



Универзитет „Св. Климент Охридски“ - Битола  
Факултет за информатички и комуникациски технологии - Битола

# Анализа и примена на машини на состојби при дизајнирање на разгранувачки системи, со посебен осврт на видео игри

- Магистерски труд -

Ментор:  
проф. д-р Виолета Маневска

Кандидат:  
дипл. инж. Мартин Петковски

Битола, септември 2022

## Апстракт

Предметот на овој магистерски труд е анализа на различни начини на имплементација на машини на состојби во различни гранки на развојот на софтвер, демонстрација на специјален начин на имплементација на машина на состојби кој е насочен кон олеснување на работата на техничките писатели и дизајнерите на видео игри, и нејзина примена во видео игра со разгранувачки дијалози специјално развиена за демонстрација на магистерскиот труд.

Дизајнот и имплементацијата на разгранувачки системи во софтвер претставува оргомен предизвик затоа што потребни се многу ресурси и време за еден дизајнер или автор да ги задржи конзистентноста и кохерентноста во рамките на системот. Овој труд нуди неколку практични решенија за намалување на комплексноста при имплементирањето на разгранувачки системи во софтвер, и нивна примена во развојот на видео игри.

Методот на имплементација на разгранувачки системи кој е цел на дискусијата во овој труд им овозможува на авторите на разгранувачките системи дискретно да ги анализираат состојбите во рамките на системот, без да се грижат за целината на системот. Овој метод има свои предности и недостатоци кои се предмет на компаративна анализа со останатите современи методи кои се користат, а исто така се предмет на анализа и од перспектива на дизајнерите и авторите.

Една од целите на овој труд е да покаже како се имплементира и уредувач за разгранувачки системи со алатки кои овозможуваат анализирање на системот од повеќе аспекти од страна на авторите. На крај, се дискутира и примената на овој метод на имплементација и алатките кои се имплементирани во области надвор од видео игрите.

## **Abstract**

The subject of this master's thesis is an analysis of different ways of implementing state machines in different branches of software development, a demonstration of a special way of implementing a state machine that is aimed at facilitating the work of technical writers and video game designers, and its application in a video game with branching dialogues specially developed for the demonstration of the master's thesis.

The design and implementation of branching systems in software is a huge challenge because it takes a lot of resources and time for a designer or author to maintain consistency and coherence within the system. This paper offers some practical solutions for reducing complexity when implementing branching systems in software, and their application in video game development.

The implementation method of branching systems which is the objective of the discussion in this paper allows the authors of branching systems to discretely analyze the states within the system, without worrying about the system as a whole. This method has its own advantages and disadvantages that are subject to comparative analysis with other modern methods that are used, and are also subject to analysis from the perspective of designers and authors.

One of the goals of this paper is to show how to implement and editor for branching systems with tools that allow the authors to analyze the system from multiple aspects. Finally, the application of this implementation method and the tools that have been implemented in areas outside of video games are discussed.

**Комисија за одбрана на магистерскиот труд „Анализа и примена на машини на состојби при дизајнирање на разгранувачки системи, со посебен осврт на видео игри“:**

проф. д-р Виолета Маневска, ментор

проф. д-р Никола Ренdevски, член

проф. д-р Лела Ивановска, член

**Кандидат:** Мартин Петковски, студент на втор циклус студии на студиската програма „Информатички науки и комуникациско инженерство - Софтверско инженерство и апликации“

Универзитет „Св. Климент Охридски“ - Битола

Факултет за информатички и комуникациски технологии - Битола

## **Изјава**

Јас, Мартин Петковски, под полна морална, материјална и кривична одговорност, изјавувам дека при изработката на магистерскиот труд со наслов „Анализа и примена на машини на состојби при дизајнирање на разгранувачки системи, со посебен осврт на видео игри“ ги почитував позитивните законски прописи на Република Македонија од областа на заштитата на интелектуалната сопственост и не користев реченици или делови од трудови на други автори без да ги почитувам методолошките стандарди и соодветниот стил на цитирање. Текстот на магистерскиот труд е мој, автентичен и самостоен труд, и е резултат на моите сознанија и практичните искуства од соодветната област и истиот, освен во деловите означени со фусноти, претставува исклучително мој труд.

За секое неовластено користење на тутг текст или труд или плагијаторство, согласен сум да сносам соодветни дисциплински, граѓански и кривични последици.

Изјавил:

---

Мартин Петковски

Битола, Република Македонија, 20.10.2022

# Содржина

<b>1 Вовед</b>	<b>10</b>
<b>2 Нелинеарни системи</b>	<b>12</b>
2.1 Нелинеарни системи во уметноста . . . . .	13
2.2 Нелинеарни системи во литературата . . . . .	13
2.3 Нелинеарни системи во филмот . . . . .	14
2.4 Нелинеарни системи во видео игрите . . . . .	16
2.4.1 Crash Bandicoot . . . . .	18
2.4.2 Jak and Daxter . . . . .	19
2.4.3 The Stanley Parable . . . . .	20
2.5 Примена на нелинеарни системи во софтвер . . . . .	22
<b>3 Избори</b>	<b>23</b>
<b>4 Компаративна анализа на нелинеарни системи според начинот на имплементирање</b>	<b>24</b>
4.1 Секвенцијален метод . . . . .	24
4.1.1 Визуелни графови . . . . .	24
4.2 Слободен метод . . . . .	25
4.3 Споредба на секвенцијалниот и слободниот метод . . . . .	25
<b>5 Структурна анализа на нелинеарните системи</b>	<b>27</b>

5.1	Систем . . . . .	28
5.2	Состојби . . . . .	28
5.3	Својства . . . . .	30
5.4	Врски меѓу состојбите . . . . .	30
5.5	Изрази за достапност . . . . .	31
5.6	Изрази за промена . . . . .	33
5.6.1	Оператори кои го регулираат начинот на промена .	33
5.6.2	Оператори кои го регулираат типот на промена . .	34
5.6.3	Оператори за користење на вредност на својство .	34
5.6.4	Оператори со специјална намена . . . . .	35
5.7	Активација на состојба . . . . .	35
5.8	Гранки . . . . .	36
<b>6</b>	<b>Тјуринг комплетност на системот</b>	<b>38</b>
<b>7</b>	<b>Имплементација</b>	<b>39</b>
7.1	Евалуација на израз за достапност . . . . .	43
7.2	Евалуација на израз за промена . . . . .	45
7.3	Структура на папки . . . . .	45
7.4	Вчитување на ресурси . . . . .	46
7.5	Дијалог состојби . . . . .	46

<b>8 Уредувач на нелинеарни приказни</b>	<b>48</b>
8.1 Уредувач на дијалог состојби . . . . .	52
8.2 Прелистувач на состојби . . . . .	58
8.2.1 Поглед базен . . . . .	59
8.2.2 Поглед табела . . . . .	60
8.2.3 Поглед граф . . . . .	61
8.2.4 Поглед патишта . . . . .	61
8.3 Преглед на приказна . . . . .	64
8.4 Уредувач на својства . . . . .	65
8.5 Прелистувач на содржини . . . . .	66
8.6 Управувач на време . . . . .	67
8.7 Уредувач на белешки . . . . .	69
8.8 Steam Работилница . . . . .	70
8.8.1 Објавување на приказна . . . . .	72
8.9 Останати компоненти . . . . .	74
8.9.1 Дневник . . . . .	74
8.9.2 Јазик . . . . .	75
8.9.3 Визуелна тема . . . . .	75
8.9.4 Историја на команди . . . . .	76
8.9.5 Резервни копии . . . . .	78
8.10 Поставки . . . . .	78

8.11 Заштита против пиратерија . . . . .	79
<b>9 Читач</b>	<b>81</b>
9.0.1 Стилизирање на компонентите на читачот . . . . .	83
<b>10 Имплементација на јавен апликациски интерфејс (API)</b>	<b>84</b>
<b>11 Интеграција со Steamworks API</b>	<b>87</b>
<b>12 Интеграција како приклучок во софтвер од трети страни</b>	<b>88</b>
12.1 Софтвери за премостување (middleware) . . . . .	88
12.1.1 FMOD . . . . .	88
12.1.2 WWise . . . . .	89
12.2 Интеграција со Unreal Engine 5 . . . . .	89
<b>13 Студија на случај - Proteus 01</b>	<b>91</b>
13.1 Имплементирање на правила со употреба на својства . . . . .	93
13.2 Имплементирање на врски меѓу состојбите . . . . .	94
13.3 Специфични случаи на глобални состојби . . . . .	95
13.4 Специфични случаи на исклучни состојби . . . . .	96
13.5 Имплементирање на состојби кои се повторуваат . . . . .	96
<b>14 Proteus 01 во Unreal Engine 5</b>	<b>99</b>
<b>15 Примена</b>	<b>103</b>

15.1 Примена во литература . . . . .	103
15.2 Примена во филм . . . . .	103
15.3 Примена во образование . . . . .	105
15.4 Примена во видео игри . . . . .	105
<b>16 Заклучок</b>	<b>107</b>

# 1 Вовед

Одлуките кои секој човек ги носи во текот на сопственото секојдневие влијаат на исходот на иднината на целата околина, па затоа нивното формално пресликување во програмски инструкции кои можат да го реплицираат носењето на одлуки е тема која е од интерес за дискутирање и анализирање. Секоја одлука е дел од некој нелинеарен систем, тоа е секвенца на дејства кои водат до некаков исход, позитивен или негативен. Секоја промена во хаотичните системи, како што е на пример промена на времето од сончево во облачно, или промена во движењето на некоја планета околу својот систем, е резултат на некоја промена во нелинеарниот систем која води до таков исход.

Нелинеарниот систем може да има повеќе разграничувања. Гранките на системот се сите можни состојби во кои системот може да биде доведен, затоа овие системи се нарекуваат и разграниувачки системи. Во вистински нелинеарен систем, бројот на гранките е бесконечен, па затоа комплексноста и тежината на дизајнирање на вакви системи е огромна.

Нелинените системи се присутни во сите уметности каде е можен избор од страна на тој што ја конзумира содржината. Постојат нелинеарни филмови, книги, поеми, музика, театарски претстави, видео игри итн. Анализата на нелинените системи во сите уметности е важна за утврдување на потребите на една имплементација на нелинеарен систем. Фокусот на овој труд е наративот во видео игри, но анализата на нелинеарни системи и нивната примена во развојот на софтвер е важна за утврдување на техничката изведба на ваков систем во видео игра.

Целта на видео игрите кои имаат нелинеарен наратив е да ги пресликаат нелинеарните својства на светот околу нас во програмски инструкции кои потоа можат да ги проектираат изборите кои луѓето ги носат во рамките на дејството на играта. За таа цел се користат различни методологии на имплементација на овие системи кои ќе бидат дискутирани во овој труд. Секоја методологија има два аспекти - аспект на техничка имплементација и перформативност и аспект на корисност за авторите.

Авторите на нелинеарни системи се соочуват со множество

на проблеми кои се поврзани со самата природа и комплексноста на нелинеарните системи. За таа цел во овој труд ќе бидат дискутиирани различни алатки кои им овозможуваат на авторите да имаат поголема контрола врз системите кои ги дизајнираат. Алатките се фокусирани на анализа на нелинеарните системи и на врските помеѓу јазлите во тие системи кои им овозможуваат различни погледи на авторите врз нелинеарниот систем.

Конечната цел на овој труд е имплементирање на систем со методологија која е соодветна и од технички аспект и од аспект на корисност на авторите. За таа цел ќе биде анализиран оригинален пристап во дизајнирање на нелинеарните системи, ќе биде дискутирана имплементацијата на овој метод во алатки кои се корисни за авторите и ќе биде демонстрирана студија на случај за овој пристап.

Нелинеарните системи постојат и надвор од рамките на видео игрите, па така имплементацијата на овие системи може да биде искористена и во други области каде е потребно да се носат избори при конзумирање на содржината. Наједноставна форма на анализирање е текстуалната форма бидејќи сите наративи во најпростата форма се сведуваат на текст, па затоа оваа форма е избрана за анализа во овој труд.

Конечно, интеграцијата со еден од најпопуларните софтвери за развој на видео игри Unreal Engine 5, ја демонстрира практичната примена на алатките и нивната улога во рамките на професионален софтвер за развој на видео игри. Со ова е комплетиран циклусот со почеток во дизајн на наратив па се до интеракција со наративот во рамките на една видео игра.

## 2 Нелинеарни системи

Нелинеарен систем е систем во кој промената на влезот е непропорционална со промената на излезот.[9] [12] Ова значи дека мала промена на влезот во системот може да предизвика огромна промена на излезот од системот. Проблемите на нелинеарните системи се од интерес на многу области во науката бидејќи сите системи се наследно нелинеарни во својата природа. [4] Разликата меѓу нелинеарните системи и поедноставните линеарни системи е во тоа што нелинеарните системи можат да доведат до конечни резултати кои се непредвидливи, хаотични или контраинтуитивни. Овие својства на нелинеарните системи ги прават многу погодни за употреба во интерактивни книги, филмови, игри и други области од уметноста во кои пожелно е тој што интерактира со делото да има различно искуство врз основа на одлуките кои ги носи при интеракцијата. Овој труд истражува како нелинеарните системи можат да се употребат за создавање на софтвер, создавање на интерактивни форми на наратив во видео игри и како ова може да се прошири на останати форми на уметноста како што се литература и филм.

Иако ваков тип на хаотичност може да изгледа како случаен и непредвидлив, во суштина не е таков. На пример, некои аспекти на временските прилики можат да се разгледуваат како случајни и непредвидливи, каде едноставни промени во еден дел од системот можат да предизвикаат комплексни промени на другите места во системот. [11] Оваа нелинеарност е една од причините зошто точни долгочорни предвидувања на временските прилики се невозможни со моменталната технологија, меѓутоа теоретски се возможни. Карактеристиките на нелинеарните системи се недостаток за научните области каде се бара одреден степен на точност, но тие карактеристики се предност во создавање на интерактивни наративи во кои секој избор на читателот може да доведе до различен исход во приказната.

## 2.1 Нелинеарни системи во уметноста

Нелинеарноста е присутна во гранки на уметноста во кои е можна интеракција меѓу човекот и делото.

Во литературата ваквиот тип на дела спаѓаат во жанрот на Книга-игра (анг. Gamebook), но исто така често се користи и терминот Хипертекст фикција за да се опише оваа гранка на литературата. [15]

Во филмската уметност постојат т.н. „интерактивни филмови“ кои имаат различни форми и механизми за постигнување на нелинеарност кои ќе бидат разгледувани во наредните поглавја. Интерактивните филмови се наоѓаат на границата помеѓу филм и видео-игра. [6]

Во природата на видео игрите е да овозможуваат различни избори за играчите. Тие избори можат да бидат скриптирани (вградени во играта со намера и точно детерминиран излез од системот) или нескриптирани (непредвиден исход и недетерминиран излез од системот). [3] Целите во играта можат да бидат строго детерминирани од развиваците на играта или да бидат детерминирани од играчот преку неговите избори во текот на играта. Дистинкција помеѓу овие типови на избори во игрите ќе бидат разгледувани во понатамошниот текст.

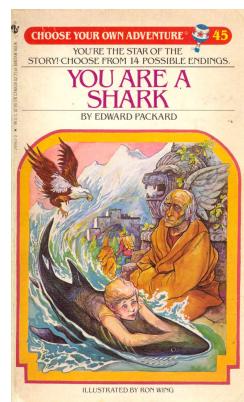
## 2.2 Нелинеарни системи во литературата

Литературните дела кои имаат нелинеарен наратив или интерактивен наратив најчесто го постигнуваат тоа преку внатрешни референци. [10] Внатрешните референции го поттикнуваат читателот да донесе одлука во одреден момент при читање со тоа што му овозможуваат избор да заврти на некое нумерирано поглавје или на специфична страница во книгата. Почетоците на ваковот стил на наратив може да се најдат во книгите на Џејмс Џојс - „Улис“ од 1924 година, Хорхе Луис Боргес - „Градина на разгранувачки патишта“ од 1941 година, Владимир Набоков - „Блед Оган“ од 1962 година, Итало Калвино - „Замокот на Испреплетени Судбини“ од 1973 година и други.

Во поп културата, на крајот од седумдесетите се издаваат книги од жанрот книга-игра (анг. Gamebook) [Book\_RoutledgeEncyclopedia], од кои најпопуларен е серијалот книги „Избери ја својата авантура“ кој содржи 184 продолженија. Кните од серијалот „Избери ја својата авантура“ го следат истиот принцип на „внатрешни референци“ за да постигнат нелинеарност.

Еспен Ј. Аарсет во неговата книга „Сајбертекст - Перспективи на ергодична литература“ го опишува терминот „ергодична литература“ кој доаѓа од грчките зборови ergon што значи „работка“ и hodos што значи „пат“.[1] Тој ја дефинира ергодичната литература како:

„Во ергодичната литература, нетривијален напор е потребен за да му се овозможи на читателот да го прочита текстот. За ергодичната литература да прави смисла како концепт, тогаш мора да постои и неергодична литература, каде напорот потребен за да се прочита текстот е тривијален, без дополнителни одговорности наметнати врз читателот освен, на пример движење на окото и периодично арбитрарно вртење на страниците.“



Слика 1: Корица на една книга од серијалот „Избери ја својата авантура“.

## 2.3 Нелинеарни системи во филмот

Во делата во филмската индустрија во кои гледачот има можност да интерактира и да ја менува содржината додека таа се случува, најчесто се категоризираат како „Интерактивно кино“. [6] Многу често овие дела можат да се категоризираат и како видео игри во зависност од нивото на слобода која гледачот ја има при интеракција со делото.

Најранитеrudimentalни примери за интерактивно кино датираат уште на почетокот на 20-тиот век. Прв успешен пример е

филм-играта „Животни Цели“ (анг. Life Targets) која била објавена во Велика Британија во 1912 година. Популарноста на овој тип на интерактивно кино го доживеал својот врв околу 1915 година, а потоа популарноста започнала да опаѓа.

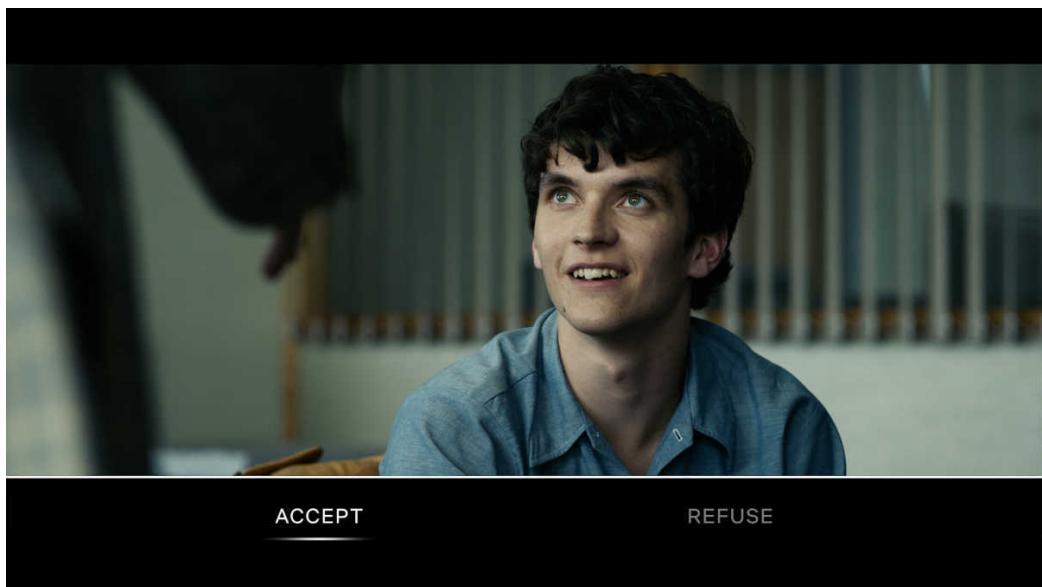
Филмот „Киноаутомат“ на Чехословачкиот режисер Радуз Цинцера се смета за првиот интерактивен филм во категоријата на „Интерактивно кино“ [14]. Во девет точки низ филмот, дејството запира, текот на филмот паузира, еден модератор се појавува на сцената и ја прашува публиката да одбере помеѓу две сцени. Откако публиката ќе гласа за една од сцените, избраната сцена се пушта како следна. Сценариото на овој филм било напишано на начин што двете линии на дејството конвергираат една кон друга кај точката на одлука. Ова значи дека во текот на филмот има само две можни сцени, наместо дупло поголем број после секоја точка на одлука. [8]



Слика 2: Сцена од филмот „Киноаутомат“.

Со развојот на технологијата напредува и начинот на интеракција на публиката со филмот. Во деведесетите години од дваесеттиот век имало обиди во кината низ САД да се вградат уреди за гласање со кои публиката би можела да гласа која би била наредната сцена во филмот. Пример за ваков филм е „Јас сум твојот човек“ (анг. I'm Your Man) издаден во 1992 година. [7]

Најскорешен пример за изведба на интерактивен филм е филмот „Црно Огледало: Бандерснеч“ (анг. Black Mirror: Bandersnatch) на видео платформата Нетфликс кој бил издаден во 2018 година. Во филмот има 150 минути уникатна видео содржина поделена на 250 сегменти кои генерираат 3 трилиони можни патишта низ кои играчот може да помине во текот на своето гледачко искуство. Во некои случаи, до истата сцена може да се стигне на различни начини, но гледачот ќе добие опции врз основа на начинот на кој стигнал до таа сцена. Филмот има 5 можни краеви врз основа на изборите кои гледачот ги носи. [13]



Слика 3: Пример за избор во филмот „Црно Огледало: Бандерснеч“.

## 2.4 Нелинеарни системи во видео игрите

Видео игра која има нелинеарен начин на играње им презентира на играчите предизвици кои можат да бидат комплетирани на различни начини. Секој играч може да избере (или да наиде) само на некои од можните предизвици, а истите предизвици можат да бидат играни и во различен редослед. Во спротивно, видео игра со линеарен начин на играње го конфронтира играчот со фиксен редослед на предизвици, а секој играч се соочува со секој предизвик во ист редослед. [2]

Нелинеарните игри му овозможуваат на играчот поголема слобода на избор од линеарните игри. На пример, нелинеарните игри овозможуваат повеќе редоследи во кои може да се заврши играта, избор помеѓу повеќе разгранети наративи за да стигнат до победа, различни типови на победа, различни избори за странични потраги и различни начини на развојот на содржината. Некои игри имаат и линеарни и нелинеарни елементи, а некои нудат и отворен свет независен од главните цели на играта, ако воопшто играта има предефинирани цели.

Игра која е значително нелинерана се опишува и како игра со отворен крај. Ваквиот тип на игри се карактеризираат со тоа што им овозможуваат на играчот да го измери својот прогрес преку цели кои самиот ги дефинира, независни од препограмираните елементи на играта.

Во игрите кои инкорпорираат линеарни приказни играчот не може да ја смени приказната или крајот на приказната. Повеќето видео игри користат линеарна структура на приказните, па така тие се многу слични и на останати форми на фикција. Меѓутоа, многу типично за такви игри е користењето на интерактивна нарација во која играчот мора да интерактира со нешто пред наративот да продолжи или нелинеарни наративи во кои настаните се портретираат во не-хронолошки редослед. [5]

Игрите кои нудат повеќе краеви го прават тоа за да го зголемат драматичниот ефект и да се нагласи моралноста или неморалноста на изборите на играчот во текот на играта. Некои игри одат многу подалеку од само мал избор или неколку краеви и во својата содржина нудат разгранети линии на наратив кои играчот може да ги контролира во критичните точки во играта. Некогаш играчот ќе има избор меѓу сите достапни разгранивања, но тие разгранивања можат да зависат и од одговорот на играчот на некој предизвик во играта. [2]

Предноста на нелинеарните наративи е тоа што му даваат на играчот чувство на контрола, односно дека свесните одлуки кои ги носи во текот на играта имаат ефект врз исходот на играта.

Недостатокот на нелинеарните наративи во игрите е тоа што потешкото и подолго се развиваат и полесно можат да доведат до апсурди во наративот за разлика од линеарните наративи. Овој труд ќе се обиде

да ги енумерира, објасни и да најде решенија за овие проблеми со цел да се олесни и забрза развојот на ваков тип на наративи.

Во наредниот дел ќе бидат анализирани типични игри со нелинеарен наратив и игри со интерактивни избори кои имаат ефект врз начинот на играње и крајниот исход, а од таму ќе бидат извлечени и потребите при конструирање на еден таков систем.

