Usporedba programskih jezika Blueprint Visual Scripting i C++ u procesu razvoja računalne igre u Unreal engine okruženju

Ravnik, David

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

DAVID RAVNIK

USPOREDBA PROGRAMSKIH JEZIKA BLUEPRINT VISUAL SCRIPTING I C++ U PROCESU RAZVOJA RAČUNALNE IGRE U UNREAL RAZVOJNOM OKRUŽENJU

Diplomski rad

Pula, srpanj 2021.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

DAVID RAVNIK

USPOREDBA PROGRAMSKIH JEZIKA BLUEPRINT VISUAL SCRIPTING I C++ U PROCESU RAZVOJA RAČUNALNE IGRE U UNREAL RAZVOJNOM OKRUŽENJU

Diplomski rad

JMBAG: 03031416, redoviti student

Studijski smjer: Informatika

Kolegij: Dizajn i programiranje računalnih igara

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, srpanj 2021.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za magistra _____ ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom _____
_____

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Potpis

_____

U Puli, _____ (datum)

Sažetak

Razvojem sve većeg broja programskih proizvoda za razvoj video igara performanse istih postaju sve bitnije kako bi razvojna okruženja ostala konkurentna na tržištu. Unreal Engine 4 (razvojno okruženje) jedna je od platformi za razvoj računalnih igara i za razliku od većine razvojnih okruženja, sadrži 2 programska jezika unutar softvera koji se značajno razlikuju po procesu izrade, tijeku rada, brzini i kvaliteti izvedbe konačne igre.

Cilj ovog diplomskog rada je usporedba programskih jezika C++ i Blueprint u Unreal 4 razvojnom okruženju. U svrhu usporedbe izrađene su dvije računalne igre koje se ne razlikuju izgledom i rasporedom razina, već je jedina razlika u pozadini, odnosno u programskim jezicima koji pokreću mehanike igara. Kroz rad je dokumentiran i uspoređen razvoj igara u oba programska jezika, te su analizirane prednosti i nedostatci finalnih igara i uspoređene su njihove performanse.

Nakon analize broja slika u sekundi i utjecaja broja objekata na performanse igara u oba programska jezika vidljivo je da C++ programski jezik ima bolje performanse u finalnoj igri, a istovremeno je razvoj igre u ovom programskom jeziku brži i u mnogim slučajevima i jednostavan u odnosu na Blueprint programski jezik.

ABSTRACT

With the development and a growing number of software products for the development of video games, their performance is becoming increasingly important in order for game engines to remain competitive in the market. Unreal Engine 4 (game engine) is one of the platforms for computer game development and unlike most development environments, it contains 2 programming languages within the software that are significantly different from each other.

The aim of this thesis is to compare programming languages C ++ and Blueprint in the Unreal Engine 4 environment. For the purpose of comparison, two computer games were made that look identical visually and do not differ in layout of levels, the only difference is in the programming languages that drive the mechanics of the game. The comparison of programming languages contains the advantages of both languages and a comparison of the performance of the finished games.

After analyzing the number of frames per second and the influence of the number and type of objects on the performance of games for both programming languages, C ++ programming language has better performance in the final game, and at the same time the game development in this programming language is in the programming language.

SADRŽAJ

1 UVOD

Razvojem sve većeg broja programskih proizvoda (dalje u tekstu softvera) za razvoj video igara (*eng. game engine*), brzina, pouzdanost i funkcionalnost igara postaju sve bitnije kako bi ostali konkurentni na tržištu. Unreal Engine 4 (razvojno okruženje) jedna je od platformi za razvoj računalnih igara. Za razliku od većine razvojnih okruženja, Unreal razvojno okruženje ima 2 programska jezika unutar softvera koji se značajno razlikuju po procesu izrade, tijeku rada, brzini i kvaliteti izvedbe konačne igre. Oba programska jezika imaju prednosti i nedostatke, pa zbog toga i posebne uloge i zadatke.

U ovom će se radu usporediti programski jezici unutar razvojnog okruženja Unreal 4, te će biti prikazane uloge, prednosti i nedostaci programskih jezika. Također, izraditi će se dvije srodne računalne igre u ovom razvojnom okruženju. Za svaku izrađenu igru biti će korišten jedan od programskih jezika kako bi se usporedio proces izrade i analizirala brzina i kvaliteta izvedbe završne igre. Programski jezici o kojima je riječ su C++ i Blueprint visual scripting. C++ je tekstualni programski jezik, dok je Blueprint vizualni ili grafički programski jezik koji se temelji na bazi čvor i veza.

Unreal je razvojno okruženje za izradu video igara. Uz Unity, jedno je od najpopularnijih besplatnih razvojnih okruženja. Indie i AAA tvrtke za razvoj igara koriste Unreal razvojno okruženje ili modificiranu verziju Unreal-a 4 za razvoj svojih igara. Neke od najpopularnijih igara koje su napravljene pomoću ovog razvojnog okruženja su Final fantasy 7 remake i Street fighter 5. Razvojno okruženje Unreal 4 sadrži mnogo razvojnih alata i funkcionalnosti koje su bitne u razvoju računalnih igara (*eng. out of the box*). Osim toga, uz softver dostupan je cijeli Unreal C++ izvorni kod (*eng. source code*) koji omogućavaju specifične izmjene na softveru. U 2020. godini Unreal 4 razvojno okruženje je postao besplatan za sve korisnike koji ne prelaze zaradu od 1000000 američkih dolara ostvarenih od jednog proizvoda u jednom tromjesečju. Korisnici koji su prešli tu granicu morali su plaćati 5% prihoda za sve proizvode. Ovi podatci preuzeti su sa službenih web stranica Epic Games-a i Unreal Enginea (Epic games, 2004 - 2021).

Budući da Unreal razvojno okruženje nudi 2 programska jezika za razvoj računalnih igara, prije same izrade bitno je poznavati prednosti i nedostatke svakoga, kako bi odluka koji jezik koristiti prilikom razvoja računalne igre bila lakša. Pri kreaciji projekta u Unreal razvojnom okruženju korisnik bira između dvije ponuđene opcije, kreaciju Blueprint projekta ili C++ projekta, što je prikazano na slici 1. Blueprint visual scripting je vizualni programski jezik koji se temelji na C++ programskom jeziku, no razlikuje se od njega u procesu izrade i u brzini i kvaliteti gotove igre. U Unreal razvojnom okruženju C++ je objektno orijentiran programski jezik koji je nadograđen novim klasama, funkcijama i varijablama karakterističnima za ovo razvojno okruženje, kao na primjer varijablama koje sadrže komponente za korisničko sučelje ili komponente igre. Poznavanje API igre (*eng. application programming interface*) i okvirne klase (*eng. API gameplay and framework classes*) je bitno tijekom razvoja. API i okvirne klase su dio skoro svakog programa za razvoj računalnih igara i služe kao glavni alat koji omogućuje

komunikaciju između korisnika i razvojnog okruženja. Oni su dostupni na službenim web stranicama Unreal razvojnog okruženja zajedno sa detaljnom dokumentacijom.

Ovaj rad sadrži usporedbu programskih jezika u Unreal razvojnom okruženju u pisanom formatu, kao i praktične primjere računalnih igara koje će također biti uspoređene tijekom izrade same igre. Izrađene će igre biti srodne i vizualno vrlo slične. Glavna će razlika biti u načinu, kvaliteti i brzini (u daljnjem tekstu performanse) rada u konačnoj verziji računalne igre. Prilikom izrade u različitim programskim jezicima razlikovao se tijek programiranja koji se očituje u brzini, pouzdanosti, jednostavnosti i čitljivosti programskog koda. Budući da je cilj rada usporedba programskih jezika, igre će vizualno biti vrlo slične i neće postajati razlika u mehanici igara iz perspektive igrača, tj. korisnicima će one izgledati jednako.

Izrada računalnih igara dokumentirana je u svrhu bolje usporedbe i prepoznavanja razlika programskih jezika tijekom izrade rada i razvoja računalnih igara. Iako se igre izrađene ovim programskim jezicima razlikuju i u performansama, glavna razlika je vidljiva tijekom rada i u ulogama i zadacima oba programska jezika. Vizualni programski jezici, kao na primjer Blueprint, su obično više orijentirani početnicima, dok tekstualni programski jezici, kao na primjer C++, dopuštaju veću slobodu u radu i zbog toga zahtijevaju više iskustva i preciznosti.

Zahvaljujući razlikama i individualnim prednostima programskih jezika, Unreal razvojno okruženje ima veći broj potencijalnih korisnika ovisno o iskustvu programera, kompleksnosti igre, ciljanoj publici igre i slično.

2 UNREAL RAZVOJNO OKRUŽENJE

Razvoj računalnih igara može se podijelit na nekoliko faza kao što su: planiranje, pred proizvodnja, proizvodnja, testiranje i prodaja. U ovome se rada uglavnom radi o fazi proizvodnje s obzirom na to da je programiranje središnji dio tog koraka i Unreal razvojno okruženje je uglavnom usmjereno produkcijskom dijelu razvoja računalne igre. Produkcijski dio igre može se također podijeliti u nekoliko koraka, a to su: izrada, animacija i integracija 3D ili 2D modela, glazba, zvukovi okoliša i likova u audio dijelu, razvoj razina, te programski razvoj mehanike. Iako se 3D modeli, animacije, glazba i slično, često razmatraju kao posebna kategorija, bitan su dio razvoja računalnih igara u Unreal-u i u izradi igre surađuju sa programskim dijelom. Osim produkcijskog dijela u ovom radu se spominje i faza planiranja koja je prikazana kroz dizajn, i faza testiranja koja je prikazana kroz analizu konačnih performansi igara.

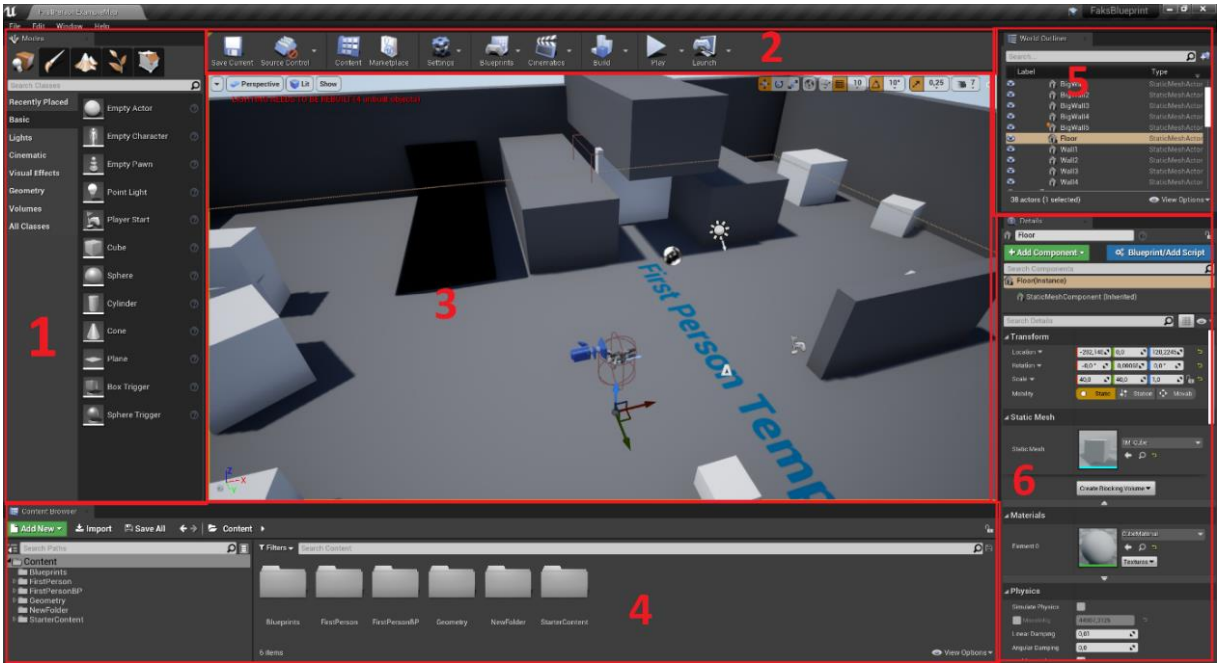
2.1 MODUL PROGRAMA ZA UREĐIVANJE RAZINA

Level editor ili Modul programa za uređivanje razina je uređivač koji služi kao početni dio u izgradnji računalne igre i razine (*eng. level*) u Unreal 4 razvojnom okruženju . U uredniku se gleda i uređuje razina tako da se dodaju, brišu i pomiču objekti i modeli unutar scene. Neki od objekata koji mogu biti u sceni su svjetla, kamere, likovi, flora, interaktivni objekti, solidni objekti i slično. Urednik se otvara odmah nakon kreacije projekta i centralni je dio izrade računalne igre. Osim za izradu i manipulaciju razina, urednik se također koristi za kreaciju, uvoz, izvoz, navigaciju i uređivanje raznih datoteka, kao i testiranje igara.

Unreal tijekom kreacije projekta korisniku nudi nekoliko baznih projekata koji služe kao temelj igre jer sadrže nekoliko baznih klasa koje služe bržem i jednostavnijem razvoju računalne igre. Izgled urednika razine ovisi o baznoj klasi. Standardni modul programa za uređivanje razina sastavljen je od šest prozora, kao što je prikazano na slici 1, te je on najfrekventniji među svim urednicima.

Glumac (*eng. actor*) je jedna od Blueprint i C++ baznih klasa. Objekt Blueprint klase glumca postavlja se unutar razine i ima lokaciju, rotaciju i veličinu. Svaki glumac može, ali i ne mora imati skriptu, a ona je tekstualna ili vizualna datoteka sa slijedom uputa. U Unreal-u se kompajlira u integriranom skriptnom kompajleru. Kako navode Jones i Liberty (2004), kompajler prevodi izvorni kod u posrednički oblik, što se naziva kompajliranjem i on stvara objektnu datoteku. Kompajler tada okida povezivač koji kombinira objektnu datoteku u izvršni program. Kompajler je program koji pretvara programske sljedove uputa u strojni jezik i izvodi ih. Skripte u Unreal razvojnom okruženju postavljaju se na glumce i druge Unreal klase. Kako ističe Ousterhout (1998), one dodaju klasama i komponentama klase sljedove uputa. Komponente su dizajnirane da budu za višekratnu upotrebu, a između njih postoje dobro definirana sučelja komponente i skripte koje olakšavaju upotrebu komponenata. Nakon kreacije

instance klase unutar igre skripta izvodi operacije inicijalizacije, a one su dio slijeda uputa koje su unesene u skriptu tekstualnim ili vizualnim putem.



Slika 1. Organizacija projekta u Unreal razvojnom okruženju

Slijedi kratki opis svakog prozora u standardnom modulu programa za uređivanje razina u Unreal 4 razvojnom okruženju po redoslijedu kao što je prikazano na slici 1 :

1. Načini rada (*eng. modes*) – prvi prozor, način rada, sadrži kartice: postavljanje (*eng. place*), slikanje (*eng. paint*), krajolik (*eng. landscape*), uređivanje prirode (*eng. foliage*) i geometrija (*eng. geometry*). Ovaj prozor sadrži alate za dodavanje i upravljanje 3D komponentama i terenom. Glavni je način izgradnje razina igre. Kartica „postavljanje“ sadrži objekte kao što su na primjer: obična kocka, cilindar, stepenice, svijetla, kamere, te efekte poput magle i slično, a oni se mogu vući i ispustiti (*eng. drag and drop*) u treći prozor koji se nalazi na centru urednika razine. Kartica „slikanje“ omogućuje korisniku podešavanje tekstura i materijala interno u Unreal aplikaciji na brz način. Teksture u Unreal razvojnom okruženju su dvodimenzionalne slike koje pokrivaju površinu trodimenzionalnog modela. Kartica „slikanje“ omogućuje manipulaciju tekstura u trodimenzionalnom prostoru, a uz to omogućuje i dodavanje kistova koji se mogu koristiti u kartici krajolika za manipulaciju krajolika. „Krajolik“ kartica dodaje teren koji se kasnije može manipulirati. Dodavanje terena je jedan od načina na koji je moguće stvarati i oblikovati razine. Drugi je način stvaranje terena pomoću statičnih modela. Za stvaranje velikih razina i krajolika teren je obično bolji izbor od statičkih objekata. Kartica „uređivanje prirode“ omogućuje dodavanje većeg broja elemenata (stabala, trave i sl.) istovremeno, uz pomoć kistova. Ti se modeli mogu dodavati i ručno

povlačenjem i ispuštanjem. Za razliku od ručnog dodavanja, korištenjem kistova dodavanje većeg broja prirodnih elemenata i modela brže je i efektivnije, što je vrlo korisno, posebno kod manjih modela, kao primjerice u slučaju trave i cvijeća. Kartica „geometrija“ omogućuje korisniku brzu kreaciju prototipa razine. Kistovi za geometriju ispunjavaju i manipuliraju 3D modele. Korišteni su u svrhu izgradnje jednostavnih oblika, građevinskih elemenata i modela za brzo dizajniranje razina. Obzirom na to da statički elementi zauzimaju sličnu ulogu, a istovremeno su učinkovitiji, brži i detaljniji, alati iz ove kartice koriste sa najviše u koraku dizajna i stvaranja brzog prototipa razina.

2. Alatna traka (*eng. toolbar*) – Kartice se razlikuju ovisno u kojemu se uređivaču korisnik nalazi. U prozoru urednika razine nalaze se kartice: spremi (*eng. Save current*), kontrola izvora (*eng. source control*), način rada (*eng. modes*), sadržaj (*eng. content*), tržnica (*eng. marketplace*), postavke (*eng. settings*), Blueprints, kinematika (*eng. cinematics*), gradi (*eng. build*), kompajliraj (*eng. compile*), igraj (*eng. play*) i pokreni (*eng. launch*). Alatna traka u Unreal-u slična je kao i u mnogim drugim razvojnim okruženjima za računalne igre i modeliranje trodimenzionalnih modela. Kartice koje su najčešće korištene za izradu igara koje su napravljene u svrhu ovog projekta su kartice: postavke, Blueprints, gradi, kompajliraj i igraj. U postavkama se nalaze postavke projekta koje se koriste u mnoge različite svrhe, kao primjerice dodavanja događaja u slučaju unosa specifične tipke sa tipkovnice, koje se kasnije u Blueprint-u ili C++ programskom kodu mogu koristiti kao događaj ili okidač za izvođenje neke akcije. Osim toga, u projektnim postavkama se odabiru i postavke vezane za pakiranje projekta igre, kao na primjer ciljana platforma. Blueprints kartica sadrži level Blueprint ili Blueprint razine, što je skripta koja je postavljena na razini računalne igre i sadrži upute koje se odnose na cijelu razinu. Kartica za izgradnju gradi modele i osvjetljenje igre. Ovaj korak vrši se po završetku izgradnje razine ili prije pakiranja (*eng. packaging*) i eksportiranja projekta. Kompajler u uredniku razine služi za kompiliranje C++ koda, a ne Blueprint koda. Obzirom na to da je Blueprint urednik dio Unreal razvojnog okruženja, Blueprint klase se kompajliraju za svaku klasu posebno u Blueprint uredniku, dok C++ koristi tekstualne urednike poput „visual studia“ za kreiranje programskog koda i zbog toga se kompajlira naknadno u uredniku razine. Kartica za igranje služi za testiranje računalne igre.

