Funkcionalno - logički programski jezik Verse i njegova primena u razvoju računarskih igara

Andrej Gašić[1]

Sadržaj **- Verse je novi funkcionalno-logički programski jezik sa elementima objektno-orijentisanog programiranja, namenjen za interaktivne 3-dimenzionalne prostore. Ovaj jezik je za sada dostupan za modifikaciju igre Fortnite. Korišćenjem programskog jezika Verse kao i alata Unreal Editor for Fortnite i Visual Studio Code, kreirane su u okviru ovog rada dve nove aktivnosti za igru Fortnite: „Igra memorije" i „Igra života". Cilj rada je da prikaže osnovne elemente i karakteristike inovativnog jezika Verse, da uvede dodatna proširenja ovog jezika konstrukcijama koje su prisutne u drugim jezicima i da prikaže njihovu primenu u razvoju pomenutih aktivnosti.**

Ključne reči **- Fortnite, funkcionalno programiranje, logičko programiranje, Verse**

I. Uvod

Verse je novi funkcionalno-logički programski jezik sa elementima objektno-orijentisanog programiranja čija je sintaksa definisana po uzoru na programske jezike C#, TypeScript i Python. Tvorci ovog jezika su Tim Svini, poznat po razvoju alata Unreal Engine, i Sajmon Pejton Džouns, glavni dizajner jezika Haskell i poznatog kompajlera GHC. Za kompletnu implementaciju ovog programskog jezika se koristi termin MaxVerse. Pošto trenutno nije završena nijedna kompletna implementacija, 2023. godine je objavljena implementacija koja odgovara podskupu definisanog programskog jezika koja se zove ShipVerse.

Ciljevi ovog jezika su da bude jednostavan i lak za razumevanje, ali i da podržava apstrakcije koje su programerima korisne u drugim programskim jezicima. Jezik je dizajniran da bude dovoljno brz za sisteme koji funkcionišu u realnom vremenu i razmenjuju veliku količinu podataka preko mreže, ali i da bude široko primenljiv. Razvijen je prvenstveno za kreiranje aplikacija koje

[1] Andrej Gašić, Računarski fakultet, Srbija (e-mail: andrej@gasic.rs)

podržavaju interaktivne 3-dimenzionalne prostore sa više korisnika. Pored toga, Verse je predviđen za velike projekte i zato ima osobine da je produktivan u timskom okruženju i optimizovan da detektuje programerske greške što pre.

U ovom radu je jezik Verse, i njegova trenutna implementacija ShipVerse, upotrebljen za modifikaciju igre Fortnite, za koju su razvijene i objavljene dve nove aktivnosti, „Igra memorije“ i „Igra života“. Pored toga, kritički je analiziran programski jezik Verse u kontekstu implementacije novih aktivnosti u igru Fortnite kao i u kontekstu drugih programskih jezika. Kako implementacija jezika Verse nije kompletna, u ovom radu su predstavljena originalna rešenja za prevazilaženje nedostataka u trenutnoj implementaciji programskog jezika Verse.

II. Programski jezik Verse

Funkcionalna paradigma je prisutna u programskom jeziku Verse prvenstveno u formi podrške za funkcije kao vrednosti. Funkcije mogu i da se prosleđuju drugim funkcijama kao argumenti. Podrazumevano ponašanje u okviru jezika je da su promenljive imutabilne, a funkcije čiste. Slično drugim funkcionalnim jezicima, naredbe ne postoje, već je izraz osnovna jedinica koda u programskom jeziku Verse. Izraz je konstrukcija čijim izvršavanjem se dobija rezultat koji može da se iskoristi u okviru drugih izraza. U izraze spadaju:

- pozivi funkcija
- poređenja vrednosti
- matematičke operacije na vrednostima
- logičke operacije na vrednostima
- operacije kontrola toka
- operacije kreiranja struktura podataka (uključujući klase)
- naredba dodela vrednosti promenljivoj
- numeričke ili tekstualne vrednosti

Koncepti iz logičke paradigme su iskorišćeni da povećaju fleksibilnost struktura kontrola toka u odnosu na druge programske jezike. Izvršavanjem ili evaluacijom izraza u Verse programskom jeziku je moguće dobiti nula, jednu ili više vrednosti [1, 2]. Termin za rezultat sa nula vrednosti u Verse programskom jeziku je neuspeh (failure). Termin za rezultat sa više vrednosti je izbor (choice). Kada rezultatu izraza odgovara više vrednosti, dalje operacije i izrazi koji se oslanjaju na rezultat se nezavisno propagiraju kroz sve vrednosti. Ovom konceptu je pridružen termin „prostiranje odluka kroz prostor“. Redosled više vrednosti u rezultatu je deterministčan.

Sistem tipova u Verse programskom jeziku je implementiran kao skup funkcija identiteta sa restrikcijama [1, 3]. Na primer, celi brojevi se predstavljaju ključnom reči int, koja je ujedno i funkcija identiteta čiji je domen

skup celih brojeva. Ako joj je prosleđena vrednost koja nije ceo broj, ona koristi prethodno definisan sistem za neuspeh, odnosno rezultat joj odgovara nula vrednosti. Klase i ostale korisničko definisane strukture podataka su funktori nad objektima definisanim kao polja.

Umesto statičkih provera saglasnosti sa tipovima, za programski jezik Verse se izvršava provera celokupnog programa pre pokretanja. Ovo je ograničenje nametnuto sistemom tipova za koji ne može da se utvrdi da li će rezultat biti neuspeh dok se ne zna vrednost, a ujedno i namerna odluka da bi logičke i sintaksne greške u kodu bile označene pre pokretanja.