#### 2.4.1 Crash Bandicoot

Crash Bandicoot е серијал на видео игри кои има повеќе продолженија, иницијално наменети за платформата Плејстејшн (анг. Playstation). Главното дејство во игрите се одвива на фиктивните острови именувани „Вумпа“, архипелаг ситуиран јужно од Австралија каде мутанти и луѓе коегзистираат. Главната механика во игрите е од типот „платформер“ каде играчот типично скока преку т.н. „платформи“ за да стигне до одредена цел. Главниот протагонист се вика Креш (анг. Crash) чиишто тивок живот на островите Вумпа многу често е попречен од главниот антагонист во серијалот, докторот Нео Кортекс (анг. Neo Cortex). Целта на играта е Креш да го порази докторот Нео Кортекс и да ги спречи неговите планови за светска доминација.

Во првата верзија на играта Crash Bandicoot е искористена типично линеарна структура во која Креш ги поминува нивоата на линеарен начин, а некои делови се достапни само со помош на специјални камења од вредност кои можат да се најдат на мапата.

Почнувајќи со верзијата наречена „Кортекс го возвраќа ударот“ (анг. Cortex Strikes Back) играта вообичаено се случува во место наречено „Ворп соба“ (анг. Warp Room), во која нивоата се поделени во множества од пет. Секој сегмент во текот на играта му овозможува на играчот слобода да одбере еден од пет нивоа. Кога играчот ќе ги помине сите 5 нивоа во редослед по негов избор тогаш играта го соочува играчот со „Шефот“ (анг. The Boss) на тој сегмент, кој откако ќе биде поразен играта продолжува во наредниот сегмент.

На овој начин се постигнува најосновното ниво на нелинеарност која му овозможува на играчот да ги минува нивоата во



Слика 4: „Ворг“ собата во Cortex Strikes Back“.

редослед по сопствен избор. Наративот во секој нареден сегмент директно зависи од тоа дали играчот ги има поминато сите претходни нивоа и сегменти, па така самите сегменти се линеарни и се алатка за прогресирање на приказната на линеарен начин без разлика од изборот на играчите кој може да биде по арбитрарен редослед.

#### 2.4.2 Jak and Daxter

Jak and Daxter е серијал на видео игри со повеќе продолженија, слично како и Crash Bandicoot, иницијално наменети за Плејстејшн 2 (анг. Playstation 2). На Jak and Daxter многу често се гледа како една од франшизите која го дефинираат Плејстејшн 2 како најпопуларна конзола од шестата генерација. Главната механика на игрите од овој серијал најчесто се дефинира како платформер и содржи мешавина од акција, тркање со возила и решавање на загатки. Серијалот е сместен во фиктивен универзум кој инкорпорира научна фантазија (анг. science fantasy), стимпанк (анг. steampunk), сајберпанк

(анг. cyberpunk) и мистични елементи.

Играчот во серијалот Jak and Daxter може да истражува мноштво од различни области во отворен свет и може да ги напаѓа непријателите на кратко растојание. Во игрите има супстанца која се нарекува Еко (анг. Eco) која може да биде манипулирана за да му овозможи на играчот некои предности или надградби. Оваа супстанца може да се најде во различни бои расфрлани низ отворениот свет, а Еко со различна боја му овозможува на играчот различна предност или надградба. Слично како супстанцата Еко, низ светот се расфрлани и т.н. „Енергетски ќелии“ (анг. Power Cells) и т.н. „Топчиња на претходниците“ (анг. Precursor orbs) кои му се потребни на играчот за да прогресира низ приказната. Ако играчот комплетира мисии кои се поврзани со приказната или собере Топчиња на претходниците тогаш играта му овозможува да прогресира кон нови локации и му овозможува на играчот да отклучи одредени „Тајни“ во наредните итерации низ светот.

Нелинеарноста во овој серијал се постигнува со тоа што на играчот му се овозможува да избере како и каде ќе се движи низ отворениот свет. Наместо изборот да го прави преку мени или специјална соба како што е случај во поедноставната варијанта на Crash Bandicoot, во случајот на Jak and Daxter изборот се прави со самото движење кон одредена цел во отворениот свет. Jak and Daxter му дава на играчот поголема слобода со тоа што не му наметнува на играчот дека мора да ги помине сите предизвици пред да помине на наредниот сегмент.

#### 2.4.3 The Stanley Parable

The Stanley Parable е видео игра која многу често се класифицира и како „интерактивна драма“ или „симулатор на одење“ (анг. walking simulator). Темите кои се провлекуваат низ играта се поврзани со изборите во видео игрите, врската помеѓу краторот на играта и играчот, предодреденост и судбина.

Играчот го води протагонистот без глас со име Стенли додека неговите дејства во текот на играта се раскажувани од нараторот. Во текот на играта, играчот е конфронтiran со патишта

кои се разграниваат. На секоја точка на разгранивање раскажувачот го наведува играчот на изборот кој е предодреден како „правилен“, но играчот може да избере да не го слуша нараторот и да избере пат кој не е предодреден како „правилен“.

Играчот во играта има перспектива од прво лице и може да се движи и интерактира со некои елементи од околината, како што се на пример притискање копчиња или отворање на врати, но нема механики кои ги карактеризираат игрите од жанрот на акциски игри.

Нараторот ја презентира приказната на играчот. Тој објаснува дека протагонистот со име Стенли е вработен број 427 во некоја деловна зграда. Работата на Стенли е да мониторира податоци на монитор и да притиска копчиња точно, без да поставува прашања. Откако мониторот на Стенли ќе прекине да прикажува податоци, Стенли ќе мора да ја истражи зградата за да открие зошто тоа се случило и зошто зградата е комплетно празна.



Слика 5: Вратите во The Stanley Parable.

Во овој момент приказната се разграничува на бројни можности, врз основа на изборите кои играчот ги носи. Кога играчот доаѓа до област каде изборот е возможен, играчот може да одлучи да не ги следи напатствијата на нараторот и да го направи спротивното дејство од тоа што нараторот го наложува.

Во воведот во играта Стенли ќе биде соочен со две отворени врати низ кои може да помине. Нараторот најавува дека Стенли ќе помине низ левата врата, но ова сè уште се нема случено. Нараторот ги зема предвид одлуките на играчот, реагирајќи со нова нарација ако играчот го следи вистинскиот пат или се обидува да го врати играчот на вистинскиот пат ако играчот оди во контрадикција со она што нараторот го наложува.

Оваа игра има повеќе од 10 различни начини да се заврши во зависност од одлуките што играчот ги носи. Одлуките во играта се носат на физички начин, со тоа што играчот носи одлука да го однесе карактерот кој го управува со движење до одредено место во просторот. При секоја одлука на играчот нараторот има контекст за одлуката што играчот ја донел и реагира соодветно врз основа на тоа што се случило.

## 2.5 Примена на нелинеарни системи во софтвер

Нелинеарните системи наоѓаат примена и во генеричниот софтвер. Најчесто се имплицитно имплементирани во форма на секвенца на настани кои треба да се случат за софтверот да биде доведен во некоја состојба. Оваа секвенца на настани може да биде избрана преку било каква интеракција со софтверот: преку дијалози, паѓачки менија, копчиња итн.

Еден типичен пример за нелинеарен систем кој применува машини на состојби се погледите во смартфон апликациите. Секој поглед претставува една состојба во системот, а интеракцијата со различните елементи на корисничкиот интерфејс прави транзиција помеѓу различните погледи во апликацијата, односно прави транзиција помеѓу различните состојби на системот.

Едно копче може да активира различен поглед на состојбата на системот. На пример, копчето именувано како „Најава“ може да води до најава преку електронска пошта или најава преку некој надворешен систем, во зависност од тоа што корисникот има избрано при регистрација. Секој од овие погледи може да биде комплетно различен и може да нуди различни опции со различни резултати. Овие опции кои се достапни само во одреден поглед се гранките на системот, а системот е нелинеарен бидејќи изборите во рамките на апликацијата

водат кон различни резултати.

### 3 Избори

Избор е множество на работи од кои некој може да избере нешто. Избирањето на нешто може да инкорпорира различни мотиви и модели. На пример, еден патник може да избере да стигне на некоја локација основано врз тоа дали ќе стигне на време, а потоа тој ги вклучува и параметрите како должината на патот, количината на гориво во возилото, густината на сообраќајот итн.

Ако ги анализираме изборите кои ги носат луѓето тогаш тие можеме да ги сместиме во 2 засебни категории: експлицитни и имплицитни избори. Експлицитните избори се оние избори кои оној кој избира ги носи експлицитно, и тие директно зависат од неговиот ментален процес. Имплицитните избори се оние избори кои оној кој избира ги носи имплицитно, односно не учествува активно во реализацијата на изборот, и најчесто не е свесен дека изборот бил донесен се додека не ги согледа резултатите од истиот. Имплицитните избори се комплексни и бараат одредена поставеност на ситуацијата, најчесто се избори да се игнорира и да не се направи нешто.

Примери за експлицитни избори се: избор на храна, избор на универзитет, избор на сопствено возило, избор на рута по која би патувал итн. Во видео игри експлицитните избори се сведуваат на избор на дијалог, избор на начин на кој ќе го поразиш главниот карактер, избор дали ќе влезеш низ левата или десната врата итн.

Примери за имплицитни избори се: разминување на улица со познаник без да се поздравите, пропуштање на театарска претстава заради поклопување на термините со нешто друго, чекање да заврши спортскиот натпревар бидејќи резултатот не може да се смени во преостанатото време итн. Во видео игрите имплицитните избори се најчесто реализирани со тајмери, кои играчот може да избере да почека да истечат. Меѓутоа, имплицитен избор може да биде и изборот да не интерактира со некој карактер контролиран од играта, при што ја губи можноста да открие некоја гранка од приказната.

## **4 Компаративна анализа на нелинеарни системи според начинот на имплементирање**

### **4.1 Секвенцијален метод**

Во секвенцијалниот метод, секој избор е обвиткан во ако-тогаш услови. Ако некој избор е донесен тогаш некој настан ќе се случи и ќе бидат достапни дополнителни избори. Овој метод се покажал како најинтуитивен за користење за програмерите и е генерално прифатениот начин на имплементирање кој се користи во широк опсег на видео игри. Релативно едноставно се имплементира во било кој програмски јазик и може брзо да се стигне до резултати, ако авторот има соодветно техничко познавање.

Проблемите со овој метод се јавуваат кога системот кој се имплементира е длабок и комплексен. Во тој случај обвиткувањата во ако-тогаш услови стануваат премногу комплексни и премногу длабоки, па имплементацијата и одржувањето на вакви системи станува прогресивно потешко. Имплементацијата на ваков начин се потпира на техничкото познавање на тој што го имплементира, па оттаму се извлекува заклучокот дека автори без претходно техничко познавање немаат лесен начин да го користат.

#### **4.1.1 Визуелни графови**

Визуелните графови се надоградба на секвенцијалниот метод кои овозможуваат визуелен приказ на ако-тогаш условите присутни во секвенцијалниот метод. Овие два методи се скоро идентични во начинот на функционирање, разликата е во начинот на кои авторите ја создаваат содржината.

Со секвенцијален метод програмерите пишуваат ако-тогаш услови во програмскиот код, меѓутоа ова може многу едноставно да се прикаже како граф на разграничувања. Предноста што се добива при користење на графовите е во тоа што јазлите во графовите можат

многу лесно да се реискористуваат со едноставно поврзување на соодветните јазли. Во програмски код тоа мора да се направи преку специјализирани функции кои додаваат дополнителна комплексност на имплементацијата.

Визуелните графови се присутни во сите популарни софтвери за пишување на разгранувачки наративи, а се присутни и во софтверите за развој на видео игри. Наоѓаат и специјализирана примена за имплементација на наративи во софтверите за развој на видео игри.

## 4.2 Слободен метод

Слободниот метод користи машини на состојби за претставување на изборите. Со користење на овој метод не се создаваат директни причинско-последични врски меѓу изборите кои се носат, а наместо тоа достапните избори зависат само од состојбата на системот. Со поедноставни зборови, изборот станува достапен само ако сите услови за неговата достапност се исполнети.

Овој начин предвидува создавање на разгранувачки системи базирани на избори без интервенција од автор кој има претходно техничко познавање. Изборите во слободниот метод не зависат директно од претходна низа на настани, туку зависат само од моменталната состојба на системот, па затоа авторите размислуваат дискретно за секоја состојба, наместо да се грижат за причинско-последичната врска која доведува до таа состојба.

Слободниот метод ќе биде предмет на понатамошна анализа и имплементација во продолжение на овој труд.

## 4.3 Споредба на секвенцијалниот и слободниот метод

Проблемот при создавање на разгранувачки системи во уметноста е во неговата предвидливост. Ова е особено видливо при развојот на видео игри бидејќи една од целите на видео игрите е да создадат уникатно искуство за секој играч, па оттука од особена

важност при развојот на наративи е неговата непредвидливост.

Главната разлика меѓу секвенцијалниот и слободниот метод е во начинот на третирање на состојбите на системот. Секвенцијалниот метод е причинско-последичен, а слободниот метод е основан врз состојбата на системот.

Во секвенцијалниот метод секогаш постои причинско-последична врска помеѓу изборите кои се носат и резултатот од нив. Па така, ако авторот одлучи дека некој настан ќе се случи, постои фиксна секвенца на избори кои ќе доведат до тој настан. Ова ја елиминира непредвидливоста бидејќи авторот на наративот директно одлучува за секвенцата на настани кои ќе се случат, без разлика на состојбата на системот.

Во слободниот метод не постои директна причинско-последична врска помеѓу изборите, таа врска е имплицитна. Ако изборите го доведат самиот систем во состојба во која некој избор ќе биде достапен, само тогаш изборот може да биде достапен. Настаните се независни од претходните избори, тие се исклучиво зависни од состојбата на системот. Ова овозможува дискретно разгледување на секоја состојба во системот, што не е случај и не може да биде реализирано со користење на секвенцијалниот метод.

Слободниот метод овозможува поголема слобода во однос на секвенцијалниот метод за авторите без техничко познавање да дизајнираат нелинеарни системи.

## 5 Структурна анализа на нелинеарните системи

Најдоминантниот метод за постигнување на нелинеарност е секвенцијалниот метод што го користат софтвери како Twine, Ink и Chat Mapper, а се користи за постигнување на нелинеарност и во видео игрите кои беа анализирани во претходните поглавја.

Овој труд истражува нов метод за имплементирање на нелинеарни системи кој го решава проблемот на комплексноста при воспоставување на врски меѓу јазлите и го сведува на линеарна комплексност - слободниот метод.

Секој нелинеарен систем кој го користи слободниот метод е составен од две структурни компоненти: состојба и својство. Состојба во слободниот метод дефинира еден момент во текот на системот, а ако состојбата е активна тогаш тоа е моменталната состојба на системот. На состојбите може да се гледа како јазли во насочениот граф кој го дефинира еден систем. Својства се именувани полиња кои можат да содржат нумерички или текстуален податок кој ја дефинира состојбата чијашто компонента е својството.

За разлика од секвенцијалниот метод каде интегрална компонента на системот се и врските меѓу состојбите, во слободниот метод не постојат експлицитни врски меѓу јазлите и ова е примарната предност на слободниот наспроти секвенцијалниот метод. Ова својство на слободниот метод овозможува врските да се дискретни за секоја поединечна состојба, а со тоа им се овозможува на авторите да се грижат само за состојбата на која работат и нејзините зависности, наместо да се грижат за системот како целина и експлицитните врски меѓу јазлите во системот. Ефектите кои произлегуваат од својствата на слободниот метод ќе бидат тема на детално анализирање и дискусија во наредните поглавја од овој труд.

Еден од недостатоците на слободниот метод е неинтуитивност при користење, што може да биде проблем кај автори кои немаат претходна техничка подготвеност. Најголем дел од авторите кои беа консултирани во рамките на истражувањето за овој труд имаат интернализирана перцепција дека секвенцијалниот метод е

единствениот „природен“ метод на кој можат да се создаваат нелинеарните системи. Овој метод го сметаат како „природен“ затоа што на прв поглед создавањето на нелинеарни системи единствено изгледа возможно на секвенцијален начин. Она што останува дополнително да се истражува и да се анализира е дали авторите го имаат истото мислење откако ќе вложат одредено време користејќи го слободниот метод.

## 5.1 Систем

Систем е множество на состојби и својства. Системот може да биде разгледуван како насочен граф иако не е експлицитно дефиниран како таков. Тој има ексклузивна контрола врз спроведувањето на зависностите меѓу една состојба и останатите состојби и меѓу една состојба и својствата. Системот води грижа за сопствениот интегритет и интегритетот на зависностите. Системот е комплетно детерминистички, сите рандомизирани вредности зависат од семето на системот.

## 5.2 Состојби

Состојба е еден момент во системот. Состојбата е пасивна, дискретна компонента на секој систем. Состојбата може да биде активна или неактивна. Во системот во еден момент може да има само една активна состојба. Активната состојба на системот ја претставува моменталната состојба во која се наоѓа комплетниот систем. Еден систем може да има толку состојби колку што овозможува РАМ меморијата. Системот почнува со состојба која е дефинирана како почетна состојба. Секое ресетирање на системот ја враќа моментално активната состојба на почетната состојба.

Состојбата може да претставува различно нешто во различни контексти. На пример, во контекст во кој имаме дијалог помеѓу два карактери, состојбата ќе го претставува напредокот во нивниот разговор, кој соговорник какви информации има добиено од разговорот, какви импликации на иднината има дознавањето на некои информации од разговорот итн.

Без разлика на контекстот во кој се користи состојбата, таа има некои основни својства кои важат во сите контексти во кои се употребува. Својства на состојбата се: име, израз за промена, израз за достапност, дополнителни опции и останати пасивни својства.

Името на состојбата е уникатниот идентификатор за состојбата. Овој уникатен идентификатор се користи за да се идентификува секоја поединечна инстанца на состојбата во рамките на системот. Не смее да постојат две состојби или својства во системот со исто име, а името не смее да содржи специјални карактери, резервирани зборови и не смее да содржи празни места.

Изразот на промена овозможува секоја состојба да има контрола врз вредноста која ја имаат својствата. Во изразот на промена стои специјално дефиниран математички израз кој дефинира каква промена на избраното свойство ќе се изврши во моментот кога состојбата ќе стане активна. Овој израз има посебен начин на дефинирање кој ќе биде понатаму разгледуван.

Изразот за достапност ги дефинира условите кои треба да се исполнети за состојбата да може да биде активирана во рамките на системот. Ако состојбата е достапна тогаш таа може да биде активирана со некакво дејство од играчот, гледачот или читателот. Изразот за достапност е дефиниран преку специјален програмски скрипт (јазик) кој ќе биде дополнително анализиран понатаму.

Дополнителните опции кои можат да се дефинираат на секоја состојба се: дали состојбата може да биде активирана повеќе од еднаш, дали состојбата е ексклузивно достапна, дали состојбата е глобална и дали кога состојбата е активна сите глобални состојби треба да бидат неактивни. Ако состојбата е ексклузивно достапна тогаш кога таа состојба е достапна таа е единствената состојба која е достапна во системот, без разлика на евалуираната достапност на останатите состојби во системот. Ако состојбата е глобална тогаш состојбата е секогаш достапна, без разлика од изразот за достапност. Ако состојбата ги исклучува сите глобални состојби, тогаш кога состојбата е активна ни една од состојбите дефинирани како глобални нема да бидат активни.

Останати пасивни својства на состојбата се: достапност на состојбата, активност на состојбата, дали некогаш состојбата била

активна и дали состојбата некогаш била достапна. Овие својства зависат од моментот во кој се наоѓа системот и системот има комплетна контрола врз нив.

### 5.3 Својства

Својство е податок во системот. Својствата можат да бидат од нумерички или текстуален тип. Тие постојат се додека постои системот и можат да бидат пристапени од било која состојба.

Својствата се идентификуваат по име, кое мора да биде уникатно и да не содржи специјални карактери, резервирани зборови или празни места. Секое својство има вредност и почетна вредност, а нумеричките својства имаат и минимална вредност, максимална вредност, можност за рандомизација, минимална вредност при рандомизација и максимална вредност при рандомизација. Вредноста на својствата може да биде променета при активација на состојба со користење на изразот за промена. Почетната вредност е вредноста која својството ја има кога системот е ресетиран или на почетокот. Вредноста на нумеричкото својство секогаш мора да биде помеѓу минималната и максималната вредност, секое прекорачување на овие граници ќе значи едначење на вредноста со минималната или максималната вредност. Ако рандомизацијата е овозможена тогаш својството при ресетирање на системот ќе добие рандомизирана вредност во границите за рандомизација. Рандомизацијата е детерминистичка, а рандомизираната вредност зависи само од семето на системот.

Својствата можат да се користат како вредност во изразот за достапност и да учесуваат во имплицитните врски на состојбите. Состојбите можат да зависат само од едно или повеќе својства, без да зависат од други состојби.

### 5.4 Врски меѓу состојбите

Врските меѓу состојбите се имплицитни, авторот никогаш не поврзува две состојби во системот експлицитно. Две состојби ги

дефинираме како поврзани од А кон Б ако состојбата Б во својот израз за достапност ја вклучува состојбата А. Во овој случај состојбата А не знае за постоењето на состојбата Б, односно таа не е експлицитно поврзана со Б. На овој начин авторот се грижи само за изразот на достапност на состојбата Б, без да посветува внимание на тоа како состојбата А е поврзана во системот. Состојбата Б се разгледува како дискретна состојба, без разлика на нивото на разграничување и без разлика на патиштата кои потенцијално би воделе до Б.

## 5.5 Изрази за достапност

Изразот за достапност е својство на секоја состојба. Секоја состојба има точно еден израз на достапност кој може или да биде исполнет или да не биде исполнет. Неговато исполнување зависи од начинот на кој врската на останатите состојби и/или својства во системот е дефинирана во самиот израз. Ако изразот за достапност е исполнет тогаш состојбата е достапна (освен ако се дефинирани некои дополнителни опции на состојбата).

Изразот на достапност се пишува со посебен јазик за скриптирање кој овозможува пристап до некои од карактеристиките на состојбите и својствата. Овој јазик за скриптирање има два типови на оператори - унарни и бинарни. Унарните оператори извршуваат операција врз една вредност, а бинарните оператори извршуваат операција меѓу две вредности.

Во контекст на изразот за достапност унарните оператори најчесто овозможуваат пристап до некое својство на состојбите. Вакви оператори се:

HAS\_HAPPENED  
CAN\_HAPPEN  
COULD\_EVER\_HAVE\_HAPPENED  
IS\_HAPPENING

По редослед овие оператори значат „се случило“, „може да се случи“, „можело да се случи во минатото“, „се случува“. Ако пред или после некој од овие оператори се вметне име на некоја состојба тој оператор ќе врати вредност „истина“ или „невистина“. На пример, ако

на состојба со име TestSostojba го додадеме операторот:

```
CAN_HAPPEN (IS_HAPPENING TestSostojba)
```

тогаш изразот за достапност ќе биде исполнет ако состојбата со име TestSostojba е моментално активната состојба во системот.

Исклучок кај унарните оператори е операторот NOT, кој е оператор на негација. Кога овој оператор се наоѓа пред некоја евалуирана вредност тој ќе ја негира вредноста, односно тоа што било „истина“ ќе стане „невистина“ и обратно. За поголема прегледност заградите околу сите унарни оператори се задолжителни. Бинарни оператори во изразот за достапност се операторите за споредба:

OR

```
IS_MORE_THAN  
IS_LESS_THAN  
IS_EQUAL_TO
```

Во превод на македонски јазик овие оператори значат: „или“, „и“, „е повеќе од“, „е помалку од“ и „е еднакво на“. Примери за употреба на бинарни оператори:

```
(Svojstvo1 IS_LESS_THAN 2)
```

Овие оператори можат да се користат и во покомплексни примери:

```
((CAN_HAPPEN Sostojba1) AND  
(NOT((Svojstvo1 IS_EQUAL_TO 1))).
```

Специјален оператор е операторот DICE. Овој оператор генерира случаен број во рамките на зададените параметри. Параметрите на овој оператор се: минимална вредност, максимална вредност и семе. Користењето на семе го прави овој оператор детерминистички и го врзува со семето на комплетниот систем.

Изразот на достапност се евалуира секој пат кога некоја состојба е активирана бидејќи тоа е единствениот момент во кој се случува промена на системот. Овој израз ги дефинира врските меѓу состојбите и својствата во системот. Секоја употреба на состојба или свойство во изразот за достапност дефинира имплицитна врска или зависност во системот.