3. Glavni prozor (*eng. main viewport*) – koristi se u svrhu vizualnog prikaza dizajna razine računalne igre i simulacije u svrhu testiranja iste. Ovdje se mogu postaviti razni objekti, svjetla, efekti i slično. Središnje je mjesto za stvaranje i uređivanje vizualnog izgleda razine. Ima nekoliko načina prikaza razine, kao na primjer žičani prikaz (*eng. wireframe*), neosvijetljen (*eng. unlit*), osvijetljen (*eng. lit*) i slično. Osim toga, ima i opciju prikaza broja slika u sekundi.

4. Preglednik sadržaja (*eng. content browser*) - sadrži sve mape i dokumente igre. U ovom prozoru nalazi se svaka datoteka glumaca, materijala, tekstura, razne vrste trodimenzionalnih modela, kosturi, Blueprint i C++ klase i slično. Nakon uvoza

dokumenta sa Unreal tržnice, dokumenti se nalaze u posebnoj mapi u pregledniku sadržaja. Kod C++ projekta na dnu se nalazi posebno odvojeni dio za sve C++ klase.

5. Lista objekata u razini (*eng. world outliner*) – Ovaj prozor prikazuje sve glumce, objekte i modele unutar scene, koji se tu dodaju automatski. Radi se o listi svih objekta u sceni, uključujući i svjetla, Bluprint klasa kao na primjer startno mjesto igrača (*eng. player start*) i slično. Svaki element u ovom prozoru može se grupirati. U listi objekata u svijetu mogu se odabrati, pomaknuti, izbrisati i sakriti objekti i grupe objekata.

6. Detalji (*eng. details*) – Ovisno o tipu glumca ili objektu koji je odabran, razlikuje se i prozor detalja. Kartica detalja sadrži sve detalje odabranog objekta. Svaki element ima rotaciju, lokaciju i veličinu koja se u ovom prozoru prikazuje. Neki drugi detalji mogu biti materijali, Blueprint ili C++ skripta koja je dodana na objekt, kao i animacijska skripta, ako je objekt sa animacijama (Epic games, 2004 - 2021).

2.2 3D MODELI, ANIMACIJA I MATERIJALI

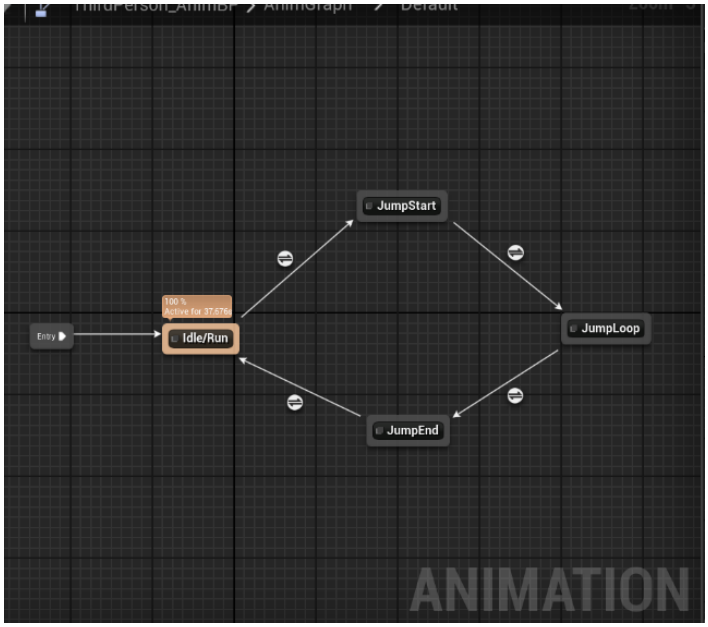
Modeli u računalnim igrama glavni su vizualni element koji igraču prikazuje događaje i promjene u igri. 3D modeli u Unreal 4 razvojnom okruženju mogu biti uvezeni u FBX formatu iz drugih 3D aplikacija poput blendera, maya i slično. Osim tih objekata Unreal ima i nekoliko objekata već ugrađenih u softver. Uvezenim objektima, ukoliko nisu unaprijed prilagođeni za Unreal, poput onih koji se nalaze na Unreal tržnici (*eng. unreal marektplace*), nakon uvožnje potrebno je ponovno dodati materijale u Unreal razvojnom okruženju. Veliki broj softvera za izgradnju 3D modela koristi posebno razvojno okruženje za renderiranje (*eng. render engine*) materijala pa zbog toga uvođenje, odnosno importiranje materijala nije moguće.

Detalji svakog modela mogu se vidjeti u prozoru detalja nakon pritiska lijeve tipke miša na objekt. Svaki objekt koji se dodaje u svijet služi kao posebna instanca klase. Svaka instanca klase ima posebnu lokaciju, rotaciju i veličinu. Detalji objekata razlikuju se ovisno o tipu objekta. Objektima se mogu dodati Blueprint ili C++ skripte. Blueprint klasama moguće je dodati varijable od kojih svaka ima svoje postavke. Obično varijable imaju istu vrijednosti za svaku instancu objekta. Neke od postavki omogućuju stvaranje varijabli koje se ponašaju drugačije i mogu se urediti i postaviti zasebno za svaki objekt (*eng. instance editable*), odnosno za svaku se instancu klase može posebno postaviti vrijednost varijable. U C++-u postoje specifikatori EditInstanceOnly ili EditAnywhere koji imaju istu funkciju.

Modeli mogu imati kosture, a kosturi se dodaju kako bi ih se moglo manipulirati i kako bi se modelu mogle dodati animacije. Unreal razvojno okruženje ima Unreal lutku (*eng. unreal mannequin*) koja ima svoje animacije i kostur. Ako dva modela imaju isti kostur, kojeg je moguće duplicirati kod gotovo svih ljudskih modela, animacija sa s jednog modela prebacuje na drugi. Modeli koji mogu preuzeti animacije Unreal lutke su modeli koji su postavljeni za Epic kostur (*eng. rigged to epic skeleton*). Unreal nudi mnoge mogućnosti vezane uz animacije, tako se primjerice dvije animacije mogu povezati u jednu miješanjem animacija (*eng. blending*),

mogu se izraditi animacije bazirane na fizici unutar igre (*eng. ingame physics based animations*) ili dodavati animacije lutaka (*eng. ragdoll*).

Animacije se postavljaju i uređuju u nacrtima animacije (*eng. animation Blueprint*). Slično kao i obični Bluprint editor, imaju event graf u sredini radnog prozora koji u ovom slučaju provjerava stanje lika. Primjerice, provjerava da li je lik kojeg igrač trenutno kontrolira u zraku, provjerava brzinu trčanja i slično, te se pomoću varijable koja vraća stanje igraća može razlikovati i prikazati razne animacije, poput trčanja ili hodanja. Ti se podaci spremaju u varijable koje su naslijeđene iz klase lika igrača. Na temelju toga se prikazuje animacija, a redoslijed i vrstu animacije određuje se animacijskim grafom. Jednostavni primjer animacijskog grafa nalazi se na slici 2.



Slika 2. Animacijski graf standardne lutke u Unreal razvojnom okruženju

U detaljima svakog objekata i komponente koja će se vizualno prikazati igraču, nalazi se opcija za dodavanje materijala. Isti materijal može dijeliti nekoliko objekata. Slično kao i varijable objekata, materijali također mogu imati svoje instance, gdje svaka instanca ima posebne postavke. Materijali u Unreal razvojnom okruženju uređuju se u uređivaču materijala. Raspored uređivača sličan je kao kod Blueprint-a. U centru se nalazi uređivač čvorova (*eng. node editor*), koji sadrži output čvor i druge čvorove za stvaranje materijala. Na desnoj strani uređivača nalazi se paleta sa raznim postavkama koja se može dodati u uređivač čvorova za manipulaciju i stvaranje materijala. Na lijevoj strani uređivača nalazi se pregled (*eng. preview*) materijala, te ispod njega prozor detalja materijala. Neki od čvora koji se dodaju u uređivač materijala su teksture, boje i slično.

2.3 WIDGET I KORISNIČKO SUČELJE

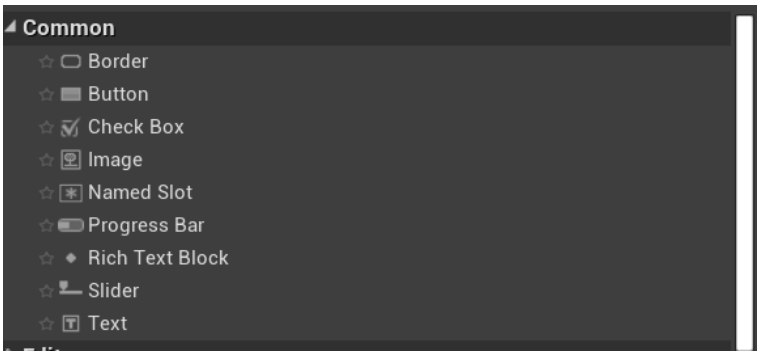
Korisničko sučelje (*eng. user interface*) ili UI omogućuje interakciju i komunikaciju sa računalom. Cilj interakcije je bolja i efikasnija suradnja računala sa čovjekom. U razvoju računalnih igara korisničko sučelje ima nekoliko zadataka koji su bitni za korisničko iskustvo (*eng. user experience*). Prilikom određivanja kvalitete sučelja, programer si postavlja sljedeća pitanja: „Mogu li kroz sučelje pronaći informacije koje tražim? Govori li mi sučelje ono što trenutno moram znati? Je li zadatak jasno određen? Ima li sučelje previše informacija?“. Tijekom cjelokupnog procesa rada programer si postavljati ova pitanja nekoliko puta.

U korisničko sučelje spada sve, od glavnog menija, menija opcija, odabira razina (*eng. level select*), pa do mape terena i iskočih prozora koji se pojavljuju tijekom igranja igre. Zadaci korisničkog sučelja variraju od jednog tipa igre do drugoga. U igrama otvorenoga svijeta, kao na primjer Legend of Zelda: Breath of the Wild ili Far cry, bitna je mapa i indikatori gdje se nalazi sljedeća misija, dok kod igara bočnog prikaza (*eng. side scroller game*), kao na primjer Super Mario ili Sonic, gdje su razine pretežito linearne, mapa i indikatori gdje se mora ići nisu toliko bitni. U igrama ove vrste ti su elementi višak, te kao posljedica toga igrač bi se teže udubio u samu igru (*eng. player immersion*). Istovremeno, premalo informacija također može otežati udubljenje u igru. Loša komunikacija kod igrača može izazvati zbunjenost i frustraciju. Zbog svega navedenog, vrlo je bitno procijeniti optimalnu količinu informacija koju će programer prenijeti korisniku kroz sučelje.

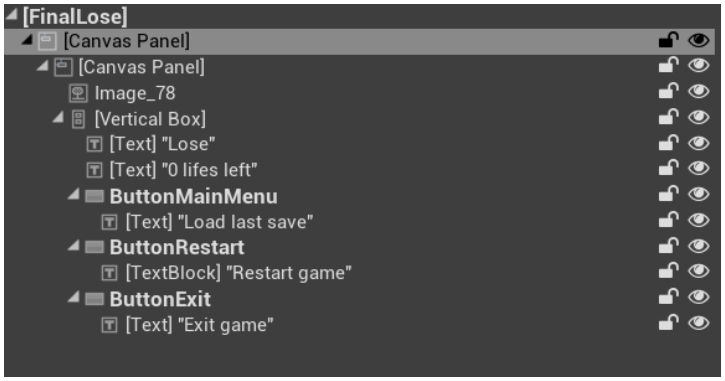
U Unreal razvojnom okruženju korisničko sučelje uređuje se u Widget uređivaču. To je uređivač sa elementima korisničkog sučelja koji se mogu dodavati u hijerarhijsku strukturu i pozicionirati kako bi se stvorilo sučelje. Elementi sučelja su sastavljeni od ne interaktivnih elementa, kao na primjer vertikalna kutija (*eng. vertical box)* ili horizontalna kutija (*eng. horizontal box*) i interaktivnih elementa kao primjerice gumb (*eng. button*) ili blok za upisivanje i uređivanje teksta (*eng. edit text block*). Slika 3 prikazuje neke od komponenata korisničkog sučelja. Slično kao i kod mnogih drugih uređivača za korisnika sučelja, za druga razvojna okruženja poput Unity-a i Godot-a, ali i za neke druge aplikacije poput Android studia (program za stvaranje android aplikacija), vizualni proces postavljanja i manipulacije elemenata u izradi korisničkog sučelja je vrlo slična, no funkcionalni dio se razlikuje.

Prilikom rada u Widget uređivaču prvo se radi na vizualnom dijelu, odnosno radi se sa komponentama i manipulacijom komponenata. Komponente koje se nalaze u prozoru komponenata se povlače i ispuštaju u glavni prozor. One su postavljene u hijerarhijsku strukturu, kao na primjer struktura koja je prikazana na slici 4. Ako se komponente dodaju na drugu komponentu, nova komponenta postavlja se na razini ispod stare komponente u hijerarhijskoj strukturi. Ako se komponenta nalazi na vrhu hijerarhijske strukture, u vizualnom prikazu komponenata nalaziti će se ispod ostalih komponenti. Na što višoj poziciji se komponenta nalazi u hijerarhijskoj strukturi, u to nižem se sloju ta ista komponenta nalazi u vizualnoj strukturi. Vizualni prikaz grafičkih elemenata ponaša se na sličan način kao i aplikacije koje rade sa uređivanjem slika i drugi programi koji rade sa vizualnim elementima i sučeljima. Neke sličnosti su primjerice rad sa slojevima, prozirnost slika i slično.

Tijek rada Blueprinta i C++-a u Widget uređivaču jednak je kod postavljanja i pomicanja komponenata u korisničkom sučelju, a razlikuje se u funkcionalnom dijelu rada. Blueprint skriptiranje za Widgete radi se u Widget Blueprint uređivaču, koji je sličan kao i standardi uređivač Blueprint klasa i Blueprint razina. Uz obične varijable, odnosno varijable koje su prisutne u standardnom C++-u, kao na primjer numeričke i tekstualne varijable, urednik korisničkog sučelje sadrži još nekoliko varijabli koje su specifične za Unreal razvojno okruženje. U ovo spadaju varijable komponenata. U uredniku korisničkog sučelja te varijable su vizualne komponente korisničkog sučelja, te im je moguće dodati i događaje, poput događaja koji se poziva pritiskom na dugme ili funkcije. Programiranje u C++ se radi uz pomoć uređivača teksta i integriranog razvojnog okruženja. Neke C++ Widget funkcije razlikuju se od onih u Widget Blueprint uređivaču. C++ ima funkcije koje rade samo sa Widget-ima i zato se koristi samo u Widget C++ klasama, iako ih je moguće naslijediti i u drugim klasama.



Slika 3. Elementi korisničkog sučelja



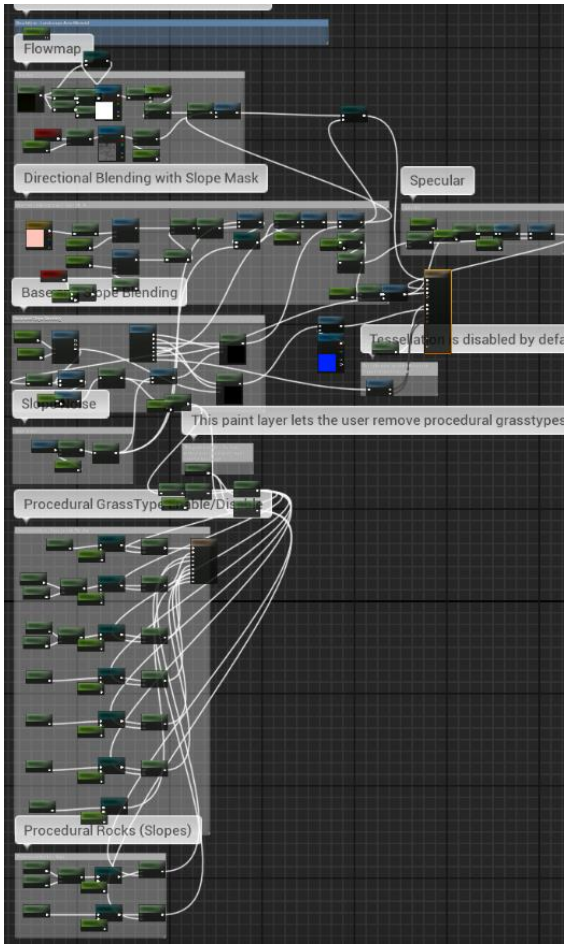
Slika 4. Primjer hijerarhija elemenata korisničkog sučelja

2.4 IZRADA TERENA

Izradi mape ili terena u Unreal razvojnom okruženju može se pristupiti na nekoliko načina ovisno o vrsti igre. Unreal razvojno okruženje ima alat za izradu terena (*eng. landscape tool*) koji obično služi kao prvi korak, nakon dizajna, za izradu mape. Osim navedenog alata za izradu

mape u Unreal razvojnom okruženju moguće je koristi i druge softvere za 3D modeliranje i za izradu mape, ili u nekim slučajevima i samo 3D modele i objekte.

Alat za izradu terena najprilagođeniji je za izradu prirodnih struktura i pejzaža. On ima nekoliko kistova koji daju mogućnosti manipulacije zemlje, na primjer za izradu brežuljaka, brda, rupa, rampi i sl. Osim toga daje mogućnost dodavanja materijala terenu kistom. Prije bojanja materijala potrebno je postaviti ga na teren i prilagoditi istome. Primjer toga vidi se na slici 5. Kist omogućuje dodavanje materijala na svaki dio terena posebno.



