Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/20BE34Bursac

ПРОГРАМСКИ МОДЕЛ ЗА ГЕНЕРИСАЊЕ ИЗАЗОВА МАКСИМАЛНЕ ИНКЛУЗИВНОСТИ У ВИДЕО ИГРАМА ЖАНРА „ТРКАЧ“

A PROGRAMMING MODEL FOR GENERATING CHALLENGES OF MAXIMUM INCLUSIVENESS IN „RUNNER“ VIDEO GAMES

Василије Бурсаћ, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Иако се тематски могу веома разликовати, све видео игре жанра „Тркач“ (енгл. Runner) одликује идентична механика игре. Имајући ово у виду, направљена је генеричка програмска основу која би се могла искористити као полазна тачка за брзи развој видео игара овог жанра. Овај програмски модел је обогаћен системом за пружање аудио-визуелно-хаптичких повратних информација играчу. Захваљујући овом систему, постигнут је висок степен инклузивности и "увучености" играча у виртуални свет видео игре. У овом раду је описан један такав програмски модел развијен у Јунити (енгл. Unity) погону, као и његове могућности и функционалности.*

Кључне речи: *Програмски модел, Видео игре жанра „Тркач“, Развој видео игара, Имерсивност, Хаптичка повратна спрега*

Abstract – *Although they can be very different thematically, all video games of the „Runner“ genre are characterized by identical game mechanics. With this in mind, a generic programming base was created that could be used as a starting point for the rapid development of video games of this genre. This programming model is enriched with a system for providing audio-visual-haptic feedback to the player. Thanks to this system, a high degree of inclusivity and "immersion" of the player in the virtual world of the video game was achieved. This paper describes one such programming model developed in Unity engine, as well as its capabilities and functionality.*

Keywords: *Programming model, Runner video games, Video game development, Immersiveness, Haptic feedback*

1. УВОД

Видео игре жанра „Тркач“ (енгл. *Runner*) одликује веома једноставна контрола која их чини погодним за играње на различитим уређајима. Захваљујући механици која је заједничка за све игре овог жанра, постоји могућност за развијање генеричке основе, тј. шаблона за развој оваквих игара.

Ово би могло значајно да убрза процес њиховог развоја, али и да омогући дизајнерима игара да прилагоде игру одређеним захтевима без програмерског предзнања и измене постојећег програмског кода.

У овом раду ће бити описан управо један такав програмски модел, развијен помоћу Јунити (енгл. *Unity*) погона за развој рачунарских игара. При изради овог програмског модела, фокус је био на постизању високог степена проширивости, ефикасности и флексибилности употребе. Самим тим, тежило се максималном поштовању конвенција, добрих пракси и принципа за писање чистог и одрживог кода.

Овај програмски модел је функционалан и без икаквих додатних измена, па и он, сам по себи, представља видео игру коју је могуће покретати на различитим платформама. Тренутно је овај програмски модел подржан на рачунарима са Виндоуз (енгл. *Windows*), МекОС (енгл. *MacOs*) и Линукс (енгл. *Linux*) оперативним системима, а могуће га је покретати и у свим модерним интернет претраживачима. Уз веома мале измене у систему и програмској скрипти за контролу и обраду уноса играча, овај програмски модел може бити прилагођен екранима осетљивим на додир, те покретан и на мобилним уређајима са Андроид (енгл. *Android*) и иОС (енгл. *iOS*) оперативним системима.

2. ПРОЦЕДУРАЛНО ГЕНЕРИСАЊЕ СТАЗЕ

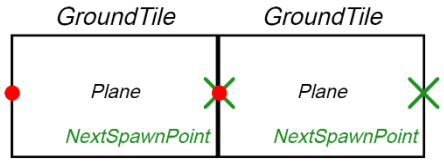
За видео игре овог жанра је карактеристично то што не постоји предефинисан ниво са унапред одређеним препрекама, већ је карактер којим играч управља приморан да трчи током неодређеног временског интервала и избегава препреке. Циљ играча је да достигне што већи број поена и преживи што дуже или да испуни неки задатак и достигне одређени циљ. То значи да играње једне партије игре на одређеном нивоу може да траје дужи временски период, те је неопходно омогућити програмско генерисање подлоге по којој играч трчи.