## 5.6 Изрази за промена

Изразите на промена се својства на секоја состојба. Секој израз на промена е поврзан со некое свойство кое тој израз го менува при активација на состојбата која го содржи. Секоја состојба може да има повеќе од еден израз на промена, но најмногу по еден за секое свойство во системот. Изразите на промена во секоја состојба се подредени според момент на извршување, па така изразот за промена кој се наоѓа прв во листата на изрази на промени секогаш ќе се изврши прв. Ова овозможува повеќе изрази на промени да зависат меѓусебно.

Изразот за промена се пишува во посебен јазик за скриптирање кој ќе биде разгледуван во понатамошниот текст. Секој израз на промена може да содржи некое свойство или повеќе свойства чијашто вредност се евалуира при извршување на изразот. Резултатот на изразот на промена секогаш е ист со типот на својството кое се менува, па така ако својството кое се менува е нумеричко тогаш и резултатот од извршувањето на изразот на промена е нумеричка вредност.

Ако во изразот на промена стои само нумеричка вредност тогаш тоа значи едноставна промена на вредноста, на пример ако изразот за промена содржи само „-2“ тогаш својството на кое се однесува тој израз ќе се промени за -2 во моментот кога состојба ќе биде активирана. Ова е најчест случај на употреба на изразите за промена, па затоа е дизајниран да биде едноставен за користење. Можни се и посложени изрази кои користат специјални оператори.

Операторите кои можат да се користат во изразот за промена се поделени во четири групи: Оператори за начин на промена, оператори за тип на промена, оператори за користење на вредност на својство и оператори со специјална намена.

### 5.6.1 Оператори кои го регулираат начинот на промена

**DELTA** (свойство = свойство + вредност) - основен оператор кој дава исти резултати како и внесување на вредност без оператор (на пример, DELTA -2 е еквивалент на внесување само -2).

**DELTANEГ** (свойство = свойство - вредност) - ја има истата улога како и операторот DELTA, само конечната вредност наместо збир - е разлика.

**SET** (свойство = вредност) - оператор кој овозможува директен внес на одредена вредност наместо нејзина промена.

### 5.6.2 Оператори кои го регулираат типот на промена

**VALUE** (вредност на свойство = вредност) - основен оператор, ако ниеден оператор од групата на оператори за тип на промена не е употребен тогаш се користи овој оператор.

**INITIAL** (почетна вредност на свойство = вредност) - оператор кој ја менува почетната вредност на својството.

**MIN** (минимална вредност на свойство = вредност) - оператор кој ја менува минималната вредност на својството.

**MAX** (максимална вредност на својството = вредност) - оператор кој ја менува максималната вредност на својството.

### 5.6.3 Оператори за користење на вредност на свойство

**GETVALUE** (пр. за користење: (GETVALUE NekoeSvojstvo)) - Основен оператор. Се користи моменталната вредност на некое свойство. Ако операторите од оваа група се изоставени тогаш се користи овој оператор.

**GETMIN** (пр. за користење: (GETMIN NekoeSvojstvo)) - се користи минималната вредност на некое свойство

**GETMAX** (пр. за користење: (GETMAX NekoeSvojstvo)) - се користи максималната вредност на некое свойство

**GETINIT** (пр. за користење: (GETINIT NekoeSvojstvo)) - се користи почетната вредност на некое свойство

#### 5.6.4 Оператори со специјална намена

**DICE** (пр. за користење SET VALUE DICE 1 6) - оператор кој генерира случајна вредност без да го користи семето на системот. Оваа вредност е комплетно случајна и различна е секој пат кога изразот ќе биде евалуиран. На операторот му се потребни минимална и максимална вредност меѓу кои се генерира случајниот број.

**DICEX** (пр. за користење SET VALUE DICEX 1 6 33) - слично како и операторот DICE, овој оператор е оператор кој генерира случајна вредност со користење на семето на системот, а дополнително може да биде специфицирано и специфично семе при евалуирање на овај специјален оператор. Исто како и операторот DICE, кај овој оператор мора да бидат специфицирани минималната и максималната вредност меѓу кои генерираните случаен број мора да се наоѓа.

### 5.7 Активација на состојба

Активација на состојба е моментот кога една состојба во системот преминува од неактивна во активна. Во понатамошниот текст ќе ги анализираме сите начини на активација на состојби кои се дефинирани и што се случува при активација на состојбата. За една состојба да може да биде активирана нејзиниот израз за достапност мора да е исполнет, односно состојбата да е достапна. Ако состојбата се активира на основниот начин и состојбата е достапна тогаш таа може да биде активирана. Исклучок кон ова правило е ако состојбата се активира на т.н. „присилен начин“ во кој случај таа не мора да биде достапна за да биде активирана, меѓутоа ова може да доведе до повреда на интегритетот на системот. Исклучок може да биде и активација во специјални случаи како што е вчитување на системот од меморија, активирање на почетната состојба при ресет на системот и останати специјални случаи кои ќе бидат понатаму разгледувани.

Во процесот на активација на состојбата треба да се случат три основни работи, редоследот е важен:

1. Состојбата да се обележи како активна и да се проследи оваа

информација во системот кој води евиденција за моментално активната состојба, претходно активираните состојби и достапноста на секоја состојба.

2. Да бидат евалуирани сите изрази за промена и да се променат својствата соодветно на резултатот од извршувањето на овие изрази
3. Да бидат евалуирани сите изрази за достапност кои се кандидати за евалуација. Кандидати за евалуација се сите состојби кои ја содржат состојбата која е цел на активација, кои содржат својства кои состојбата која е цел на активација ги менува и состојби кои се означени како глобални ако состојбата не е означена како состојба која ги изоставува глобалните состојби.

## 5.8 Гранки

Гранките се компонента на секој нелинеарен систем кои овозможуваат контрола врз специфичен пат на разгранување на јазлите (состојбите) во рамките на системот. Секоја гранка е секвенцијална листа на активација на состојби. Гранката се генерира со редоследно запишување на состојбите кои се активираат во рамките на системот. Системот може да има бесконечно многу гранки. Секоја гранка е дефинирана со нејзиното име кое мора да биде уникатно и листата од состојби кои се редоследно запишани.

Гранките им овозможуваат на авторите на системот да тестираат специфичен пат на активација на состојбите кој е именуван и може да се отвори во било момент. Кога некоја гранка во системот е активна тогаш таа гранка може да биде навигирана нанапред и наназад во времето. Системот се грижи за интегритетот на гранките во моментот кога тие се навигирани нанапред и наназад. Бидејќи системот е комплетно детерминистички тоа значи дека било кој момент дефиниран во гранката ќе ги даде истите резултати. Оттаму, својствата во било кој момент ќе имаат еднакви вредности без разлика на итерацијата, а тоа имплицира дека и пасивните својства и достапноста на состојбите во системот ќе бидат исти во ист момент од времето иако итерацијата е различна.

Гранките се дел од системот и нивната состојба може да биде

зачувана во рамките на системот. Секоја гранка има активна состојба која е почетна состојба на системот кога гранката ќе биде вчитана.

## 6 Тјуринг комплетност на системот

За да се покаже Тјуринг комплетноста на еден систем доволно е да се покаже дека системот може да симулира некој Тјуринг-комплетен систем. На пример, некој императивен јазик е Тјуринг-комплетен ако има:

1. условно разгранивање
2. арбитрарна количина на меморија на располагање (може да се користи целосно расположливата РАМ меморија)

Имајќи ги предвид овие два услови за Тјуринг-комплетност може да се каже дека нелинеарниот систем обработуван во овој труд е Тјуринг-комплетен.

Во системот кој е цел на анализа во овој труд условното разгранивање не е експлицитно дефинирано како во императивните јазици бидејќи не постојат експлицитни „ако-тогаш“ (анг. if-else) и „оди-до“ (анг. goto) изрази. Ако императивните јазици ги разгледуваме како насочен граф составен од меѓусебно насочено поврзани јазли, тогаш и системот разгледуван во овој труд можеме да го сведиме на ист таков насочен граф со меѓусебно поврзани јазли кои претходно ги нарековме состојби на системот. Секој поединечен јазол во графот (состојба) може да биде разгранет и да биде поврзан со било кој друг јазол (состојба), а тоа значи дека условното разгранивање на системот е комплетно.

Својствата на системот овозможуваат користење на РАМ меморијата на системот. Системот може да има онолку својства колку што има РАМ меморија на располагање, па од таму и вториот услов за Тјуринг-комплетност е исполнет.

Понатамошната анализа на Тјуринг-комплетноста на системот и утврдување на типот е надвор од опсегот на овој труд.

## 7 Имплементација

Една од целите на овој труд е имплементација на софтвер за управување со нелинеарни системи. Софтверот овозможува пишување на нелинеарен текст кој може да биде во форма на проза, поезија или други литературни форми. Текстот како форма е избран затоа што е најлесен за демонстрација на сите аспекти на системот кои подоцна можат да се интегрираат и на друг начин со други мултимедијални системи. Секоја приказна или текст е еден систем и овие поими се еквивалентни, а во понатамошниот текст ќе бидат користени наизменично. За постигнување на оваа цел ќе биде искористен програмскиот јазик C++ на оперативниот систем Windows. Има мноштво причини за овој избор, а во продолжение ќе бидат анализирани неколку.

Програмскиот јазик C++ е многу побрз и поефикасен во извршување на огромна количина на инструкции во краток временски период од останатите програмски јазици во кои би можел да се развие ваков систем. Нелинеарните системи имплементирани на методот кој се анализира во овој труд потенцијално би можеле да имаат милиони јазли поврзани со милиони врски за чијашто анализа единствено е погоден јазикот C++ бидејќи овозможува соодветен баланс помеѓу перформанси и комплексност на изведба.

Уште една причина за избор на овој јазик е тоа што C++ е јазикот на позначајните софтвери за развивање на видео игри (анг. game engines) каде овој систем потенцијално би можел да биде интегриран. Јадрото на моторот Unity е напишано во C++ иако главен јазик кој се користи за развој во Unity е C#. Unreal Engine 4, Unreal Engine 5 и CryEngine се напишани во C++ и видео игрите во овие мотори се развиваат во јазикот C++.

C++ лесно се структуира во C API, кое може да биде извезено во DLL датотека. Ова DLL датотека подоцна може да биде вчитана во софтвер од трети страни и да биде искористена за интегрирање на системот во тој софтвер.

Од аспект на архитектура на софтверот, тој е поделен на 5 програмски целини:

**Engine (мотор)** Оваа програмска целина е јадрото на нелинеарниот систем каде се имплементирани сите аспекти дискутирани во претходните поглавја.

**Game (игра)** Во оваа програмска целина се сместени нивоа на апстракција над моторот на нелинеарниот систем. Тука се сместени сите елементи кои овозможуваат моторот да се трансформира во мотор за пишување на нелинеарен текст. На сличен начин може да се имплементираат и други програмски целини кои овозможуваат управување и со поинакви мултимедијални содржини како интерактивни филмови, 3д видео игри и други содржини.

**Editor (уредувач)** Програмска целина која им овозможува на авторите уредување на нелинеарниот систем и сите негови аспекти. Оваа целина детално ќе биде разгледувана во наредните поглавја.

**Reader (читач)** Читачот им овозможува на читателите интеракција со нелинеарниот систем со цел прогресирање на приказната. Читачот не овозможува уредување на содржината.

**API (апликациски интерфејс)** Апликацискиот интерфејс нуди можност за интеграција со софтвери од трети страни како што се софтвери за развој на видео игри.

**Generator (генератор)** Програмска целина која податоците од системот ги претвора во HTML формат кој подоцна се испртува во читачот.

Во контекст на оперативниот систем Windows овие програмски целини се дефинирани како:

**Engine** - динамички поврзана библиотека (DLL)

**Game** - динамички поврзана библиотека (DLL)

**Editor** - извршна програма (EXE)

**Reader** - извршна програма (EXE)

**API** - динамички поврзана библиотека (DLL)

## **Generator** - динамички поврзана библиотека (DLL)

За реализација на софтверот се искористени и надворешни зависности:

**Библиотеката ImGui** дел од целината Editor која овозможува цртање на визуелни контроли за управување со податоците. ImGui е библиотека која работи со непосреден режим на цртање на контролите на корисничкиот интерфејс (анг. immediate mode) како спротивност на задржаниот режим на цртање на контролите на корисничкиот интерфејс (анг. retained mode). Предноста на непосредниот режим наспроти задржаниот режим е во тоа што овозможува управување со податоци во корисничкиот интерфејс со метод без состојби (анг. stateless method). Ова овозможува лесно управување со огромните количини на податоци на екранот без визуелна десинхронизација која е основна потреба на нелинеарните системи.

**Библиотеката WPF** е генерична библиотека за развој на кориснички интерфејси. Меѓудругото, овозможува цртање и управување на модерен HTML код<sup>1</sup> кој се користи во читачот за приказ на податоците. Библиотеката е специфично наменета за програмскиот јазик C#, а врската меѓу C++ и C# се прави на ниво на C API кое се вчитува преку динамички поврзана библиотека (DLL).

Комбинацијата на јазиците HTML и CSS за приказ на читачот е избрана затоа што огромен процент од дизајнерите веќе имаат некакво искуство во овие јазици и за нив е едноставно брзо да се прилагодат за работа на читачот.

**Библиотеката base64** се користи за енкодирање на бинарни ресурси во base64 и декодирање на текстуални нишки од base64. Base64 енкодингот се користи за брз приказ на мултимедијални содржини во HTML.

**Библиотеката tinyXML** се користи за вчитување и зачувување на XML програмски структури.

---

<sup>1</sup>Кога се употребува фразата HTML код, во суштина се мисли и на сите компоненти кои можат да бидат интегрирани во HTML код како што се CSS и JavaScript.

За секоја структурна компонента на системот е дефинирана една C++ класа. Од најголем интерес на анализа е класата на состојбите во нелинеарниот систем бидејќи за понатамошна имплементација потребно е да се креираат специфични класи за специфичните состојби. Примери за специфична состојба се: „карактерот во видео игра допира копче“, „карактерот во видео игра влегува во дефиниран простор“, „публиката на интерактивниот филм одбира сцена“, „страница во роман“ итн. За имплементирање на специфична состојба потребно е основната класа на системот да биде наследена. Состојбите од системот ја следат шемата на „фабрика“ (анг. factory pattern), па така инстанците од било која класа на состојби се креираат со повик до фабриката на состојби. Фабриката на состојби се грижи класата да биде регистрирана во системот и да може динамички да биде создадена. При серијализација, системот преку оваа шема динамички го претвора името на класата во нишка од карактери, за потоа при десеријализација да може да биде креирана класа од соодветниот тип дефинирана преку нишка од карактери.

Секоја наследена класа од основната класа на состојби мора да го декларира својот тип со користење на дефинирано препроцесорско макро. Постојат мноштво на функции кои можат да бидат преоптоварени (анг. override) од основната класа за состојби со кои на наследената класа ѝ се овозможува контрола врз сите процеси на инстанците на состојбите.

Минималните побарувања на конечната имплементација на уредувачот на нелинеарни приказни се:

- Intel Celeron G1000 процесор или еквивалент
- 11 мегабајти РАМ меморија
- Intel HD Graphics
- DirectX 11 со Shader Model 5.0
- Windows XP или понова верзија
- 60 мегабајти слободен простор на диск

Овие побарувања се резултат на избраните технологии при имплементирање на уредувачот и затоа тој лесно може да биде премостен на друга платформа, па дури и такви платформи кои немаат високи перформанси, но поддржуваат извршување на библиотеката на моторот.

Минималните побарувања на конечната имплементација на читачот зависат од имплементацијата на библиотеката Ultralight и нејзините побарувања.

## 7.1 Евалуација на израз за достапност

Евалуацијата на изразот за достапност се прави во неколку чекори и има неколку оптимизации кои ќе бидат анализирани во понатамошниот текст. Првиот чекор е лексичката анализа на изразот која структурно ќе го подготви изразот за наредниот чекор. Во првиот чекор може да се смета дека изразот е во префикс нотација како што наведува и стандардот за скриптирање на овој израз. Вториот чекор е изразот да биде доведен од префикс нотација во постфикс нотација.

Приоритетот на операторот се дефинира преку статичка листа во која редоследот на операторите го дава нивниот приоритет, па така операторот кој е прв во листата има највисок приоритет, а операторот кој е последен во листата има најнизок приоритет. Операторите подредени по приоритет се:

```
(  
OR  
AND  
NOT  
HAS_HAPPENED  
CAN_HAPPEN  
COULD_EVER_HAVE_HAPPENED  
IS_HAPPENING  
IS_MORE_THAN  
IS_LESS_THAN  
IS_EQUAL_TO  
)
```

Во следниот чекор постфикс нотацијата се евалуира, меѓутоа

крајниот резултат од евалуацијата на постфикс нотацијата е кеш на покажувачи кон функција (анг. function pointer cache) наместо нејзината вистинитост. Овој чекор во понатамошниот текст ќе биде рефериран како компајлирање на изразот.

Компајлирањето на изразот овозможува лексичката анализа и евалуацијата на постфикс изразот да се изведе само кога изразот е променет, наместо да се изведува секој пат кога изразот за достапност треба да биде евалуиран. Лексичката анализа и евалуацијата на постфикс изразот во комбинација се спори операции, па од таму извира потребата за нивна оптимизација. Изразите за достапност постојат на секоја состојба така што со секоја наредна создадена состојба перформансите при евалуација опаѓаат експоненцијално. Оптимизацијата на евалуацијата на изразите за достапност е есенцијална за перформансите на целиот мотор за нелинеарни системи.

Откако ќе биде создаден кешот на покажувачи кон функции при секој повик за евалуација на изразот се итерира низ листата која го содржи кешот и секвенцијално се повикуваат сите функции кои ги содржи. Со ова времето на извршување и на најкомплексните изрази со повеќе од 1000 лексички компоненти се сведува под 0.5 милисекунди по израз на модерни системи. Кешот на покажувачи кон функција е дефиниран како тип

```
std::unordered_map<std::string, std::queue<  
    std::function<const void* const()>>>
```

Дефиницијата `std::unordered_map` креира неподредена хеш мапа меѓу нишката од карактери на изразот (сите карактери од изразот) и кешот на покажувачи кон функција. Дефиницијата `std::queue` овозможува функциите да се извршуваат по принципот FIFO, односно првата функција која постфикс евалуацијата ќе ја најде ќе биде и првата функција од кешот која ќе се изврши. Дефиницијата `std::function<const void* const()>` дефинира дека функцијата може да има било каков тип се додека типот е константен покажувач кон константна вредност. Ова овозможува да се дефинираат функции со тип `const bool* const` или функции со тип `const float* const` или функции со некој генеричен тип на покажувач.

Во изразите за достапност може да се вметнат и константни броеви кои евалуаторот мора да ги чува во посебен кеш. Тој кеш е

дефиниран со тип `std::unordered_map<std::string, std::queue<float>>*`, а тоа значи дека првиот број кој ќе биде пронајден при евалуација на постфикс изразот ќе биде првиот број со кој ќе биде евалуиран некој израз кога ќе се повика покажувачот кон функцијата која бара константен број.

Резултатот од евалуацијата на изразот на достапност секогаш е вистина или невистина.

## 7.2 Евалуација на израз за промена

Евалуацијата на изразот за промена се прави секвенцијално, односно секој оператор се евалуира во линијата во која се наоѓа (анг. *inline evaluation*). Овој израз е едноставен и бара просто заменување на параметри во форма на макро, па затоа евалуацијата може да се одвива секвенцијално со цел избегнување на непотребна комплексност. Овие изрази не се компајлираат бидејќи замената на параметрите е доволно брза за да не мора да се заменуваат со покажувачи кон функции.

Резултатот од евалуацијата на изразот за промена во оваа имплементација е нумеричка вредност.

## 7.3 Структура на папки

Во основната папка се наоѓаат сите EXE и DLL датотеки кои се потребни за функционирање на уредувачот и читачот. Овие датотеки се: `NSTGame.dll`, `NSTEEngine.dll`, `NSTEeditor.exe`, `NSTReader.exe` и сите зависности од надворешните библиотеки. Датотеките кои не се во оваа група на датотеки се генериирани од уредувачот при неговото извршување. Структурата на папки се состои од папките:

**backup** папка во која се чуваат сите резевни копии кои уредувачот ги создава во текот на работењето.

**exports** папка во која се чуваат сите датотеки кои се увезуваат или извезуваат од уредувачот.

**lang** папка во која се чуваат датотеките за јазик на уредувачот.

**stories** папка во која се чуваат „компајлираните“ датотеки во некои некоја верзија на NST форматот.

Приказните можат да бидат вчитани или увезени во уредувачот од било која локација на диск, но уредувачот води грижа само за организацијата на датотеките кои се наоѓаат во папките кои тој ги препознава.

## 7.4 Вчитување на ресурси

Во системот можат да се вчитаат мултимедијални ресурси од различен тип. Откако ресурсите ќе се вчитаат тие можат да се користат во прегледот на состојбите. Начинот на интегрирање во состојбите ќе биде разгледуван во следните поглавја. Секој ресурс има сопствено име преку кое може да биде референциран во состојбата. Кога некоја состојба се прикажува во читачот, вметнатите ресурси се прикажуваат во нивната изворна форма како текст, слика, видео или аудио.

Вчитаните ресурси можат да бидат од следните миме-дефинирани типови: "apng", "avif", "gif", "jpg", "jpeg", "jfif", "rjpeg" "pjp", "png", "svg", "webp", "wav", "wave", "webm", "ogg", "css", "csv", "webm", "js", "json", "mp3", "mpeg", "oga", "ogv", "otf", "ttf", "txt", "xml", "mp4".

При вчитување на ресурсите тие се енкодираат во base64 формат кој може многу едноставно да се вметне во HTML код кој го користи читачот. Вчитувањето и енкодирањето на секој ресурс се одвива во засебна процесорска нишка.

## 7.5 Дијалог состојби

За имплементација на нелинеарен текст во системот е имплементирана наследена класа на основната состојба наречена дијалог состојба. Дијалог состојбата дефинира дополнителни параметри над основните параметри на состојбата. Овие параметри ја дефинираат текстуалната ситуација која ја претставува инстанцата на

состојбата, исказите на состојбата кои служат како врски до останати состојби, специјално поле за стилот на состојбата кога таа сocoјба се прикажува во читачот и обележани својства.

Ситуацијата на дијалог состојбата е во форма на текст кој може да содржи изрази со специјална намена. Овој текст се црта во читачот секогаш кога е активна состојбата за која таа ситуација е врзана. Изразите за специјална намена стојат или помеѓу средни или помеѓу големи загради. Ако изразот е помеѓу средни загради тогаш при цртање на текстот во читачот, средните загради заедно со изразот се заменуваат со својството кое изразот го претставува. На пример, ако меѓу текстот се најде израз „[Svojstvo1]“ тогаш комплетниот израз ќе биде заменет со моменталната вредност на Svojstvo1. Ако изразот е помеѓу големи загради во формат „resource=Resurs1“, тогаш комплетниот израз се заменува со base64 вредноста за претходно вчитаниот ресурс.

Изјавите на дијалог се специјални параметри кои ги имаат дијалог состојбите. Преку изјавите на дијалог читателот избира одлука. Тие се наоѓаат во читачот во форма на копчиња на чиишто клик читателот носи одлука и избира пат по кој ќе продолжи понатаму. Состојбите дефинирани како дијалог состојби се основните компоненти во нелинеарниот текст.

Обележаните својства се својства кои постојат во системот но имаат име кое е читливо, наспроти името кое системот го користи за референцирање на својството. Името може да содржи било кој карактер поддржан од HTML. Обележаните својства секогаш се прикажуваат во читачот. Редоследот на обележаните својства е важен бидејќи тие редоследно се прикажуваат во читачот. Секое својство во системот може да има само едно обележано својство односно една етикета.

## 8 Уредувач на нелинеарни приказни

Уредувачот на нелинеарни приказни е софтвер кој овозможува креирање, уредување, управување, тестирање, серијализација и десеријализација на нелинеарни интерактивни форми на текст. Имплементиран е во јазикот C++, исто како и останатите програмски целини. Корисничкиот интерфејс е имплементиран со библиотеката ImGui, меѓутоа тој нема да биде анализиран во понатамошниот текст. Фокусот на дискусијата ќе биде насочен кон управување на нелинеарните системи со помош на корисничкиот интерфејс.

Постои цврсто одделување на логичките сегменти во програмски целини, па така уредувачот на нелинеарните приказни има улога исклучиво како интерфејс меѓу корисникот и моторот за нелинеарни системи. Од уредувачот на нелинеарни приказни единствено се практикаат инструкции кон останатите програмски целини и се прикажува одговорот на инструкциите.