Programski jezik Verse koristi sistem algebarskih efekata da modeluje bočne efekte [4]. Za svaki izraz postoje efekti koji određuju šta može da se desi pri izvršavanju izraza. Ovi efekti dozvoljavaju kontekstu izraza da reaguje na različite slučajeve pri izvršavanju tog izraza. Na primer, pristupanje elementu niza je često neuspešna operacija zbog upotrebe nevalidnih indeksa, i u ovom jeziku je ta operacija naznačena efektom <decides>. Izraze sa ovim efektom nije moguće izvršavati u kontekstu koji ne uzima u obzir mogućnost neuspešnosti operacije [5].

Većina modernih programskih jezika dozvoljava kontrolu toka pomoću if naredbe za grananje i for petlje za iteraciju. U programskom jeziku Verse se dozvoljava kontrola toka tako što se ove tradicionalne strukture koriste kao konteksti koji podržavaju podizraze sa efektom <decides>, jer imaju različite rezultate u odnosu na uspešnost podizraza. If struktura u Verse programskom jeziku je izraz koji određuje granu koju treba izvršiti u odnosu na rezultat izvršavanja podizraza. Ako rezultat podizraza ima jednu ili više vrednosti, izvršava se blok koda vezan za if ključnu reč. Ako je rezultat podizraza neuspeh, onda se izvršava blok koda vezan za else ključnu reč ako ona postoji. For struktura u Verse programskom jeziku pakuje vrednosti izvršenog podizraza u niz.

Značajni efekti koji su trenutno implementirani u programskom jeziku Verse su:

- converges – izraz koji će prekinuti sa izvršavanjem posle nekog vremena jer će se za njega odrediti konačni rezultat
- computes – izraz koji ne mora nužno da se zaustavi (recimo, računanje sledbenika za svakog člana beskonačnog niza)
- varies – izraz koji ne vraća isti rezultat pri svakom izvršavanju, tj. nije čista funkcija
- transacts – izraz koji može da menja vrednosti u memorijskim lokacijama
- no rollback – izraz čije posledice izvršavanja ne mogu da se ponište
- suspends – izraz koji se izvršava asinhrono
- decides – izraz koji može da bude neuspešan, tj. da ne vrati vrednost

Izrazu je moguće pripisati više efekata, ali neki efekti su uzajamno isključivi. <converges>, <computes>, <varies>, <transacts> i <no_rollback> formiraju hijerarhiju efekata, gde svaki efekat obuhvata ponašanje prethodnog efekta kao podskup ponašanja obuhvaćenog trenutnim efektom. To znači da izraz koji je naznačen sa <transacts> efektom dozvoljava ponašanje koje može da se desi izrazima naznačenim sa <converges> ili <varies> efektom.

U okviru jezika nije moguće definisati nove efekte za razliku od većine jezika koji trenutno podržavaju algebarske efekte, što je ograničenje uvedeno kako bi se olakšala prethodno pomenuta analiza celokupnog programa pre pokretanja. Efekti koji su dostupni u Verse programskom jeziku su modelovani po uzoru na funkcionalnosti koje pružaju monade za ulaz-izlaz, logiku i softversku transakcionu memoriju u Haskell programskom jeziku.

Svaka korisničko definisana funkcija može da se naznači efektima pomoću specifikatora. Specifikatori su jezička konstrukcija koja ne služi samo za definisanje efekata, već i za definisanje raznih karakteristika funkcija i struktura podataka. Funkcije podržavaju specifikatore <constructor>, koji određuje da funkcija isključivo kreira instancu strukture podataka, i <override>, koji dozvoljava polimorfizam, tako što zamenjuje implementaciju funkcije koja se pojavljuje u interfejsu ili klasi koja predstavlja pretka trenutne klase. Postoje i specifikatori koji definišu pristupna prava za članove struktura podataka, to su private, public, protected, internal.

Programski jezik Verse je osmišljen tako da olakša pisanje distributivnih i konkurentnih programa, a to je postignuto kombinacijom <suspends> efekta i ugrađenim izrazima za kontrolisanje asinhronih izraza [6]. Kao i za druge efekte, izraze sa <suspends> efektom je moguće izvršavati samo u kontekstu koji podržava asinhrone izraze. Ovaj kontekst je moguće napraviti preko spawn izraza. U kontekstu koji podržava asinhrone izraze je moguće koristiti izraze:

- race – pokreće više asinhronih izraza i kada jedan završi izvršavanje, vrati njegov rezultat kao rezultat celokupnog izraza, a zaustavi ostale i poništi posledice njihovog izvršavanja
- rush – pokreće više asinhronih izraza i kada jedan završi izvršavanje, vrati njegov rezultat kao rezultat celokupnog izraza, a ne prekida izvršavanje ostalih izraza
- sync – pokreće više asinhronih izraza i čeka da se izvršavanje svih završi, i vrati njihove rezultate kao članove sekvence (tupla)
- branch – pokreće asinhroni izraz na ispali pa zaboravi (fire-and-forget) način

Svi neasinhroni izrazi koji se pojavljuju u bloku koda zajedno i nisu rastavljeni asinhronim izrazom se izvršavaju u muteksu.

Sve ovo čini jezik Verse pogodnim za kreiranje novih aktivnosti u okviru računarskih igara preko mreže.

III. Igra Fortnite i okruženje za razvoj

Fortnite je igra razvijena od strane kompanije Epic Games koja omogućava interakciju velikog broja igrača u raznim aktivnostima, uglavnom kompetitivnim, ali postoje i aktivnosti kolaborativne prirode [7]. Glavna aktivnost u igri Fortnite je takmičenje 100 igrača u isto vreme. Tim Svini ima želju da razvija igru do nivoa da ima neograničen broj igrača i aktivnosti u zajedničkom prostoru. Fortnite je jedna od retkih igara koja nudi platformu za kreiranje, menjanje i eksperimentisanje sa okolinom i sistemima u okviru igre. Korisnici mogu da se oslone na već postojeće sadržaje u igri, i da ih dopune svojim modifikacijama kako bi formirali nove aktivnosti. Modifikacije se mogu deliti sa drugim igračima. Epic Games podržavaju i angažuju korisnike da modifikuju igru Fortnite u cilju opstajanja igre na tržištu. Verse je pogodan za razvoj i unapređenje postojećih računarskih igara. Planirano je da do 2025. godine, Verse postane standardni jezik za kreiranje i modifikaciju računarskih igara pomoću Unreal Engine alata [8]. Za sada se programski jezik Verse može koristiti samo za modifikaciju igre Fortnite.