Slika 5. Primjer Blueprinta materijal za izradu terena izvor: JoeGarth

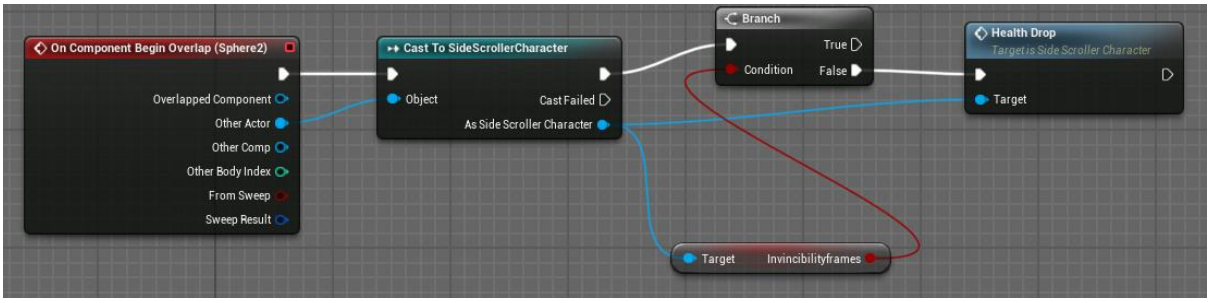
U slučaju kada nije potrebna izgradnja cijelog terena, za izradu mape također se može koristi samo 3D modele, kao na primjer kada se radnja igre odvija u gradu, zgradama ili sličnim strukturama. Kod izrade mape, osim izrade terena i dodavanja glumaca, u igru je potrebno dodati i svjetlo sa neba (*eng. sky light*), svjetlo (*eng. light source*) i maglu (*eng. atmospheric fog*).

3 USPOREDBA VIZUALNIH I TEKSTUALNIH PROGRAMSKIH JEZIKA

Programski jezik je formalni jezik koji sadrži niz uputa. McGuire i College W. (2019*)* navode kako je cilj programskog jezika izraziti algoritam na način na koji je nedvosmislen ljudima i strojevima. Programski jezik definira sintaksa i semantika. Sintaksa je gramatika jezika, dok je semantika njegovo značenje. Postoji mnogo različitih programskih jezika od kojih svaki ima specifičnu upotrebu, zadatak, terminologiju i sintaksu. Također, razlikujemo nekoliko tipa programskih jezika. U Unreal razvojnom okruženju imamo dvije vrste programskih jezika, vizualni i tekstualni. Primjer tekstualnog programskog jezika je vidljiv na slici 6, a vizualnog programskog jezika na slici 7.

```
void ASpinBase::OnSpinHit(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveCompone
    AFaksCPlusPlusCharacter* sideScrollerChar = Cast<AFaksCPlusPlusCharacter>(OtherActor);
    if (sideScrollerChar != nullptr){
        sideScrollerChar->HealthDrop();
    }

}
```

Slika 6. Kod za C++ događaj koji se okida u slučaju kolizije



Slika 7. Blueprint događaj koji se okida u slučaju kolizije

Vizualni programski jezik je programski jezik koji omogućuje programerima stvaranje i dodavanje uputa programu manipulacijom grafičkih elemenata, a ne tekstualnim instrukcijama. Velik broj vizualnih programskih jezika kao i Blueprint jezik radi na bazi čvor i veza. Čvor u ovom slučaju predstavlja funkciju, a veza je output koji vraća rezultat funkcije i povezuje ih kako bi se stvorio redoslijed i tijek rada programa. Prednosti vizualnih programskih jezika su da su jednostavniji za razvoj i upotrebu od tekstualnih programskih jezika, u slučajevima kada se radi o manjim funkcijama mogu biti čitljiviji, lako ih je naučiti, sprječavaju stvaranje programa sa problemima u sintaksi, manji je rizik programa sa većim pogreškama, imaju već ugrađene objekte, klase i funkcije.

Vizualni programski jezici zbog navedenih prednosti najviše odgovaraju početnicima. Iz tog se razloga oni često koriste u edukaciji i kod učenja programskih jezika. Vizualni i grafički elementi intuitivni su i lako ih je upotrebljavati, te smanjuju rizik stvaranja pogrešaka u programskom kodu. Vizualni programski jezik pomaže programeru na razini sintakse i semantike (Boshernitsan i Downes, 2004).

Sintaksa odgovara na pitanje „Kako mogu stvoriti valjani programski kod?“. Jedan od primjera u tradicionalnim programskim jezicima je odvajanje dijelova koda sa točkom i zarezom. Razina sintakse koristi čvorove, prozore, ikone, dijagrame, obrasce, veze među elementima i slično koji smanjuju rizik sintaktičkih pogrešaka. Mnogi sintaktički problemi u vizualnim programskim jezicima zbog rada sa grafičkim elementima nisu mogući, ali je zbog toga ograničena i sloboda rada.

Razina semantike sadrži značenje programa i način ponašanja programa. Za razliku od sintakse, koja odgovara na pitanje kako konstruirati i urediti programski kod, semantika odgovara na pitanje je li programski kod točan i zašto. U slučaju Blueprint programskog jezika u Unreal razvojnom okruženju, stvaranje referenci na objekte ili komponente i jednako tako dodavanje Blueprint funkcija, ide samo uz pomoć prozora za čvorove. To je prozor koji se otvara i automatski nudi postojeće čvorove koji neće rezultirati u pogreškama. Iz tog razloga nije moguć unos pogrešnih imena, pogrešaka u pisanju i slično. Osim toga, program sprječava spajanje određenih nepodudarnih čvorova i spajanje krivih tipova varijabli na inpute koji traže druge varijable. Drugi vizualni programski jezici često imaju slične osobine (Boshernitsan i Downes, 2004).

Većina programskih jezika su tekstualni programski jezici, te su, za razliku od vizualnih koji su bazirani na grafičkim elementima, bazirani na tekstualnim uputama. Oni se značajno razlikuju ovisno o kojem se jeziku radi. Neki od najpopularnijih tekstualnih programskih jezika su C++, Java, JavaScript i Python. Unreal razvojno okruženje sadrži C++ tekstualni jezik.

Zbog svojih prednosti vizualni programski jezici u mnogo slučajeva mogu izgledati pristupačniji i jednostavniji za početnike. Međutim, i tekstualni programski jezici imaju svoje uloge i prednosti koje se razlikuju ovisno o kojem se programskom jeziku radi.

Neke od prednosti tekstualnih programskih jezika su brzina rada, odnosno brža izvedba, funkcija i operacija, bolja čitljivost u kompleksnijim klasama, brži rad programa, veća fleksibilnost i veća sloboda rada.

4 BLUEPRINT VISUAL SCRIPTING

Blueprint visual scripting je jedan od dva programska jezika za razvoj igre u Unreal razvojnom okruženju. Blueprint je vizualni programski jezik koji omogućava korisnicima programiranje dodavanjem, brisanjem, spajanjem i drugom manipulacijom grafičkih elemenata. Mnogi vizualni programski jezici temelje se na ideji čvora i veze. Čvorovi su događaji, funkcije, varijable, a strelice služe za povezivanje čvora i određuju tijek programa. Čvorovi u vizualnom programiranju prikazani su kao crna kutija (*eng. black box*), znaju se inputi i outputi, ali ne i sam proces i događaji unutar čvora. Jedini način na koji se može vidjeti točno što radi čvor je otvaranjem baznog C++ koda na temelju kojeg Blueprint čvor radi. Još jedan način na koji se može gledati čvor i vezu u vizualnom programiranju je kao stabla. Blueprint čvorovi zauzimaju isto ulogu kao i čvorovi u stablu. Čvorovi idu linearno, no u nekim slučajevima se dijele. Jedan od glavnih ciljeva vizualnih programskih jezika je napraviti programiranje pristupačnijim za početnike.

Skripta se odvija linearno od događaja na dalje, ovisno o tome kako su spojene veze među čvorovima. Svaki output čvor može imati samo jednu vezu prema sljedećem input čvoru, a input čvor može imati nekoliko ulaznih veza. Postoje 2 vrste veza, od kojih jedna služi za kontrolu tijeka događaja, odnosno u kojem redoslijedu se događaji odvijaju, a druga služi za spajanje varijable s čvorom. Output čvora određuje da li će veza prenositi podatke o varijablama ili da li će odrediti koji se čvor okida nakon trenutnog. Svaki čvor ima najmanje jedan output čvor koji služi za upravljanje tijeka događaja, dok neki čvorovi imaju više od jedne output veze koje služe u tu svrhu. Jedan od primjera za prikaz čvora sa nekoliko vrsta inputa i outpura je Branch čvor. On zauzima istu ulogu kao što if funkcija zauzima u C++-u. Ovaj čvor ima jedan output u slučaju istine i jedan u slučaju laži. Oba outputa mogu, ali i ne moraju imati vezu koja ih povezuje na drugi čvor. Samo jedan od dva outputa se nastavi odvijati, ovisno o rezultatu funkcije. U ovom slučaju se skripta grana i postaje kompleksnija jer se mora dodati čvor za svaki mogući rezultat. Uz to Branch čvor ima dvije vrste inputa, od kojih prva vrsta prima vezu koja upravlja tijekom događaja iz prijašnjeg čvora ili događaja, a drugi input je veza koja prima varijablu. U slučaju Branch čvora varijabla koju prima je tipa Boolean.

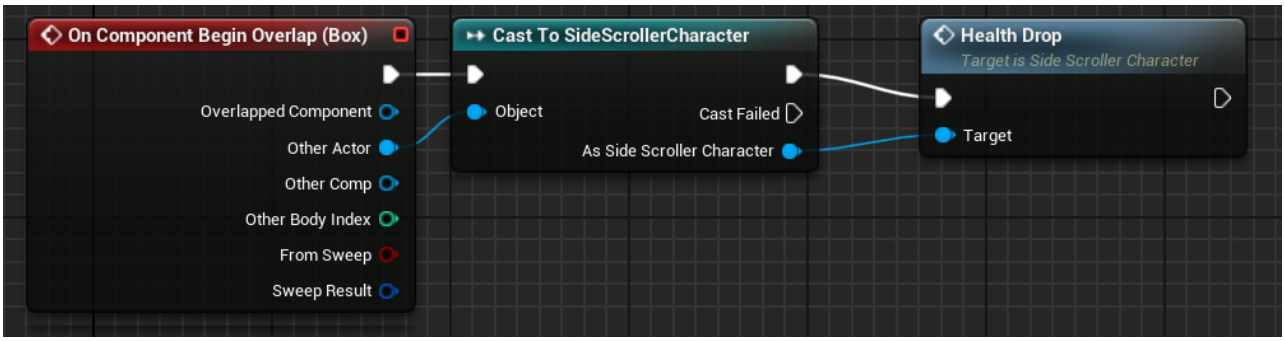
Blueprint sustav je vizualni način skriptiranja računalne igre u Unreal razvojnom okruženju koji koristi čvor i veze među njima kako bi programiranje bilo jednostavnije i u nekim situacijama brže. Budući da Blueprint sustav koristi vizualno skriptiranje, cilj mu je učiniti programiranje računalnih igara pristupačnije početnicima. Uz to se koristi i za brzu izradu jednostavnih mehanika. Blueprint je programski jezik baziran na C++-u i proces skriptiranja ima sličnosti sa procesom programiranja u C++-u. Ovaj vizualni programski jezik ima opciju stvaranja Blueprint klasa i objekata.

Jedna dodatna karakteristika koja je također moguća u Blueprint sustavu je nasljeđivanje. Svaka Blueprint klasa ima baznu ili roditeljsku klasu (*eng. parent class*) koja je po zadanom C++ klasa i koja nasljeđuje Blueprint sustav. Blueprint klasa koristi funkcije koje su naslijeđene iz roditeljske klase. Funkcijama i varijablama iz bilo koje druge klase se također može pristupiti, ovisno o razini sigurnosti. Varijable u Blueprint klasi mogu biti: privatno, javno ili zaštićeno.

Za pristupanje podacima iz drugih klasa potrebna je referenca koja pokazuje točno na objekt iz kojega se podaci uzimaju. Za dijeljenje podataka iz jedne klase u drugu u Blueprint sustavu koristi se čvor za pristup funkcijama u drugim klasama ili *cast to node*. On kao inpute uzima referencu za instancu objekta koju se poziva. Primjerice, ako se čvor za pristup funkcijama poziva nakon događaja koji se poziva u slučaju kolizije igrača sa objektom druge klase (*eng. begin overlap event)*, onda se može uzet referencu na igrača koji je bio u kontaktu sa ovom klasom iz outputa događaja i povezati na input od čvora za referenciranje druge klase. Iz outputa čvora za pristup funkcijama može se dobiti pristup bilo kojoj funkciji iz te klase. Funkcije kojima se može pristupiti ovise o njihovoj razini sigurnosti.

Blueprint sustav može se podijeliti na Blueprint klase i Blueprint razine. Blueprint klase sadrže nekoliko kategorija i svaka ima svoje posebne zadatke. Uglavnom ih se koriti kao skriptu na komponentama u razini koji izvršavaju neki zadatak. U ovo spadaju i lik igrača (*eng. player character*), NPC-evi (*eng. non playable* character) i protivnici koji koriste umjetnu inteligenciju (*eng. AI enemy*). Blueprint razine je skripta koja se postavlja zasebno na svaku razinu i posebno za nju izvršava zadatke. Ova skripta obično sadrži procese koje se odnose na razinu kao cjelinu, a ne na objekte unutar nje. Nacrt razine sadrži funkcije i varijable kao i ostale Blueprint klase. U ovom Blueprintu obično se dodaju funkcije koje se odvijaju tijekom cijele razine, kao na primjer preostalo vrijeme za završiti razinu, referencu na ime ili broj razine i Widget koji se pojavi ako je isteklo vrijeme. Osim toga, glazba koja svira i sl. također može biti dodano u Blueprint razine.

Programiranje u Blueprint editoru temelji se na događajima (*eng. event*), a događaji su čvorovi koji se nalaze na početku svakog procesa. Oni se okidaju kao posljedica određenih događaja i omogućuju Blueprintu da izvrši niz radnji, kao npr. kada igra započne (OnBeginPlay) i kada objekt dodirne drugi objekt (OnBeginOverlap), koji je prikazan na slici 8.



Slika 8. Primjer OnCOmponentBeginOverlap čvora u Blueprint uredniku

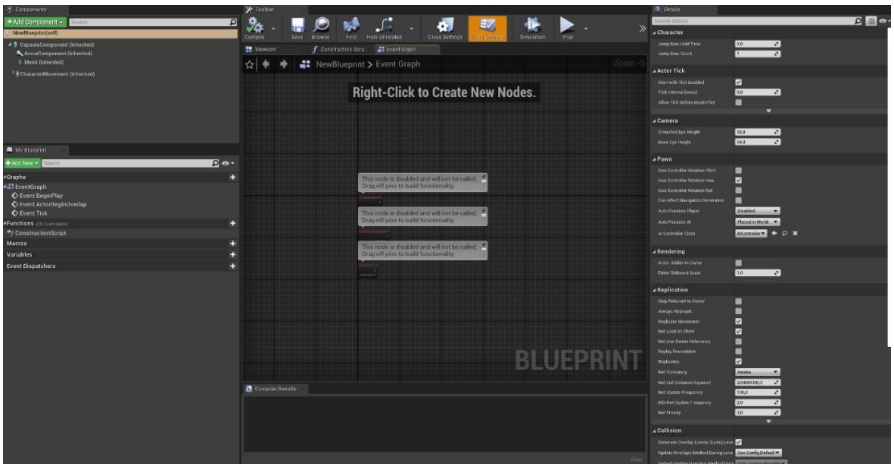
4.1 BLUEPRINT EDITOR

Skriptiranje u Blueprintu razine i Blueprintu klase se izvršava u Blueprint editoru. To je uređivač u kojemu se može raditi na skriptama i koristi se u nekoliko svrha. Osim za Blueprint razine i Blueprint klase koristi se također za stvaranje stabla animacija i AI stabla ponašanja (*eng. AI behaviour tree*). Ovisno u koju se svrhu koristi, uređivač i alati u njemu biti će različiti. Međutim, bez obzira koji se uređivač koristi, tijek rada u Blueprint uređivaču uvijek ostaje sličan. On omogućuje stvaranje i uređivanje čvora uz pomoć vizualnog skriptnog jezika, a rad u uređivaču obično je brz i jednostavan.

Glavni prozori u Blueprint uređivaču su centralni uređivač, u kojemu se radi sa objektima koje će skripta koristiti, skripta izgradnje (*eng Construction script*), u kojoj se pomoću čvora stvara skripta i sadrži output, i graf događaja (*eng. event graph*). U radu sa Blueprint sustavom graf događaja uvijek je prisutan dok, na primjer, viewport nije prisutan u uređivaču Blueprint razine.

Uređivač sadrži graf događaja u centru u kojem se stvaraju i uređuju čvorovi. Na lijevoj strani nalazi se prozor sa komponentama (*eng. components tab*) i „moj nacrt“ (*eng. my Blueprint tab*). Prozor sa komponentama sadrži komponente koje se nalaze u glavnom uređivaču i koje skripta može koristit. Neki primjeri komponenata koje može sadržati su osnovni oblici, kao sfera i kocka, AI komponente, kao na primjer AI percepcija, kamere, blokovi kolizije, likovi, kosturi i slično. Iz tog prozora se mogu povući i spustiti reference na te objekte i koristiti ih kasnije u skripti. Primjer toga bila bi komponenta kolizije koja se može koristit za prepoznavanje kada dolazi do kolizije među objektima u igri i kao rezultat toga može se napraviti skripta koja se izvršava u slučaju kolizije.

U prozoru „moj nacrt“ stvaraju se funkcije i varijable. Pri stvaranju funkcija kreira se novi prozor u kojemu se može pomoću čvora i veza stvoriti funkcija koja se kasnije može pozvati u grafu događaja. Funkcije imaju inpute i outpute, ovisno o vrsti funkcije, kao i obična funkcija u C++-u.