Постојат 3 различни типови на датотеки кои може уредувачот на нелинеарни приказни да манипулира:

- NSS
- XML
- NTX

Основниот тип на датотека е бинарна, компајлирана, криптирана датотека со екstenзија NSS. Овој тип на датотеки ги дефинира NSS стандардот кој е многу сличен на XML форматот. Овие датотеки имаат минимална големина, најбрзо се вчитуваат во програмска структура и се зачувуваат во надворешна датотека. При зачувување оваа датотека се криптира со константно дефиниран клуч како претпоследен чекор пред запишувањето на диск. При вчитување датотеката се декриптира со истиот константно дефиниран клуч по што може да се чита во програмските структури. Овој тип на датотека е интегриран во моторот за нелинеарни системи и може да се користи без разлика од типот на состојби кои се регистрирани во системот.



Слика 6: Дел од интерфејсот на уредувачот на нелинеарни приказни

Друг тип на поддржани датотеки се XML датотеките. Тие се мемориски потешки и поспори при вчитување и зачувување во споредба со NSS датотеките. Овие датотеки не се криптирани и не се бинарни, односно можат да бидат уредувани со текстуален уредувач или да бидат споделувани преку некој систем за контрола на верзии.

Последниот тип на датотеки кој е поддржан од уредувачот на нелинеарни приказни е форматот NTX кој единствено може да биде увезен, но не и да биде изведен. Оваа лимитација е резултат на самиот формат кој им овозможува на авторите комплетниот систем да го создадат во уредувач на текст по избор. Во понатамошниот текст ќе биде анализиран стандардот NTX.

Секоја NTX датотека започнува со дефинирање на својствата на системот. Секое свойство се дефинира во формат "[име на свойство]: вредност минимална максимална" (пр. [Svojstvo1] 0 -1 1). Секој сегмент во NTX датотеката е одделен со форматот "====" во празен ред. Откако својствата ќе бидат дефинирани тие се одделуваат од состојбите со "====" во празен ред. Потоа во наредниот празен ред се дефинира името на состојбата во средни загради (пр. [Sostojba1]). Откако состојбата ќе биде дефинирана во наредните редови се дефинира ситуацијата на состојбата во текстуален формат, а секоја

дефиниција на ситуацијата мора да заврши со "—" во празен ред. Конечно, по дефиницијата на "—" се испишуваат функционалните зависности, секоја во нов празен ред. Секоја функционална зависност е дефинирана со идентификатор кој стои во средни загради, проследен со ":" и дефиницијата на зависноста (пр. [Sostojba2]: Ova vodi kon sostojba 2). Функционалните зависности можат да бидат состојби или својства. Исклучок е операторот "#" преку кој може да се вметне израз за достапност за состојбата (пр. [#]: (HAS\_HAPPENED Sostojba1) AND (HAS\_HAPPENED Sostojba3)) Уредувачот на нелинеарни приказни овозможува:

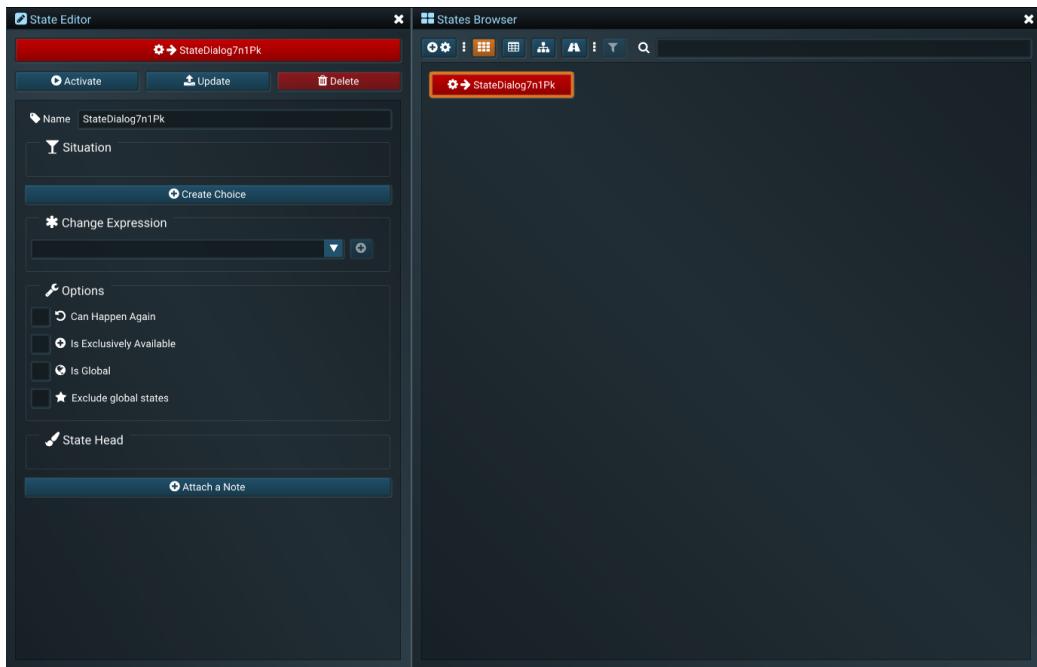
- Ефикасна манипулација со датотеки, вчитување и ревчитување, зачувување, извезување во специјални формати како што е XML форматот, отворање на скрешни датотеки, увезување од XML и NTX форматите.
- Управување со резервни верзии на датотеки и фреквенција на зачувување на резервни копии.
- Управување со контекстот на датотеките, управување со структурата на диск.
- Историја на извршени команди.
- Создавање, читање, управување и бришење на состојби и својства.
- Управување со системот, ресетирање на комплетниот систем или ресетирање на активната гранка.
- Паузирање на системот.
- Извршување на читачот и синхронизација на системот во уредувачот и во читачот.
- Избор на јазик.
- Избор на тема.
- Размена на приказни создадени во уредувачот преку платформата Steamworks.
- Алатки за управување и уредување на системот во кои се вбројуваат уредувачот на состојби, уредувачот на својства, прелистувачот на состојби, читачот вграден во уредувачот на

нелинеарни приказни, прелистувачот на содржини и управувачот на времето.

- Во специјалните алатки на уредувачот на нелинеарни приказни се вбројуваат дневникот, прегледот на перформанси и историјата на команди. Овие алатки овозможуваат пронаоѓање на проблеми во уредувачот и воглавно се насочени кон него.

## 8.1 Уредувач на дијалог состојби

Уредувачот на дијалог состојби (анг. State Editor) е одделен прозорец кој се наоѓа во рамките на уредувачот на нелинеарни приказни. Прозорецот на уредувачот на состојби може да биде прикажан со кликање на субменито „Уредувач на состојби“ во менито „Прозорец“.

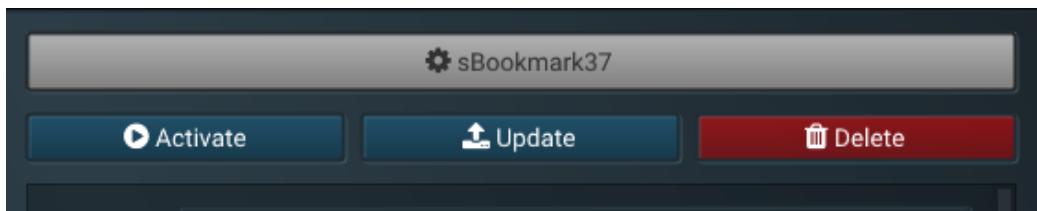


Слика 7: Уредувачот на дијалог состојби заедно со прелистувачот на состојби.

Уредувачот на дијалог состојби има два режими на приказ. Едниот режим е едноставен, а другиот е напреден. Нелинеарни приказни можат да се создадат само користејќи го едноставниот поглед, меѓутоа сите напредни опции кои можат да се користат се содржани во напредниот поглед. Режимот на работа може да се промени во прозорецот „Поставки“. Во натамошниот текст овој прозорец ќе биде дискутиран во напредниот режим на работа кој ги содржи сите опции.

Првиот елемент во уредувачот на дијалог состојби е елементот на самата состојба. Елементот на состојбата е копче кое кога ќе биде кликнато ги отвора деталите на состојбата во уредувачот на состојби. Ова копче има различна боја врз основа на параметрите на

состојбата. Ако состојбата е моментално активната состојба тогаш копчето има црвена позадина, ако состојбата е достапна тогаш копчето има зелена боја, ако состојбата е селектирана и прикажана во уредувачот на дијалог состојби тогаш копчето има портокалова рамка, а ако состојбата некогаш била активирана тогаш бојата на копчето е темно црвена. Секоја поединечна состојба во сите уредувачи е дефинирана со елементот на состојбата.



Слика 8: Копчиња за акции на состојбата.

Со десен клик на елементите се отвора контекстуално мени кое нуди неколку опции кои се однесуваат на состојбата. Опциите кои можат да се одберат со лев клик на глушецот се:

**Уреди** - ја отвора состојбата во уредувачот на состојби и овозможува детален преглед на параметрите на состојбата.

**Ресетирај** - со клик на ова субмени состојбата се доведува до основните вредности на параметрите. Ова дејство може да предизвика нарушување на интегритетот на системот бидејќи параметрите на состојбата се местат назад на основните вредности без да се внимава на интегритетот на системот.

**Пробај активација** - системот ќе се обиде да ја активира состојбата. Ако состојбата е достапна тогаш активацијата ќе биде успешна, во спротивно активацијата нема да биде успешна.

**Присили активација** - активацијата ќе биде присилна и без разлика на параметрите состојбата ќе биде активирана.

**Намести почетна состојба овде** - почетната состојба ќе биде наместена на состојбата во чијшто контекст е отворено субмениито.

**Дуплирај** - состојбата ќе биде дуплицирана со длабоко копирање (анг. deep copy).

**Копирај** - овозможува копирање на името на состојбата, изразот за достапност, ситуацијата или главата на состојбата.

**Избриши** - состојбата ќе биде избришана од системот.

**Создај избор** - создава нова состојба и автоматски ја поврзува новата состојба со состојбата од каде потекнува контекстуалното мени каде било притиснато ова копче.

**Зависи од** - ги прикажува сите состојби од кои оваа состојба зависи.

**Зависност кон** - ги прикажува сите состојби кои зависат од оваа состојба.

**Својства кои се менуваат** - ги прикажува сите својства на системот кои оваа состојба ги менува на некаков начин преку изразот за промена.

**Белешки** - ги прикажува сите белешки кои се прикачени на состојбата од каде потекнува ова контекстуално мени и овозможува нивно уредување.

Во наредниот сегмент од овој прозорец се наоѓаат копчиња кои се најчесто користени: Активирај (со клик присилно се активира состојбата), Ажурирај (со клик се ажурира состојбата со променетите параметри) и Избриши (со клик се бриши состојбата од системот).

Во полето „Име“ кое се наоѓа во наредниот сегмент се впишува името на состојбата. Името мора да ги почитува ограничувањата дефинирани во поглавјето „Состојби“.

Во полето „Ситуација“ се вметнува ситуацијата на дијалог состојбата. Таа може да биде уредена со еден од уредувачите на текст кои се веќе поставени во прозорецот „Поставки“. За секој од овие уредувачи постои копче. Со клик на копчето уредувачот стартува дете-процес (анг. child process) во рамките на оперативниот систем Windows. Ова дете-процес е интегрирано во визуелниот изглед и во функционалноста на уредувачот на нелинеарни приказни. Позицијата на дете-процесот на еcranот е цврсто врзана со позицијата на интерниот систем за прозорци на уредувачот на нелинеарни приказни. При секој почеток на уредување на состојбата се создава привремена текстуална датотека која се отвора во една од избраните програми.

Оваа датотека континуирано се мониторира од уредувачот на нелинеарни приказни за промена во времето на уредување, параметар кој го има секоја датотека управувана од оперативниот систем Windows. Откако авторот ќе заврши со уредување на датотеката и ќе го притисне копчето за зачувување во избраната програма за уредување на текст, уредувачот на нелинеарни приказни препознава дека датотеката била променета и вредноста од таа привремена датотека ја читува во полето „Ситуација“. Откако оваа промена ќе биде направена може веднаш да се прегледа во полето „Преглед“ кое се наоѓа веднаш под копчињата. Овој принцип во понатамошниот текст ќе биде рефериран како „принципот за уредување на датотеки во надворешни уредувачи на текст“.

Под полето „Ситуација“ е копчето именувано како „Создај избор“. Со клик на ова копче се создава нова состојба и автоматски се поврзува новосоздадената состојба со состојбата од каде било притиснато ова копче. Брската може да се види во полето именувано како „Израз за достапност“ на состојбата каде било притиснато ова копче.

Во полето „Израз за промена“ се внесуваат сите изрази за промена кои треба да бидат евалуирани кога состојбата која е во контекстот ќе биде активирана. Секој израз за промена мора да содржи својство од системот кое ќе биде променето со резултатот од изразот за промена. Својствата можат да се одберат од паѓачкиот елемент кој содржи листа на својства која може да биде филтрирана. Откако соодветното свойство кое треба да се промени е избрано, со клик на копчето „+“ на десната страна се додава изразот за промена во листата за избраното свойство. Изразот е променет кога авторот ќе кликне на копчето „Ажурирај“.

Секое свойство во листата има сопствен елемент во форма на копче. Лев клик на копчето ќе ги обележи преку анимација сите состојби кои тоа свойство го имаат во својот израз за достапност. Со десен клик на копчето за својството се отвора субмени кое овозможува уредување на сите параметри на својството. Секоја промена завршува со клик на „Ажурирај“ за промените да бидат пренесени во системот. Клик на „Означи зависни состојби“ е еквивалент на лев клик на самото копче на својството.

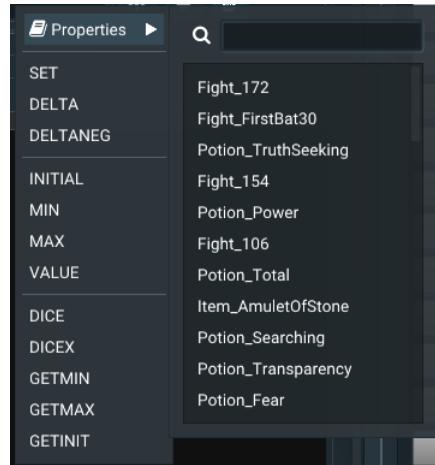
Десен клик на полето за уредување на изразот за промена

отвора мени кое нуди контекстуална помош при пишување на изразот за промена. Во субменито „Својства“ можат да се најдат сите својства кои се достапни во системот и тие можат да бидат филтрирани преку полето за филтрација. Со клик на некое од својствата тоа својство ќе се впише во изразот. Останатите опции во ова мени, слично како и за својствата, се однесуваат на автопополнувањето на некој од операторите.

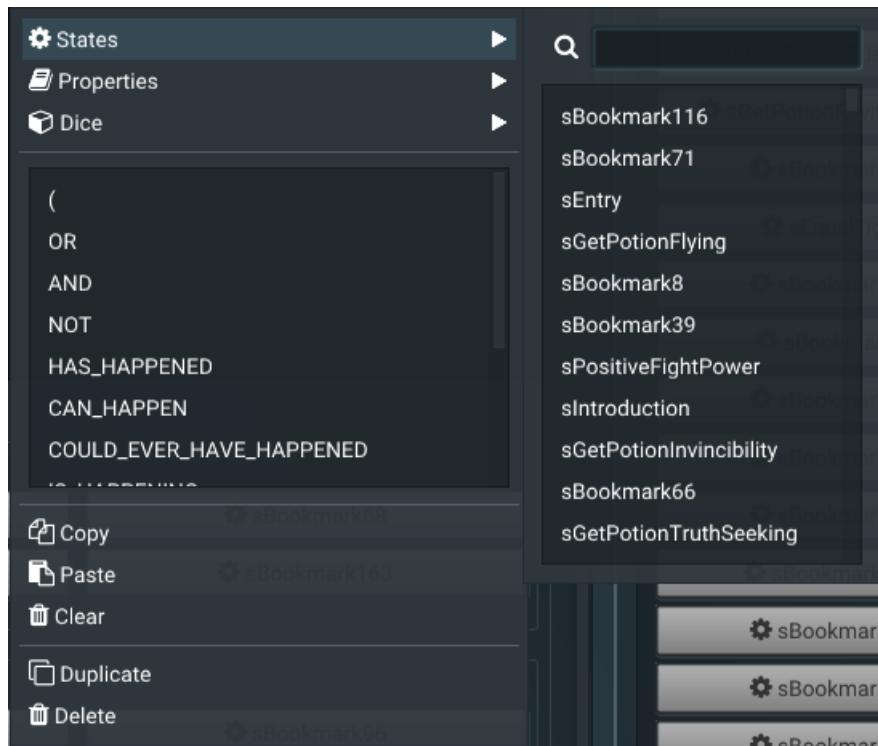
Во полето „Израз за достапност“ се впишува изразот за достапност за состојбата која е контекст, во форма на скрипт дефиниран во поглавјето „Изрази за достапност“. Ова поле има способност да ги обележува со различна боја операторите, состојбите, својствата и константните броеви. За полесно уредување на овој израз автопополнување на параметри е овозможено преку контекстуалното мени кое се повикува на десен клик на глушецот во рамките на елементот на полето. Ова контекстуално мени овозможува автопополнување преку лев клик на имињата на состојбите, имињата на својствата, операторот DICE со неговите специфични параметри, загради, сите достапни оператори во изразот за достапност. Автопополнувањето има способност и за замена на параметри. Ако селекцијата при отворање на контекстуалното мени е врз некој параметар, тогаш автопополнувањето автоматски ќе го замени тој параметар. Параметрите преку ова мени можат да бидат и копирани, залепени, избришани или дуплицирани.

Откако состојбата ќе биде ажурирана, се ажурираат и податоците во наредните две полиња кои ги листаат сите состојби и својства од кои состојбата која е во контекст зависи и состојбите кои зависат од состојбата која е во контекст.

Во следниот сегмент се листаат сите изјави на дијалог (анг. dialog statements). Изјавите на дијалог се дефинирани со кратки сегменти на текст врзани со некоја состојба. Ако состојбата со која се врзани е достапна, тогаш изјавата за дијалог е прикажана како копче за избор во читачот. Состојбата на која изјавата на дијалог се врзува



Слика 9: Контекст мени на изразите за промена.



Слика 10: Контекст мени на изразите за достапност.

може да се избере преку паѓачкото мени кое може да биде филтрирано. Ако состојбата е дефинирана како „ALL“ тогаш таа изјава на дијалог важи за сите состојби, ако понатаму не е дефинирано поинаку.

Во сегментот „Опции“ може да се изберат дополнителните опции на состојбата дефинирани во поглавјето „Состојби“.

Во последниот сегмент „Глава на состојбата“ може да се дефинира стилот на состојбата која е во контекст кога таа би се прикажувала во читачот. Во ова поле може да се внесе било каков HTML код кој вообичаено би се наоѓал во главата на веб страница, измеѓу HTML етикетите „`<head>`“ и „`</head>`“.

Најдолу во овој прозорец е копчето „Прикачи белешка“ преку кое се создаваат нови белешки специфични за моментално селектираната состојба. Белешките немаат никаква функционална важност, тие им служат на авторите како бележник кои им е важен само нив. Во бинарната верзија на датотеките, овие белешки не се

State	Dialog Statement	Actions
sBookmark12	Go back North?	Delete
sBookmark13	North?	Delete

Слика 11: Уредување на изјави за дијалог.

зачувуваат. Секоја белешка може да биде уредена со претходно дефинираните уредувачи на текст (исто како и за полето „Ситуација“). Тие дополнително можат да бидат и копирани, исечени, залепени, избришани. Белешките можат да бидат прикачени на врвот преку копчето „Прикачи“, или можат да бидат означени како „Важно“ преку истоименото копче, при што нивната боја се менува во црвена.

## 8.2 Прелистувач на состојби

Основна функција на прелистувачот на состојби (анг. states browser) е подреден, филтриран или специјално организиран преглед на состојбите во системот и врските меѓу нив. Прелистувачот на состојби нуди 4 погледи врз состојбите и секој од овие погледи овозможува различен пресек на состојбите и врските меѓу нив и се користи за различен тип на побарувања при пребарување.

Сите погледи можат да бидат филтрирани по име преку полето за филтрирање кое се наоѓа во горниот дел од прозорецот. Специјалните случаи на филтрирање ќе бидат разгледувани во понатамошниот текст. Сите пребарувања се одвиваат во реално време, односно во моментот кога во полето за филтрирање се внесува текст. Една од предностите на непосреден режим на претставување на контролите на корисничкиот интерфејс е ефикасноста и брзината при филтрирање на огромна количина на податоци како во овој случај. Четирите погледи врз состојбите кои ќе бидат разгледувани во понатамошниот текст се: базен (анг. pool), табела (анг. table), граф (анг. graph) и патишта (анг. paths).

Преку прелистувачот на состојби може да се создаде и нова состојба со клик на копчето кое се наоѓа најлево во листата на копчиња.

### 8.2.1 Поглед базен

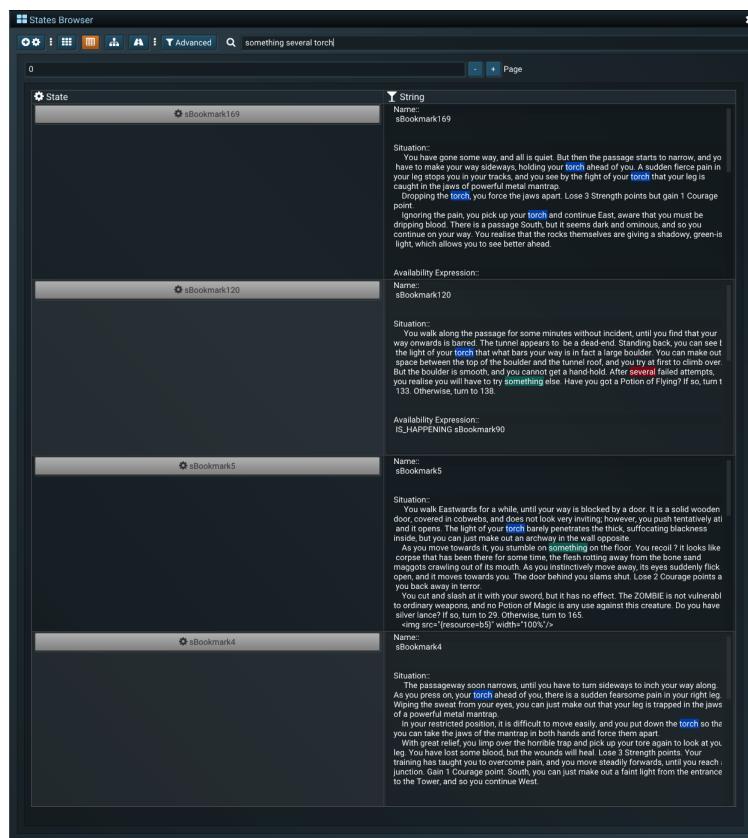
Погледот „базен“ овозможува поглед од горе (анг. top-down) поглед врз сите состојби кои постојат во системот, без разлика на врската меѓу нив. Овој поглед е корисен за брзо пребарување низ состојбите кога се знае името на состојбата или множеството состојби кои се бараат. Приказот е едноставна листа која ги содржи сите филтрирани состојби. Овој поглед не овозможува напредно пребарување.