A. Unreal Editor for Fortnite

Unreal Editor for Fortnite (UEFN) je računarska aplikacija za dizajniranje i razvoj novih aktivnosti i njihovo objavljivanje u igri Fortnite [9]. UEFN ima pristup raznim alatima koji su deo Unreal Engine 5 aplikacije za pravljenje igara. Ova aplikacija uključuje razne alate kao što su alati za pravljenje modela, tekstura, materijala i vizuelnih efekata, kao i alata za unos unapred pripremljenih grafičkih i zvučnih sadržaja. UEFN sadrži i kompajler za programski jezik Verse, što dozvoljava pisanje Verse koda i njegovo izvršavanje u okviru aktivnosti.

Fortnite kreatori mogu da koriste i kombinuju postojeći skup alata iz igre i alate koje nudi UEFN kako bi kreirali nove aktivnosti, a timovi kreatora mogu da rade zajedno na računaru, konzoli i telefonu kako bi razvili i testirali aktivnosti u realnom vremenu.

UEFN je dostupan na aplikaciji Epic Games launcher na kojoj je i dostupna sama igra Fortnite. Instaliranje UEFN alata zahteva Epic Games nalog i već instaliranu igru Fortnite. Prostor potreban za instalaciju UEFN alata je oko 6 GB. Trenutno postoji samo Beta verzija koja je još u fazi razvoja. Međutim , i ova verzija pruža mnogo mogućnosti od kojih su u ovom radu korišćene sledeće:

- pravljenje modela, tekstura, materijala i vizuelnih efekata
- dizajniranje karakteristika za nove aktivnosti kroz programski jezik Verse

- organizovanje promena pomoću Unreal Revision Control sistema

Unreal Revision Control je sistem napravljen po uzoru na Git koji onemogućava da se izgube promene koje bilo koji dizajner unosi u projekat preko UEFN alata. Ovaj sistem omogućava sinhronizaciju rada čitavog tima na istoj aktivnosti.

B. *Visual Studio Code*

Visual Studio Code je editor koda namenjen za pisanje modernih aplikacija [10]. Razvijen je od strane kompanije Microsoft i dostupan je na svim operativnim sistemima za računare. Ima ugrađenu podršku za JavaScript i TypeScript programske jezike, ali omogućava korišćenje i drugih programskih jezika kao što su C, C# i Python, i to pomoću ekstenzija.

Instaliranjem UEFN alata se automatski dodaje ekstenzija za programski jezik Verse u editor koda Visual Studio Code. Ekstenzija za programski jezik Verse je unela promene u bojama za reči koje su deo sintakse jezika, kao i podvlačenje i naglašavanje grešaka u kodu.

IV. Implementacija „Igre memorije“

Ova aktivnost je zasnovana na poznatoj dečijoj igri koja se igra sa skupom karata. Karte su predstavljene kao dugmad na ekranu. Igrači mogu da pritisnu dugme da otkriju broj na karti. Igrač može otvoriti 2 karte u isto vreme, a ako obe karte imaju jednake brojeve, karte ostaju otvorene na ekranu. Brojevi koji su povezani sa kartama su fiksni. Kada igrač uspešno otkrije sve karte, igra se završava. Aktivnost se može podesiti tako da je broj karata na ekranu, kao i broj karata koje se otvaraju u isto vreme, promenljiv.

Analiza dokumentacije UEFN alata i jezika Verse je pokazala da jezik Verse omogućava dodavanje „Igre memorije“ u igru Fortnite, što ranije nije bilo moguće. Iz tih razloga, ova dobro poznata igra nije do sada postojala u igri Fortnite.

Implementacija „Igre memorije“ u jeziku Verse je zahtevala pisanje originalnog koda koji se sastoji iz generalnih pomoćnih funkcija i struktura podataka kao i funkcija i struktura podataka za kontrolu toka igre.

A. *Generalne pomoćne funkcije*

Definicije funkcija za manipulisanje struktura podataka analogne funkcijama u drugim programskim jezicima su prikazane na Sl. 1. Većina pomoćnih funkcija i klasa za organizaciju struktura podataka je analogna klasama i funkcijama višeg reda iz drugih programskih jezika. To su funkcije:

Flatten – Flatten (Haskell),

Map – Map (Haskell),

Reduce – Fold (Haskell),

Rectangular_array – pravougaona matrica (C#)

```
(Items:[][]t where t:type).Flatten<public>()<computes>:[]t =
    for (NestedArray : Items, Item : NestedArray):
        Item

Map<public>(Array:[]input, Func:type{_(:input):result} where
input:type, result:type):[]result =
    for (Element : Array):
        Func(Element)

MMap<public>(Matrix:[][]input, Func:type{_(:input):result} where
input:type, result:type):[][]result =
    for (Array : Matrix):
        for (Element : Array):
            Func(Element)

Reduce<public>(Array:[]input, Func:type{_(:tuple(input,
result))<transacts>:result}, Accumulator:result where input:type,
result:type)<transacts>:result =
    var AccumulatorVariable:result = Accumulator
    for (Element : Array):
        set AccumulatorVariable = Func(Element, AccumulatorVariable)
    AccumulatorVariable

rectangular_array(t:type) := class:
    UnderlyingArray:[]t
    Rows:int
    Columns:int

    ToMatrix()<computes>:[][]t =
        for (Row := 0..Rows - 1):
            for (Column := 0..Columns - 1, Element := GetElement[Row,
Column]):
                Element

    ToArray()<computes>:[]t =
        UnderlyingArray

    GetElement(Row:int, Column:int)<computes><decides>:t =
        UnderlyingArray[Column + Row * Columns]
```

SL. 1. DEFINICIJE FUNKCIJA ZA MANIPULISANJE STRUKTURA PODATAKA ANALOGNE FUNKCIJAMA U DRUGIM PROGRAMSKIM JEZICIMA

Na Sl. 2 su prikazane funkcije i klase za konkurentnost koje su namenjene da olakšaju upravljanje većim brojem tokova akcija, što u ovom slučaju podrazumeva i upravljanje procesom otvaranja karata.