Да би се ово постигло, направљени су појединачни сегменти који представљају делове подлоге, појединачне препреке и ствари за сакупљање, а од којих ће се затим програмски генерисати стаза. Ови сегменти се чувају као префабриковани објекти (енгл. *Prefabs*) класе *GameObject* и могуће их је подесити и поново употребити колико год пута је то потребно.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Драган Иветић, ред. проф.

2.1. Сегменти подлоге

Појединачни сегмент подлоге је назван *GroundTile* и он се састоји од два повезана објекта – од равни (енгл. *Plane*) по којој се играч креће и празног објекта који означава где треба да се позиционира следећи сегмент подлоге (назван *NextSpawnPoint*). На слици 1 је црвеним кругом приказан пивот сегмента подлоге, а он је референтна тачка за њено позиционирање. Дакле, када се позиција једног сегмента подлоге позиционира у *NextSpawnPoint* тачку претходног сегмента подлоге, равни по којима играч треба да се креће ће се потпуно поравнати и наставити једна на другу, као на слици 1. На равни које чине сегменте подлоге су текстурисањем постављене слике камене стазе. Ова текстура се једноставно може заменити другом сликом или се на раван могу поставити различити 3Д објекти и на тај начин се може креирати другачији изглед стазе и амбијента.



Слика 1. *Садржај једног сегмента подлоге и међусобно повезивање више оваквих сегмената*

2.2. Препреке и вероватноће њиховог појављивања

На крају сваког сегмента подлоге се налазе по три празна објекта који су равномерно распоређени и истих су димензија, те покривају једнаке делове стазе. Дакле, у свакој траци стазе се налази по један празан објекат. Препреке се такође чувају као префабриковани објекти, те се након генерисања насумично позиционирају у ове празне објекте.

Ради разноврсности, те додатног отежавања игре, омогућено је постојање више различитих типова препрека које ће се појављивати са различитом вероватноћом у зависности од тренутног нивоа и резултата играча. Како би се то омогућило, имплементирана је класа *SpawnableObstacle*. Ова класа садржи префабриковани објекат типа *GameObject* који представља 3Д модел препреке, вероватноћу за појављивање датог објекта, као и горњу и доњу границу вероватноће за појављивање тог објекта.
Скрипта која управља генерисањем стазе (*GroundSpawner*) садржи низ објеката *SpawnableObstacle* класе, што омогућава лако додавање нових типова препрека. Довољно је у Јунити Инспектору (енгл. *Unity Inspector*) у низ превући одговарајуће префабриковане објекте за препреке, као и унети вероватноће њиховог појављивања. За ово није потребно никакво програмерско знање, па дизајнери игре могу једноставно да формирају префабриковане објекте препрека и унесу њихове вероватноће појављивања, преко корисничког интерфејса Јунити едитора.

Тренутно у игри постоје три типа препрека које су направљене у демонстративне сврхе. Ове препреке су креиране од 3Д модела коцке различитих димензија на који је потом постављена текстура у виду слике. Приликом дизајна игре, ову текстуру је могуће заменити другом или се комплетан 3Д модел препреке може заменити другим, а могуће је креирати још неограничено много типова препрека.

2.3. Ствари за сакупљање

Ствари за сакупљање се могу опционо генерисати на сегментима подлоге и број генерисаних ствари за сакупљање по сегменту подлоге је могуће програмски променити. У игри тренутно постоји само један тип ствари за сакупљање, а то су дрвене цепанице.