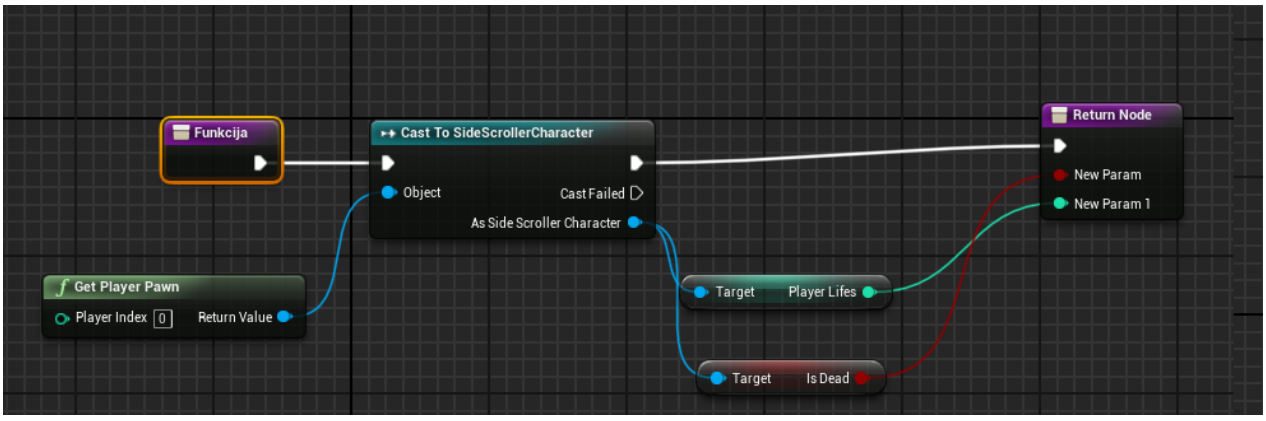
Slika 9. Blueprint uređivač

4.2 BLUEPRINT FUNKCIJE I VARIJABLE

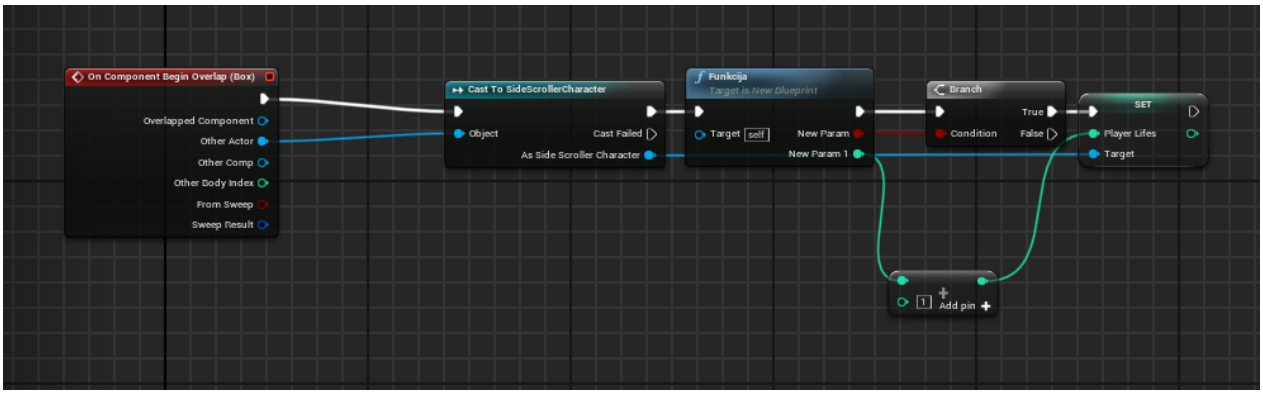
Funkcije i varijable u Blueprint uređivaču stvaraju se u uredniku „moj nacrt“ (*eng. my Blueprint*) i one su centralni dio tijeka rada u Blueprint skriptiranju. Funkcije u Unreal razvojnom okruženju uređuje se u posebnom prozoru od onoga u kojemu ih se koristi.

Prvi prozor je prozor u kojemu se određuje što funkcija radi nakon pozivanja, što su inputi funkcije i što su outputi funkcije. Na slici 10 prikazan je primjer obične Blueprint funkcije. Ovaj prozor je uređivač funkcija i sadrži upute i akcije koje funkcija odrađuje nakon njezinog pozivanja, odnosno moglo bi se reći da se u ovome prozoru funkciju stvara. Funkcija je sastavljena od početnog čvora, funkcionalnog dijela i output čvora. Početni čvor služi kao početak izvođenja funkcije. Osim toga, početnom čvoru se mogu dodati varijable kao outpute, te one služe kao uvjeti funkcije. Funkcionalni dio sadrži sljedove vizualnih uputa koje program izvodi. Output čvor ima ulogu kao i return komanda u C++-u. On označava kraj funkcije i vraća potrebne podatke. Uređivač funkcija nema čvorove događaja jer se u ovom uređivaču ne određuje trenutak u kojemu će se funkcija okinuti, već samo što će funkcija raditi, odnosno ovaj uređivač ima samo ulazni čvor koji ima ulogu okidanja funkcije. Trenutak u kojemu će se funkcija okinuti određuje se u drugom prozoru.

U drugom prozoru nalazi se glavni Blueprint uređivač. Primjer događaja i okidanje funkcije koja je kreirana na slici 10 je prikazan na slici 11. U ovome prozoru se koristi kreirana funkcija i unosi se u tijek rada skripte. Sličnu ulogu u C++ kodu imala bi glavna funkcija (*eng. main function*). U ovom prozoru vidljiv je samo čvor funkcije sa potrebnim inputima i outputima. Varijable koje su u prvom prozoru unesene u početni čvor kao uvjet funkcije, u Blueprint uređivaču služe kao input u čvor, a varijable koje su unesene u završni čvor služe kao output čvora.

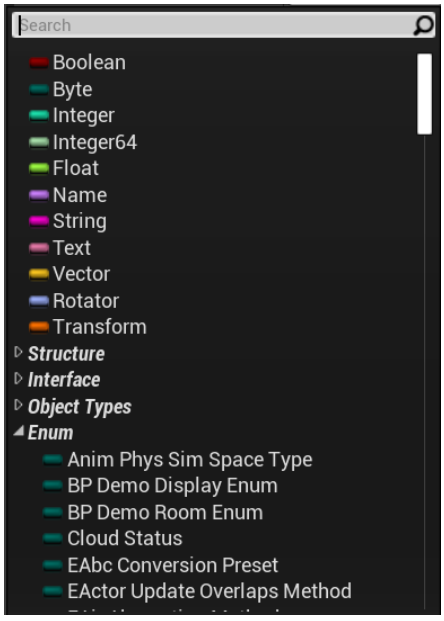


Slika 10. Primjer Blueprint funkcije



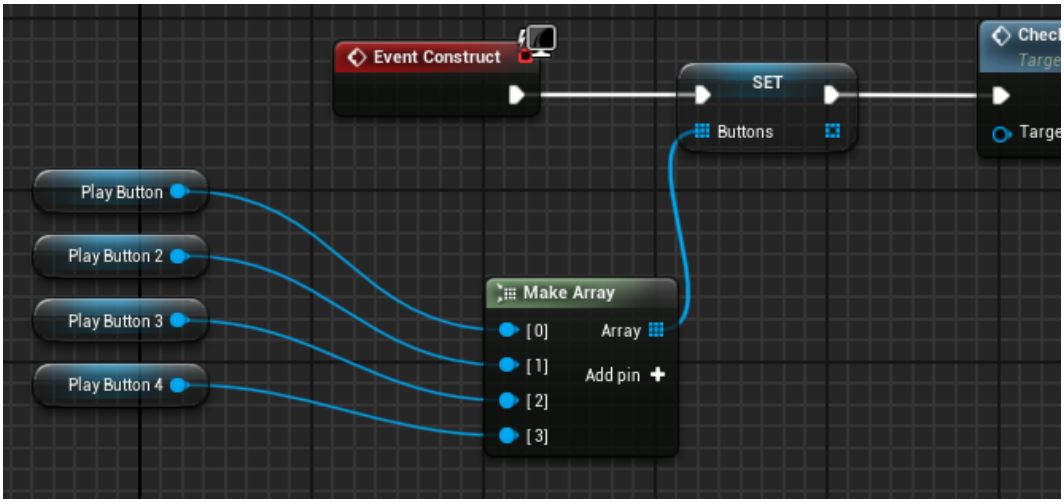
Slika 11. Primjer Blueprint događaja

Osim funkcija, u prozoru „moj nacrt“ se također stvaraju varijable, a za stvaranje varijabli u Blueprintu potreban je samo jedan pritisak na lijevu tipku miša. Varijable u Unreal razvojnom okruženju zauzimaju nekoliko uloga. Osim običnih vrsti varijabli koje su prisutne u standardnom C++ programskom jeziku, kao na primjer Boolean, intiger i String, Blueprint uređivač i Unreal C++ sadrže varijable koje služe kao reference objektima, lokacije, reference za elemente i slično. U poglavlju 3.3 već su spomenute varijable komponenata u korisničkom sučelju koje označavaju primjerice gumbove i tekstna polja. Blueprint uređivač raspolaže još većim brojem varijabli nego uređivač korisničkog sučelja. Neke od vrsta varijabli koje se nalaze u ovom uređivaču su prikazani na slici 12. Osim varijabli komponenata iz korisničkog sučelja, Blueprint uređivač ima još druge varijable komponenata, primjerice komponente statičkih objekata i detektora kolizije.



Slika 12. Izbornik vrste varijable u Blueprint uređivaču

Varijable unutar Blueprint uređivača imaju ugrađene dvije osnovne operacije, Get i Set operacije, uz pomoć kojih se može uzeti i postaviti element u varijabli. Get čvor sadrži samo output, dok Set čvor sadrži i input i output i može ga se postaviti u tijek skripte. Varijable se mogu postaviti kao array ili lista kao što je prikazano na slici 13, zatim kao set, map, i može se postaviti kao samu varijablu. Punjenje varijabli može se raditi ručno u prozoru detalja ili pomoću Set čvora u Blueprint uređivaču. Varijable mogu biti privatne, zaštićene i javne. Jednako kao i u C++-u, javnim varijablama može svatko pristupiti i vidjeti ih. Privatnim varijablama može pristupiti samo funkcija unutar klase, dok niti jedan objekt ili funkcija izvan nje ne može. Zaštićena varijabla je slična privatnoj, izvan klase joj se ne može pristupiti osim ako je druga klasa dijete te klase.



Slika 13. Primjer kreacije liste u Blueprint programskom jeziku

4.3 BLUEPRINT KLASE

Kao i Unreal razvojno okruženje C++ i Blueprint sadrži klase. Pri kreiranju klase odabire se roditeljsku klasu ili baznu klasu koju će Blueprint klasa nasljeđivati. Neke od klasa koje se mogu naslijediti pri kreaciji prikazane su na slici 14. Blueprint klase u razvoju mogu zauzeti mnogo uloga ovisno koja se bazna klasa odabere. Klase imaju širok raspon funkcija: od kontrole igrača (*eng. player controller*) pa do pravila igre *(eng. game mode base*). Na službenoj stranici doc Unreal razvojnog okruženja, uz dokumentaciju navedeno je da su Actor, Pawn, Character, Player Controller i Game Mode najčešće korištene roditeljske klase.

Slika 14. Izbornik Unreal Klasa

Actor je bazna klasa koja se koristi za izradu interaktivnih objekata poput svjetla, vrata, lifta i slično. Osim toga, također se koristi za prepreke i protivnike kojima nije potreban AI (umjetna inteligencija) controller, odnosno koji imaju jednostavne zadatke, na primjer pomicanje od jedne točke do druge i u slučaju kontakta izvršavanje nekog zadatka. Pawn klasa je klasa koja se koristi za kreaciju AI-a ili objekta kojim igrač može upravljati. Character i player controller su klase koje usko surađuju. Character klasa je klasa koja je postavljena na lika kojim igrač igra i ona upravlja nad varijablama i funkcijama poput kondicija (*eng. stamina*), zdravlja (*eng. health*) i slično. Player Controller je klasa kojoj je uloga upravljanje i postavljanje varijabli vezanih uz kretanje lika igrača. Iz ove klase podaci idu u animacijski Blueprint (Epic Games, 2004-2021).

4.4 LEVEL BLUEPRINT

Level Blueprint ili Blueprint razine je vrsta Blueprinta koja radi na nivou cijele razine. Roditelj Blueprint razine je uređivač razine glumca (*eng. level script Actor*). Blueprint razine izrađuje se posebno za svaku razinu i nije moguće imati istu skriptu za dvije razine. Nakon kreiranja razine automatski se kreira Blueprint razine, te joj se pristupa kroz Blueprint ikone na alatnoj traci. Alatna traka je detaljnije opisana u kratkom opisu elemenata u poglavlju 2.1 i vidljiva je na slici 1. Jednako kao i u Blueprint klasama, Blueprint razine radi sa čvorovima i

vezama. Sam Blueprint uređivač ponaša se na isti način kao i u Blueprint klasama, odnosno rad se u njemu temelji na čvorovima događaja koji nakon okidanja događaja idu na čvor funkcije. Glavna razlika je u njihovim ulogama. Uloga level Blueprinta je da sadrži funkcije i upute koje se odnose na tu specifičnu razinu. Osim toga, postoji još nekoliko manjih razlika između Blueprint uređivača klasa i razine, kao na primjer u nekim ugrađenim funkcijama i slično. Događaji u Blueprintu razine se odnose na cijelu razinu. Blueprint razine se može koristi u slučaju level streaminga ili nasljeđivanja varijabli u instancu igre na kraju razina.

5 C++ ZA UNREAL RAZVOJNO OKRUŽENJE

C++ je objektno orijentiran programski jezik koji, osim objektno orijentiranih funkcionalnosti, također sadrži funkcionalnosti iz programskog jezika C. Razvijen je tijekom 1980-tih godina u Americi kao nasljednik C programskog jezika. Kao rezultat toga ima sve značajke C jezika kao npr. lak prijenos na druge platforme i univerzalne i modularne programe. Osim toga sadrži i objektno orijentirane funkcionalnosti. U C++ programski jezik dodani su apstrakcija, nasljeđivanje, polimorfizam i enkapsulacija (Kirch-Prinz i Prinz, 2002).

Bit objektno orijentiranog programiranja je modeliranje "objekata" (odnosno stvari ili koncepata), a ne "podataka". Objekti koji se modeliraju mogu biti Widgeti na zaslonu, poput gumba i okvira s popisima, ili stvarni objekti, poput kupaca, bicikala, aviona, mačaka i vode. Objekti imaju karakteristike, tj. svojstva ili atribute, poput starosti, brzine, prostranosti, boje i sl. Oni također imaju sposobnosti, operacije ili funkcije, kao na primjer kupnja, ubrzanje, letenje, presvlačenje ili puhanje mjehurića. Zadatak objektno orijentiranog programiranja je predstavljanje tih objekata u programskom jeziku (Jones i Liberty, 2004).

Objektno orijentirano programiranje (dalje u tekstu OOP) stavlja fokus na objekte što je jako atraktivno za izradu računalnih igara i lako je raditi na izmjenama programskog koda. Zahvaljujući objektima lakše je rasporediti i urediti program i kod, odnosno pojednostavljeno je razumijevanje, uređivanje i proširivanje programskog koda. Primjerice kod igre bilijara kugle se može zamisliti kao objekt sa značajkama kao rotacija i lokacija. Nije potrebno svaki objekt izraditi posebno, već se izrađuje kao objekt instance klase.

Koncepti OOP-a, apstrakcija, enkapsulacija, nasljeđivanje i polimorfizam, su također pristupni u razvoju računalnih igara. Apstrakciji je cilj minimiziranje složenosti i generalizacija aspekata klase. Ovaj koncept omogućuje svođenje cijelog procesa programiranja na najosnovnija načela, jednako tako osigurava da se ne ponavlja isti koncept tijekom izrade igre. Ako igra ima nekoliko objekata sa srodnim elementima, smanjivanje koda i ujedinjavanje u jednu klasu uvelike poboljšava performanse igre i urednost koda (Tykhomyrov, 2002).

Polimorfizam je koncept koji je koristan u nekoliko različitih konteksta. On omogućuje objektima različitih vrsta pristup kroz isto sučelje. Unreal razvojno okruženje C++ ne omogućuje apstraktne klase i funkcije poput uobičajenog C++-a.

Ideja enkapsulacije je grupiranje varijabli i metoda. Grupirani podatci u C++ zovu se klase. Osim grupiranja, enkapsulacija također skriva podatke. Implementacija objekata skrivena je od ostatka programa koji poznaje objekt, inpute, outpute i ulogu objekta, ali ne i svaki aspekt objekata i što se događa unutar njega. Implementacijom enkapsulacije koriste se i Get i Set funkcije, koje omogućavaju odstupanja i izmijene elemenata u objektu. U Unreal razvojnom okruženju kod kombiniranja C++ i Blueprinta, enkapsulirani objekti i klase nalaze se u C++ roditeljskoj klasi, dok se neke od funkcionalnosti rade u Blueprintu. U ovom slučaju koriste se Get i Set funkcije kako bi se pristupilo varijablama objekta.

Nasljeđivanje je koncept koji omogućuje stvaranje klase kao ekstenzija druge klase. U tom slučaju klasa koja se nasljeđuje ima ulogu djeteta, a klasa od koje se nasljeđuje je klasa roditelj. U Unreal razvojnom okruženju svaka klasa koja se koristi u razvoju računalnih igara je naslijeđena od klase UObjekt. Ako se kod razvoja igara koristi prethodno spomenuta tehnika kombiniranja C++-a i Blueprinta, onda se postavlja C++ klasa kao roditelj, a Bluprint klasa kao dijete. Detaljnije o korištenju ovih koncepata u Unreal razvojnom okruženju biti će objašnjeno kasnije (Tykhomyrov, 2002).

Iako je izrada kompletne igre u C++-u moguća, najbolji način rada je koristiti C++ za programere koji rade na baznim sustavima igre, dok je dizajnerima omogućena nadogradnja u Blueprint sustavu. C++ ima ulogu roditelja, dok Blueprint ima ulogu djeteta. Ako se koristi takva tehnika bitno je znati da Blueprint može čitati i naslijediti C++ kod, dok C++ ne može čitati Blueprint kod. Klasa nasljednika može čitati iz klase roditelja dok roditelj ne može čitat iz klase nasljednika. Prije nego što se može pristupiti podatcima i funkcijama u C++ kodu mora se odrediti dostupnost uređivača (*eng. editor accessibility*). Ovisno o razini dostupnosti, podacima se može pristupiti iz nekoliko različitih mjesta, na primjer direktno iz Blueprinta ili za svaku instancu u viewportu posebno. Dostupnost ovisi o tipu podataka i razini sigurnosti.

U Unreal razvojnom okruženju osnovna klasa je UObject. To je bazna klasa za velik broj klasa poput APawn ili AGame mode. UClass također je bazirana na UObject i koristi se pri stvaranju novih klasa kako bi se odredili specifikatori klase (*eng. class specifires*). UClass jedan je od meta specifikatora, a u njih također spadaju UStruct, UField, UProperty, UEnum, UFunction, UIntarface i slično.