Async_closure - klasa koja omogućava pakovanje funkcije sa prethodno definisanim stanjem iz okruženja (tj. zatvorenje)

DynamicRace – funkcija koja izvršava niz funkcija tako da osigura da će promene samo jedne da se primene, a rezultati ostalih će se poništiti

RecursiveRace – funkcija koja predstavlja rekurzivni/indukcioni korak za DynamicRace funkciju

```
async_closure<public>(t : type) := class:
    Execute<public>:type{_()<suspends>:t}

(Closures:[]async_closure(t) where
t:type).DynamicRace<public>()<suspends>:?t =
    if (Closure := Closures[0]):
        WinningResult := RecursiveRace(Closure, Closures, 0)
        option{WinningResult}
    else:
        false

RecursiveRace(Closure:async_closure(t), Closures:[]async_closure(t),
Index:int where t:type)<suspends>:t =
    if (NextClosure := Closures[Index + 1]):
        race:
            Closure.Execute()
            RecursiveRace(NextClosure, Closures, Index + 1)
    else:
        Closure.Execute()
```

SL. 2. DEFINICIJE FUNKCIJA I KLASA NAMENJENE ZA OLAKŠAVANJE KONKURENTNIH OPERACIJA

Kreirane su pomoćne funkcije za prikaz i sastavljanje grafičkih elemenata koje su namenjene da olakšaju prevođenje i prikazivanje stanja igre korisniku.

ToUIMessage – prevodi tekst ili broj u poruke koje mogu da se prikažu na ekranu

CreateBackground – pravi podlogu za druge grafičke elemente na osnovu prosleđene boje

Stack – slaže niz grafičkih elemenata jedan preko drugog

ArrangeGrid – pravi tabelu grafičkih elemenata tako da se prostire preko celog ekrana i tako da svaki element zauzima jednaku količinu prostora

B. Funkcije i strukture podataka za kontrolu toka igre

Na Sl. 3 je prikazana funkcija PlayGame. Tok igre se odvija u okviru ove funkcije koja se oslanja na sve prethodno opisane funkcije i strukture podataka. Potrebno je istaći da svaka karta sadrži svoj tok koji prati da li je ona izabrana ili ne, koji se dobija funkcijom GetTileAsyncClosure. Ovi tokovi se međusobno isključuju pomoću DynamicRace funkcije, čime se postiže mogućnost da se samo jedna karta otvori u isto vreme.

IsAllEqual – proverava da li su svi članovi liste identični, implementirano korišćenjem IsEqual i Reduce funkcije

IsEqual – proverava da li je prosleđen objekat jednak zapamćenom objektu

Tile - klasa koja povezuje broj karte i grafički prikaz karte

```
PlayGame()<suspends> : void =
    Values := Map(Tiles, GetTileValue)
    Closures := Map(Tiles, GetTileAsyncClosure)
    loop:
        OpenedTiles := for (OpenCount := 1..NumberOfCopies):
            MaybeTile := Closures.DynamicRace()
            if (Tile := MaybeTile?):
                Tile.Button.SetVisibility(widget_visibility.Hidden)
                Tile
            else:
                break
        OpenedValues := Map(OpenedTiles, GetTileValue)
        if(OpenedValues.IsAllEqual()?):
            Visibility := Map(Tiles, GetTileButtonVisibility)
            if (Visibility.IsAllEqual()?):
                NotifText.SetText(ToUIMessage("Congratulations!"))
                break
        else:
            Sleep(FlipDelay)
            for (Tile : OpenedTiles):
                Tile.Button.SetVisibility(widget_visibility.Visible)
```

SL. 3. DEFINICIJA FUNKCIJE KOJA IZVRŠAVA TOK IGRE

Na Sl. 4 je pokazano kako tok igre može da se zaustavi u bilo kom momentu. To se može desiti ako igrač izađe iz igre, a može i da se zaustavi igra ako igraču istekne vreme. Funkcije i strukture podataka koje upravljaju ovim događajima su:

ExecuteGameLoop - funkcija koja kontroliše događaje i zaustavlja tok u zavisnosti od njih

Countdown_timer – klasa koja sadrži vreme koje je preostalo igraču da završi igru i samostalno ažurira vrednosti

RunCountdown – funkcija klase Countdown_timer koja prati protok vremena

UpdateUI – funkcija klase Countdown_timer koja grafički prikazuje preostalo vreme

```
ExecuteGameLoop(State:game_state,
Timer:countdown_timer)<suspends>:void =
    race:
        State.PlayGame()
        Timer.RunCountdown()
    Sleep(10.0)

ExecuteGameLoop(Exit:button_loud, State:game_state,
Timer:countdown_timer, GUI:gui)<suspends>:void =
    race:
        Exit.OnClick().Await()
        ExecuteGameLoop(State, Timer)
    GUI.Remove()

countdown_timer := class:
    RemainingTimeWidget:text_block
    NotifText:text_block
    TimerTickPeriod:float = 1.0
    var RemainingTime:float

    RunCountdown()<suspends> : void =
        UpdateUI()
        loop:
            Sleep(TimerTickPeriod)
            set RemainingTime -= TimerTickPeriod

            UpdateUI()

            if (RemainingTime <= 0.0):
                NotifText.SetText(ToUIMessage("Time is up!"))
                break

    UpdateUI() : void =
        if (RoundedTime := Int[RemainingTime]):
            RemainingTimeWidget.SetText(ToUIMessage(RoundedTime))
```

SL. 4. DEFINICIJE FUNKCIJA I STRUKTURA PODATAKA KOJE ZAUSTAVLJAJU IGRU

C. *Pristup „Igri memorije“*

Aktivnosti može da se pristupi preko računara, konzole ili telefona. Prethodno je potrebno instalirati igru Fortnite. U okviru igre je moguće ukucati

kod za aktivnost koji igrača vodi do „Igre memorije“. Na Sl. 5 je prikazana slika sa ekrana „Igre memorije“ u okviru igre Fortnite.