2.4. Потенцијални проблем и оптимизација

У играма овог жанра се константно креира и уништава велики број објеката. Ово значајно оптерећује хардверске ресурсе уређаја на којима се игра извршава. Ради бољег управљања меморијом врши се њено ослобађање од објеката који се више не користе (енгл. *garbage collection*). Када год Јунити треба да изврши „сакупљање смећа“, он прекида извршавање програмског кода и наставља нормално извршавање тек када је процес ослобађања меморије завршен [1]. Самим тим, број приказаних фрејмова по секунди опада.

Како би се оптимизовало коришћење ресурса и спречио наведени проблем, у овом програмском моделу је примењен образац удруживања објеката (енгл. *object pooling pattern*). Овај пројектни образац уместо сталног креирања и уништавања, користи колекцију већ иницијализованих објеката спремних за коришћење [2]. Након што се престане са коришћењем објекта, он се не уништава, већ се враћа у колекцију. У наставку ће бити описана примена овог обрасца приликом генерисања стазе са препрекама и стварима за сакупљање.

2.5. Генерисање стазе

Генерисање сегмената стазе са препрекама и стварима за сакупљање контролише скрипта *GroundSpawner*. У овој скрипти се за сваки тип објеката који се генеришу налази по једна пулинг колекција објеката за реализацију *object pooling* обрасца. Будући да у игри постоји више типова препрека, праве се појединачне колекције за сваки о тих типова препрека, при чему су све те колекције налазе обједињене у једном речнику. Елементи овог речника су парови кључ-вредност, где је кључ назив префабрикованог објекта одређене препреке, а вредност пулинг колекција префабрикованих објеката тог типа. Речник је веома повољна структура података за имплементацију оваквог система, јер он омогућава константно време приступа елементима, без обзира на број елемената у њему.

При покретању игре, врши се пролазак кроз низ са типовима препрека и формира се поменути речник. Ово нема значајан утицај на перформансе, јер се ради само на почетку, при првом покретању игре, а низ са свим типовима препрека садржи релативно мало елемената. На овај начин је омогућено генеричко руковање са препрекама, без обзира на број њихових различитих типова.

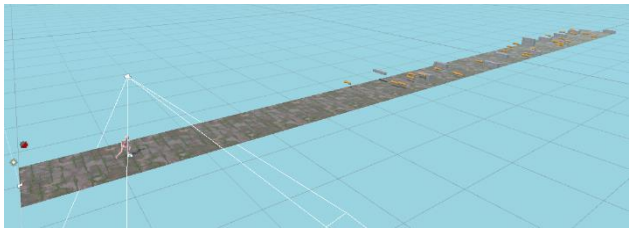
На почетку игре се генерише одређени број сегмената, како корисник не би могао да види генерисање

подлоге испред себе. Поред очигледне естетске предности, ово побољшава и употребљивост, јер ће на овај начин корисник увек моћи да се ментално припреми за препреке које га очекују на наредним сегментима. Из истог разлога, првих неколико сегмената је генерисано без препрека.

Око читавог *GroundTile* објекта се налази *BoxCollider* компонента којој је укључено *IsTrigger* својство. Када играч напусти *BoxCollider* одређеног сегмента подлоге, догађа се следеће:

1. Из пулинг колекције се извлачи нови сегмент подлоге.
2. На основу вероватноћа се одлучује који тип препреке ће бити генерисан. Препрека тог типа се извлачи из одговарајуће пулинг колекције и поставља се на сегмент подлоге.
3. Из пулинг колекције са стварима за сакушљаање се извлачи тражени број објеката и они се постављају на сегмент подлоге.
4. Формирани сегмент подлоге се поставља на крај постојеће стазе.
5. Пар секунди од напуштања одређеног сегмента стазе, тај сегмент стазе, препрека и ствари за сакупљање које су биле на њему се уклањају са сцене и враћају у одговарајуће пулинг колекције.

Оваква имплементација генерисања подлоге омогућава да у сваком тренутку на сцени буде приказан приближно константан број сегмената подлоге, те се приказује само онолико сегмената колико је потребно, као што је приказано на слици 2.