5.1 UFUNCTION, UCLASS I UPROPERTY

UFunction se deklarira zajedno sa funkcijama. To je parametar za postavljanje detaljnijih specifikacija nove funkcije i postavlja se na liniju iznad deklaracije funkcije. Specifikacije se mogu podijeliti na dva dijela, od kojih prvi sadrži specifikacije, a drugi meta specifikacije. Primjer strukture specifikatora u kodu je prikazan u tablici 1 koja je preuzeta sa stranice Epic Games (2004 – 2021.).

UFUNCTION([specifier1=setting1, specifier2, ...], [meta(key1="value1", key2, ...)])

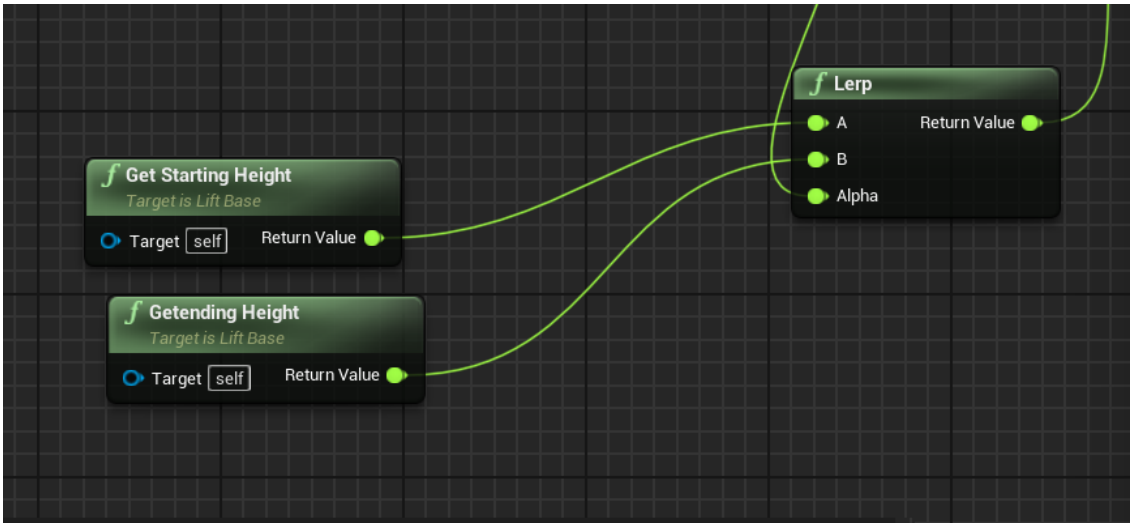
Tablica 1. Primjer specifikatora funkcije Izvor: Epic Games

Specifikatore funkcije pozivamo za kontrolu ponašanja funkcije sa velikim brojem aspekata Blueprint grafa, uređivača i samog razvojnog okruženja. Specifikatori koji su najčešće korišteni u praktičnom primjeru su BlueprintPure, BlueprintCallable, CallInEditor i Category. BluprintPure i BlueprintCallable omogućuju pristup funkciji iz Blueprint uređivača. Ove funkcije pozivaju se na isti način kao i drugi Bluprint čvorovi. CallInEditor omogućuje pozivanje funkcije u instanci objekta u uređivaču. Ona se poziva u prozoru sa detaljima instance

objekta. Category dodaje funkciju u kategoriju koja se prikazuje u Blueprint uređivaču. Na slici 15, nakon specifikacije da se radi o UFunction baznoj klasi, dodani su specifikatori BlueprintPure i Category. Na slici 16 te iste C++ funkcije dodane su u Blueprint uređivač. BlueprintPure specifikator omogućio je dodavanje C++ funkcije u Blueprint grafu. BlueprintPure funkcije su funkcije koje nemaju input i output koji upravlja tijekom događaja nego samo outpute i inpute koji uzimaju i vraćaju varijable.

```
UFUNCTION(BlueprintPure, category = "LiftBase")
    float GetStartingHeight() const {
    return StartingHeight;
}
UFUNCTION(BlueprintPure, category = "LiftBase")
    float GetendingHeight() const {
    return EndingHeight;
}
```

Slika 15. Primjer Get funkcije sa specifikatorom



Slika 16. Primjer C++ funkcija pozvanih u Blueprintu

Meta ili metadata specifikatori kontroliraju kako se klase, sučelje, varijable i funkcije ponašaju prema aspektima razvojnog okruženja. Za svaki tip podataka postoje posebne specifikacije podataka.

UClass i UProperty se postavljaju slično kao i UFunction. UClass se koristi kod deklaracije klase, a UProperty kod deklaracije varijabli, koje također uključuju statički objekt (*eng. StaticMesh*). Glavna razlika je da svaki tip podataka ima različite specifikatore.

5.2 C++ KLASE U UNREAL RAZVOJNOM OKRUŽENJU

C ++ programski jezik podržava enkapsulaciju stvaranjem korisnički definiranih tipova podataka, zvanih klase. Nakon stvaranja, definirana klasa djeluje kao potpuno enkapsulirani entitet i koristi se kao cjelina. Korisnici definirane klase ne trebaju znati kako klasa funkcionira, samo trebaju znati kako se njome služiti (Jones i Liberty, 2004).

Klase u Unreal 4 razvojnom okruženju stvaraju se u Class Wizardu, koji omogućuje dodavanje nativnih Unreal C++ klasa. Svaka klasa se nakon stvaranja sastoji od datoteke klase zaglavlja (*eng. header*) i izvorne datoteke (*eng. cpp*). Datoteke zaglavlja su tekstualne datoteke koje sadrže deklaracije i makronaredbe (Kirch-Prinz i Prinz, 2002).

Zaglavlje klase sadrži deklaraciju klase, deklaracije varijabli, funkcija i glavnih događaja. U izvornoj datoteci su definirane funkcionalnosti klase. Nove funkcije se također deklariraju prvo u header datoteci, te se zatim deklariraju funkcionalnosti u izvornoj datoteci. UFunction i UProperty se također dodaju iznad deklaracija funkcija u header datoteci. Na slici 17 prikazana je nova header klasa.

Unreal ima standardizirani način imenovanja klasa. Klase koje su nasljednici osnovne klase objekata koji se mogu stvarati (*eng. spawnable gameplay objects*) imaju prefiks A, a klase sa prefiksom U ne mogu biti direktno dodane u razinu igre kao instanca klase ili objekt, već samo kao ekstenzija ili komponenta druge klase.

```
3    #pragma once
4
5    #include "CoreMinimal.h"
6    #include "AIController.h"
7    #include "MyAIControllerBase.generated.h"
8
9    /**
10   *
11   */
12   UCLASS()
13   class FAKSCPLUSPLUS_API AMyAIControllerBase : public AAIController
14   {
15       GENERATED_BODY()
16
17   };
18
```

Slika 17. Prikaz novo generirana klasa u C++ programskom jeziku

6 IZRADA IGRE

Razvoj igara obuhvaća stvaranje igre, uključujući dizajn igre, programiranje, umjetnost, pisanje, dizajn zvuka, dizajn razina, produkciju, testiranje, marketing, razvoj poslovanja i još mnogo toga (Macklin i Sharp, 2016).

U svrhu kvalitetnije usporedbe procesa izrade igre, za ovaj rad izrađene su dvije srodne igre. Jedna je napravljena koristeći pretežito programski jezik C++, a druga koristeći Blueprint visual scripting sustav. 3D modeli i drugi elementi koji nisu vezani direktno uz programiranje, a korišteni su za izradu igre, isti su u oba slučaja, odnosno, nema razlike između igara u dizajnu, rasporedu razina ili 3D modelima, već samo u programskim jezicima. Na primjer različiti modeli mogu imati različit broj lica ili poligona. Veći broj poligona može značiti bolji vizualni izgled 3D modela, ali jednako tako znači i sporiji rad igre. Cilj rada je usporedba C++ i Blueprint programskih jezika, a što je manje razlika koje nisu vezane uz programske skripte, to je točnija usporedba kvalitete programskih jezika.

6.1 DIZAJN

„Dizajn igre praksa je začeća i stvaranja načina na koji igra funkcionira, uključujući jezgru akcije, teme, i što je najvažnije, iskustvo i ugođaj tijekom igranja igre“ (Macklin i Sharp, 2016). Za ovaj rad izrađena je igra koja je nazvana Laboratory runner koja predstavlja računalnu igru bočnog prikaza. Igra je podijeljena u četiri levela ili razine. Igrač se nalazi unutar laboratorija. Cilj igre je pobjeći iz laboratorija u ograničenom vremenu i tijekom bijega izbjegavati i prelaziti preko prepreka. Namijenjena je za igru samo jednog igrača (*eng. single player*) s Windows operativnim sustavom i na PC platformi.

Pri pokretanju igre prvo s čime se igrač susreće je glavni izbornik. On ima tri gumba: play (igraj), options (opcije) i end (kraj). Gumb igraj vodi u izbornik za odabir razina. Igra je podijeljena u 4 razine. Pritiskom lijeve tipke miša na gumb „igraj“ otvara se odabrana razina. Pritiskom lijeve tipke miša na gumb „opcije“ u glavnom izborniku otvara se prozor sa opcijama, koja uključuje opcije rezolucije, zvuka i slično. Pritiskom lijeve tipke miša na gumb „kraj“ izlazi se iz igre. Nakon svake razine pojavljuje se Widget kojim se može ići na sljedeću razinu, ponovno pokrenuti razinu ili izaći u glavni izbornik. Kada lik igrača umre poziva se Widget koji također daje opcije ponovnog otvaranja razine ili povratak u glavni izbornik.

Igra ima nekoliko prepreka i izazova koje se trebaju prijeći kako bi se došlo do kraja mape ili razine i pobijedilo. Prvi izazov koji ima svaka razina je vremenski izazov, tj. svaka se razina mora prijeći u ograničenom vremenu. Vremensko ograničenje i zdravlje igrača (*eng. player health*) su jedina dva načina na koji igrač može izgubiti igru. Kada varijabla zdravlja igrača padne na 0 ili niže, igrač je izgubio. Drugi izazovi su podijeljeni i postavljeni kroz cijelu razinu i smanjuju zdravlje igrača. Klase tipa Actor koje služe kao druge prepreka su TurretBase,

SpinObsticle i SpikeObsticle. Klasa TurretBase je toranj koji puca projektile u smjeru igrača u redovnim intervalima, SpinObsticle klasa se vrti oko same sebe i SpikeObsticle je klasa koja se kreće između dvije točke koje mogu biti na Y osi ili na Z osi. Osim ovih prepreka ima i prepreka koje služe kao izazovi, ali ne smanjuju zdravlje igrača, primjerice klasa LiftBase (dizalo).

Igre koje su služile kao inspiracija za izradu ove igre su SpeedRunners, Super Mario 3D World i Celeste. Sve navedene igre su igre bočnoga prikaza i igre u kojima je izbjegavanje prepreka glavni izazov koja igra donosi.

6.2 IZRADA MAPE

Izrada mape u Unreal razvojnom okruženju, osim u nekim iznimkama kao na primjer automatski generiranim mapama, izrađuje se bez programskog koda. Zbog točnije usporedbe računalnih igara obje mape i tereni izrađene u ovome projektu izgledati će jednako. Izrada mape je u oba slučaja pratila isti proces, te stoga nije potrebno posebno opisati proces izrade razine u dokumentaciji C++ i Blueprint implementacije igre.

Za izradu terena korišteni su isključivo 3D modeli i objekti za osvjetljenje terena. 3D modeli korišteni u izradi mape su modeli iz Unreal Engine tržnice (Unreal Engine marketplace), 3D modeli izrađeni posebno za ovaj projekt u vlastitoj izradi i polazni UE4 modeli koji su integrirani u programu. Modeli koji su korišteni s Unrela Engine tržnice su:

- Soul: City (Epic Games)
- GOOD SKY (UNEASY)
- Landscape Backgrounds (Gokhan Karadayi)
- Platformer Starter Pack (Platfunner)
- IES Light Profile Pack (Construct Games)
- Brushify - Arctic Pack (JoeGarth)
- Science Laboratory (SilverTm)
- Stylized Character Kit: Casual 01 (ROCKETARTS)

Izrada uz pomoć pejzažnog alata (*eng. landscape tool*) nije bila potrebna jer se radi o igri bočnog prikaza u kojima se obično ne koriste organski oblici u terenima, nego 3D modeli i slično. Veliki dio terena je izgrađen uz pomoć platformi koje su istovremeno i dio mehanike igara, a više o njima biti će dokumentirano u poglavljima 6.3, 6.4 i 6.5.

6.3 IMPLEMENTACIJA BLUEPRINT SUSTAVA

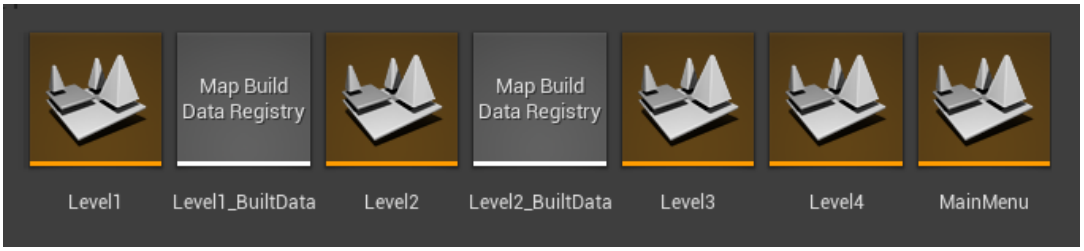
Bluprint sustav za razvoj računalnih igara u početku je bio izrađen i usmjeren dizajnerima i početnicima bez prevelikog znanja o programiranju, no u posljednjim godinama značajno se poboljšao. Računalna igra u ovoj je verziji napravljena isključivo koristeći Blueprint programski jezik. Cilj izrade bio je bolje upoznavanje se Blueprint jezikom i njegovim mogućnostima, prednostima i nedostacima. On ima nekoliko baznih kategorija, koje su prikazani na slici 18. Za izradu Blueprint sustava korištena je Side Scroller bazna kategorija. Side Scroller bazni tip igre sadrži klase potrebne za igre bočnog prikaza. Ova bazna kategorija već sadrži neke ugrađene klase i funkcije koje pomažu korisniku u bržem razvoju računalne igre.

Slika 18. Izbornik baznih igara u Unreal razvojnom okruženju

6.3.1 Podjele razina

Razina, u kontekstu Unreal razvojnog okruženja, može biti definirana kao zbirka predmeta i njihovih svojstava koji zajedno definiraju područje igranja (Nixon, 2017). Igra napravljena u svrhu ovog projekta je podijeljena na nekoliko razina koje su prikazane na slici 19.

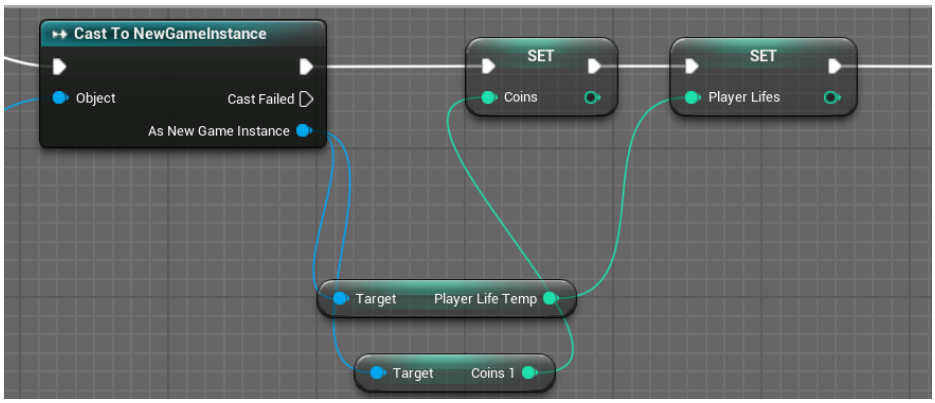
Otvaranjem igre otvara se prvo glavni izbornik. Razvoj igre sa nekoliko razina u Unreal razvojnom okruženju može se ostvariti na nekoliko načina. Igre sa nekoliko razina često imaju varijable koje trebaju biti podijeljene među razinama. Svaki put kada se pozove nova razina sve varijable koje su bile spremljene u Blueprintu razine i Blueprint klase ponovno se pokreću. Ovaj problem može se riješiti na nekoliko načina, koristeći instancu igre ili level streamingom.



Slika 19. Primjer podjele razina u Unreal razvojnom okruženju

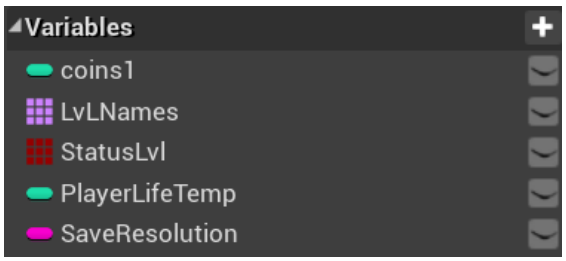
Level streaming omogućuje spremanje i učitavanje razine ili mape u memoriju, odnosno omogućuje učitavanje razine, skrivanje i prikazivanje istih. Neki primjeri u koju svrhu se koristi level streming bile bi igre otvorenoga svijeta, gdje se ne mora učitati cijeli svijet odjednom nego se učitavaju samo određeni dijelovi ovisno o poziciji igrača, ili igre sa više razina, kako bi se bitne varijable, koje se moraju slati među razinama, mogle tijekom igre spremati u jednoj razini i učitavati u ostale, bez nepotrebnog koda za slanje i spremanje podataka u memoriju ili instancu igre. Za igre otvorenoga svijeta level streaming omogućuje stvaranje velikog svijeta sa dobrim performansama. Osim performansa ova tehnika može također ubrzati izradu mapa. Budući da je razina podijeljena na nekoliko slojeva koji su postavljeni jedan preko drugoga, nekoliko dizajnera mogu istovremeno raditi svatko na svom sloju. Prije streaminga potrebno je postaviti centralnu razinu koja će služiti kao čvorište varijabli, klasa, funkcija i drugih slojeva. U slučaju igre izrađene za ovaj projekt to je Glavni meni ili MainMenu razina. Kao djeca razine koja su uvijek učitana bile bi postavljene razine od 1 do 4.

Drugi način komunikacije među razinama je kroz instancu igre (*eng. game instance*). Instanca igre je klasa u kojoj se stanje varijabli i funkcija ne mijenja kod ponovnog učitavanja razine. U klasama poput Actor ili Character svaka razina stvara novu instancu klase, čak i klase poput PlayerController i game moda nakon učitavanja nove mape ponovno se pozovu, dok u slučaju klase instance igre *(eng. game instance*) to nije tako. Instanca igre je klasa na razini cijele igre, odnosno stanje se mijenja tek u slučaju ponovnog otvaranja igre, a ne pri otvaranju nove razine.