States Browser					
• sBookmark116	• sBookmark71	• sEntry	• sGetPotionFlying	• sBookmark8	• sBookmark39
• sIntroduction	• sGetPotionInvincibility	• sBookmark66	• sGetPotionTruthSeeking	• sBookmark12	• sBookmark166
• sGetPotionSearching	• sBookmark21	• sGetPotionTransparency	• sGetPotionPower	• sBookmark196	• sTheJourneyBegins
• sBookmark35	• sGetPotionCalm	• sBookmark24	• sOptionsChosen	• sBookmark43	• sBookmark3
• sBookmark119	• sGetPotionFear	• sGetPotionIntuition	• sBookmark27	• sGetPotionDuality	• sBookmark41
• sBookmark30	• sGetPotionElusiveness	• sBookmark57	• sGetPotionMadness	• sGetPotionRevitalization	• sBookmark16
• sBookmark42	• sBookmark13	• sBookmark167	• sEqualFight	• sBookmark42	• sBookmark2
• sBookmark5	• sBookmark34	• sBookmark6	• sBookmark37	• sBookmark7	• sBookmark36
• sBookmark38	• sBookmark10	• sBookmark11	• sBookmark14	• sBookmark15	• sBookmark17
• sBookmark200	• sBookmark19	• sBookmark20	• sBookmark22	• sBookmark23	• sBookmark25
• sBookmark28	• sBookmark29	• sBookmark40	• sBookmark31	• sBookmark45	• sBookmark46
• sBookmark48	• sBookmark49	• sBookmark50	• sBookmark51	• sBookmark52	• sBookmark53
• sBookmark55	• sBookmark56	• sNegativeFight	• sBookmark59	• sBookmark59	• sBookmark60
• sBookmark63	• sBookmark64	• sBookmark65	• sBookmark67	• sBookmark68	• sBookmark69
• sBookmark70	• sBookmark115	• sBookmark72	• sBookmark114	• sBookmark73	• sBookmark113
• sBookmark112	• sBookmark25	• sBookmark111	• sBookmark76	• sBookmark110	• sBookmark77
• sBookmark79	• sBookmark149	• sBookmark80	• sBookmark148	• sBookmark81	• sBookmark82
• sBookmark84	• sBookmark85	• sBookmark86	• sBookmark87	• sBookmark141	• sBookmark88
• sBookmark89	• sBookmark90	• sBookmark91	• sBookmark92	• sFightFirstBat30	• sBookmark93
• sBookmark95	• sBookmark96	• sBookmark97	• sBookmark98	• sBookmark99	• sBookmark100
• sBookmark102	• sBookmark103	• sBookmark104	• sBookmark105	• sBookmark106	• sBookmark107
• sBookmark109	• sBookmark118	• sBookmark120	• sPositiveFightDouble	• sBookmark121	• sBookmark122
• sBookmark124	• sBookmark125	• sBookmark126	• sBookmark127	• sBookmark128	• sBookmark129
• sBookmark131	• sBookmark132	• sBookmark133	• sBookmark134	• sBookmark135	• sFightThirdBat30
• sBookmark137	• sBookmark138	• sBookmark139	• sBookmark142	• sBookmark143	• sBookmark144
• sBookmark146	• sBookmark147	• sBookmark150	• sBookmark151	• sBookmark152	• sBookmark153
• sBookmark155	• sBookmark156	• sBookmark157	• sBookmark158	• sBookmark159	• sBookmark160
• sBookmark163	• sBookmark164	• sFightSecondBat30	• sBookmark165	• sBookmark168	• sBookmark169
• sBookmark171	• sBookmark172	• sBookmark173	• sBookmark174	• sBookmark175	• sBookmark176
• sBookmark178	• sBookmark179	• sBookmark180	• sBookmark181	• sBookmark182	• sBookmark183
• sBookmark185	• sBookmark186	• sBookmark187	• sBookmark188	• sBookmark189	• sBookmark190
• sBookmark192	• sBookmark193	• sBookmark194	• sBookmark195	• sBookmark197	• sBookmark198
• sDeathStrength	• sPositiveFight	• sFlight	• sNegativeFightElusiveness	• sPositiveFightSilverLance	• sUseInvincibility30
• sUsePower30	• sNegativeFightInvincibility	• sNegativeFightDouble	• sStopForAMeat		• sUseElusiveness30

Слика 12: Поглед наречен „базен“

### 8.2.2 Поглед табела

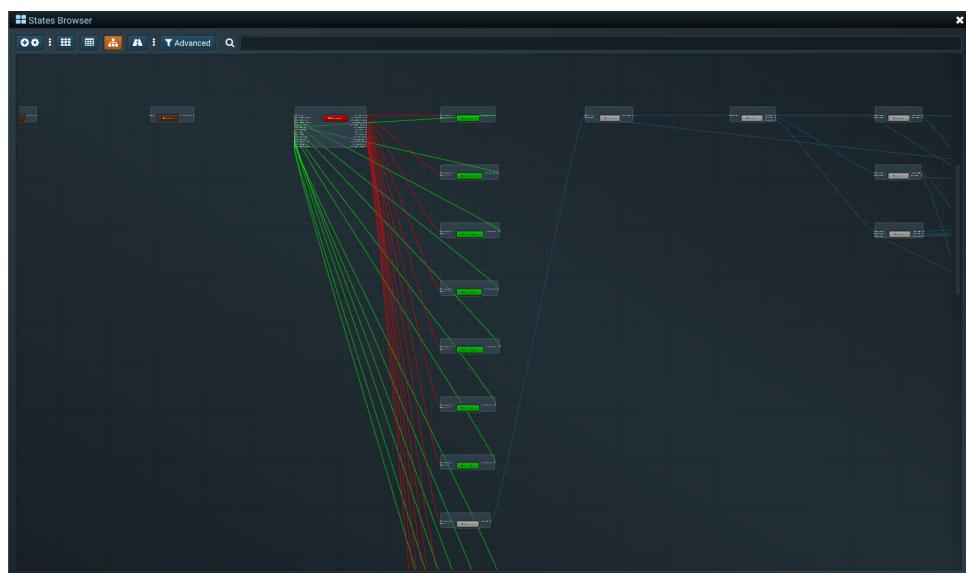
Погледот „табела“ овозможува детален поглед врз состојбите, вклучително и вредностите на нивните параметри. Кога се користи филтерот за пребарување во овој поглед се земаат предвид сите параметри на состојбите, а секое празно место во нишката на карактери на филтерот се третира како посебен филтер. Секој посебен филтер се обележува со различна боја во табелата на состојби. Овој преглед е корисен кога се бара нешто кое се наоѓа не само во името на состојбата туку и во нејзините внатрешни параметри. Табелата може да биде и филтрирана преку напредните филтри кои вклучуваат филтрирање според пасивните параметри на состојбите. Дополнително, може да се филтрираат и прикажат само специфични параметри на состојбата.



Слика 13: Табеларен поглед

### 8.2.3 Поглед граф

Погледот „граф“ овозможува визуелен приказ на комплетниот систем во форма на граф. Овој поглед визуелно ги прикажува врските меѓу состојбите. Бојата на секоја врска зависи од пасивните параметри на состојбата од која врската излегува. Овој поглед најчесто се користи за создавање на ментална мапа на состојби при пишување на системот бидејќи многу лесно може да се утврди просторната позиција на секоја од состојбите. Овој поглед овозможува напредно пребарување описано во погледот „Табела“.



Слика 14: Погледот на насочен граф

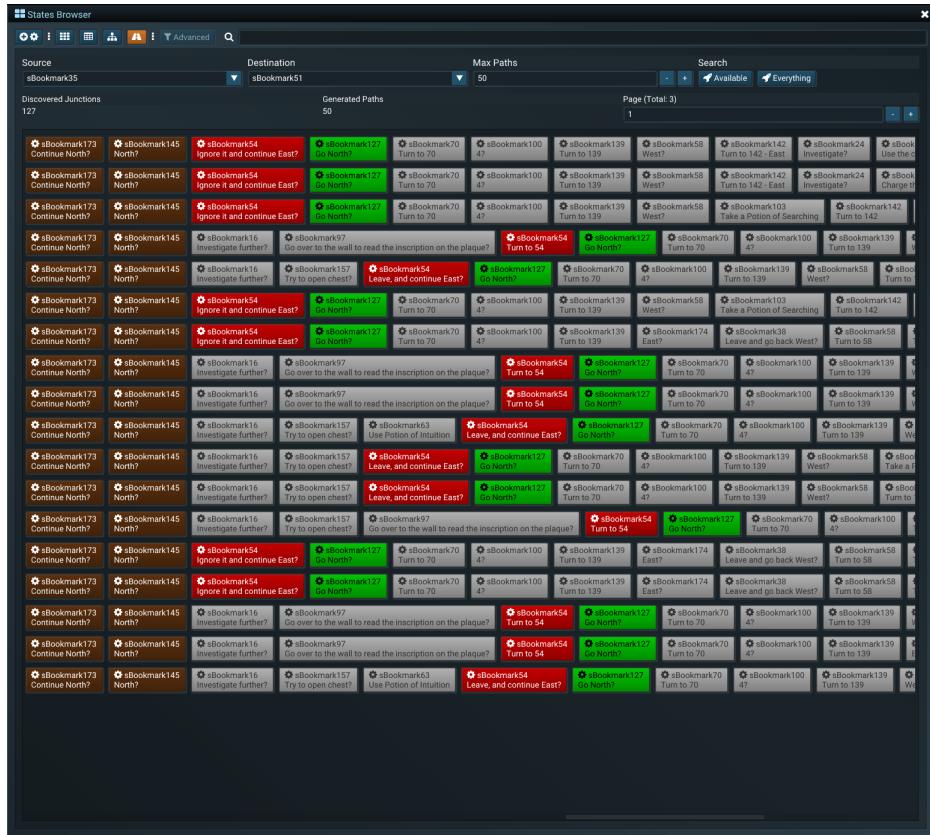
### 8.2.4 Поглед патишта

Погледот „патишта“ овозможува приказ на сите патишта кои водат од една состојба до друга состојба во рамките на системот. Постојат бесконечно многу начини на кои може да се премине од една состојба во системот во друга состојба во системот, па од таму проблемот на генерирање на сите патишта кои водат од една до друга состојба е нерешлив. Како резултат на ова свойство на системот, невозможно е да се најдат сите патишта секогаш кога системот ќе биде променет, бидејќи тоа е инструкција која никогаш нема да заврши. За

да се реши овој проблем погледот „Патишта“ во прелистувачот на состојби овозможува генерирање на дефиниран број на патишта на клик на копче. Секое генерирање на патишта мора да има дефинирани извор, дестинација и максимален број на патишта кои треба да се генерираат. Секое извршување на генерирањето на патишта се извршува на посебна процесорска нишка, па така генерирањето на патишта може да се следи во реално време во уредувачот.

За поминување на графот на состојби во системот се користи модифицирана верзија на алгоритмот „пребарување во ширина“ (анг. breath first search). Пребарувањето во ширина е избрано наместо пребарувањето во длабочина бидејќи е попогодно за користење во корелација со модификациите кои ќе бидат дискутирани во понатамошниот текст. Најзначајната модификација на алгоритмот е кеширање на патиштата кои се пронајдени. Секој јазол во графот чува информација за патиштата преку кои може да се стигне до таму. Откако ќе биде пронајден валиден пат помеѓу две состојби, графот се поминува во обратна насока, а притоа се обележуваат сите јазли како јазли преку кои може да се стигне до целта. Потоа, при понатамошното пребарување на графот ако се наиде на јазол кој е обележан како јазол преку кој може да се стигне до целта, пребарувањето по таа гранка завршува и повторно графот се поминува во обратна насока и сите поврзани јазли се маркираат како јазли преку кои може да се стигне до целта. Конечно, откако ќе се поминат сите јазли во графот се генерира минимално распространет граф од сите јазли кои се обележани како јазли преку кои може да се стигне до целта. Понатаму, од тој граф се извлекуваат сите можни начини на кои може да се помине графот или максималниот број на патишта кој е дефиниран. На модерни системи, за систем кој содржи околу 500 состојби и околу 5000 врски меѓу нив, овој алгоритам успева да го пронајде субграфот на можни патишта во помалку од 1 милисекунда и да генерира 10000 патишта во 3.3 секунди.

Конечниот резултат од генерирањето на патишта е вертикална листа на патишта, кои и самите се хоризонтална листа на состојби. Патиштата се подредени според бројот на состојби кои ги содржат, па така првоприкажаниот пат е патот кој содржи најмалку меѓусостојби. Ако почетната состојба биде активирана, на секоја наредна состојба е прикажан начинот на кој може да се стигне до неа. Ова е многу корисно за авторите кои сакаат да имаат пресек на начините на кои може да се стигне од една во друга состојба.



Слика 15: Погледот на генерираните патишта

Овој начин на генерирање на патишта не е идеалниот бидејќи во него не се вклучени својствата на системот, па така може да се случи состојби кои се премостуваат само со употреба на својства да се изземени од листата на патишта. Ова е доста тежок проблем за решавање и во поглед на ефикасност и брзина и во поглед на искористување на меморија. За комплетната слика на патиштата секој јазол во системот би ја чувал комплетната состојба на системот во тој момент. Комплетната состојба на системот, во зависност од содржината на состојбите може да биде меѓу 120 бајти и неколку мегабајти, па така ако оваа состојба на системот се чува во секој јазол од графот би била потребна меморија која го надминува капацитетот на модерните системи. Од тука извира одлуката да се користат само зависностите меѓу состојбите за тестирање на валидноста на патот наместо комплетна евалуација на состојбата.

Постојат две копчиња за генерирање на патишта. Копчето

„Достапна“ ќе ги генерира само патиштата чијашто прва состојба е достапна. Копчето „Сè“ ќе ги генерира сите патишта кои ќе ги пронајде.

### 8.3 Преглед на приказната

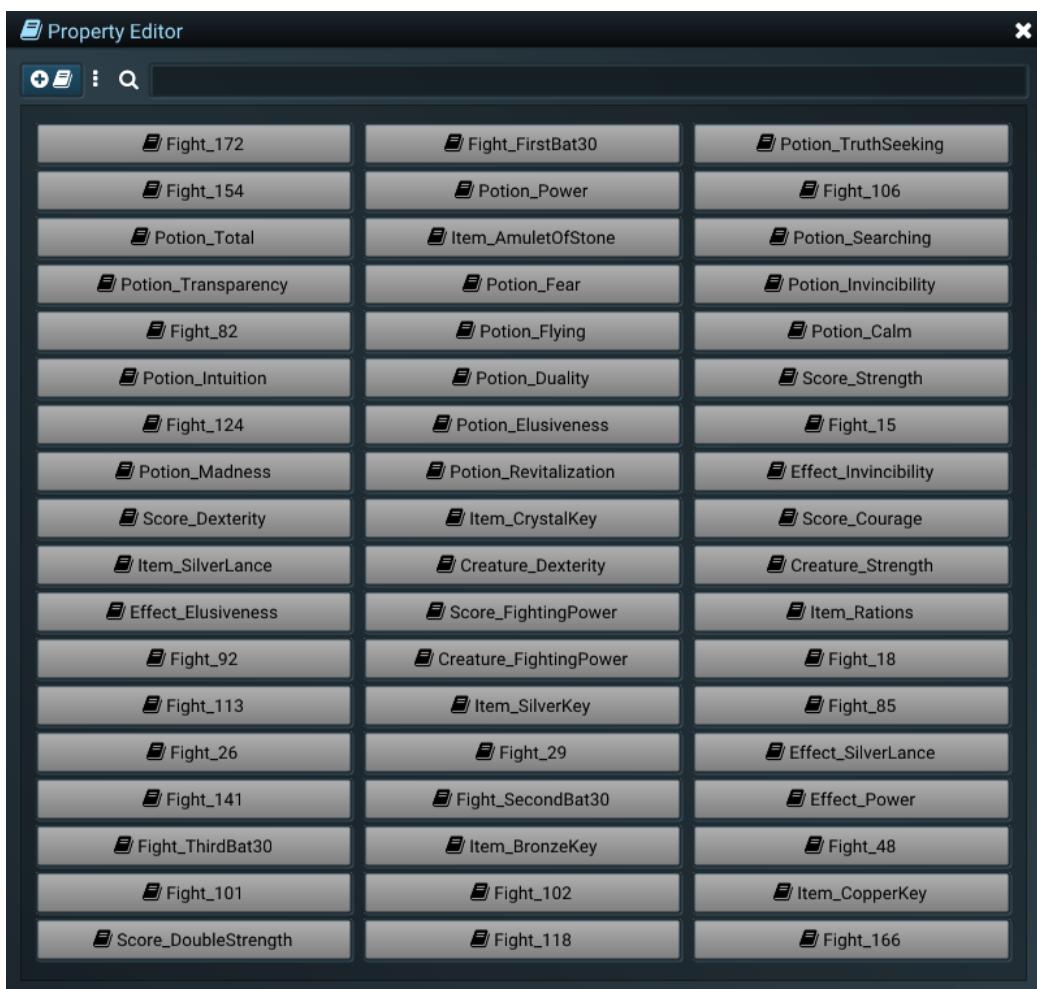
Прегледот на приказната се прави во прозорецот наречен „Читај во уредувачот“. Овој преглед е едноставен и содржи два визуелни сегменти. Първият визуелен сегмент е нишката на карактери на ситуацията на активната дијалог состојба. Вториот визуелен сегмент се копчињата за избор каде се презентираат нишките од карактери на изјавите за дијалог на активната дијалог состојба. Со клик на едно од овие копчиња играчот (читателот) ја активира дијалог состојбата поврзана со изјавата за дијалог состојбата и на тој начин прави прогрес во приказната.



Слика 16: Прозорецот за преглед на приказната.

## 8.4 Уредувач на својства

Функцијата на уредувачот на својства е уредување на својствата во системот. Погледот кон својствата во овој прозорец е сличен на погледот кон состојбите во погледот „базен“ во прелистувачот на состојби. Ново својство може да се додаде на копчето кое се наоѓа лево од полето за филтрирање. Полето за филтрирање ги филтрира својствата по име. Функциите на копчето на секое својство се претходно дискутирани и важат низ комплетниот уредувач на нелинеарни приказни.

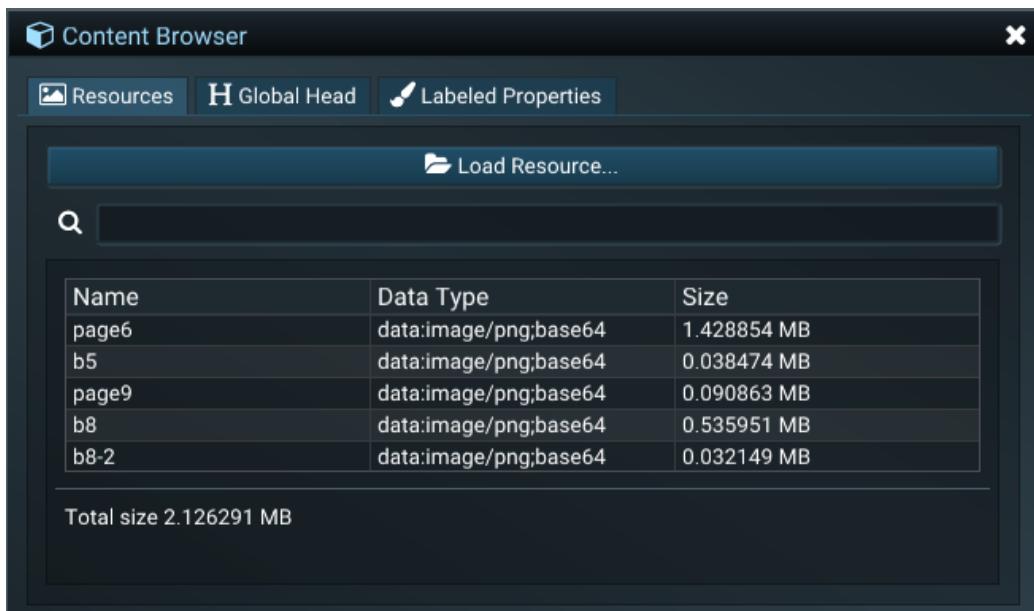


Слика 17: Уредувачот на својства

## 8.5 Прелистувач на содржини

Прелистувачот на содржини (анг. content browser) овозможува три различни функционалности: вчитување на надворешни ресурси, вметнување на HTML код во главата на секоја состојба и обележување на својства со имиња. Во овој прозорец предвидени се три јазичиња: Ресурси, Глобална глава, Обележани својства. Функционалноста на сите три јазичиња ќе биде дискутирана во продолжение.

Вчитувањето на надворешни ресурси од диск се одвива преку дијалогот за селектирање на датотека од диск кој може да се повика на копчето за вчитување на ресурси. Дијалогот ги филтрира затекнатите датотеки на патот на диск според достапните миме-дефинирани типови кои се наведени во поглавјето „Вчитување на ресурси“. Откако ќе се избере соодветна датотека, таа се вчитува во РАМ меморијата, се енкодира во base64 формат и се чува во листата на ресурси на системот од каде може да биде едноставно референцирана подоцна.



Слика 18: Прелистувач на содржини

Листата на вчитани надворешни ресурси може да биде филтрирана, а секој ресурс од листата може да биде преименуван

преку полето за преименување, може да биде избришан или вчитан повторно. Со клик на копчето за копирање се копира HTML кодот за специфичниот ресурс во зависност од неговиот миме-дефиниран тип.

Глобалната глава се однесува на кодот кој се вметнува помеѓу етикетите „`<head>`“ и „`</head>`“ на HTML кодот на состојбата кога таа се прелистува во читачот. Глобалната глава се вметнува во главите на сите состојби во системот, меѓутоа има помал приоритет од главите дефинирани во специфичните состојби кои се однесуваат само на состојбата. Ова поле може да биде уредено преку принципот за уредување на датотеки во надворешни уредувачи на текст.

Обележаните својства имаат три важни карактеристики кои можат да се уредуваат: Својството на кое тие се однесуваат, нивната етикета (читливото име) и нивниот редослед. За секое својство може да се додаде нова етикета преку копчето кое се наоѓа најлево, но секое својство може да има точно една етикета, односно да биде обележано еднаш. Ова се рефлектира во достапните опции за избор на својство во паѓачкото мени каде се листаат својствата кои можат да се одберат. Обележаните својства можат да бидат филтрирани по етикета преку полето за филтрирање.

## 8.6 Управувач на време

Управувачот на време манипулира со гранките и нивната состојба. Има две основни функционалности: управување со покажувачот на гранките и управување со самите гранки. Во самиот прозорец постојат три јазичиња: Контрола на време, Активна гранка и Прелистувач на гранки.

Во јазичето „Контрола на време“ е предвидена функционалноста за премотување и пребарување низ времето на гранките. Таму има две копчиња „Премотај“ и „Напредни“ кои го контролираат времето по еден чекор. Времето на гранката може да биде контролирано и преку алатката под овие копчиња која може директно да го пребара времето на одреден нумерички чекор. Копчето „Ресетирај“ ја доведува гранката во почетната состојба. Под ова копче се наоѓа комплетната историја на активирани состојби во гранката која се разгледува.

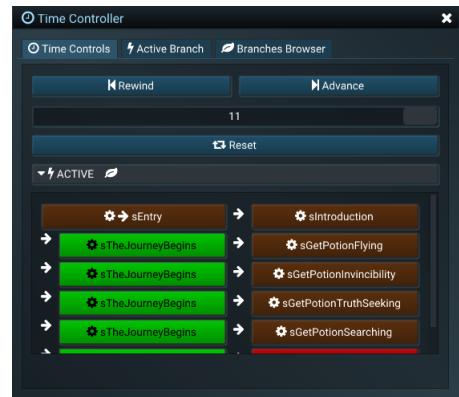
Во јазичето „Активна гранка“ се разгледува моментално активната гранка, ако таква постои. Копчето „Ресетирај ја гранката“ ќе ја доведе гранката во почетната состојба, при тоа задржувајќи го нејзиното нумеричко семе. Копчето „Зачувай ја гранката“ ја зачувува гранката заедно со нејзиното семе. Зачувувањето ќе биде дискутирано во понатамошниот текст. Семето на гранката може да биде рачно променето и да биде зачувано.

Во јазичето „Прелистувач на гранки“ може да се прелистуваат сите постоечки гранки. Гранките се управуваат преку користење на контекстуалното мени.

Преку управувачот на време секоја гранка може да биде ресетирана. Кога гранката се ресетира тогаш системот го задржува своето семе, а активната состојба се носи на почетната состојба и притоа се ресетираат сите состојби во системот. Ресетирањето на системот се прави преку копчето за ресетирање и се разликува од ресетирање на гранката во тоа што во тој случај семето на системот се генерира случајно. Семето на специфичната гранка во системот може да се избере преку полето „Семе“.

На сличен начин, гранката може да биде премотана или напредната. Кога гранката се премотува тогаш претходно зачуваната активна состојба се активира, а сите останати состојби се носат во моментот кога состојбата била активна. Многу сличен е и процесот на напреднување на гранката, само наместо гранката да се дивижи наназад низ времето, таа се движи нанапред низ времето. На лизгачот време може да се дефинира специфична точка во времето каде треба да се намести покажувачот на гранката. Сите овие процеси се одвиваат со задржување на интегритетот на системот.