Слика 2. *Дужина стазе у једном тренутку играња*

3. КОНТРОЛА И КРЕТАЊЕ ИГРАЧА

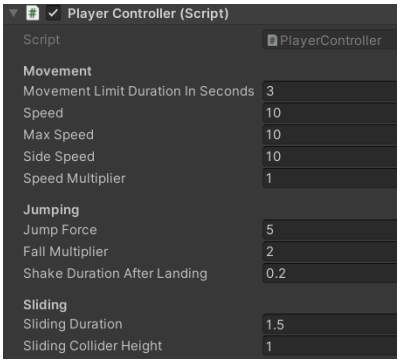
При развоју овог програмског модела коришћен је Јунитијев нови систем за обраду улаза (енгл. *Input System*). Захваљујући томе, мапирање свих контрола на тастатури или контролеру је могуће променити унутар *InputActions* објекта. Преузимање уноса корисника је извршено у скрипти *InputHandler*. Све скрипте које зависе од уноса корисника поседују референцу на ову скрипту, што у великој мери олакшава руковање контролама игре.

Игре развијене на основу овог програмског модела се на тренутно подржаним платформама могу контролисати користећи тастатуру или контролер за видео игре.

Контролер карактера је базиран на физици и налази се у скрипти *PlayerController*. Из тог разлога, читав код задужен за кретање карактера је смештен у *FixedUpdate* функцију. Ова функција се позива у фиксном временском интервалима по којем ради и Јунитијев погон за физику (енгл. *Unity Physics Engine*), што омогућава глатко и физички правилно кретање карактера.

Преузимање уноса корисника се врши у *Update* функцији, јер се она позива сваки фрејм, а не у фиксним интервалима, па не постоји ризик да неки унос корисника не буде регистрован.

Мноштво поља из скрипте *PlayerController* је изложено у инспектору, па се дизајнерима игре нуди широк спектар могућности за модификацију понашања карактера којим играч управља. Нека од тих поља су приказана на слици 3.



Слика 3. *Нека од својстава скрипте PlayerController која су доступна за измену у инспектору*

3.1. Оптимизација кода за кретање

Програмски код за кретање карактера се ослања на множење тродимензионалних вектора са брзинама кретања карактера по осама тродимензионалног координатног система и протеклим временом између два суседна позива *FixedUpdate* функције. Ове операције су комутативне, па је њихов редослед у овом програмском моделу такав да се увек изврши најмањи могући број операција потребних да се постигне одређени резултат. Дакле, прво се измноже сви скалари, па се затим добијени резултат помножи са векторима.

3.2. Модел и анимације карактера

3Д модел карактера и анимације коришћене у игри су преузете са сајта Миксамо (енгл. *Mixamo*) [3]. Будући да се померање 3Д модела врши програмски, било је неопходно да се све анимације карактера извршавају у месту. Због тога су са преузетих анимација обрисани фрејмови на којима је померање 3Д модела по z-оси.

4. КОНТРОЛА МУЗИКЕ И АУДИО ЕФЕКАТА

Контрола звучних ефеката у игри се врши помоћу *AudioManager* скрипте која је постављена на истоимени празан објекат у сцени. У овој скрипти се налазе референце на аудио изворе за репродукцију музике и звучних ефеката, као и имплементације различитих функција за контролу аудио репродукције. Ова скрипта је синглтон (енгл. Singleton), што значи да јој се може глобално приступати и да постоји само једна њена инстанца [4]. На тај начин је у великој мери олакшано коришћење аудио-клипова из различитих делова видео игре.

5. ПОЛОЖАЈИ И КОНТРОЛА КАМЕРЕ

У програмском моделу су имплементирани углови гледања, тј. камере из трећег (слика 4) и првог лица (слика 5) који у потпуности прате кретање играча. Могуће је променити активну камеру притиском на одређени тастер тастатуре или контролера.