SL. 5. SLIKA SA EKRANA „IGRE MEMORIJE“ U OKVIRU IGRE FORTNITE

D. Evaluacija programskog jezika Verse u implementaciji „Igre memorije“

Ugrađene konkurentne konstrukcije u programskom jeziku Verse, naročito race konstrukcija, pružaju interesantne načine za modeliranje toka igara. Tako, na primer, u ovoj implementaciji „Igre memorije“ je korišćena race konstrukcija koja omogućava otvaranje samo jedne karte u isto vreme. Ako igrač pokuša da pritisne više karata, samo se rezultat jedne prikaže jer se rezultat ostalih funkcija poništi. Izraz race nema analognih konstrukcija u poznatim modernim programskim jezicima. Preostale dve konkurentne konstrukcije imaju analogne konstrukcije u C# programskom jeziku: rush konstrukcija je analogna Task.WhenAny konstrukciji, a sync konstrukcija je analogna Task.WhenAll konstrukciji.

Konkurentne konstrukcije u programskim jezicima su generalno namenjene da mogu da se koriste sa proizvoljnim brojem konkurentnih izraza. Task.WhenAll iz C# programskog jezika dozvoljava programeru da prosledi niz od proizvoljnog broja konkurentnih izraza i da vrati rezultat svih konkurentnih izraza zajedno. Konstrukcije u okviru Verse programskog jezika trenutno nemaju ovu mogućnost i autori ovog jezika su istakli to kao nedostatak. U ovom radu kao deo implementacije „Igre memorije“ je rešen ovaj problem tako što je napisana DynamicRace funkcija koja za proizvoljni niz funkcija rekurzivno poziva race konstrukciju.

Programski jezik Verse obezbeđuje podskup funkcionalnosti u odnosu na Unreal Engine vezane za grafičke korisničke interfejse. Prednost ovoga je da su grafički korisnički interfejsi performantni i mogu da se skaliraju nezavisno od veličine ekrana, ali mana je da podskup funkcionalnosti izabranih za ovaj programski jezik sakriva ove mogućnosti. U implementaciji „Igre memorije“ je ovaj problem delimično rešen oslanjajući se na stack_box grafičke elemente koji proizvoljan broj elemenata preraspoređuju prema jednoj dimenziji ekrana.

V. Implementacija „Igre života“

Ova aktivnost je implementacija poznatog ćelijskog automata Konvejeve „Igre života“ koja koristi boje na podnim poljima da bi prikazala da li su ćelije „žive“ ili „mrtve“. Igrači mogu da „ožive“ nove ćelije pucajući na polja pomoću puške. Oni takođe mogu da zaustave simulaciju da bi pripremili složenije obrasce, kao i da osveže simulaciju tako da sva polja odgovaraju „mrtvim“ ćelijama. Implementacija prati standardna pravila „Igre života“ gde:
- Susedi ćelije uključuju ćelije koje su dijagonalno postavljene od nje
- „Žive“ ćelije koje imaju 2 ili 3 „živa“ suseda ostaju „žive“
- „Mrtve“ ćelije koje imaju 3 „žive“ ćelije kao susede postaju „žive“ ćelije

Konvejeva „Igra života“ je poznat primer matematičke simulacije za koju, u generalnom slučaju, ne može da se odredi da li će ikad da se zaustavi. Nije bilo moguće pokretanje matematičkih simulacija u igri Fortnite pre pojave programskog jezika Verse. Ova matematička simulacija je implementirana da dokaže da kroz Verse može da se simulira Tjuringova mašina.

Implementacija „Igre života“ u programskom jeziku Verse je zahtevala pisanje originalnog koda koji se sastoji iz generalnih pomoćnih funkcija kao i funkcija i struktura podataka za predstavljanje simulacije.

A. Generalne pomoćne funkcije

Kao i za prethodnu implementaciju, dosta pomoćnih funkcija za organizaciju struktura podataka je napisano po uzoru na funckije iz drugih programskih jezika (Sl. 6).

GetOrDefault – Lookup (Haskell)

Contains – Elem (Haskell)

Sem toga, implementirane su funkcije za konstruisanje čestih oblika mapa.

CreateHashMap – funkcija koja konstruiše mapu heševa i vrednosti, gde je heš generisan pomoću funkcije koja se primenjuje na vrednost

CreateFrequencyMap – funkcija koja konstruiše mapu objekata i broja pojavljivanja tog objekta u nekoj strukturi podataka

InsertOrIncrement – funkcija koja dodaje objekat u mapu ili povećava broj pojavljivanja objekta u mapi ako se ranije već pojavio

