirane i u samu aplikaciju Godot.


Slika 16: Prikaz konačnih vremenskih efekata sa detaljima

Dodavanjem detalja u samu igru za efekte promjene vremena može se dočarati kako neizrazita logika stvarno može doprinjeti realističnosti igre ako se koristi na ispravan način. Nasumičnost i neizrazita logika zajedno stvaraju zanimljivu, no korisnu interakciju igrača sa samom igrom. Vješti igrači će imati konzistentno, no uvijek novo iskustvo, a novi igrači mogu biti impresionirani intuitivnošću implementacije.

# 6. Zaključak

Umjetna inteligencija je postala jedna od ključnih komponenti u izradi različitih dijelova videoigara. Alati koji omogućuju kreiranje sadržaja osobama koje nemaju iskustva s umjetnošću daje priliku stvarati iznimno realistične i zanimljive videoigre.

Neizrazita logika pruža jedan od ključnih komponenti umjetne inteligencije te se može koristiti u različitim domenama računalnih znanosti. Ona se koristi za prikaz nejasnih, dvosmislenih informacija, koje imaju određenu razinu istinitosti u više različitih skupina tvrdnji te ga računalo može teško shvatiti, ali ga čovjek može lako objasniti.

Način na koji je grafika promijenjena u primjeru ove igre je samo jedan od mnogih načina na koji neizrazita logika može utjecati na iskustvo igrača. Neizrazita pravila mogu biti različita: određena slika može biti povećana ili smanjena na temelju udaljenosti od igrača, drugog objekta u igri ili samog prikaza prozora igre na monitoru. Grafike se mogu smanjivati u kvaliteti na temelju udaljenosti kako bi se sačuvao memorijski prostor, a i objekti mogu imati utjecaj jedan na drugog u samoj mehanici igre kroz pravila neizrazite logike. Mogućnosti koje neizrazita logika nudi u kreiranju zanimljivih interakcija su beskonačne te to služi kao dokaz da koncepti koji možda nemaju jasnu svrhu u trenutku kada su nastali ne moraju nužno biti beskorisni.

U početku je naveden kratki uvid u povijest tehnologija računalnih grafika i neizrazite logike. Nakon toga je detaljno razrađena sama mehanika neizrazite logike na već spomenutom primjeru sustava za klimatizaciju kako bi se dobio potpuno razumljiv dojam o tome što je neizrazita logika. U zadnjem dijelu je prikazana igra te su objašnjene njene različite funkcionalnosti. Također je objašnjena primjena neizrazite logike u računalnoj grafici na praktičnom primjeru kako bi se prikazao utjecaj na dojam realističnosti videoigre.

# Popis literature


[1] J. M. Mendel, „Fuzzy logic systems for engineering: a tutorial," *Proceedings of the IEEE*, sv. 83, br. 3, str. 345–377, 1995.

[2] P. Badoni, A. Katal, M. S. Reddy i M. Bhargava, „Graphics vs Gameplay: A Comparative Analysis in Gaming," *2022 2nd International Conference on Intelligent Technologies (CONIT)*, IEEE, 2022., str. 1–8.

[3] C. Machover, „A Brief, Personal History of Computer Graphics.," *Computer*, sv. 11, br. 11, str. 38–45, 1978.

[4] J. A. Castellano i D. E. Mentley, „Large-screen display industry: market and technology trends for direct view and projection displays," *Projection Displays II*, SPIE, sv. 2650, 1996., str. 2–8.

[5] F.-h. Hsu, M. S. Campbell i A. J. Hoane Jr, „Deep Blue system overview," *Proceedings of the 9th international conference on Supercomputing*, 1995., str. 240–244.

[6] G. N. Yannakakis i J. Togelius, *Artificial intelligence and games*. Springer, 2018., sv. 2.

[7] M. Pirovano, „The use of Fuzzy Logic for Artificial Intelligence in Games," 2012. adresa: `http://www.maxis.com/`.

[8] A. Garrido, „A brief history of fuzzy logic," *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, sv. 3, br. 1, str. 71–77, 2012.

[9] *Lotfi A. Zadeh citáty (18 citátov) | Citáty slávnych osobností*. adresa: `https://citaty-slavnych.sk/autori/lotfi-a-zadeh/`.

[10] B. Kosko i S. Isaka, „Fuzzy logic," *Scientific American*, sv. 269, br. 1, str. 76–81, 1993.

[11] Y. Bai i D. Wang, „Fundamentals of fuzzy logic control—fuzzy sets, fuzzy rules and defuzzifications," *Advanced fuzzy logic technologies in industrial applications*, str. 17–36, 2006.

[12] *Fuzzy logic - Wikipedia*. adresa: `https://en.wikipedia.org/wiki/Fuzzy_logic`.

[13] L. A. Zadeh, „Fuzzy sets as a basis for a theory of possibility," *Fuzzy sets and systems*, sv. 1, br. 1, str. 3–28, 1978.

# Popis slika

# Popis tablica