Слика 4. *Уклизавање карактера гледано камером из трећег лица*

Слика 5. *Уклизавање карактера гледано камером из првог лица*

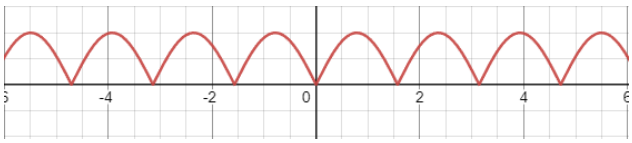
5.1. Потрес камере

У скрипти *CameraController* имплементиране су две функције за потрес камере у циљу повећања имерсивности приликом одређених радњи у игри. Обе функције су засноване на контроли амплитуде и фреквенције Перлиновог шума камере током одређеног временског периода. Прва функција омогућава добијање грубих потреса камере, па је искоришћена приликом ударца у препреку. У другој функцији се врши постепено умањење потреса камере помоћу интерполације вредности амплитуде шума, па је она искоришћена за симулацију суптилнијих потреса приликом доскока и уклизавања.

6. ХАПТИЧКА ПОВРАТНА СПРЕГА

У овом програмском моделу је тренутно имплементирано неколико различитих шаблона вибрације који контролишу вибрационе моторе у дршкама контролера и служе за пружање повратних информација. Неки од имплементираних шаблона су константна вибрација, линеарна вибрација, степенаста вибрација, таласаста вибрација и друге. Ове функције се налазе у скрипти *GamepadVibrationController* и помоћу њихових параметара је могуће фино подешавање одговарајућих шаблона вибрације контролера.

У случају овог програмског модела, вибрација контролера је примењена на неколико места. Константна вибрација различитог интензитета и трајања је примењена приликом доскока и при удару у препреку. Степенаста вибрација је примењена уз тајмер који се приказује пре почетка игре, па играч реко ње добија повратну информацију о преосталом времену до почетка игре чак и ако не гледа директно у екран. Приликом уклизавања примењена је посебна варијација таласасте вибрације, која је описана синусном функцијом приказаном на слици 6. Ова функција се показала веома погодном за симулацију трења и неравнина на подлози који се могу осетити и у стварном свету приликом оваквог вида кретања.



Слика 6. *Синусна функција коришћена за имплементацију варијације таласасте вибрације при уклизавању*

7. ЗАКЉУЧАК

Резултат рада је програмски модел који представља добру полазну тачку за развој видео игара жанра „Тркач“ са високим степеном имерсивности. Добијени програмски модел је проширив и прилагодљив.

Главни циљ у даљем развоју овог програмског модела јесте прилагођавање мобилним уређајима са екраном осетљивим на додир, уз евентуалну имплементацију подршке за контролу помоћу жироскопа.

У будућности се планира увођење нових типова сегмената стазе као што су скретања и раскрснице. Поред тога, постојећи програмски модел се може унапредити системом за генерисање окружења стазе, као и увођењем шејдера (енгл. *Shader*) који би се могао програмски контролисати, како би се створио ефекат закривљености стазе у неком смеру.

8. ЛИТЕРАТУРА

[1] Unity Technologies, „Garbage Collector Overview“, https://docs.unity3d.com/Manual/performance-garbage-collector.html, Званична документација о процесу ослобађања меморије (приступљено у септембру 2022.)

[2] Unity Technologies, „Introduction to Object Pooling“, https://learn.unity.com/tutorial/introduction-to-object-pooling, Званична документација са објашњењем обрасца удруживања објеката (приступљено у септембру 2022.)

[3] Adobe, Mixamo, https://www.mixamo.com/, Веб сајт за преузимање карактера и анимација (приступљено у септембру 2022.)

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Boston, Addison-Wesley, p. 127, October 1994.

Кратка биографија:

Василије Бурсаћ рођен је 1998. године у Новом Саду. Дипломирао је на Факултету техничких наука 2021. године. Мастер рад на Факултету техничких наука из области Електротехнике и рачунарства одбранио је 2022. године.