Potrebno je istaći da u trenutnoj implementaciji jezika Verse, korišćenje funkcije koja podržava više tipova podataka za tip ključa u okviru mape izaziva nagli prekid u izvršavanju, što nije posledica ovog koda već mana u implementaciji. Zbog toga je u stvarnoj implementaciji ove igre bilo neophodno da se kod iz isečka u nekim slučajevima prilagodi specifičnim tipovima korišćenim u okviru ostatka koda.

```
CreateHashMap<public>(Array:[]t, Func:type{_(:t)<transacts>:int} where
t:type)<transacts>:[int]t =
    var HashMap:[int]t = map{}
    for (Element : Array):
        if (set HashMap[Func(Element)] = Element) {}
    HashMap

GetOrDefault<public>(Map:[key]val, Item:key, Default:val where
key:subtype(comparable), val:type)<computes>:val =
    if (Val := Map[Item]):
        Val
    else:
        Default

Contains<public>(Array:[]t, Item:t where
t:subtype(comparable))<computes>:logic =
    for (ArrayItem : Array):
        if (ArrayItem = Item):
            return true
    false

Reduce<public>(Array:[]input, Func:type{_(:tuple(input,
result))<transacts>:result}, Accumulator:result where input:type,
result:type)<transacts>:result =
    var AccumulatorVariable:result = Accumulator
    for (Element : Array):
        set AccumulatorVariable = Func(Element, AccumulatorVariable)
    AccumulatorVariable

InsertOrIncrement<public>(Item:t, FrequencyMap:[t]int where
t:subtype(comparable))<transacts>:[t]int =
    CurrentOccurrences := GetOrDefault(FrequencyMap, Item, 0)
    var NewFrequencyMap:[t]int = FrequencyMap
    if (set NewFrequencyMap[Item] = CurrentOccurrences + 1):
        NewFrequencyMap
    else:
        FrequencyMap

(Array:[]t where
t:subtype(comparable)).CreateFrequencyMap<public>()<transacts>:[t]int
=
    Reduce(Array, InsertOrIncrement, map{})
```

SL. 6. DEFINICIJE FUNKCIJA ZA KONSTRUISANJE I MANIPULACIJU STRUKTURA PODATAKA

Pomoćne funkcije za matematičke operacije su dodate da olakšaju računanje koraka u matematičkoj simulaciji.

Szudzik – funkcija koja implementira matematičku funkciju koja izračunava celobrojni heš za par brojeva tako da je mapiranje 1-1 [11].

StdDiv – funkcija koja predstavlja operaciju deljenja celih brojeva, koja se ponaša kao operacija deljenja celih brojeva u drugim programskim jezicima, tako da rezultati teži ka nuli

AddTuple – funkcija koja sabira dva para brojeva tako da se prva koordinata prvog para sabere sa prvom koordinatom drugog para, a druga koordinata prvog para sabere sa drugom koordinatom drugog para

WrapToBounds – funkcija koja „preliva“ cele brojeve ili par brojeva prema definisanom opsegu

LeavesBounds – funkcija koja proverava da li ceo broj ili par brojeva izlazi van definisanog opsega

FromXYZ – funkcija koja konstruiše instancu vector3 strukture pomoću 3 koordinate

GetRangeWithCenter – funkcija koja određuje opseg kordinata na osnovu veličine prostora

B. Funkcije i strukture podataka za predstavljanje simulacije

Pravila „Igre života“ su definisane pomoću funkcija. Pošto pravila igre nisu strogo definisana za ograničen broj polja, u ovoj implementaciji su napisane dve varijacije pravila.

GetNeighboursClassic – funkcija koja određuje koordinate susednih polja prema koordinatama trenutnog polja u horizontalnom, vertikalnom i dijagonalnom smeru

GetNeighboursBounded – funkcija koja određuje koordinate susednih polja kao GetNeighboursClassic ali odstrani susede koji se nalaze van ograničenog prostora

GetNeighboursWrapped – funkcija koja određuje koordinate susednih polja kao GetNeighboursClassic ali „preliva“ koordinate ako se one nalaze van ograničenog prostora

ShouldStayAliveClassic – funkcija koja određuje da li polje treba da bude živa ćelija u sledećoj iteraciji

Sama simulacija je upakovana u strukturu podataka koja samostalno može da izračuna sledeće stanje u odnosu na trenutno stanje (Sl. 7).

Board – struktura podataka koja sadrži celokupno stanje simulacije, uključujući trenutno „žive“ ćelije, opseg prostora simulacije i pravila igre

Update – funkcija koja izračunava sledeću iteraciju simulacije na osnovu pravila igre

GetFromMaps – funkcija koja određuje ćelije koje treba da budu „žive“ u sledećoj iteraciji na osnovu mape broja pojavljivanja susednih ćelija

U klasi Board su pravila igre definisana u promenljivama GetNeighbours i ShouldStayAlive. GetNeighbours ima tip funkcije koja se primenjuje na polje ćelije i ograničenja table, a kao rezultat daje listu ćelija koje odgovaraju susedima date ćelije. ShouldStayAlive ima tip funkcije koja se primenjuje na

polje ćelije, listu trenutno „živih“ ćelija i broj pojavljivanja među susedima, tako da je izvršavanje funkcije neuspešno ako data ćelija ne treba da bude „živa“ u sledećem ciklusu.

```
board<public> := class:
    XRange<public>:point
    YRange<public>:point

    GetNeighbours:type{_(:tuple(point, point,
point))<computes>:[]point} = GetNeighboursBounded
    ShouldStayAlive:type{_(:[]point, :point,
:int)<decides><computes>:void} = ShouldStayAliveClassic

    LivingCells<public>:[]point = array{}

    Update<public>(AddedCells:[]point)<transacts>:board =
        Cells := AddedCells + for (Cell : LivingCells, not
Contains(AddedCells, Cell)?):
            Cell
        Neighbours := for:
            Cell : Cells
            CellNeighbours := GetNeighbours(Cell, XRange, YRange)
            Neighbour : CellNeighbours
        do:
            Neighbour
        Hashes := for (Neighbour : Neighbours):
            Szudzik(Neighbour)
        HashMap := CreateHashMap(Neighbours, Szudzik)
        FrequencyMap := Hashes.CreateFrequencyMap()
        NewLivingCells := GetFromMaps(Cells, FrequencyMap, HashMap)
        board{LivingCells := NewLivingCells, XRange := XRange, YRange
:= YRange}

    GetFromMaps(Cells:[]point, FrequencyMap:[int]int,
HashMap:[int]point)<computes>:[]point =
        for:
            Hash->Occurrences : FrequencyMap
            Cell := HashMap[Hash]
            ShouldStayAlive[Cells, Cell, Occurrences]
        do:
            Cell
```

SL. 7. DEFINICIJE FUNKCIJA I STRUKTURA PODATAKA ZA KONTROLU STANJA IGRE

Simulacija mora grafički da se prikaže u okviru igre. To se postiže povezivanjem polja u simulaciji sa objektima u igri. Simulacija mora i da reaguje na interakciju od strane korisnika (Sl. 8).