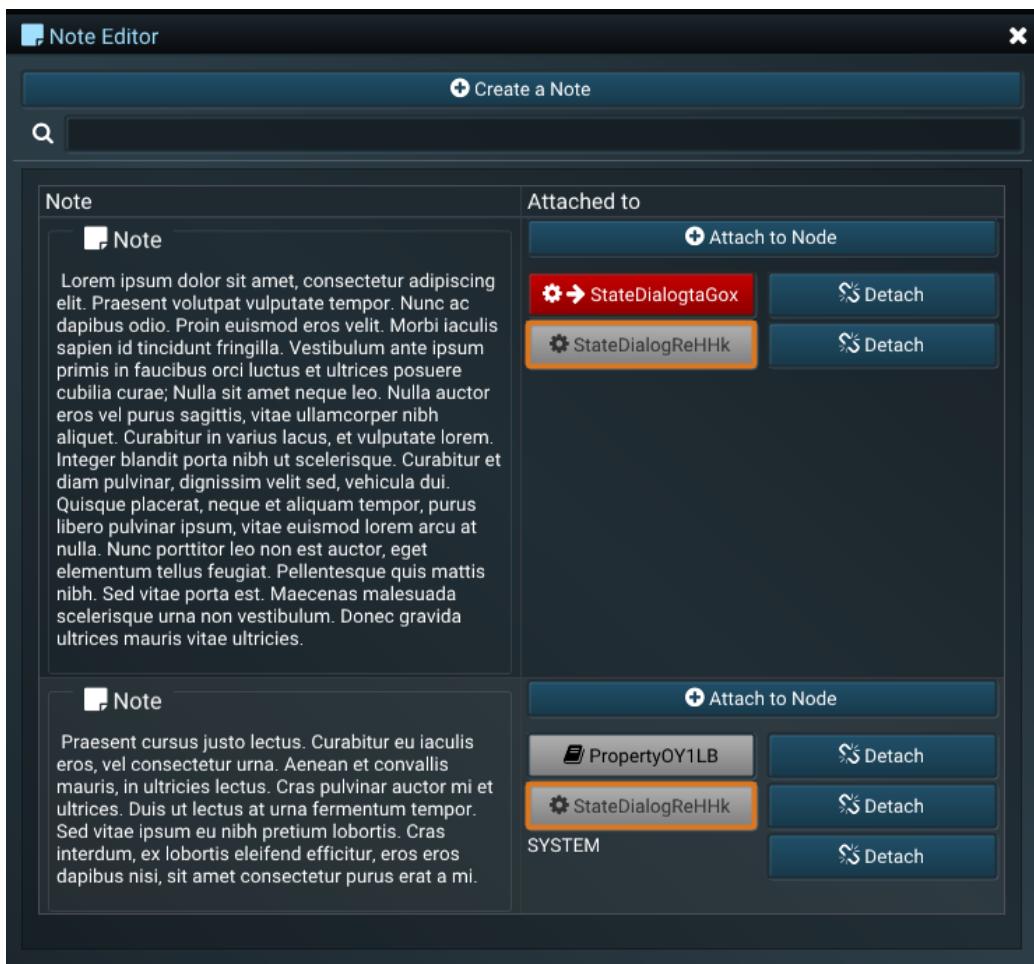
Секоја гранка може да се зачува преку копчето „Зачувай“. Кога гранката ќе биде зачувана таа добива случајно име кое може да се промени преку прелистувачот на гранки во управувачот со време. Во прелистувачот на гранки може да се види композицијата на состојби за секоја гранка и може да се управува со гранките: да се менува нивното



Слика 19: Погледот на управувачот со време

име, гранката да се избрише или гранката да се вчита. При вчитување на гранката, таа гранка станува активна во управувачот со време при што контролите за управувањето на времето се однесуваат на таа специфична гранка. Гранките во прелистувачот на гранки можат да бидат филтрирани по име преку полето за филтрирање.

## 8.7 Уредувач на белешки



Слика 20: Погледот на уредувачот на белешки

Уредувачот на белешки овозможува поглед врз сите белешки кои постојат во системот. Белешките можат да бидат прикачени на било која компонента на системот, вклучително и состојби и својства,

но можат да бидат прикачени и на самот систем, во кој случај тие се нарекуваат и глобални белешки.

Преку копчето „Создај белешка“ може да се додаде нова белешка во системот. Белешките можат да се пребаруваат според нивната содржина, преку полето за пребарување под копчето „Создај белешка“.

Во долниот дел од прозорецот се излистани сите белешки кои се достапни во системот. Секоја белешка може да се уредува преку соодветното контекст мени кое е достапно за белешката. Таа може да се уредува преку било кој текстуален уредувач претходно дефиниран во прозорецот „Поставки“.

Постојат и дополнителни опции кои овој поглед ги нуди, насочени кон управувањето со белешките. Копчето „Прикачи на јазол“ овозможува белешката да биде прикачена на една или повеќе компоненти наречени „јазли“ на системот. Клик на копчето отвора листа на компоненти од кои може да се избере. За секоја веќе прикачена белешка постои опцијата да се „Откачи“ од јазолот со клик на истоименото копче.

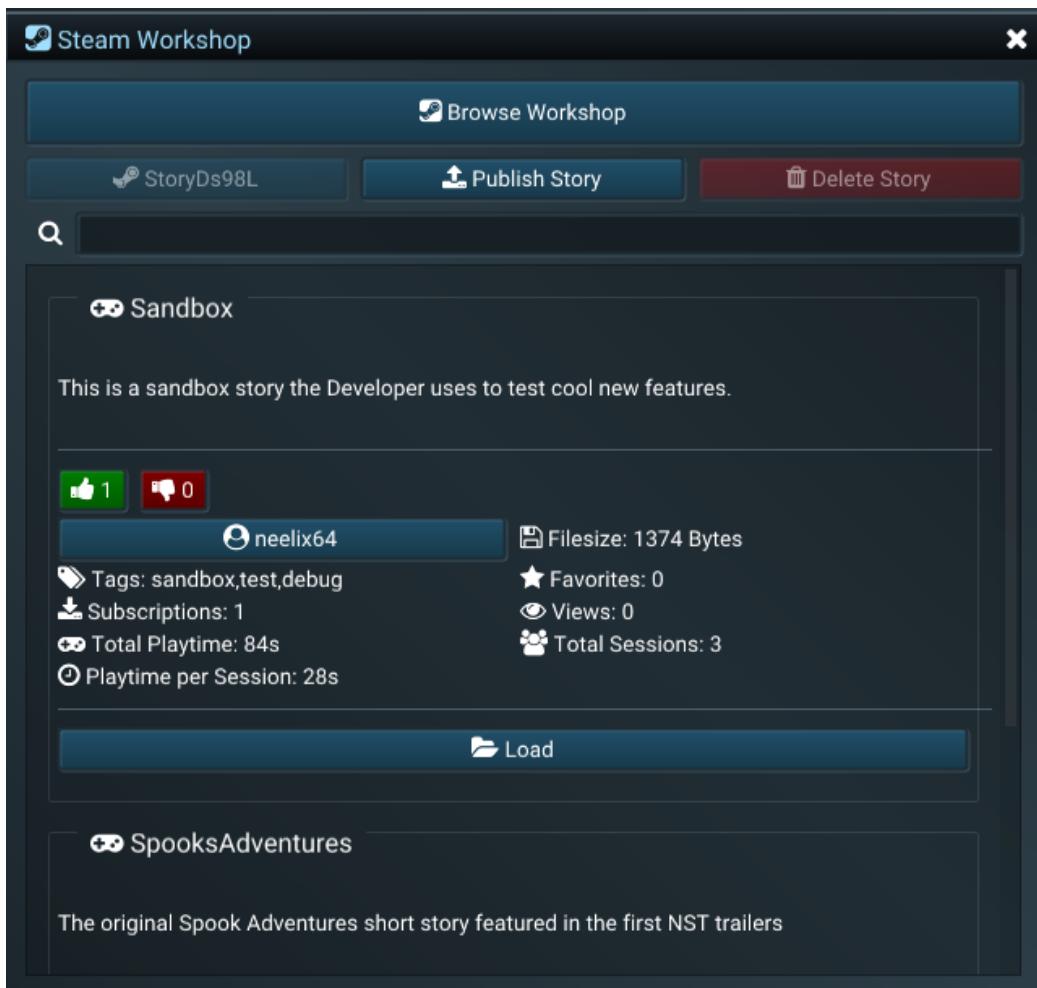
## 8.8 Steam Работилница

Прозорецот Steam Работилница овозможува пристап до функционалностите кои произлегуваат од интеграцијата со платформата Steam. Преку интеграцијата им се овозможува на корисниците да ги споделуваат авторските приказни преку алатката Steam Работилница (анг. Workshop) на платформата Steam. Овој прозорец бара активна конекција на интернет и активна конекција кон сервисите на платформата Steam.

Содржините кои се достапни на работилницата можат да се прегледаат преку клик на копчето „Прегледај ја Работилницата“. Кликтот отвора нов прозорец над главниот прозорец и води кон работилницата на платформата Steam.

Името на приказната има сопствено копче. Ова копче е достапно само ако приказната е веќе прикачена на Steam

работилницата и води до страницата на приказната на Steam работилницата.



Слика 21: Погледот на прозорецот на Steam работилницата

Копчето „Избриши“ е достапно само ако приказната е присутна на Steam работилницата. Со клик на ова копче приказната би била избришана од Steam работилницата.

Копчето „Симни“ е достапно за сите приказни кон кои корисникот е претплатен. За корисникот да симне некоја приказна и да може да ја прелистува и уредува, тој мора да е претплатен на приказната. Откако приказната ќе биде симната, таа е достапна и во папката на Steam работилницата, но и во папката за приказни од каде

може многу едноставно да се отвори со користење на опцијата за вчитување во уредувачот.

Копчето „Ажурирај и вчитај“ е достапно за сите приказни кои се веќе претходно симннати во локална папка. Клик на ова копче гарантира дека најновата верзија на приказната ќе биде вчитана во уредувачот.

Под копчињата се наоѓа листа на сите приказни на кои корисникот се има претплатено. Секоја приказна има сопствено контекстуално мени преку кое се управува со приказната. Ова контекстуално мени нуди три опции: отварање на приказната во Steam работилницата, бришење на претплатата и бришење на локалните датотеки кои биле симннати преку копчето за симнување или преку копчето за ажурирање.

### 8.8.1 Објавување на приказна

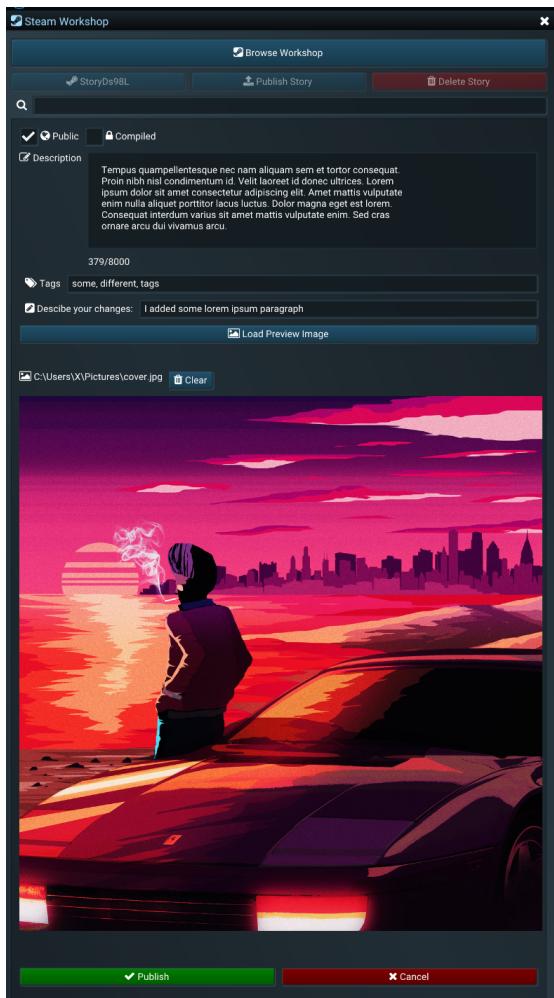
За да се објави една приказна преку Уредувачот, таа прво треба да е отворена во уредувачот. Откако приказната ќе биде отворена треба да се кликне на копчето „Објави ја приказната“. Овој клик отвора ново јазиче со опции за објавување на приказната на Steam работилницата.

Овој поглед нуди неколку можности:

**Јавно:** Дали приказната да биде објавена јавно или приватно. Ако полето за избирање е селектирано тогаш приказната ќе биде објавена јавно на Steam работилницата.

**Компајлирана:** Дали приказната да биде компајлирана или не. Ако приказната е компајлирана тогаш таа не може да биде декомпајлирана, односно е перменентно криптирана. Ако е некомпајлирана тогаш таа може да биде уредувана со Уредувачот, а во спротивно може да биде вчитана само преку Читачот.

**Опис:** Опис на приказната кој ќе им биде прикажан на сите корисници кои ќе ја посетат страницата на приказната на Steam работилницата.



Слика 22: Објавување на приказна на Steam работилницата

**Етикети:** Етикетите на приказната, разделени со запирка. При пребарување на Steam работилницата се користат етикетите за да се добијат подобри резултати при пребарувањето.

**Опис на промените:** Секоја промена на приказната мора да има опис. Овој опис им служи само на авторите како ориентир, за да би можеле да се вратат кон некоја промена во иднина.

**Вчитај слика за преглед:** Сликата за преглед е основната слика која им се прикажува на сите корисници на Steam работилницата кога пребаруваат приказни или кога ја прелистуваат страницата на самата приказна. Ова копче овозможува означување и

прикачување на слика од локален диск на Steam работилницата.

Откако корисникот ги пополнил овие полинја треба да кликне на копчето „Објави“ за да ја објави приказната на Steam работилницата. Постои и опција да се врати назад без да ги зачува промените со клик на копчето „Откажи“.

## 8.9 Останати компоненти

Останатите компоненти на уредувачот на нелинеарни приказни се однесуваат на компоненти кои се насочени кон самиот уредувач наместо кон системот за нелинеарни приказни. Во понатамошниот текст ќе бидат анализирани само најважните од овие компоненти за функционирањето на уредувачот.

#	Ниво	Време	Подсистем	Листа	Порака
1	STATEMENT	26-08-2021+08-15-01	NSTEeditor	292	Скорешна датотека е прочитана Proteus_01_v1
1	STATEMENT	26-08-2021+08-14-58	NSTEeditor	292	Увезена датотека D:\Projects\NST\NST\NSTEditor\exports\Pro

Слика 23: Дневникот на дејства

### 8.9.1 Дневник

Дневникот е едноставна листа на дејства кои се случуваат во текот на работата со уредувачот на нелинеарни приказни. Сите грешки

кои се случуваат при работа со уредувачот се печатат во дневникот и подоцна можат да бидат анализирани. Секоја состојба на дневникот може да биде зачувана во датотека на диск. Оваа состојба на дневникот автоматски се зачува на диск ако уредувачот неправилно прекине со работа (англ. to crash).

### 8.9.2 Јазик

Уредувачот на нелинеарни приказни може да има дефинирани јазици. Еден јазик е дефиниран со неговата специфична датотека која се наоѓа во папката `lang`. Форматот на оваа датотека е многу сличен на форматот `ini`. Во првиот ред се дефинира изворниот начин на кој јазикот се идентификува, а сите наредни редови се парови од клуч и вредност одделени со знакот за еднаквост. Клучот секогаш е на английски јазик, а вредноста е на изворниот јазик. Дефиницијата за сите можни комбинации на клучеви и вредности се наоѓа во посебна датотека именувана како `NSTStrings.txt`.

Во имплементацијата има посебни C++ макро команди кои за дадена нишка од карактери на английски јазик ја враќа соодветната вредност на јазикот кој е избран од корисникот. Во меморија јазиците постојат како хеширани мапи помеѓу нишката на карактери на английски јазик и нишката на карактери на избраниот јазик. Ако при барање во соодветниот јазик не е пронајдена нишката на карактери што треба да се преведе тогаш се избира оригиналната нишка на карактери на английски јазик.

Јазикот во уредувачот на нелинеарни приказни може да се избере во прозорецот „Поставки“, во јазичето „Уредувач“, а за целите на овој труд достапни се английскиот, македонскиот, рускиот и украинскиот јазик.

### 8.9.3 Визуелна тема

Библиотеката за создавање на кориснички интерфејс ImGui овозможува модифицирање на мноштво параметри кои се однесуваат на визуелниот изглед и боите на визуелните елементи. Во уредувачот

на нелинеарни приказни овие параметри се сведени на четири основни параметри кои можат да се проследат кон секоја визуелна тема. Секоја визуелна тема има свое име и вредности на четирите параметри. Вредностите на параметрите се контролирани од код и не можат да се уредуваат од корисниците. Кога уредувачот се стартува или кога темата е променета овие четири вредности се проследуваат до библиотеката ImGui која потоа контролира на кој начин се пртаат визуелните контроли во прозорецот.

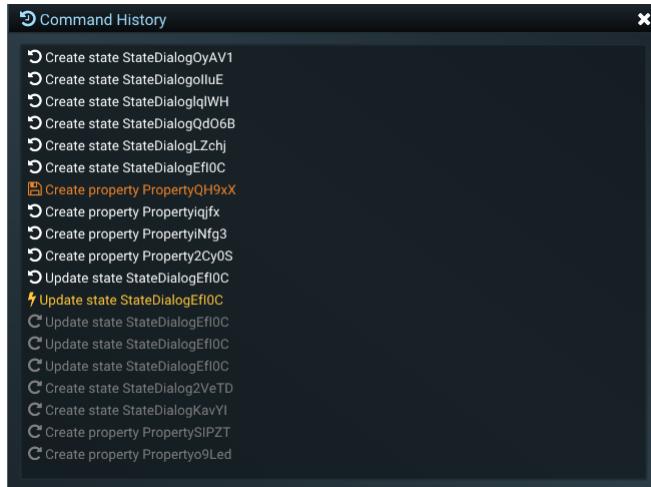
#### 8.9.4 Историја на команди

Било какво дејство во рамките на уредувачот на нелинеарни приказни го дефинираме како извршување на команда. За секоја промена која се случува при работата мора да биде дефинирана уредувачка команда. Уредувачот на нелинеарни приказни чува историја од секоја извршена команда во текот на работењето.

Комплетната историја на извршени команди може да се прегледа преку посебен прозорец кој може да се прикаже преку менито „прозорец“, субменито „алатки на уредувачот“ и конечно субменито „историја на команди“. Преку овој прозорец, со клик на специфичната команда може да се оди наназад или нанапред во времето на извршените команди во текот на работата.

Овој систем на команди се грижи за интегритетот на уредувачот со тоа што секоја последователна комадна се запишува во листата на извршени команди. Од тука произлегува дека извршувањето на командите не може да го наруши интегритетот на системот и да доведе до неправилности бидејќи сите команди се извршуваат во ист редослед секој пат.

Секоја команда е дефинирана со параметри на командата, процедура на извршување на командата нанапред и процедура за враќање на командата наназад. Ако корисникот сака да се врати наназад во извршувањето на командите тогаш сите команди се извршуваат со нивната обратна операција. На пример, обратна операција од пишување на текст е бришење на текстот. Па така за враќање на командата на пишување на текстот, истиот тој текст ќе биде избришан, односно доведен во претходната состојба.



Слика 24: Визуелниот интерфејс на историјата на команди

Овој систем на историја на команди овозможува имплементирање на различни системи кои им помагаат на авторите да не се грижат за грешките кои ги направиле во текот на работата. Преку системот за историја на команди многу лесно се имплементира undo / redo функционалност која е основа на секој модерен софтвер. Операцијата undo е враќање една команда наназад во историјата на команди, а операцијата redo е одење една команда нанапред во историјата на команди.

Преку историјата на команди може да се следат и промените кои се направени во текот на работењето, па така ако покажувачот кон последно зачуваниот момент при работата се разликува од покажувачот кон последната извршена команда тогаш сигурно настанала некаква промена. Следствено на ова, ако авторот се обиде да создаде нов фајл без да ги зачува промените уредувачот мора да го предупреди за промените кои не се зачувани.

Авторот преку овој систем на историја на команди може во било кој момент да одлучи да ги отфрли промените што ги направил од последниот момент кога зачувал.

### 8.9.5 Резервни копии

Резервните копии произлегуваат како резултат од чувањето на историјата на команди. При одредена фреквентност на команди која авторот може да ја утврди преку менито „поставки“ и субменито „резервни копии“, може да се чува резервна копија на комплетната состојба на системот за нелинеарни приказни по извршувањето на н-тата команда. Резервната копија во основа е еквивалентна на зачувување на приказната од авторот, меѓутоа се случува автоматски, а самиот уредувач на нелинеарни приказни се грижи за верзиите кои се чуваат.

## 8.10 Поставки

Поставките на уредувачот имаат сопствен прозорец. Прозорецот за поставки има три јазичиња: Уредувач, Пишување и Надворешни уредувачи. Сите поставки можат да се преbaraат преку полето за преbaraување, без разлика на јазичето во кое се наоѓаат. Поставките можат да се зачуваат, да се отфрлат без да се затвори прозорецот или да се откажи промената при што се затвора и прозорецот. Овие акции можат да се извршат преку копчињата на врвот од прозорецот.

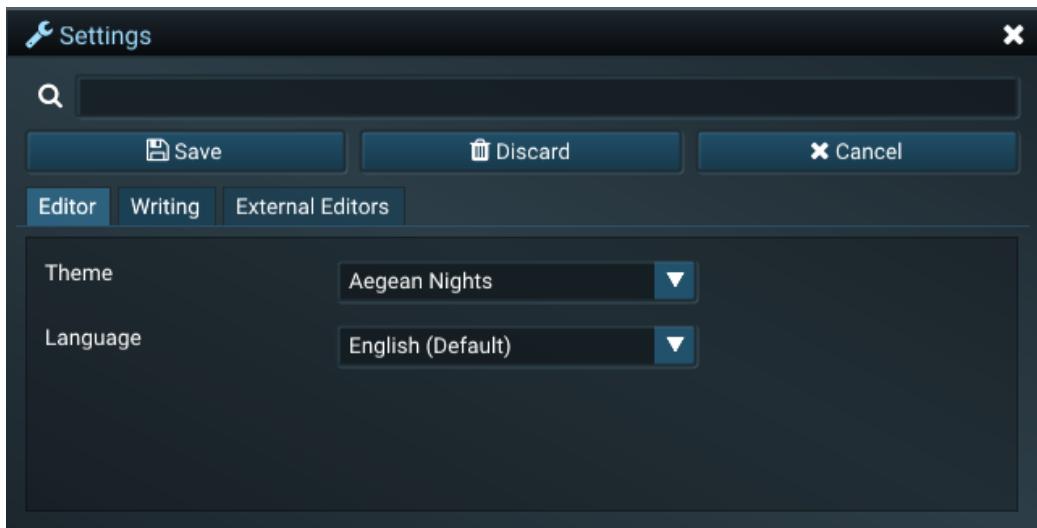
Секоја поставка во системот има сопствена основна вредност која секогаш може да биде избрана преку копчето со стрелка наназад. Ако поставката е веќе на својата основна вредност ова копче е недостапно.

Достапни поставки се:

**Тема** Се избира визуелната тема на уредувачот.

**Јазик** Се избира јазикот на уредувачот помеѓу македонски, английски, руски или украински.

**Честина на резервни копии** Колку често уредувачот прави резервна копија од приказната што моментално се уредува.



Слика 25: Прозорецот за поставки

**Напреден уредувач на состојби** Ако ова поле за избор е избрано тогаш уредувачот на состојби ги содржи сите опции.

**Надворешни уредувачи** Содржи листа на надворешни уредувачи кои уредувачот на нелинеарни приказни може да ги користи. Потребни се 3 податоци за да може уредувачот да се појави во листата на достапни уредувачи: име на уредувачот, локација на извршната датотека и аргументи кои се користат при извршување на извршната датотека. Извршните аргументи најчесто се користат за да се овозможи „чисто“ извршување на уредувачот на текст, без да се вчитуваат претходни фајлови и дополнителни опции кои не се потребни.

## 8.11 Заштита против пиратерија

Системот за заштита против пиратерија се потпира на системот што го нуди платформата Steam. За секоја копија на уредувачот и читачот на нелинеарни приказни постои уникатен клуч кој се врзува само за таа инстанца на инсталацијата. Предноста на користење на овој начин е во тоа што платформата Steam веќе има систем за заштита од пиратерија кој може многу лесно да се интегрира. Недостатокот е што за овој систем против пиратерија да работи,

потребна е активна интернет конекција и сметка на платформата Steam.