Slika 20. Prikaz djela funkcije za spremanje podataka u instancu igre

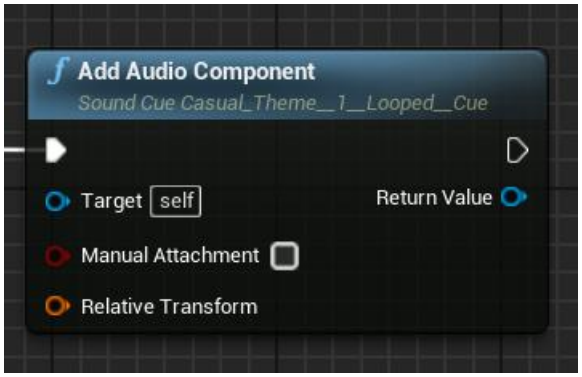
Za izradu ove igre odabrana je instanca igre kao način dijeljenja podataka među razinama. U instanci igre spremljene su sve bitne varijable koje se moraju slati među razinama. Varijable poput skupljenog novca, preostalog života i slično spremaju se u instanci igre. Na kraju svake razine, prije otvaranja nove razine, kroz Blueprint se spremaju podatci iz player character klase u instancu igre i na početku nove razine se ponovno učitaju iz instance u player character klasu. Za pozivanje instance u Blueprint sustavu koristi se čvor za pristup funkcijama iz drugih klasa, u ovom slučaju klasa instance igre (*eng. cast to game instance*). Kao referenca koja ulazi u Object input ide referenca na trenutnu instancu igre. U slučaju spremanja podataka iz instance u PlayerCharacter klasu nakon čvora CastToGameInstance idu Set funkcije iz PlayerCharacter klase, a Get funkcije iz Instance. Ovo se poziva nakon BeginPlay događaja.



Slika 21. Prikaz varijable NewGameInstance klase

6.3.2 Audio

Zvukovi i glazba u obje implementacije igre su isti. Kao i kod 3D objekta zvukovi i glazba su sa Unreal Engine tržnice ili integrirani u sam softver. Zvukovi u računalnim igrama imaju velik spektar uloga. U ovoj igri objekti poput tornja (*eng. turret*) i svaka mapa imaju svoj zvuk, a svaka zvučna komponenta ima svoju ulogu. Toranj ima zvučnu komponentu koja se poziva kada toranj puca. U Blueprintu zvučna komponenta dodaje se sa čvorom AddAudioComponent. Čvor u Blueprintu u slučaju turret klase dodan je direktno nakon Fire čvora, kako bi se direktno nakon njega čuo zvuk. Čvor koji dodaje zvuk je prikazan na slici 22.



Slika 22. Prikaz čvora za dodavanje zvuka

Igra ima i opciju smanjenja i pojačanja zvuka glazbe, a to se regulira u opcijama. Nakon promijene glasnoće, razina na kojoj je postavljena se sprema u varijablu, koja se iz Widget Bluprint klase opcija šalje i sprema u instancu igre. Za upravljanje glasnoćom zvuka glazbe u Blueprint sustavu Unreal 4 razvojno okruženje ima AdjustVloume čvor. Čvor prima referencu na audio komponentu i varijablu tipa Float koja predstavlja razinu zvuka kao input. Varijablu za postavljanje razine zvuka vuče se iz instance igre.

6.3.3 SideScrollerCharacter klasa

Unreal 4 razvojno okruženje za razvoj računalnih igara nudi nekoliko baznih tipa igara. One služe samo kao bazni sloj, koji sadrži osnovne mehanike za jednostavno kretanje glavnog igrača, svijet i kameru. Bazni tipovi igara služe bržem i jednostavnijem razvoju računalne igre. SideScrollerCharacter jedna je od baznih klasa Unreal razvojnog okruženja za igre bočnog prikaza (*eng. side scroller*). Ona je dijete Character klase koja je korištena kao bazna klasa i za druge bazne tipove igara poput igre prvog i trećeg lica. Kao jedna od centralnih klasa za razvoj ove računalne igre, ona sadrži nekoliko funkcija i događaja. Neke od varijabli i komponenti su vidljive na slici 23.

Klasa SideScrollerCharacter imala je prioritet u razvoju jer igrač najveći dio vremena igrajući igru provodi kontrolirajući glavnog lika igre. Ova klasa je u centru tijekom cijele igre. Kontrole i mehanike navedene klase iznimno su bitne za zadovoljstvo i zabavu igrača. Kao centralni dio igre u SideScrollerCharacter klasi spremljene su mnoge varijable, funkcije i događaji koje se pozivaju u drugim klasama. Varijable koje ova klasa sadrži pune se na početku svake razine iz instance igre. Na kraju svake igre podaci iz varijabli spremaju se u instanci igre i ponovno pune u ovu klasu na početku iduće razine. Na slici 23 vidljive su neke od varijabli i komponenata ove klase.

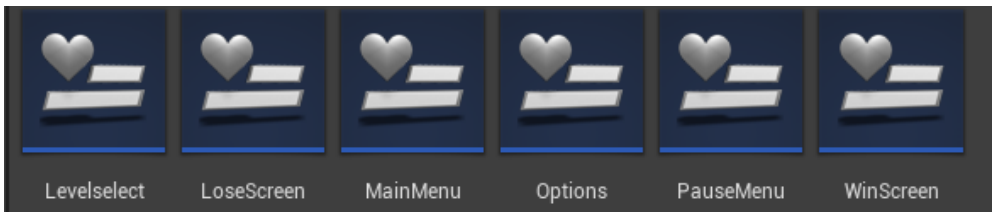
Klasa ima događaje koji se naknadno u drugim klasama okidaju. Događaji „dodaj život“ ili „smanji život“ ovise o prethodnom događaju i smanje ili povećaju broj života lika igrača. Događaji kretanja, skoči i trči su događaji koji primaju input sa tipkovnice i na temelju toga pomiču lika. Događaji za otvaranje Widgeta okidaju se kada je igra izgubljena ili pobijeđena i za otvaranje menija za pauzu. Zadnji događaji se okidaju na početku svake razine i imaju ulogu postavljanja nekih varijabli poput: preostalo vrijeme, preostalo života i preostalo zlata. Ovi događaji su zasebno napravljeni i okidaju se u Blueprint skripti u drugim klasama za razliku od zadanih događaja koji se pozivaju kroz igru.

Slika 23. Prikaz varijabli u SideScrollerCharacter klasi

Varijable spremljene u ovu klasu koriste se kao baza koja se zatim nasljeđuje na druge klase i događaje. Svaka naslijeđena varijabla se u drugoj klasi manipulira i vraća. Varijabla Player Acceleration je tipa Float. Koristi se u animacijskom nacrtu kako bi nacrt znao koju animaciju trčanja prikazati. PlayerFuel i PlayerHealth su tipa Float i prikazani su u ProgressBaru na vrhu korisničkog sučelja. Coins je varijabla tipa intiger koja se nasljeđuje na glumca CoinsActor koji povećava broj varijable za 1 u slučaju kontakta sa igračem. PlayerLives se koristi nekoliko puta kroz izrađenu igricu: za dodavanje života u slučaju da broj novca prijeđe 100, za oduzimanje života ako Timer bude 0, u slučaju da PlayerHealth bude 0 i u instanci igre za nasljeđivanje broja života na sljedeću razinu. Boolean varijable IsDead, DiedToTime i InvnicbilityFrames se koriste kako bi se utvrdilo stanje lika.

6.3.4 Korisničko sučelje i Blueprint Widget klase

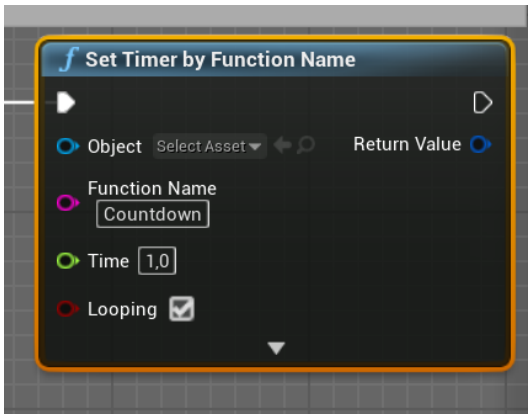
Zajedno sa Heads-up zaslonom ili HUD-om (*eng. heads-up display*), koji sadrži vizualne komponente unutar razina, Blueprint projekt ima 7 Blueprint Widgeta, koji se pozivaju u slučaju nekih događaja u razinama ili u glavnom meniju. Neke od Blueprint Widget datoteka prikazane su na slici 24. Uloga HUD-a je prikaz preostalog vremena, života, novaca i mane. On se dodaje u PlayerCharacter Blueprintu na početku igre i prikazuje na ekranu.



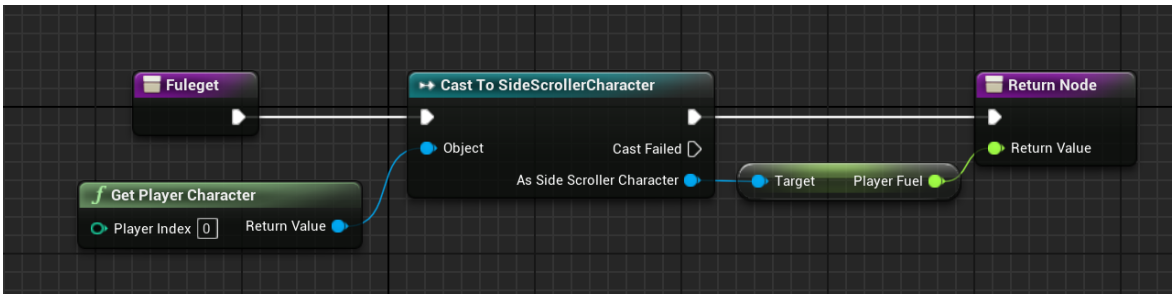
Slika 24. Prikaz Blueprint Widget klasa izrađenih računalnih igara

Za prikaz preostalog vremena koristi se SetTimerByFunctionName čvor koji svaku sekundu ponavlja događaj ili funkciju. Sama funkcija u sebi sadrži brojač koji broji od 150 na niže. Ovaj Blueprint čvor zajedno sa njegovim inputima i outputima prikazan je na slici 25. Kada je brojač

jednak nuli, Timer se briše i oduzme se 1 život. Podaci brojača se istovremeno šalju u HUD Widget Blueprint i prikazuju na ekranu igrača. Prikaz vremena kroz HUD koristi funkciju koja je vezana za TextBlock. TextBlock i funckije koje su vezane uz njih stvaraju se automatski i traže input u return čvor. Ta funkcije poziva se za svaku prikazanu sliku u sekundi. Ostale funkcije koje prikazuju informacije na HUD, rade na sličan način kao i Timer, tj. uzimaju podatke iz SideScrolerCharactera i prikazuju ih na HUD. Ovo je prikazano na slici 26.



Slika 25. Prikaz Timer čvora

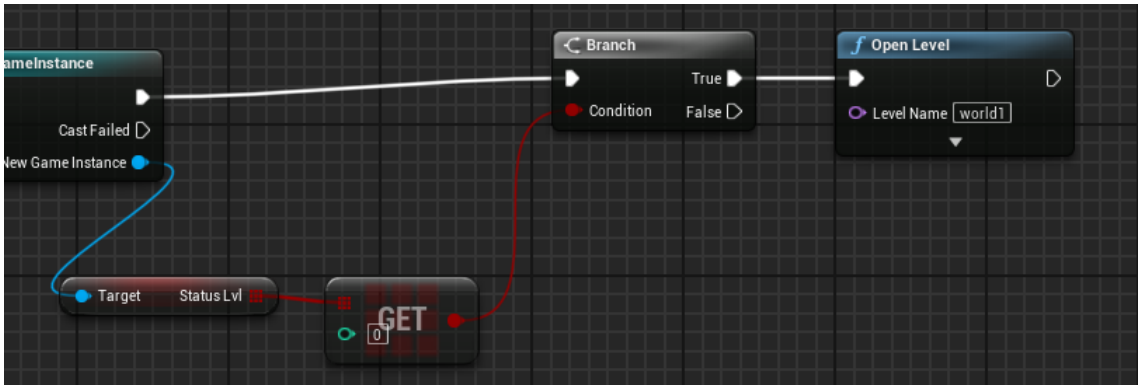


Slika 26. Prikaz funkcije za ispis na korisničko sučelje

Elementima u izgradnji korisničkog sučelja može se dodati Binding ili vezu. On stvara novu Blueprint funkciju koja ima početni čvor, koji se poziva kao i Tick event, i return čvor koji vraća output. Output čvor se razlikuje ovisno o kojem se elementu radi. Osim Bindinga neki elementi u Blueprint Widgetima imaju i događaje. Jedan od događaja koji se koristi kod UBurtton elementa je on Click event. Button ili gumb ima još nekoliko događaja koji su povezani sa pritiskom lijeve tipke miša, samo se pozivaju u različitim trenutcima. Primjerice, jedan se poziva u trenutku pritiska, dok se drugi poziva u trenutku puštanja. Princip događaja u korisničkom sučelju sličan je kao i u izgradnji aplikacija u nekim drugim softverima koji koriste slušatelja (*eng. listener*). Događaji gumbova su OnClicked, OnPresed, OnRelesed, OnHoverd i OnUnhoverd. OnClicked i OnPresed okidaju se na pritisak lijeve tipke miša. Razlika je da se OnPressed okida na početku pritiska, a OnClicked na kraju.

MainMenu je Blueprint Widget koji se otvara na početku igre, te se u njega može ući i u svakom trenutku igre kroz izbornik pauze, nakon pobjede i nakon poraza. Glavni izbornik ima 3 gumba od kojih svaki vodi u svoj posebni Widget Blueprinta: izbornik razine, izbornik opcija i izlazak iz igre. Za otvaranje razine, na pritisak gumba uglavnom se koriste tri čvora. RemoveFromParent je čvor koji se koristi u svrhu brisanja trenutnog Widgeta prije nego što se pozove novi. Drugi čvor je CreateWidget. On kreira Widget i prima klasu Widgeta kao input. Treći čvor u nizu je AddToViewport čvor. Funkcija ovog čvora je prikaz kreiranog Widgeta na ekranu igrača. Kao input prima objekt koji je kreiran sa prijašnjim čvorom.

LevelSelect je Widget sa izbornikom razina. Widget ima četiri gumba za otvaranje razina, jedan za povratak u glavni izbornik i jedan za učitavanje spremljenih podataka. Glavne funkcije korištene u izradi ove klase su OpenLevel funkcija, DoesSaveGameExist, SetVisibility i LoadGameEvent. OpenLevel funkcija poziva se samo u slučaju da je prijeđena prijašnja razina. Provjera se vrši branch funkcijom, koja je Blueprint verzija If funkcije. Podatak za provjeru dobije se iz liste tipa Bool sa prijeđenim razinama.

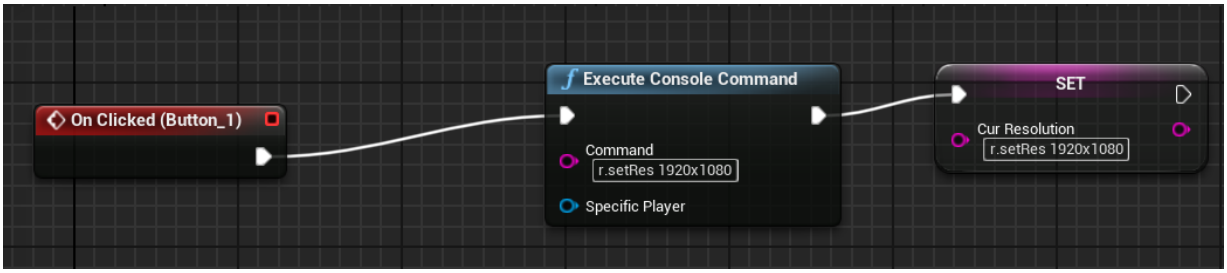


Slika 27. Prikaz funkcija za provjeru uvjeta za otvaranje nove razine

DoesSaveGameExist je Unreal Blueprint funkcija koja prima ime utora i indeksa spremljenih podataka i na temelju njih vraća informaciju da li ima spremljenih podataka. Ova funkcija vraća varijablu tipa Bool. SetVisibility je funkcija koja prima kao input komponentu tipa gumb ili neku drugu Widget komponentu. Funkcija skriva ili prikazuje komponentu igraču. U ovoj klasi podatak koji vraća DoesSaveGameExist funkcija ide u If funkciju. Ovisno o rezultatu, prikazuje se gumb za učitavanje spremljene igre ili ne. Osim ovih funkcija poziva se i događaj iz druge klase, LoadGameEvent i nalazi se u instanci igre. LoadGameEvent okida funkcije koji učitaju spremljene podatke. Više o ovome biti će napisano u dokumentaciji klase instance igre.

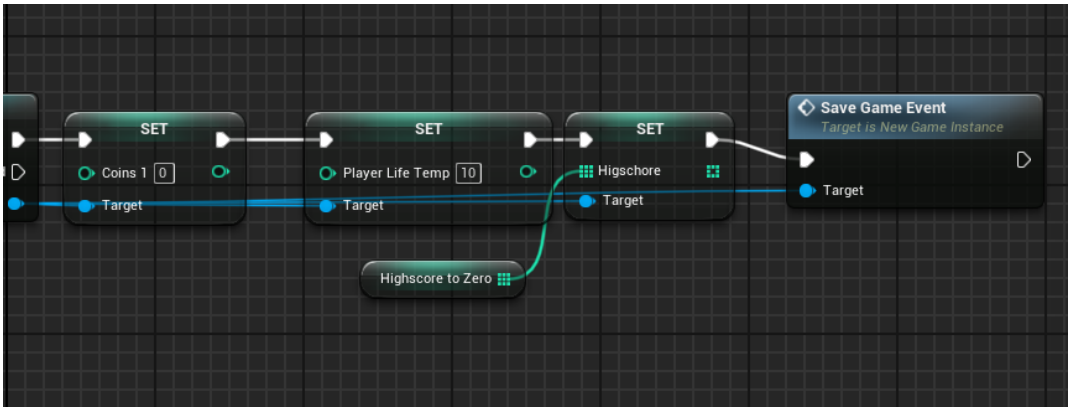
Izbornik opcija ima 3 dijela ili sekcije. Prvi dio radi i mijenja rezoluciju, drugi radi sa spremanjem i brisanjem podatka i treći dio sa kontrolama. U prvom dijelu za spremanje na gumb komponente dodan je OnClick događaj koji na pritisak mijenja rezoluciju ekrana. Nakon promjene trenutne rezolucije, nova rezolucija se sprema u varijablu. Sadržaj te varijable se

nakon zatvaranja izbornika sprema i prenosi u novu varijablu koja se nalazi u instanci igre. Sekcija sa kontrolama otvara novi Widget sa listom i opisom kontrola.