Monitorable – interfejs koji definiše funkciju koja treba da se izvršava redovno u određenom intervalu

Ticker – klasa koja izvršava funkcije implementacija interfejsa Monitorable u intervalu od tridesetine sekunde

Tile – klasa koja pakuje grafički i matematički prikaz polja i dozvoljava ažuriranje grafičkog prikaza na osnovu promena u simulaciji

Kada igrač pokuša da „oživi“ ćeliju, ona mora da se doda u listu ćelija koje ulaze u sledeći ciklus simulacije. Ovo se postiže funkcijom RegisterHit, koja predstavlja zatvorenje nad globalnom listom novih ćelija.

```
tile := class(monitorable):
    Cell:tuple(int, int)

    Model:creative_prop
    Asset:creative_prop_asset

    RegisterHit:type{_(:tuple(int, int))<transacts>:void}

    var Prop<private>:creative_prop = creative_prop{}

    Spawn<private>():void =
        Result := SpawnProp(Asset, Model.GetTransform().Translation +
FromXYZ(0.0, 0.0, 50.1), IdentityRotation())
        if (set Prop = Result(0)?) {}

    Instantiate():void =
        SetDead()
        Spawn()

    Update<override>():void =
        if (not Prop.IsValid[]):
            RegisterHit(Cell)
            SetLiving()
            Spawn()

    SetLiving():void =
        Model.SetMaterial(LivingCell)

    SetDead():void =
        Model.SetMaterial(DeadCell)
```

SL. 8. DEFINICIJA STRUKTURE PODATAKA KOJA ODGOVARA PRIKAZU ĆELIJE U IGRI

Na stanje matematičke simulacije može da se utiče na više načina. Igrač može da doda novu „živu“ ćeliju tako što puca na polje, može da zaustavi i nastavi simulaciju kako bi uneo složenije oblike, a može i da „ubije“ sve ćelije da bi počeo ispočetka (Sl. 9).

WaitForTurnOff – funkcija koja prati da li je igrač zaustavio simulaciju

Update – funkcija koja usklađuje simulaciju sa promenama korisnika i grafičkim prikazom u okviru igre

```
WaitForTurnOff()<suspends>:void =
    if (PauseSwitch.GetCurrentState[]):
        PauseSwitch.TurnedOffEvent.Await()

Update(Board:board, Tiles:[]tile)<suspends>:void =
    MaybeBoard:?board := race:
        block:
            Sleep(IterationDelay)
            option{Board.Update(Hits)}
        block:
            ResetButton.InteractedWithEvent.Await()
            option{board{XRange := Board.XRange, YRange :=
Board.YRange}}
        block:
            WaitForTurnOff()
            false
    NextBoard := if (NewBoard := MaybeBoard?):
        set Hits = array{}
        for (Tile : Tiles):
            if (Contains(NewBoard.LivingCells, Tile.Cell)?):
                Tile.SetLiving()
            else:
                Tile.SetDead()
        NewBoard
    else:
        PauseSwitch.TurnedOnEvent.Await()
        Board
    Update(NextBoard, Tiles)
```

SL. 9. DEFINICIJE FUNKCIJA KOJE OPISUJU TOK SIMULACIJE I REAGUJU NA INTERAKCIJU IGRAČA

C. Pristup „Igri života“

Aktivnosti može da se pristupi preko računara, konzole ili telefona. Prethodno je potrebno instalirati igru Fortnite. U okviru igre je moguće ukucati kod za aktivnost koji igrača vodi do „Igre života“. Na Sl. 10 je prikazana slika sa ekrana „Igre života“ u igri Fortnite.

SL. 10. SLIKA SA EKRANA" ŽIVOTA" U IGRI FORTNITE

D. Evaluacija programskog jezika Verse u implementaciji „Igre života“

Mape u Verse programskom jeziku nisu trenutno implementirane do kraja jer su ograničene na ključeve primitivnog tipa, a namera dizajnera jezika je bila

da objekat svakog tipa koji implementira comparable interfejs može da bude ključ. U ovoj implementaciji je to ograničenje zaobiđeno pomoću funkcije za heširanje para brojeva.

Igrač u igri Fortnite može na razne načine da interaguje sa prostorom i elementima igre, ali većina interakcija ne može da se prati kroz programski jezik Verse. U ovo se ubraja nedostatak mehanizama da se izvrši funkcija ako korisnik puca, niti postoji mehanizam da se izvrši funkcija ako metak pogodi metu. U ovoj implementaciji je omogućeno praćenje i provera da li je neki objekat uništen, što omogućava nameštanje objekata tako da objekat nestane kada igrač puca na njega, a da se konstanto proverava da li objekat još uvek postoji, kao i da se reaguje kada se primeti promena. Postavljanjem velike količine objekata se može i odrediti gde je igrač pucao.

Zbog prirode Verse programskog jezika kao alata za pisanje konkurentnih mrežnih aplikacija, problem stanja trke se potpuno zaobilazi u nekim slučajevima koji bi u drugim programskim jezicima zahtevali dodatno razmatranje. U implementaciji „Igre života“ gde igrači mogu da „ožive“ ćelije, u svaku iteraciju se ubaci niz novih ćelija koje su nastale putem korisničke interakcije sa okruženjem. Ovaj niz se ažurira tako što se na kraj niza dodaje nova ćelija svaki put kada igrač pogodi polje. U drugim programskim jezicima je potrebno osigurati da dva korisnika koja dodaju novu ćeliju u isto vreme neće obrisati promene jedni drugog. U Verse programskom jeziku je definisano da se svaki podskup instrukcija koji ne sadrži asinhronu instrukciju izvršava u muteksu, što znači da nijedna druga instrukcija neće prekidati izvršavanje tog podskupa koda. Ovo je trenutno postignuto tako što se instrukcije Verse programskog jezika izvršavaju na jednoj niti, ali će ubuduće biti rešeno tako što će kompajler da prerasporedi instrukcije koje međusobno mogu nepredvidivo da utiču na izvršavanje.