## 9 Читач

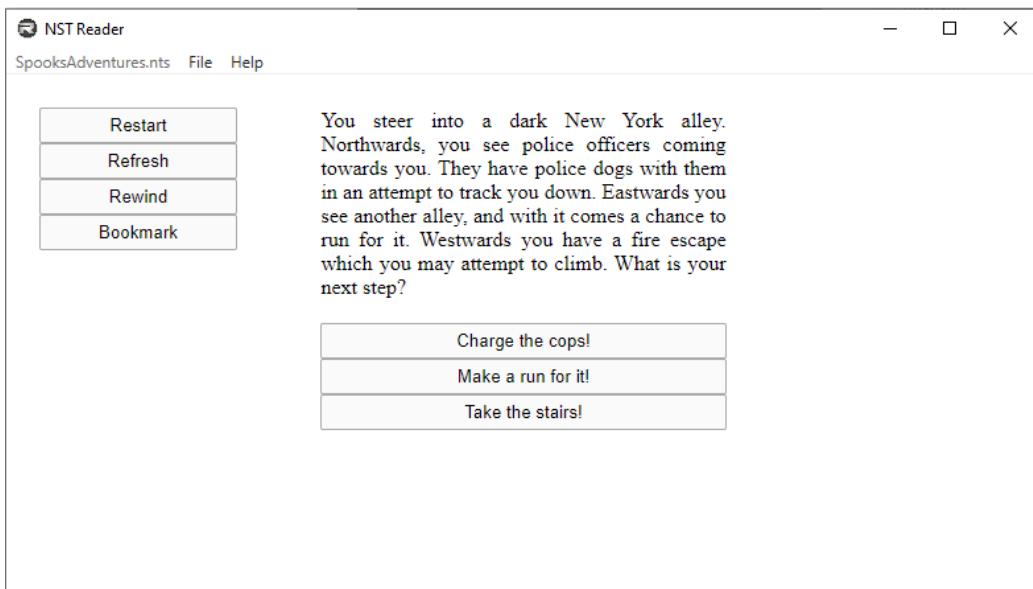
Читачот е засебен проект со сопствена извршна EXE датотека. Тој единствено овозможува читање на приказната која е создадена со помош на уредувачот на приказни. Читачот ја користи библиотеката за кориснички интерфејси WPF за прикажување на HTML код, напишана за програмскиот јазик C#. HTML и CSS кодот се генерира преку динамички поврзаната библиотека наречена NSTGenerator. За секоја состојба во системот читачот генерира засебен HTML код кој подоцна се проследува кон WPF за да може да биде нацртан на еcranот. WPF за пртање на HTML и CSS компоненти го користи субсистемот на пребарувачот Edge, кој во својата позадина го користи Chromium моторот за пртање и обработка на веб содржини.

Кога читачот се извршува во првиот поглед играчот (или читателот) има можност да избере која приказна од достапните приказни сака да ја вчита во читачот. Откако приказната ќе биде вчитана сите системски контроли се преземени од моторот на нелинеарниот систем, а задачата на читачот е да ја „преведи“ моменталната состојба на системот во HTML код кој подоцна ќе се прикажи во читачот.

Од функционален аспект, читачот е многу сличен на прегледот на состојба во уредувачот на нелинеарни приказни. Поделен е на два визуелни сегменти: во првиот сегмент се црта текстот и мултимедијалната содржина на моментално активната состојба, а во вториот сегмент се цртаат копчињата преку кои играчот (или читателот) има можност да носи избори при играњето.

Единствената разлика е што во левиот дел од читачот има специјални функционални копчиња за контрола преку кои може играчот да се врати на главното мени, да ја зачува играта, да го ресетира системот итн. Под овие функционални копчиња се излистани и обележаните состојби со нивните етикети (читливи имиња) и вредноста која ја имаат во тој момент.

Постои црвста логичка одделеност меѓу генераторот на HTML код и цртачот на HTML код во читачот. Генераторот на HTML код може да се користи во комбинација со било кој цртач на HTML наместо тој на WPF/Edge/Chromium, па така, во теорија, читачот би можел да биде



Слика 26: Читач

имплементиран на било која платформа или оперативен систем кој има поддршка за цртање HTML. За целта на ова истражување и дискусија, читачот работи само на оперативниот систем Windows.

За да се вчита приказна во Читачот постојат повеќе начини. Ако се читува датотека од локална адреса тогаш доволно е да се кликне на менито „Датотека“, па на копчето „Вчитај“. Откако во дијалогот кој ќе се отвори, се избере соодветна приказна, со клик на копчето „Ок“ таа датотека се читува во Читачот.

Приказна може да се чита и преку Steam работилницата, за што е потребно да се кликне на менито „Датотека“, па на копчето „Steam Workshop“. Во новиот прозорец кој ќе се отвори се излистани сите приказни кон кои корисникот е претплатен. Откако корисникот ќе избере приказна која сака да ја чита, тој треба да кликне на копчето „Прочитај“ за да ја чита приказната од Steam работилницата во Читачот.

### **9.0.1 Стилизирање на компонентите на читачот**

Во Читачот можат да се користат CSS и JS кодни делови за да се измени изгледот на приказната во Читачот. Основниот изглед на приказната во читачот може да се измени преку промена на следните CSS компоненти:

**button** За означување на сите копчиња

**#nst-Situation** За означување само на текстот на ситуацијата

**#nst-labeledPropertiesTable** За означување на табелата со означените својства

**#nst-labeledPropertiesTable td** За означување на секој поединечен елемент на табелата со означените својства

**#nst-MainMenuWrapper** За означување на главното мени

**#nst-MainMenuWrapper button** За означување на копчињата на главното мени

## 10 Имплементација на јавен апликациски интерфејс (API)

Јавниот апликациски интерфејс овозможува комуникација помеѓу внатрешните системи на еден софтвер со софтвери од трети страни. Најчесто, софтверите од трети стани имаат начини за создавање на приклучоци (англ. plugin) кои комуницираат со друг софтвер преку вчитување и извршување на функциите во јавните апликациски интерфејси. Интеграцијата со софтвери од трети страни зависи исклучиво од апликациското интерфејс.

За постигнување на целта на интеграција на системот со некој од популарните софтвери за развој на видео игри (специфично Unreal Engine 5) развиен е јавен апликациски интерфејс кој овозможува комуникација со функциите на нелинеарниот систем.

Јавниот апликациски интерфејс користи C API функционална парадигма, која се врзува на објектно-ориентираниот C++ дел. Функциите се извезуваат во DLL датотека која го користи исклучиво програмскиот јазик C. Причината за кој е избран овој начин е бидејќи вака извешените C функции можат да се врзат на било кој друг програмски јазик и да се користат како „домородни“ (англ. native).

Главната компонента на секој ентитет на апликацискиот интерфејс е светот. Светот содржи нелинеарен систем и други помошни компоненти кои се потребни за негова правилна иницијализација.

За да се создаде еден свет преку јавниот апликациски интерфејс треба да се повика функцијата именувана како `NST_CreateNewWorld()`. Оваа функција создава нова инстанца од светот и ја враќа адресата на која инстанцата постои во меморија во форма на 32-битен број. Овој број е „кваката“ (англ. handle) која клиент-апликацијата ќе ја користи секој пат кога сака да повика некоја функција на светот.

Светот постои во меморискиот простор на вчитаната DLL датотека, клиент-апликацијата никогаш не пристапува директно во овој мемориски простор, а наместо тоа ја користи „рачката“ која претходно ја добила за да го идентификува светот кој сака да го користи.

Конечно, кога клиент-апликацијата која го користи јавниот апликациски интерфејс, сака да се затвори, таа мора да ја повика функцијата `NST_DeleteWorld(handle)` од јавниот апликациски интерфејс за да се ослободи меморијата во која се чува светот.

Останати функции кои се достапни во јавниот апликациски интерфејс се:

`size_t NST_GetNumAvailableStates(size_t handle)` - Резултатот е 32-битен број кој го претставува бројот на моментално достапни состојби во системот.

`const char* NST_GetStateNameAtIndex(size_t handle, size_t index)`  
- Резултатот е низа од карактери која го дава името на состојбата на дадениот индекс.

`bool NST_IsStateAvailable(size_t handle, const char* stateName)`  
- За дадено име на состојба, функцијата резултира со одговор на прашањето дали состојбата со тоа име е достапна.

`const char* NST_GetDialogStatementAtIndex(size_t handle, size_t index)`  
- За даден индекс на изјава на дијалог, функцијата резултира со низа од карактери за таа изјава на дијалог.

`const char* NST_GetActiveStateSituation(size_t handle)` -  
Резултатот е низа од карактери која ја претставува ситуацијата на моментално активната состојба.

`void NST_ActivateStateByIndex(size_t handle, size_t index)` -  
Повикот кон оваа функција ја активира состојбата која го има дадениот индекс. Индексот е резултат од повик кон соодветната функција која резултира со овој индекс.

`void NST_ActivateStateByName(size_t handle, const char* stateName)`  
- Слична како функцијата со име `NST_ActivateStateByIndex`, само наместо индексот, операцијата ја прави со име на состојба која постои во системот.

`void NST_ResetStory(size_t handle)` - Повикот кон оваа функција резултира со ресетирање на приказната.

`void NST_ActivateStateByName(size_t handle, const char* stateName)`  
- Слична како функцијата со име `NST_ActivateStateByIndex`,

само наместо индексот, операцијата ја прави со име на состојба која постои во системот.

**void NST\_LoadFromString(size\_t handle, const char\* str)** -

Повикот на оваа функција со енкриптирана низа на карактери ќе резултира со обид да се вчита приказната преку таа низа на карактери.

## 11 Интеграција со Steamworks API

Steamworks апликацискиот интерфејс им овозможува на софтверите и игрите да направат комплетна интеграција со платформата Steam преку овозможување на пристап до нејзините системи. Steam е дигитална продавница за видео игри и алатки за видео игри. Интеграцијата со Steam овозможува и пристап до специјализирани алатки за размена на содржина создадена во игрите или алатките, наречена Steamworks.

Целта на интеграцијата со Steamworks е двојна. Првата цел е едноставното споделување на содржината создадена со Уредувачот. Корисниците преку платформата Steam работилница можат да ги споделуваат меѓусебно приказните кои ги создаваат, а можат и да ги продаваат за одредена цена. Втората цел е анти-пиратерија. Steamworks API-то нуди пристап до интерните системи на платформата Steam за борба против пиратерија кои беа дискутирани во поглавјето „Заштита против пиратерија“.

Интеграцијата е во уредувачот и во читачот. Во уредувачот корисниците можат да вчитуваат приказни кои други корисници ги имаат создадено, да прикачат нови приказни и да ги уредуваат сопствените приказни. Во читачот корисниците можат да ги отворат и да ги прелистуваат приказните кои други корисници ги имаат прикачено преку уредувачот на платформата Steam.

## 12 Интеграција како приклучок во софтвер од трети страни

Модерниот наратив во видео игрите сè повеќе се основа на разгранувачки дијалози. Постојат многу примери на видео игри кои имаат разгранувачки дијалог со различна длабочина. Примери за вакви игри се: Cyberpunk 2077 издадена во 2020 година, No Man's Sky издадена во 2016, Red Dead Redemption II издадена во 2018 година итн. Како резултат на ова постои мотив за да се интегрираат алатки за развој на ваков тип на наративи во модерните софтвери за развој на видео игри.

### 12.1 Софтвери за премостување (middleware)

Стандардниот начин за интегрирање на надворешни алатки во софтвер за развој на игри се таканаречени софтвери за премостување (анг. middlewares), па затоа овој начин е погоден за интеграција и на системот за нелинеарни приказни во софтвери за развој на видео игри.

Софтверите за премостување се самостојни софтвери кои не се дел од софтвер од трети страни, меѓутоа нудат начин за интеграција во софтвер од трети страни. Има многу примери за ваков софтвер, кај видео игрите најчесто овој софтвер се однесува на создавање на содржини кои подоцна се користат во самата видео игра како што се текстури, анимации, аудио, музика итн.

#### 12.1.1 FMOD

FMOD е софтвер за создавање, управување и пуштање на аудио датотеки во видео игри и друг софтвер. FMOD се состои од неколку различни компоненти кои го дефинираат како middleware:

**FMOD Studio:** Алатка за креирање на аудио за видео игри, дизајнирано во формат како дигитална аудио работна станица (анг. Digital Audio Workstation).

**FMOD Studio run-time API:** апликациски интерфејс кој им овозможува на програмерите едноставен начин за комуникација со FMOD Studio.

**FMOD Studio low-level API:** апликациски интерфејс за свирење на звучни датотеки, со можност да се додадат специјални ефекти и тродимензионален звук.

### 12.1.2 WWise

Wwise (Wave Works Interactive Sound Engine) е софтвер за создавање на интерактивни содржини за видео игри, со посебен фокус на видео игри. Wwise како и FMOD има мноштво на компоненти кои го категоризираат како middleware:

**Wwise Authoring** Софтвер за создавање на аудио датотеки која работи на повеќе платформи.

**SoundFrame API** Апликациски интерфејс кон Wwise Authoring алатката.

Структурата на FMOD и Wwise е многу слична на таа на софтерот обработуван во овој труд, па затоа и природна е одлуката да се земе овај формат како пример за интеграција во софтвери од трети страни. FMOD Studio и клиентот на Wwise ја имаат улогата на Уредувачот, а FMOD Studio low-level API и SoundFrame API ја имаат улогата на апликацискиот интерфејс за управување со нелинеарните системи.

## 12.2 Интеграција со Unreal Engine 5

Unreal Engine е софтвер за развој на 3D видео игри развиен од компанијата Epic Games. Софтерот е напишан во програмскиот јазик C++ и нуди можност за пренесување на многу други оперативни системи и платформи. Најновата верзија на Unreal Engine која го носи бројот 5 била лансирана во 2022 година.

Unreal Engine нуди можност за създавање на приклучоци (анг. plugins) и има можност за вчитување на неограничен број на динамички поврзани библиотеки од трети страни, што го прави погоден за интегрирање на софтвери за премостување. Приклучоците можат да се споделуваат на специјален пазар на приклучоци развиен од истата компанија која го развива и софтверот.

Unreal Engine поддржува различни типови на содржини како што се: текстури, модели, материјали, анимации, звуци итн. Секоја од овие типови на содржини има сопствен тип на податок дефиниран во Unreal Engine. Новиот тип на податок кој дефинира една нелинеарна приказна NSS, исто така треба да биде дефиниран како тип на податок. Овој тип на податок е наречен приказна (анг. Story).

Секоја приказна може да биде вчитана или ре-читана исто како и секоја друга содржина во Unreal Engine. Вчитувањето може да се направи и преку соодветниот дијалог за вчитување, но може да се направи и преку едноставна влечи-и-пушти акција во уредувачот на содржини во Unreal Engine.

Приказните во Unreal Engine се вчитуваат како енкриптирана текстуална низа од карактери која подоцна преку апликацискиот интерфејс содржан во динамички поврзаната библиотека на системот за нелинеарни приказни се вчитува во програмскиот простор на извршената динамички поврзана библиотека. За секоја функција од апликацискиот интерфејс постои функција која се врзува на системите на Unreal Engine. Покрај можноста за повик од C++ сегментите на Unreal Engine, овие функции можат да се повикаат и од нацртите (анг. Blueprints) на Unreal Engine.

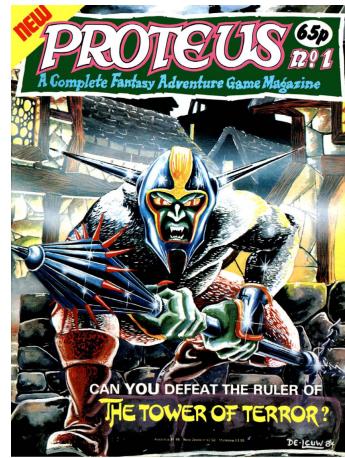
Функциите кои се изложени на повик од надвор можат да се искористат за приказните да се моделираат на било кој начин. Состојбите од нелинеарниот систем можат да се користат и за създавање на кориснички интерфејси, приказни, движење на геймплеј, динамичка контрола на аудио итн. Имплементацијата и примената на функциите за работи кои не се поврзани со наративот и игривоста се надвор од опсегот на овој труд, па затоа во понатамошниот текст и примери ќе се задржат на градењето на наративи и нивната интеграција во Unreal Engine 5.

## 13 Студија на случај - Proteus 01

Протеус е списание во форма на авантуристичка игра кое периодично било објавувано во осумдесеттите години на дваесеттиот век. Ова списание било рекламирано како „Комплетна соло авантуристичка игра во форма на магазин во која вие ја играте улогата на Херојот“. Фокусот на овие списанија е фантазијата и научната фикција, а секој различен број на списанието нудел различни правила и опции.

Ова списание е погодно за имплементирање во системот описан во овој труд бидејќи ги вклучува сите елементи на системот, вклучително состојбите и својствата и сите нивни параметри. За демонстрација во овој труд е земен првиот број на списанието именуван како „The Tower of Terror“. Списанието е на англиски јазик, па од таму и имплементацијата која ќе биде разгледувана во понатамошниот текст е на англиски јазик.

Во оригиналните правила за играње на списанието се потребни две зарчиња и пенкало. Како што играчот прогресира низ т.н. „Кула на Теророт“ тој многу веројатно ќе наиде на разни замки или чудовишта. Играчот исто така, ќе добие информации или ќе најде одредени предмети кои ќе му помогнат во неговото патешествие. Списанието бара од играчот овие информации да ги запише на лист хартија, а како што приказната прогресира играчот ќе треба да прави промени на овој лист хартија кои ги содржи информациите за неговото патешествие. Списанието бара од играчот и да направи мапа како што се качува на „Кулата на Теророт“ која ќе му биде од корист во патешествието. На крај, списанието го предупредува играчот дека можеби неговиот прв обид нема да биде успешен, но дека од тој обид ќе извлече информации кои ќе му помогнат во следниот обид.



Слика 27: Насловната страница на списанието „Протеус“

Списанието содржи нумерички обележан текст кој ако се чита во нумеричкиот редослед не прави смисла. Секој текст на крајот има насока за играчот на кој број следно да заврти за да ја продолжи приказната.

На почетокот на секоја игра, играчот поседува „Умешност“, „Сила“ и „Храброст“ како нумерички вредности. Играчот фрла едно зарче и додава 8 на вредноста на зарчето за да ја добие почетната вредност на „Умешноста“. Играчот фрла две зарчиња и додава 15 на вредноста на зарчето за да се добие почетната вредност на „Силата“. Играчот фрла едно зарче и додава 6 на вредноста на зарчето за да се добие почетната вредност на „Храброста“. Почетните вредности никогаш не можат да се надминат. Некои од вредностите можат да се променат во негативна или позитивна насока.

Во текот на играта играчот се соочува со различни видови на чудовишта кои имаат сопствени вредности за „Сила“ и „Умешност“. Битките со чудовиштата се одвиваат според специфичен алгоритам. Играчот прво фрла две зарчиња, а вредноста ја додава на моменталната вредност на „Умешноста“ на чудовиштето. Оваа вредност се нарекува и „моќ за борба“ на чудовиштето. Потоа играчот уште еднаш фрла две зарчиња и ова го додава на сопствената вредност на „Умешноста“ добивајќи ја сопствената „моќ за борба“. Ако „моќта за борба“ на играчот е поголема од таа на чудовиштето тогаш во тој момент играчот му задава удар на чудовиштето и чудовиштето губи 2 поени од својата „Сила“. Ова важи и обратно, кога „моќта за борба“ на чудовиштето е поголема од таа на играчот, со тоа што во тој случај се одземаат 2 поени од „Силата“ на играчот. Ако двата играчи имаат иста „моќ за борба“ тогаш никој не губи поени од својата „Сила“. Кога на играчот или на чудовиштето „Силата“ е помала или еднаква на 0, тогаш битката завршува, а победник е тој чијашто „Сила“ е поголема од 0. Ако играчот ја потроши „Силата“ целосно тогаш патешествието за него е завршено.

Играчот во текот на патешествието може да добие или да изгуби поени според тоа што е индицирано во текстот. На пример, играчот може да изгуби 2 поени од својата „Умешност“ или може да добие 1 поен на својата „Храброст“. Ако играчот го следи списанието треба да ги бележи на својот лист хартија со информации за патешествието.

Играчот на почетокот на играта зема 5 парчиња храна кои може да застане и да ги изеде во било кој момент од патешествието. Кога играчот ќе застане да јаде тогаш треба да одземе 1 поен од бројот на парчиња на храна кои ги поседува и да додаде 5 „Сила“ на листот хартија со информации за патешествието, но не повеќе од почетната „Сила“ која ја стекнал на почетокот на патешествието. Играчот не смее да застане да јаде додека некоја битка е во тек.

Во понатамошниот текст ќе бидат анализирани начините на кои можат да бидат имплементирани овие правила на списанието во системот за нелинеарни приказни.

### 13.1 Имплементирање на правила со употреба на својства

Прв чекор при имплементирање на правилата се употребата на својства. Секој параметар кој може играчот да го има како што се „Силата“, „Умешноста“ и „Храброста“ дефинира по едно својство во системот. Својствата имаат можност случајно да ја генерираат својата вредност на почетокот на приказната. Оваа случајно генерирана вредност може да биде помеѓу зададена минимална и максимална вредност. Во понатамошниот текст параметарот „Сила“ за играчот ќе биде разгледуван како основен пример кој подоцна ќе се примени и врз другите параметри на сличен начин. Во примерот на параметарот „Сила“ во кој играчот фрла две зарчиња и додава вредност 15 за да ја добие конечната вредност на параметарот „Сила“ може да се употреби минималната и максималната вредност на случајно генерираната нумеричка вредност за да се симулира фрлање на зарчиња. Минималната вредност секогаш ќе биде 15 плус минималната вредност од исходот на фрлање на две зарчиња која е 2, вкупно 17. Максималната вредност ќе биде 15 плус максималната вредност од исходот на фрлање на две зарчиња која е 12, вкупно 27. Исходот ќе биде во рамките на правилата. Овој метод може да се примени на сите останати параметри од овој вид.

Сите предмети кои играчот може да ги добие во текот на играта се дефинираат со едно својство. Овие својства имаат една од двете возможни вредности - или 0 или 1. Ако предметот е во посед на

играчот тогаш неговата вредност е 1, а ако предметот не е во посед на играчот тогаш неговата вредност е 0.

На сличен начин како и предметите можат да се третираат и настаните кои се случиле. На пример, една битка меѓу играчот и чудовиште може да се третира како завршена ако својството специфично дефинирано за битката добие вредност 1, наместо вредност 0 која ја има кога битката не е завршена.

На овој начин се третираат и предметите кои играчот може да ги користи и нивните ефекти кои можат да траат и неколку рунди. Ако некој ефект во текот на битката може да трае 3 рунди тогаш во моментот кога играчот ќе го „активира“ предметот, својството поврзано за ефектот на предметот добива вредност 3, која со секоја наредна рунда во текот на битката се намалува.

## 13.2 Имплементирање на врски меѓу состојбите

Секој сегмент во приказната на Протеус е нумерички обележан. Овие сегменти кои ги содржи Протеус можат да се третираат како една состојба во рамките на системот за нелинеарни приказни. Сите енумериирани сегменти се една состојба, но тоа не се сите состојби во рамките на системот. Почетните параграфи кои го воведуваат играчот во патешествијата на Протеус се исто така третирани како состојби. Изборот на предмети кои играчот ќе ги понесе со себе се, исто така, состојби. Секоја битка која играчот ја има со чудовиштата е составена од мношто на состојби кои се повторуваат. Овој тип на поврзување е специјален тип и ќе биде разгледуван во наредните поглавја.

Секој нумериран сегмент кој не е крај на патешествието најдолу содржи можни избори кои може да ги донесе играчот во текот на патешествието. Овие избори се користат како основа за создавање на изразите за достапност и за создавање на изјавите за дијалог за секоја состојба. На пример, ако нумеририранот сегмент 12 дефиниран со состојбата sBookmark12 го содржи нумеририранот сегмент 42 дефиниран со состојбата sBookmark42 во листата на можни избори, тогаш состојбата sBookmark42 во својот израз за достапност го има изразот IS\_HAPPENING sBookmark12. Ова значи дека кога состојбата

sBookmark12 е активна дека следниот можен избор ќе биде состојбата sBookmark42. На овој начин зависностите меѓу сите нумериирани сегменти се создаваат во системот за нелинеарни приказни.

Во некои специјални случаи, покрај зависноста од некоја состојба на играчот ќе му биде потребно и поседување на некој од предметите, па така во тој случај изразот на достапност ќе биде:

```
(  
    (IS_HAPPENING sBookmark24 )  
    AND  
    (Item_CrystalKey IS_EQUAL_TO 1)  
)
```

Ако изборите на повеќе состојби водат до некоја состојба тогаш изразот за достапност ќе содржи израз сличен на изразот:

```
(IS_HAPPENING sBookmark35 )  
OR  
(IS_HAPPENING sBookmark44 )  
OR  
(IS_HAPPENING sBookmark193 )
```

Ова се основните начини на имплементација, меѓутоа постојат и покомплексни случаи кои ќе бидат разгледувани во наредните поглавја.