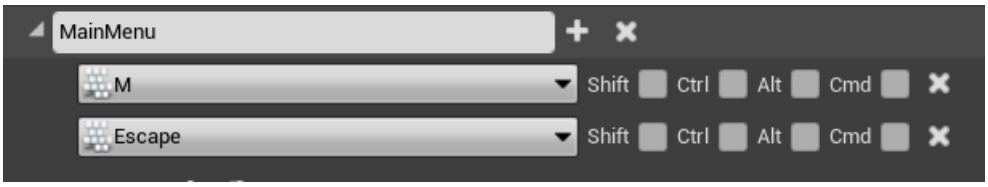
Slika 28. Prikaz funkcije za promjenu rezolucije

Sekcija za spremanje i brisanje u izborniku opcija ima dva gumba. Prvi je običan gumb za spremanje. Na pritisak lijeve tipke miša okida se događaj koji sprema podatke u klasi instance igre. Drugi gumb je za brisanje spremljenih podataka i vraća sve podatke na nulu.



Slika 29. Prikaz dijela funkcije za okidanje događaja SaveGameInstance

U postavkama projekta postavljeni su novi inputi tipki koji su u Blueprint klasi lika igrača pozvani i naknadno su im dodane funkcionalnosti u Blueprint uređivaču. Dodane tipke otvaraju izbornik za pauzu. Tipke se mogu i posebno dodavati u Blueprintu, ali na ovaj način se uvijek mogu dodati i mijenjati kontrole, bez da se mijenja blueptint ili C++ kod. Izbornik pauze ima opcije za nastavak igre, spremanja igre, ponovni start razine i izlazak na glavni izbornik. Spremanje radi na isti način kao i izbornik opcija, tako što se poziva događaj za spremanje iz instance igre. Ponovni start razine sprema podatke života u SaveGameInstance i ponovno pokreće igru nakon čega se podaci iz klase lika igrača opet pune sa spremljenim podacima iz SaveGameInstance.

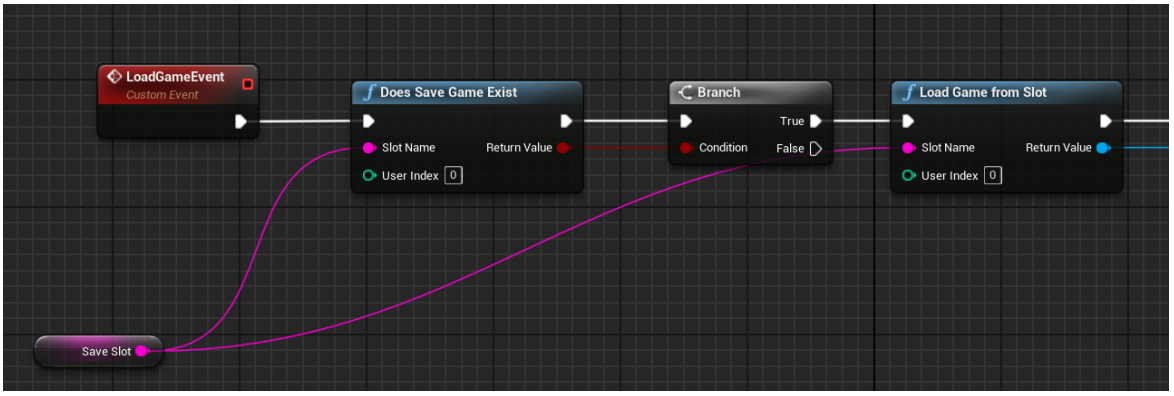


Slika 30. Prikaz vezanja tipki za događaj

6.3.5 Spremanje i instanca igre

Blueprint klasa za spremanje podataka sadrži isključivo podatke, a roditeljska klasa na kojoj je Blueprint klasa bazirana sadrži polazne funkcije. Zadatak Bluprint klase za spremanje podataka je prijenos i spremanje podataka u Slot. Događaj i pozivanje funkcija odvijaju se u instanci igre.

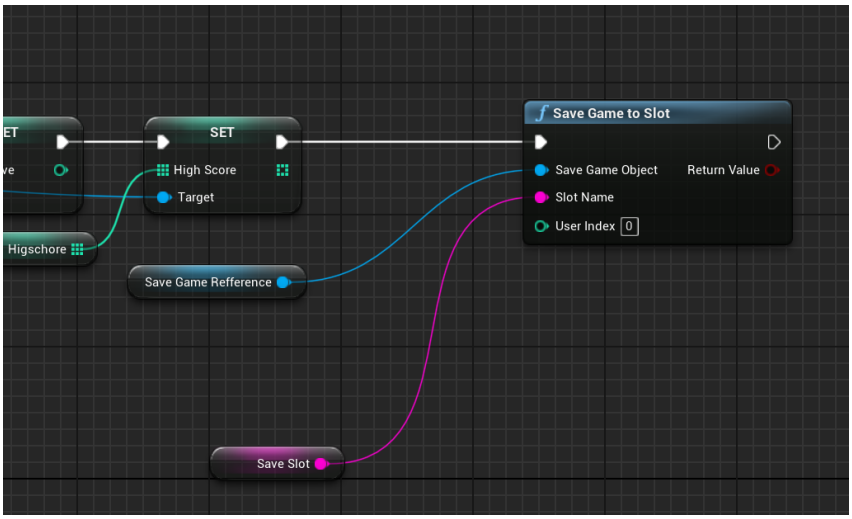
Instanca igre, osim događaja i funkcija koje okidaju spremanje igre, ima i druge podatke koje je potrebno slati iz jedne razine u drugu. U klasi igrača instanca se poziva na početku i na kraju svake igre. Na početku svake razine poziva se CastToGameInstance funkcija koja kao referencu uzima instancu igre. Ona omogućuje pozivanje drugih funkcija, događaja i varijabli iz klase instance igre ukoliko oni nisu privatni. Unreal ima ugrađene set i Get funkcije za varijable. Na početku svake razine, odnosno nakon otvaranja nove razine, koriste se set funkcije iz klase igrača i uzimaju podatke koji su spremljene u instanci igre, a na kraju razine, odnosno prije zatvaranja razine, koriste se Get funkcije iz klase igrača, podaci dobiveni Get funkcijom spremaju se u instancu igre. Na ovaj način se podaci prenose iz jedne razine u drugu.



Slika 31. Prikaz događaja za učitavanje spremljenih podataka

Instanca igre ima 2 događaja: LoadGameEvent i SaveGameEvent. Ovi događaji nisu polazni događaji koji su napravljeni tijekom igrom i već imaju okidače, nego su naknadno napravljeni. Oni se okidaju u Widget funkcijama na pritisak gumba za spremanje ili učitavanje spremljenih podataka. Nakon pritiska na gumb okida se događaj iz instance igre. Nakon što se događaj za učitavanje spremljenih podataka okida, poziva se funkcija DoesSaveGameExist koji provjerava postoji li spremljeni podatak i vraća podatak tipa Bool. Podatak koji se vraća ide u branch funkciju i u slučaju istine učitava se spremljena igra funkcijom LoadGameFromSlot. Ona kao

input ima ime slota i broj indexa. Slot je spremljen kao varijabla u instanci igre. Funkcija vraća podatak tipa SaveGame iz kojeg se mogu izvući spremljeni podaci. SaveGameEvent događaj nakon okidanja kreira SaveGameObjekt kao input prima SaveGame klasu. Nakon što se objekt kreirao, podaci iz instance spremaju se u varijable u objektu klase SaveGame. Nakon spremanja podataka objekt se stavlja kao input za funkcjiu SaveGametoSlot.



Slika 32. Prikaz funkcije za spremanje podataka

6.3.6 Blueprint Mehanike

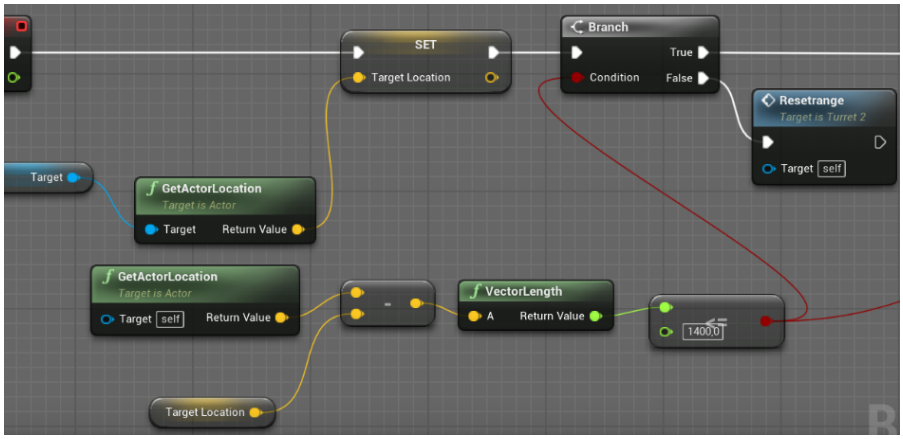
Mehanike su operacije i funkcije, sa velikim brojem uloga, koje su postavljeni na nekim objektima. U Unreal razvojnom okruženju objekti sa mehanikama zovu se Actori. „Igrajući igru, igrači traže izazov, majstorstvo i nagradu, sve upakirano u podrazumijevajuće i motivirajuće aktivnosti. Iz toga proizlazi važnost igranja igre kao ključnog kamena temeljaca igre, a mehanika igre kao alata s kojima igrač mora komunicirati kako bi provodio aktivnosti igranja“ (Fabricatore, 2007).

6.3.6.1 TurretBase Klasa

Turret je UE4 Blueprint klasa tipa Actor. Turret ima tri glavna stvorena događaja. Svaki događaj se pod određenim uvjetima poziva u Tick događaju. Turret traži lokaciju igrača, provjerava je li igrač u dometu i ako je, tada poziva AimAtTarget događaj, te nakon što nacilja igrača, poziva Fire događaj. AimAtTarget događaj je prikazan na slici 33. Nakon što igrač izađe iz dometa Turreta poziva se ResetRange događaj.

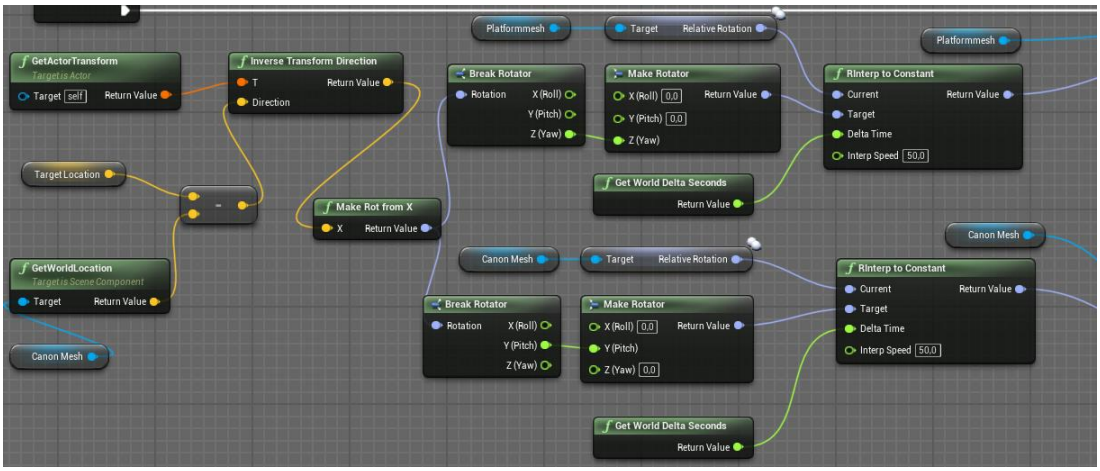
Provjera udaljenosti igrača i dometa radi se u Tick događaju, a on se okida se za svaku sliku (*eng. frame*). Funkcija GetActorLocation je funkcija koja prima objekt klase tipa Actor i vraća njegovu lokaciju. U implementaciji Blueprint igre ova funkcija vraća lokaciju igrača i lokaciju

tornja. Vektori su varijable sa lokacijom, a kad se jedna pozicija oduzima od druge, dobivena udaljenost ide u VectorLength varijablu koja pretvori varijablu u Float format. Nakon toga slijedi provjera udaljenosti i ovisno o rezultatu slijedi okidanje događaja.



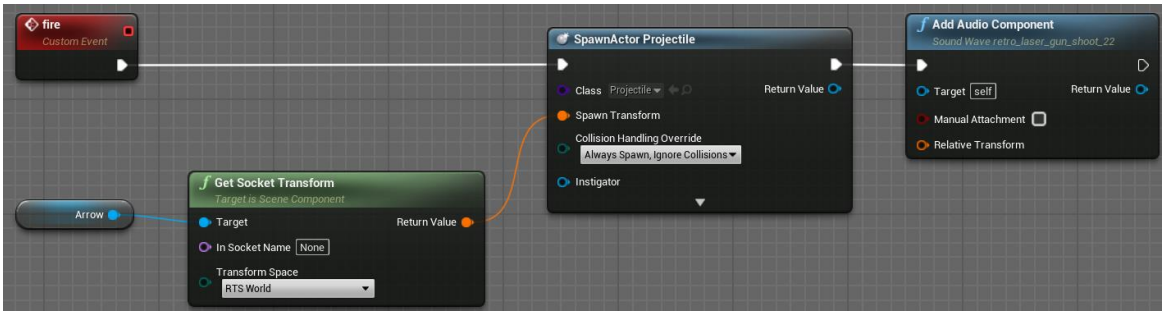
Slika 33. Prikaz funkcije za spremanje lokacije u TurretBase klasi

Rotacija i smjer tornja u slučaju da je igrač u dometu kalkulira se u AimAtTarget događaju. Nakon kalkulacije rotacije i rotatora na temelju lokacija igrača i rotacije toranja, dobivene rotacijeidu u Rinterp funkciju zajedno sa trenutnom rotacijom komponenata toranj. Rotinterp funkcija uzima trenutnu rotaciju, ciljanu rotaciju i ciljanu brzinu kretanja i na temelju toga vraća rotaciju sa konstantnom brzinom i tečnom animacijom. Nakon kalkulacije rotacije vraćena vrijednost zajedno sa željenom komponentom ide u SetRelativeRotation funkciju koja rotira komponentu po željenoj rotaciji. Kalkulacija rotacije prolazi kroz dvije faza. U prvoj se fazi kalkulira pozicija i smjer igrača, a u drugoj se koriste funkcije Brake i Make rotator koji rotaciju dijele po svakoj osi. Platforma se rotira samo po lokalnoj Z osi, a top samo po lokalnoj Y osi.



Slika 34. Prikaz funkcije za kalkulaciju rotacije u TurretBase klasi

Fire događaj okida se nakon AimAtTarget događaja. Između okidača događaja za ciljanje i događaja za pucanje aktivna je Delay funkcija koja odgađa sljedeći događaj za nekoliko sekundi. Sekunde se ručno unose ili prenosi iz varijable tipa Float. Funkcija služi da bi ograničila broj projektila. Bez funkcije Delay u svakoj slici u sekundi pucao bi jedan projektil, a u ovom slučaju puca u redovitim intervalima. Nakon okidanja poziva se SpawnActor funkcija. Funkcija prima kao input smjer, rotaciju i veličinu komponente. Također, dodaje objekt klase u razinu, u smjeru i na poziciju u kojoj se nalazi komponenta. Brzina kretanja dodana je u klasi koja se poziva kao komponenta, a u istom trenutku se poziva i audio komponenta sa zvukom pucanja.

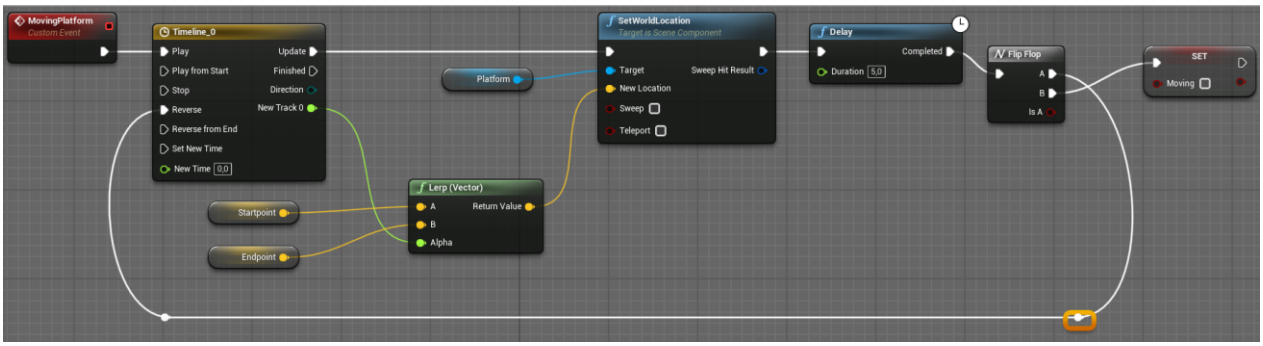


Slika 35. Prikaz funkcije Fire događaja u TurretBase klasi

ResetRange događaj se okida u slučaju kada igrač nije u dometu tornja. Događaj uzima originalnu rotaciju komponenata i istom funkcijom kojom pomiče toranj prema igraču, vraća toranj u početnu poziciju. Događaj prati slični postupak kao i AimAtTarget događaj, a razlika je samo u varijablama rotacije koji u ovom slučaju vraćaju toranj na početni položaj tj. vraćaju lokalnu rotaciju komponenata na nulu.