Matematičke simulacije mogu da se koriste kao temelj za razne igre. Na primer, pravila „Igre života“ mogu da se prošire tako da svaki igrač ima svoje ćelije. Ovakvim proširenjem bi mogla da se napravi igra u kojoj se meri čije su ćelije preuzele veći udeo prostora u određenom vremenu.

VI. Zaključak

Programski jezik Verse je jezik opšte namene, i po rečima Sajmon Pejton Džounza, ima za cilj da promeni temelje programiranja. Imperativne i objektno-orijentisane paradigme su preovladale u većini programskih jezika koji se danas koriste. Za razliku od njih, programski jezik Verse se oslanja na funkcionalne i logičke paradigme kako bi ostvario apstrakcije koje omogućavaju efikasno korišćenje računarskih resursa, kao i potvrđivanje tačnosti koda. Sadrži elemente Icon programskog jezika koji nikad nije zaživeo [11]. Međutim, programski jezik Verse ima budućnost jer je preuzeo mnoge

osobine iz različitih jezika. Tako Verse predstavlja jezik koji je široko primenljiv i lak za korišćene, i očekuje se da bude prihvaćen od strane programera. Ipak, treba skrenuti pažnju da je sintaksa ponekad zbunjujuća i da jezik nije u potpunosti razvijen.

U cilju implementacije novih aktivnosti u igru Fortnite bilo je neophodno prevazići nedostatke u trenutnoj implementaciji jezika Verse kao i prihvatiti drugačiji način razmišljanja. Tako ovaj rad nudi originalna rešenja za nedostatke programskog jezika Verse. Ova rešenja su trenutno dostupna na GitHub veb servisu za deljenje koda [12], a u budućnosti bi bilo poželjno da postanu deo standardne biblioteke jezika. Nove aktivnosti čija je implementacija opisana u ovom radu su već prihvaćene od strane učesnika u igri (Sl. 11), i predstavljaju stvarni doprinos ovog rada igri Fortnite (Sl. 12).

SL. 11. SLIKA SA EKRANA KOJA POKAZUJE BROJ IGRAČA ZA OBJAVLJENE AKTIVNOSTI OD MOMENTA OBJAVLJIVANJA

SL. 12. PRIKAZ „KREATOR“ STRANICE SA NOVIM AKTIVNOSTIMA U OKVIRU IGRE FORTNITE KOJE SU OBJAVLJENE KAO REZULTAT OVOG RADA

LITERATURA

[1] Neal Glew, Tim Sweeney, and Leaf Petersen. A multivalued language with a dependent type system. In Proceedings of the 2013 ACM SIGPLAN workshop on Dependently-typed programming (DTP '13). Association for Computing Machinery, New York, NY, USA, 25–36. 2013. https://doi.org/10.1145/2502409.2502412

[2] Lennart Augustsson, et al. The Verse Calculus: A Core Calculus for Deterministic Functional Logic Programming. Proceedings of the ACM on Programming Languages, Volume 7, Issue ICFP, Article No.: 203pp 417–447

[3] Brian McKenna. Type system of Fortnite's Verse language. https://brianmckenna.org/blog/verse_types (01.01.2024.)

[4] Epic Games. Verse Specifiers and Attributes. https://dev.epicgames.com/documentation/en-us/uefn/specifiers-and-attributes-in-verse (01.01.2024.)

[5] Epic Games. Verse Failure. https://dev.epicgames.com/documentation/en-us/uefn/failure-in-verse (01.01.2024.)
[6] Epic Games. Verse Time Flow and Concurrency. https://dev.epicgames.com/documentation/en-us/uefn/concurrency-in-verse (01.01.2024.)
[7] Epic Games. Fortnite. https://store.epicgames.com/en-US/p/fortnite (01.01.2024.)
[8] Tim Sweeney. What is Verse for and where is its place in the ecosystem? https://forums.unrealengine.com/t/actual-custom-games-possible-with-uefn-or-just-advanced-map-editor/886043/3 (01.01.2024.)
[9] Epic Games. Unreal Editor for Fortnite available in Public Beta. https://www.unrealengine.com/en-US/blog/unreal-editor-for-fortnite-is-now-available-in-beta (01.01.2024.)
[10] Microsoft. Visual Studio Code. https://code.visualstudio.com/ (01.01.2024.)
[11] Matthew P. Szudzik. The Rosenberg-Strong Pairing Function. 2019. https://doi.org/10.48550/arXiv.1706.04129
[12] Andrej Gašić. Testing ground for the Verse language in Unreal Engine for Fortnite. https://github.com/Androoideka/uefn-verse-lab (19.01.2024.)

ABSTRACT

Verse is a new functional logic programming language with elements of object-oriented programming, intended for interactive 3-dimensional spaces. This language is currently available for modding the game Fortnite. Using the Verse programming language as well as the Unreal Editor for Fortnite and Visual Studio Code tools, two new experiences for Fortnite were created within this work: "Memory Game" and "Game of Life". The aim of the paper is to show the basic elements and characteristics of the innovative programming language Verse, to introduce additional extensions for this language based on constructions that are present in other languages and to show their application in the development of the mentioned experiences.

Keywords - Fortnite, functional programming, logic programming, Verse

FUNCTIONAL LOGIC PROGRAMMING LANGUAGE VERSE AND ITS APPLICATION IN THE DEVELOPMENT OF COMPUTER GAMES

Andrej Gašić