### 13.3 Специфични случаи на глобални состојби

Постојат неколку глобални состојби во случајот на Протеус 01. Едната состојба е состојбата која му овозможува на играчот да јаде за да ја регенерира „Силата“. Играчот е во можност да јаде во било кој момент освен на почетокот кога сè уште ги нема добиено парчињата храна и во текот на битка со чудовиште. Изразот на достапност на оваа состојба зависи од својството за достапна храна, па така ако играчот ја потроши достапната храна преку активирање на оваа состојба, состојбата нема да биде достапна.

Втората состојба е состојбата која станува достапна во моментот кога играчот ќе ја изгуби во целост „Сила“. Оваа состојба се

вика sDeathStrength. Состојбата sDeathStrength е единствената состојба која е ексклузивно достапна и ги исклучува глобалните состојби. Тоа значи дека кога оваа состојба е достапна е единствената достапна состојба и ги игнорира останатите глобални состојби ако се достапни, како што на пример е состојбата преку која играчот може да јаде.

### 13.4 Специфични случаи на исклучни состојби

Исклучните состојби се состојби во кои глобалните состојби не се достапни. Примери за вакви состојби се: sFight, sNegativeFight, sPositiveFight и други. Овие состојби ги исклучуваат глобалните состојби бидејќи глобалните состојби ја содржат состојбата за регенерирање на „Силата“ која според правилата не треба да биде дозволена додека битката трае. Исклучни состојби се и состојбите кои се крај на патешествието бидејќи тогаш играчот или не може да продолжи или веќе го завршил патешествието.

### 13.5 Имплементирање на состојби кои се повторуваат

Состојби кои се повторуваат во случајот на Протеус 01 се состојбите во кои играчот е во битка со некое чудовиште. Во секоја битка има повеќе рунди, но бројот на рундите не е конечен, односно рундите се одвиваат се додека еден од учесниците не ја загуби целата „Сила“ која ја поседува.

За оваа цел има една почетна состојба која се нарекува sFight и во која секој од учесниците ја добива својата „моќ за борба“ во форма на специјално својство. Ако битката е почната тогаш состојбата sFight е активна. Во зависност од споредбата меѓу „моќ за борба“ на играчот и чудовиштето можат да бидат три достапни состојби. Првата состојба која се вика sPositiveFight во својот израз на достапност го има следниот израз:

```
(IS_HAPPENING sFight)  
AND  
(Score_FightingPower
```

IS\_MORE\_THAN  
Creature\_FightingPower )

Втората состојба се вика sNegativeFight и го има следниот израз во својот израз за достапност:

(IS\_HAPPENING sFight)  
AND  
(Score\_FightingPower  
IS\_LESS\_THAN  
Creature\_FightingPower)

И конечно ако „моќ за борба“ е еднаква и за играчот и за чудовиштето ја имаме состојбата за еднаквост која го има следниот израз за достапност:

(IS\_HAPPENING sFight)  
AND  
(Score\_FightingPower  
IS\_LESS\_THAN  
Creature\_FightingPower)

Преку овие три состојби се покриени сите можи исходи на битката. Состојбата sPositiveFight во својот израз за промена ја менува „Силата“ на чудовиштето за -2 поени. На ист начин, состојбата sNegativeFight во својот израз за промена ја менува „Силата“ на играчот за -2 поени.

Кога „силата“ на играчот ќе се спушти под 0 поени или е еднаква на 0 поени единствената состојба која ќе остане достапна е состојбата sDeathStrength. Ако „силата“ на чудовиштето се спушти под 0 единствена состојба која ќе биде достапна ќе биде состојбата која следи по борбата според упатството во нумериралиот сегмент. Ова се постигнува преку специјално својство кое следи дали битката е завршена во комбинација со својството за силата на чудовиштето. Изразот на достапност во ваква состојба ќе биде:

(  
    (Fight\_18 IS\_EQUAL\_TO 1)  
    AND  
    (Creature\_Strength IS\_LESS\_THAN 0.1)  
)  
OR

```
(  
    (Fight_85 IS_EQUAL_TO 1)  
    AND  
    (Creature_Strength IS_LESS_THAN 0.1)  
)  
OR  
(  
    (Fight_92 IS_EQUAL_TO 1)  
    AND  
    (Creature_Strength IS_LESS_THAN 0.1)  
)  
OR  
(  
    (Fight_118 IS_EQUAL_TO 1)  
    AND  
    (Creature_Strength IS_LESS_THAN 0.1)  
)
```

Конечно, оваа состојба во своите изрази за промена ќе ги содржи сите својства со кои се следи дали битката е завршена и нивните вредности ќе ги промени во 0.

## 14 Proteus 01 во Unreal Engine 5

Конечната цел на овој труд е да се најде начин кој ќе овозможи нелинеарен систем со разгранивачки наратив направен со слободен метод, да се интегрира во некој современ софтвер за развој на игри и да може преку интеракција со внес од периферните единици да се следи наративот на Proteus 01 во Unreal Engine 5.

Во главата „Интеграција со Unreal Engine 5“ беше дискутирано како може да се создаде интеграција на системот за нелинеарни приказни во Unreal Engine 5, а во ова поглавие ќе се искористи таа интеграција за да се имплементира интеракцијата со Proteus 01 во 3D сцена, создадена со Unreal Engine 5.

3D сцената во Unreal Engine 5 ја рефлектира ерата во која беше создаден Proteus 01. Содржината која е искористена за да се создаде сцената е создадена од Epic Games и е достапна комплетно бесплатно. Конфигурацијата на сцената е специјално создадена за потребите на овој труд.

Приказната Proteus 01 се вчитува во уредувачот на Unreal Engine 5 како NSS бинарна датотека. За интеракција со приказната се создадени специјални Нацрти (анг. Blueprints) кои овозможуваат внес од периферните единици како тастатура и глушец.

За интеракција со светот во играта се користат левиот и десниот клик на глушецот. Приказната се прикажува на мониторите кои се поставени во сцената. Интеракцијата со мониторот се избира кога камерата е насочена кон мониторот, а се притиска на левото копче на глушецот. За излез од интеракцијата се користи десното копче на глушецот. За избор на дијалог опција на еcranот на мониторот се впишува број и се стиска на копчето Ентер на тастатурата.

Откако интеракцијата ќе биде избрана, изборот се пропагира до системот за нелинеарни приказни, се процесира, а резултатот се враќа на еcranот на мониторот.

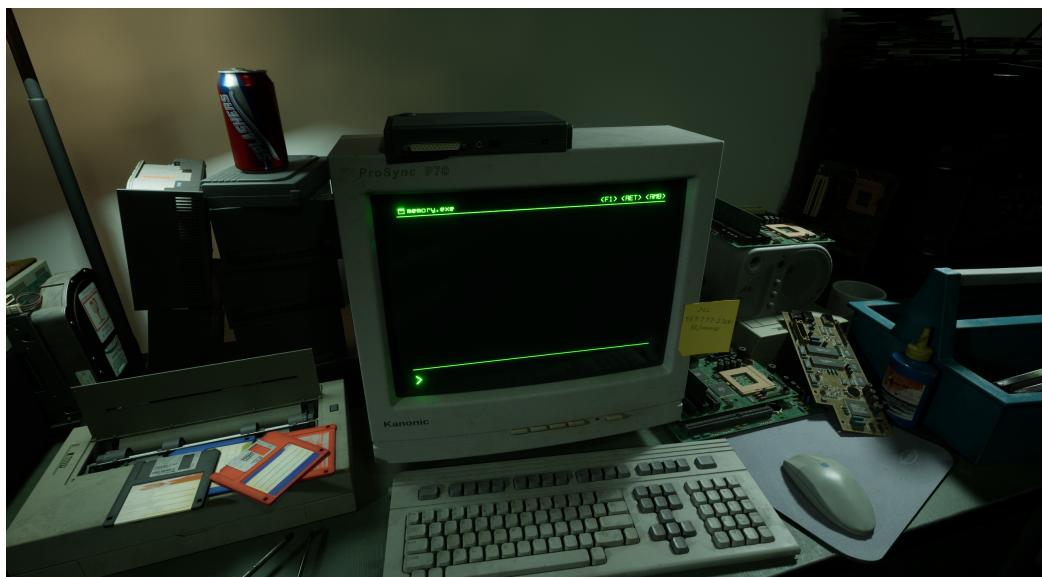
Интеракцијата со состојбите во системот може да биде имплементирана на различни начини, во зависност од потребите на видео играта во која се имплементира. Таа може да биде физичка, со

влез во одреден простор, со таймер кој активира состојба откако ќе истече, со интеракција на кориснички интерфејс итн. Системот за нелинеарни приказни е агностичен во однос на начинот на интеракција.

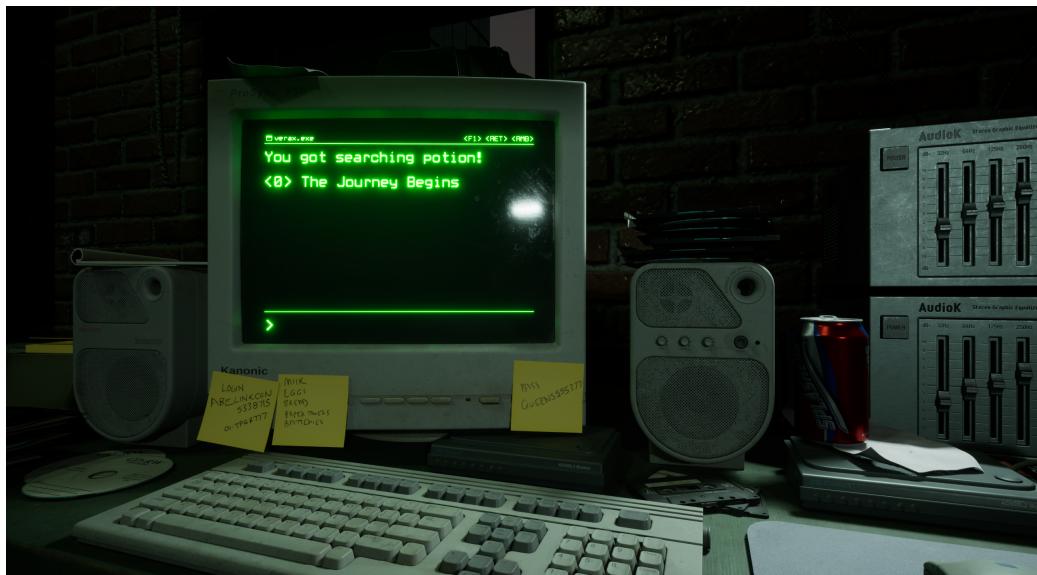
На овој начин е реализиран комплетниот циклус на интеграција на системот за нелинеарни приказни во софтверот за развој на видео игри Unreal Engine 5. На сличен начин интеграцијата може да се направи и во други софтвери за развој на видео игри, но и во останати софтвери кои не се поврзани со видео игри.



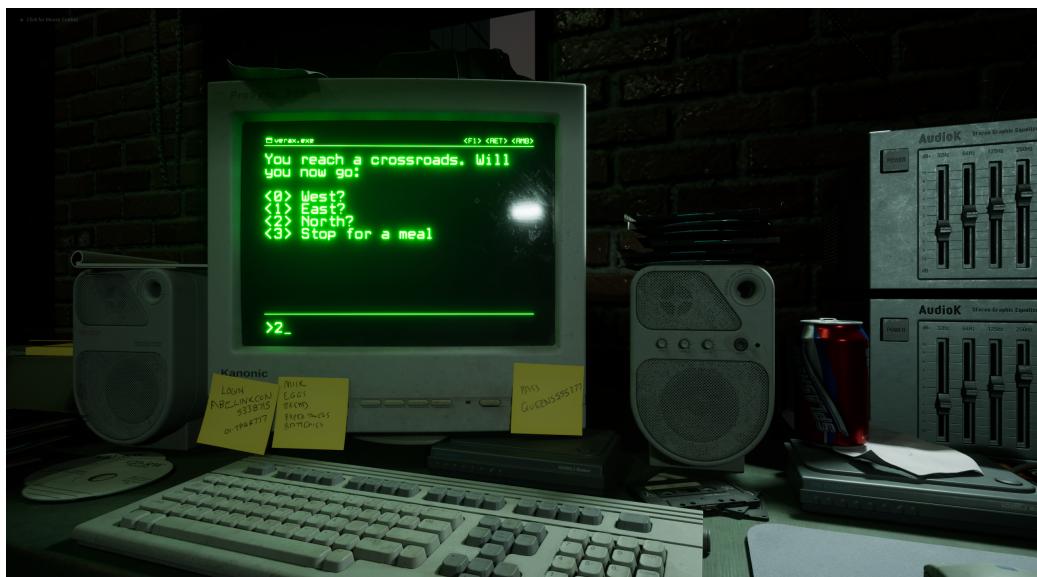
Слика 28: Статична слика од имплементацијата на Proteus 01 во Unreal Engine 5



Слика 29: Статична слика од имплементацијата на Proteus 01 во Unreal Engine 5



Слика 30: Статична слика од имплементацијата на Proteus 01 во Unreal Engine 5



Слика 31: Статична слика од имплементацијата на Proteus 01 во Unreal Engine 5

## 15 Примена

Примената на овој систем не е ограничена само на видео игри и не е ограничена само на формата текст. Формата текст е погодна за едноставно демонстрирање на можностите кои ги нуди овој начин на имплементирање на мотор за нелинеарни системи, меѓутоа таа форма може многу лесно да се замени со други мултимедијални форми како што се видео, тродимензионален простор, аудио итн.

Во претходните поглавја беше анализирано како е можно состојбите на системот да бидат надоградувани за да можат да содржат било кој формат на податоци и да имаат било каква улога во рамките на системот. Во ова поглавје ќе бидат анализирани начините на кои состојбите во системот би можеле да бидат надоградени за да се постигнат различни цели во различни области.

### 15.1 Примена во литература

Примерот на Протеус 01 кој е анализиран во овој труд е основен пример како системот за нелинеарни приказни може да се употреби за создавање на литературни дела. Со систем кој е имплементиран на начин описан во овој труд авторите можат на едноставен начин да вметнуваат логички правила кои ги поврзуваат сегментите во литературното дело. Литературното дело може да ја содржи било која форма на текст. Иако во овој труд формата е проза, на сличен начин системот може да се користи за пишување и во останатите литературни форми.

### 15.2 Примена во филм

За комплетна имплементација на системот за употреба во филм, единствено треба да се имплементира наследена класа на класата на состојбите која би имала инструкции кои се специфични за форматот на филмовите. Состојбата би имала можност да чува датотека во видео формат, која датотека би била прочитана и пуштена во моментот кога состојбата би се активирана во рамките на системот.

Филмови од категоријата на „Киноаутомат“, филмот кој беше анализиран во едно од претходните поглавја, можат многу лесно да се имплементираат со правилата на системот од овој труд. Филмот „Киноаутомат“ има неколку точки на разгранивање во кои публиката (или во модерно време, гледачот) може да донесе одлука во која насока да продолжи филмот. Целосната содржина на филмот пред и по точките на разгранивање може да се смета како една состојба во системот. Врските меѓу овие состојби во системот дефинираат што гледачите можат да одберат во една точка на разгранивање на системот.

На сличен начин може да се имплементира и прикажување на филмот „Црно Огледало: Бандерснеч“. Во овој филм како и во филмот „Киноаутомат“ постојат точки на разгранивање во кои гледачот може да направи некаков избор. Гледачот избира насока во филмот со својот далечински управувач. Содржините на филмот меѓу точките на разгранивање се состојби во системот, а врските меѓу нив би се правеле на стандарден начин. Системот е контролер-агностичен, па така било каков внес кој системот може да го регистрира може да го контролира изборот во системот. Во примерот на уредувачот на нелинеарни приказни изборот се прави со левиот клик на глушецот врз копче во корисничкиот интерфејс. Кога копчето ќе биде притиснато, уредувачот праќа сигнал до моторот за нелинеарни системи да ја активира соодветната состојба. Ова може да биде лесно заменето со друг начин на внес.

Предноста на системот дискутиран во овој труд е во тоа што состојбите се дискретни, па така при имплементацијата авторите не се грижат за директните врски меѓу состојбите туку се грижат само за врските на состојбата која ја уредуваат. Дополнителна предност е што во состојбите можат да користат својства на системот кои можат да рефлектираат некоја промена која е резултат на нивниот избор. Типичен пример е следење на расположението на некој лик, кој може да стане пријателски или агресивно расположен кон гледачот или останатите ликови во филмот ако гледачот ги прави изборите кои водат до тоа. Врз основа на тоа расположение на ликот можат некои избори да бидат овозможени или оневозможени и филмот да добие сосема поинаква насока.

### **15.3 Примена во образование**

Системот би можел да се користи и во образовниот процес како начин на учење преку тестирање. Еден пример за примена на системот во образовен процес е имплементација во која на ученикот преку системот би му биле понудени неколку избори за некое прашање или ситуација од кој еден или повеќе се точни, а некои од изборите би биле погрешни. Ако ученикот избере погрешно тогаш системот би прикажал обrazложение зошто тој избор е грешен, би го навел на точниот одговор, и крајно би го вратил ученикот повторно на истото прашање се додека не го избере точниот одговор чијашто состојба би нудела дополнително објаснување зошто тоа е точниот одговор. Со употреба на својствата на системот и изразите за промена на крајот би можел ученикот да се вреднува според изборите кои ги донел.

### **15.4 Примена во видео игри**

Системот описан во овој труд би можел да најде најширока примена во видео игрите бидејќи тие имаат најширок спектар на мултимедијални содржини кои можат да бидат искористени во еден нелинеарен систем. Видео игрите се комбинација од сите претходно споменати форми на содржини кои дополнително овозможуваат интеракција во виртуелен простор.

Најосновен пример за демонстрација е изборот на нивоа во видео игрите кој се базира врз разгранувања. Секој избор на ниво е едно разгранување кое може во системот да биде имплементирано како негова состојба. Ако играчот има 3 нивоа од кои може да избере тогаш тие 3 нивоа би биле имплементирани како состојби на нелинеарниот систем. Дополнително, ако некое од наредните нивоа зависи од комплетирањето на некое претходно ниво, ова многу лесно може да биде изразено преку врските во рамките на системот.

Јадрото и основните механики поврзани со нелинеарност во видео игри слични на форматот на игрите Crash Bandicoot, Jak and Daxter и The Stanley Parable, а дополнително и игри како претходно неспоменатите Uncharted (серијал), Life is Strange (серијал), Tacoma, Firewatch, Dr. Langeskov, The Tiger, and The Terribly Cursed Emerald,

The Witness, L.A. Noire, Subsurface Circular, би можеле комплетно да бидат имплементирани со користење на системот обработен во овој труд. Повеќето од гореспоменатите видео игри би можеле да бидат имплементирани само со користење на состојби, без употреба на својствата и дополнителните можности кои ги нуди системот.

Следствено, со имплементација на системот би можела да се создаде хипотетичка игра во која состојбите на системот би биле директно контролирани од дејствата на играчот (или недејствувањето на играчот) во виртуелниот простор на видео играта. Па така самото движење на играчот од еден простор во друг простор би можело да ја води состојбата на системот. Било какво дејство на играчот како притискање на копче, отворање на врата, интеракција со ликови кои не се контролирани од играчот (анг. NPC), па дури и создавање на звук при специфично движење или поглед во некоја насока би можело да доведе до промена на активната состојба на системот, а следствено на тоа и што играчот може да прави понатаму и како видео играта реагира на изборите на играчот. Можностите за примена на таков систем во видео игри се неограничени.

## 16 Заклучок

Овој труд покажува дека слободниот метод на имплементација на нелинеарни системи е ефикасен во имплементирање на нелинеарни наративи во видео игри. Преку комплетната имплементација на студијата на случај Proteus 01, и негова интеграција во модерен софтвер за развој на видео игри како што е Unreal Engine 5, на методот описан во овој труд се прикажува начинот на кој може да биде имплементиран било кој нелинеарен систем од тој тип, користејќи го овој метод.

Самиот метод на имплементација не е доволен за да ја ублажи комплексноста која произлегува од природата на нелинеарните системи. За таа цел потребни се авторски алатки кои дополнително помагаат во анализата на нелинеарните системи. Алатките кои беа цел на имплементација на овој труд овозможија ефикасно имплементирање на студијата на случај Proteus 01 од страна на автор кој има техничка подготвеност. Потребно е дополнително истражување и развој на алатките за автори кои не се технички подгответи.

Слободниот метод на имплементација за автори кои не се технички подгответи делува контраинтуитивно при првото соочување со него, но потребни се дополнителни студии дали при користење во подолг период овој метод е поефикасен од секвенционалниот метод кај автори кои не се технички подгответи. Кај технички подгответи автори слободниот метод се покажа како поефикасен отколку секвенцијалниот метод бидејќи предноста од дискретно анализирање на состојбите овозможува дифузирање на комплексноста при работа со системи кои имаат поголем број на состојби.

Имплементацијата на нелинеарните системи отвара свет на можности кој допрва треба да се истражува. Овој труд нуди одговори на само мал дел од прашањата поврзани со имплементација на нелинеарните системи. Комплетно имплементираната студија на случај Proteus 01 е доказ дека оригиналниот метод описан во овој труд може да најде примена при имплементирање на комплетни нелинеарни системи.

## Референци

- [1] Espen J. Aarseth. *Cybertext: Perspectives on Ergodic Literature*. JHUP; UK ed. edition, 1997. ISBN: 978-0801855795.
- [2] Andrew Rollings. Ernest Adams. *Fundamentals of Game Design*. Prentice Hall, 2006. URL: [https://wps.prenhall.com/bp\\_gamedev\\_1/54/14053/3597646.cw/index.html](https://wps.prenhall.com/bp_gamedev_1/54/14053/3597646.cw/index.html).
- [3] Tim Bostan Barbaros; Marsh. *Fundamentals of interactive storytelling*. Online Academic Journal of Information Technology, 2012.
- [4] Javier de Canete. Galindo Cipriano. Garcia-Moral Inmaculada. *System Engineering and Automation: An Interactive Educational Approach*. Springer, 2018. ISBN: 978-3642202292. URL: <https://books.google.com/books?id=h8rCQYXGGY8C&q=most+systems+are+inherently+nonlinear+in+nature&pg=PA46>.
- [5] Dani Cavallaro. *Anime and the visual novel: narrative structure, design and play at the crossroads of animation and computer games*. McFarland & Company, 2010. ISBN: 0-7864-4427-4.
- [6] Michael Cowan. *Interactive media and imperial subjects: Excavating the cinematic shooting gallery*. NECSUS. European Journal of Media Studies, 2010. URL: <https://doi.org/10.25969/mediarep/3438>.
- [7] William Grimes. “When the Film Audience Controls the Plot”. In: (1993). URL: <https://www.nytimes.com/1993/01/13/movies/when-the-film-audience-controls-the-plot.html?pagewanted=all&src=pm>.
- [8] Chris Hales. *Cinematic interaction: From kinoautomat to cause and effect*. Digital Creativity, 2005. DOI: 10.1080/14626260500147777.
- [9] Larry Hardesty. “Explained: Linear and nonlinear systems”. In: (2010). URL: <https://news.mit.edu/2010/explained-linear-0226>.
- [10] Ursula K. Heise. *Chronoschisms: Time, Narrative, and Postmodernism*. Cambridge University Press, 1997. ISBN: 0-521-55544-2.
- [11] Edward N. Lorenz. “Deterministic Nonperiodic Flow”. In: *Journal of the Atmospheric Sciences* (1963).
- [12] Dr Warren Smith. Dr Jamal Uddin. Dr Alexandra Tzella. Dr John Meyer. “Nonlinear Systems”. In: (2018). URL: <https://www.birmingham.ac.uk/research/activity/mathematics/applied-maths/nonlinear-systems.aspx>.

- [13] Peter Rubin. In: (2018). URL: <https://www.wired.com/story/black-mirror-bandersnatch-interactive-episode/>.
- [14] Ian Willoughby. “Groundbreaking Czechoslovak interactive film system revived 40 years later”. In: (2007). URL: <https://english.radio.cz/groundbreaking-czechoslovak-interactive-film-system-revived-40-years-later-8607007>.
- [15] Jane Douglas Yellowlees. *The End of Books—or Books Without End?: Reading Interactive Narratives*. University of Michigan Press, 2001. ISBN: 0472088467.