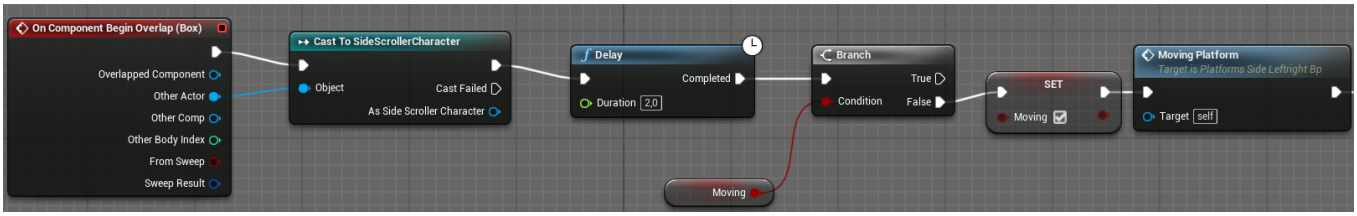
6.3.6.2 LiftBP Klasa

LiftBP Blueprint je klasa tipa Actor. Zadatak LiftBP klase je premještaj igrača od pozicije A na poziciju B pomicanjem platforme. Za kalkulaciju pozicija i premještaj klase na drugu poziciju korištene su Blueprint funkcije. Timeline, SetActorLocation i Lerp rade zajedno kako bi pomicali platformu od točke A do točke B. Timline je funkcija koja prima graf od 2 dimenzije i nekoliko točaka. U grafu os X simbolizira prolazak vremena, a os Y postotak pokreta. Minimalna vrijednost je nula, a maksimalna je jedan. Timeline se koristi kako bi animacija i okret između dvije točke bili tečniji. Output Timeline funkcije ide u Lerp vektor zajedno sa dva podatka tipa vektor koji imaju spremljene lokacije za početak i kraj kretanja. Lerp ili Linear Interpolate kalkulira put između dvije lokacije. Ulaz iz Timeline funkcije opisuje brzinu i tijek kretanja. Lerp vraća lokacije koje onda idu u SetActorLocation funkciju, koja zatim pomiče klasu sa dobivenim lokacijama.



Slika 36. Prikaz funkcije za pokretanje platforme u LiftBP klasi

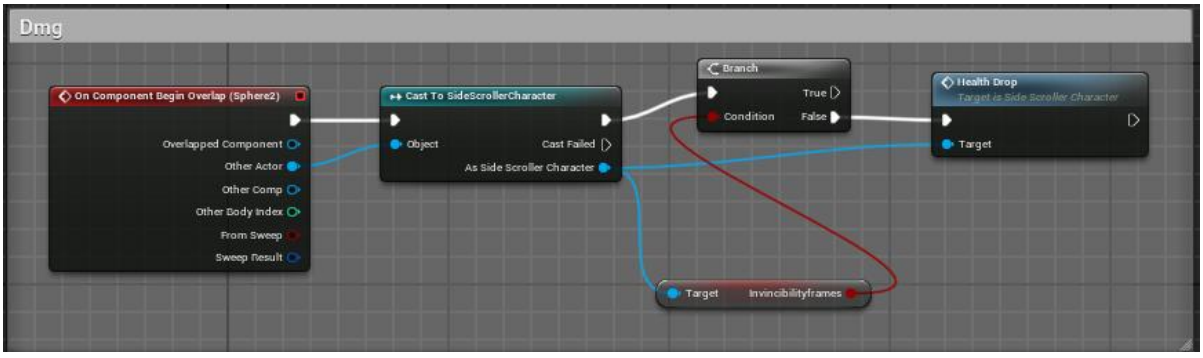
Ova klasa ima BoxComponent komponentu ili komponentu koja u slučaju kolizije sa drugim objektom u sceni poziva događaj.



Slika 37. Prikaz okidanja funkcije za pomicanje platforme

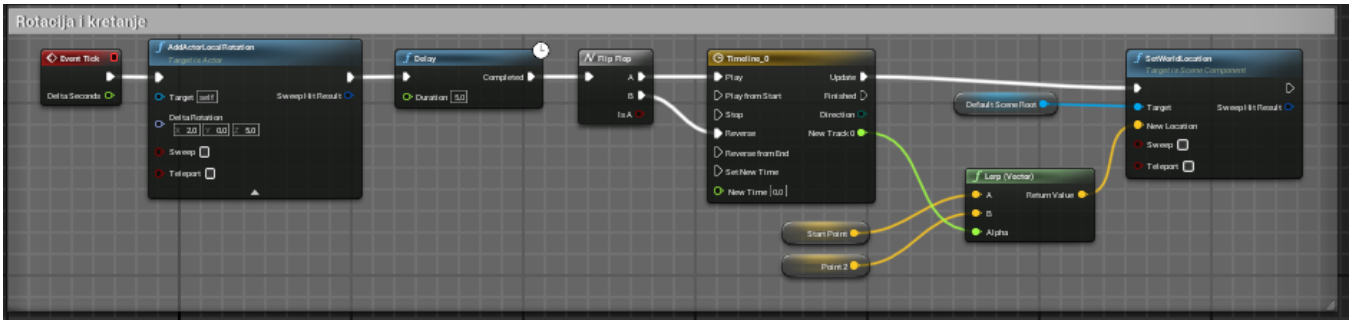
6.3.6.3 Spike Klasa

Spike je jedna od Blueprint klasa bazirana na Actor klasi. Zadatak joj je kretanje između dvije točke i u slučaju kolizije sa igračem okida događaj HealthDrop u SideScrollerCharacter klasi. Klasa LiftBP koja pomiče platformu sa jedne pozicije na drugu radi na sličan način. Razlika je u tome što na kontakt sa igračem pozove funkciju za pomicanje, a ne funkciju HealthDrop koja smanjuje život igrača. Komponente LiftBP klase su također jako slične. Klasa ima dvije StaticMesh komponente i jednu BoxComponent komponentu.



Slika 38. Prikaz okidanja događaja HealthDrop

U slučaju kontakta poziva se funkcija iz klase SideScrollerCharacter. Klasa koja se poziva je HealthDrop i smanjuje varijablu Health. Ona provjerava padne li vrijednost varijabla ispod nule, te u slučaju da padne, poziva se Widget koji informira igrača da je izgubio igru.

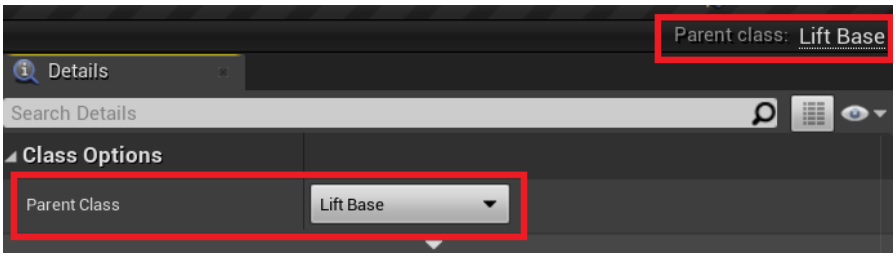


Slika 39. Prikaz funkcije za lokalnu rotaciju i kretanje objekta klase

Pomicanje platforme okida se u Tick događaju. Za pomicanje platforme koriste se iste funkcije kao i za objekt LiftBP klase - Timeline, Lerp i SetActorLocation. FlipFlop funkcija je funkcija koja se izmjenjuje između outputa A i B. Prvi output iz funkcije FlipFlop ide u Timeline funkciju u Play input, a drugi output ide u Reverse input Timeline funkcije. Play radi pokret u jednom smjeru, a Reverse vraća objekt na početnu poziciju. U Tick događaju osim pomicanje objekta LiftBP klase ima i funkcija AddLocalRotation koja dodaje lokalnu rotaciju objektu klase.

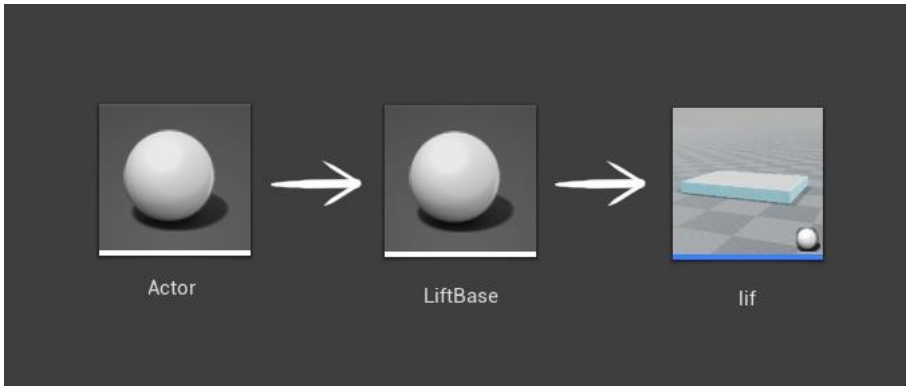
6.4 C++ Razvoj

Za C++ razvoj igre korištena je tehnika koja postavlja C++ klasu kao roditelja Blueprint klase. Sve su varijable i funkcije spremljene u C++ klasi, te su tu inicijalizirani i događaji i komponente. U Blueprint klasi se uređuju samo komponente koje su dodane kroz C++ baznu klasu i dodaju se neke manje funkcije ukoliko je to potrebno. Primjer jednog odnosa među klasama, u ovome slučaju LiftBase klasa, prikazan je na slici 41. Actor je bazna klasa na kojoj je LiftBase, odnosno C++ klasa bazirana. Nakon toga LiftBase Blueprint klasa nasljeđuje LiftBase klasu iz C++-a. Postavljanje roditeljske klase za Blueprint klase prikazano je na slici 40. Za izradu igre u C++ korišten je program Visual studio.



Slika 40. Primjer izbora roditeljske klase

Jedna C++ klasa može se koristiti za nekoliko interaktivnih objekata, primjerice za podizanje novčića (CoinActor), zdravlja (HealthActor), izdržljivosti (Stamina/Fuel). Veći dio procesa, sa nekoliko manjih iznimka, izrađen je u C++ programskom jeziku.



Slika 41. Prikaz odnosi između klasa u C++ projektu

6.4.1 C++ Klase

Klasa je osnovni element u objektno orijentiranom programiranju, te se na temelju klase stvara objekt, odnosno instanca te klase. Sljedeća poglavlja sadrže podatke i dokumentaciju koja je pisana tijekom izrade C++ verzije računalnih igara, odnosno tijekom izrade igre koja je programirana u C++ kodu.

6.4.1.1 SideScrollerCharacter Klasa

SideScrollerCharacter u C++ implementaciji igre zauzima sličnu ulogu kao i u Blueprint implementaciji. U C++ implementaciji igre, C++ klasa igrača je postavljena kao roditeljska klasa Blueprint SideScrolerCharacter klase. Objekt ove klase je centar igre, te se iz perspektive igrača cijela se igra vrti oko glavnog lika igre. Osim iz perspektive igrača, u programskom kodu ova klasa je također jedna od centralnih klasa igre. Klasa sadrži velik broj varijabli i funkcija. Funkcije iz ove klase pozivaju se iz većine drugih klasa. Primjerice, funkcija HealthDrop iz ove

klase, kojoj je zadatak smanjiti životne bodove igraču, koristi se u klasama ProjectileBase, SpikeBase i TurretBase.

Svi elementi prikazani na HUD-u su inicijalizirani i spremljeni u ovoj klasi. U ove elemente spadaju broj života, novca i preostalog vremena i ProgressBar sa preostalim životom (*eng. health*) i kondicijom (*eng. stamina*). Sve varijable imaju Get funkcije koje se onda u Widgetu pozivaju i ispisuju na ekranu. Varijabla u kojoj se sprema vrijeme puni se kroz nekoliko funkcija i događaja. Za izradu odbrojavanja vremena u C++ implementaciji koristile su se funkcije GetWorldTimeManager i SetTimer, koje ponavljaju određenu funkciju svake sekunde. Funkcija za inicijalizaciju brojača stavljena je u EventBeginPlay događaj u C++ klasi SideScrollerCharahctera, koja se poziva na početku svake razine. Funkcija koja je odabrana u deklaraciji Timera ponavlja se svake sekunde. U funkciji je brojač koji se prilikom svakog ponavljanja smanjuje za jedan. Timer se briše nakon što brojač dođe do nule. Vrijednosti tog brojača pristupa se u korisničkom sučelju i ispisuje ga se. Ostale varijable koje su prikazane na korisničkom sučelju tijekom igre pune se sa običnom Set funkcijom. Varijable ove klase za novac i život spremaju se u instanci na kraju igre, a na početku se pune sa podacima iz instance igre.

```
void AFaksCPlusPlusCharacter::SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent)
{


    InputComponent->BindAction("Pause", IE_Pressed, this, &AFaksCPlusPlusCharacter::OpenMenu);

    InputComponent->BindAction("Sprint", IE_Pressed, this, &AFaksCPlusPlusCharacter::SpeedUp);
    InputComponent->BindAction("Sprint", IE_Released, this, &AFaksCPlusPlusCharacter::SpeedDown);
    // set up gameplay key bindings
    PlayerInputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
    PlayerInputComponent->BindAction("Jump", IE_Released, this, &ACharacter::StopJumping);
    PlayerInputComponent->BindAxis("MoveRight", this, &AFaksCPlusPlusCharacter::MoveRight);

    PlayerInputComponent->BindTouch(IE_Pressed, this, &AFaksCPlusPlusCharacter::TouchStarted);
    PlayerInputComponent->BindTouch(IE_Released, this, &AFaksCPlusPlusCharacter::TouchStopped);
}
```

Slika 42. Prikaz događaji koji se okidaju u slučaju pritiska na određene tipke u C++ kodu

Osim postavljanja varijabli i funkcija, u ovoj klasi se također postavljaju postavke za tipke i događaje. Prije postavljanja tipki u C++ klasi, dodaje se ime događaja i koja tipka poziva koji događaj. To se radi u postavkama projekta. Nakon toga, tipke se inicijaliziraju u C++ klasi i dodaje se odgovarajuća funkcionalnost događaju. Tipke imaju dva događaja, kada je je tipka u stisnutom i kada je puštenom stanju. U nekim slučajevima potreban je samo jedan od ta dva događaja. Ova klasa je napravljena na baznoj klasi PlayerCharacter. Zbog toga klasa već ima neke zadane postavke tipke kao npr. Jump i MoveRight. Osim ovih, dodana su još tri događaja. Pause događaj se poziva pritiskom na tipku ESC ili M, čime se okida funkcija OpenMenu koja otvara Widget izbornika pauze. Ostala dva događaja su povezani sa Sprint funkcijom. Prvi kada je tipka pritisnuta i druga kada je puštena. Svaki događaj poziva svoju funkciju koja ubrzava ili vraća na početno stanje brzinu kretanja igrača.

6.4.1.2 Korisničko sučelje

Razine i glavni izbornik su na isti način raspoređen kao i u Blueprint implementaciji ove igre. Igra ima jednu razinu koji služi kao glavni izbornik i četiri razine u kojima se odvijaju akcije i mehanike igre. Druga razina se ne može započeti dok prva nije dovršena, što također vrijedi za ostale razine. Vrijeme koje je bilo potrebno za prijelaz razinu se sprema i prikazuje kao najbolji rezultat *(eng. high score*) na kraju svake razine. Widgeti su postavljeni u 8 BlueprintWidget klasa i svi Widgeti su djeca C++ klase MyWidgetBase. Blueprit Widget se koristi za vizualni uređivač elemenata, kao primjerice tekst blokovi i slično, dok se C++ koristi za izradu i inicijalizaciju funkcija i varijabli.

6.4.1.3 MyUserWidgetBase Klasa

MyUserWidgetBase C++ je klasa koja je bazirana na UUserWidget klasi, koja je dio Unreal baznih C++ klasa. Klasu MyWidgetBase nasljeđuju sve klase za prikaz korisničkog sučelja. Ona se može podijeliti u funkcije koje koriste sva djeca klase i na klase koje su napravljene posebno za jedno dijete klase.

Funkcija koju svi nasljednici navedene klase koriste je PostText. To je funkcija tipa void koja ima BlueprintCallable specifikator i kao input uzima objekt klase tipa Widget. Zadatak ove funkcije je postavljanje unesene klase za korisničko sučelje na ekran igrača i prilagođavanje nekih od postavki za rad u izborniku, a ne u igri. Ekvivalent ove C++ funkcije u Blueprintu zauzima četiri čvora, dok se ova funkcija može pozvati u jednom čvoru.

```
void UMyUserWidgetBase::PostText(UUserWidget* A) {

    A->AddToViewport();

    pc = UGameplayStatics::GetPlayerController(GetWorld(), 0);

    pc->SetInputMode(FInputModeUIOnly());
    pc->bShowMouseCursor = true;
    pc->bEnableClickEvents = true;
    pc->bEnableMouseOverEvents = true;
}
```

Slika 43. Primjer funkcija za dodavanje korisničkog sučelja

ClickOptionRes i ClickOptionState su dvije funkcije koja su dio MyUserWidget-a, ali koriste se samo u prozoru opcija koji ima nekoliko zadataka. Jedan od tih zadataka je postavljanje i promjena rezolucije ekrana. Jedna funkcija vraća rezoluciju ekrana, a druga vraća stanje ekrana, ovisno o tome je li igra preko cijelog ekrana (*eng. fullscreen*) ili ne (*eng. windowed*). Funkcija koja dodaje rezoluciju na ekran prima podatak tipa FString sa

kombinacijom rezolucije i stanja ekrana. Ove dvije funkcije se pozivaju prilikom promjene rezolucije. Pritiskom miša na novu rezoluciju poziva se ClickOptionRes funkcija koja prima unesenu rezoluciju, traži za zadnjim postavljenim stanjem i spaja rezoluciju i stanje u jedan String, koji onda unosi u funkciju za promjenu ekrana. ClickOptionState radi suprotno tome: prima informacije o sadašnjem stanju, uzima rezoluciju i spaja ih.

Sljedeća funkcija je funkcija Clicked. Ona je tipa void i prima broj razina kao input. Koristi se u Widgetu koji se pojavi nakon što se prijeđe razina i u glavnom izborniku pritiskom na gumb za otvaranje razine. Funkcija prima podatak tipa cijeli broj koji je unesen kako bi se u glavnom izborniku, pri izboru razine otvorila točna razina. Jer igra ima samo 4 levela, unesen je ručno za svaki gumb. U drugom slučaju, u Widgetima nakon pobjede, broj razina se dobije u Actor klasi koji se nalazi na kraju razine i koji u sebi ima spremljen broj razine. Iz te klase poziva se ova funkcija i onda se unosi broj razine. Nakon unosa broja u funkciju, funkcija uzima taj broj i otvara level pod imenom „level“ plus uneseni broj. Dodatno, kao što je prije u funkciji PostText način unosa bio promijenjen i prilagođen za rad u glavnom meniju, sada se vraća opet na način unosa prilagođen za igru.

Funkcija koja provjerava je li prijeđena prijašnja razina i je li moguće pritisnuti gumb je zadnja veća funkcija u MyUserWidget-u. Funkcija je tipa Bool i vraća istinu u slučaju da je prijašnja razina prijeđena. Ona prima broj razina i poziva instancu igre u kojoj su spremljeni podaci o tome je li razina prijeđen ili nije. Unutar instance se uzima uneseni broj i traži se element sa tim indexom minus jedan u proslijeđenom nizu podataka tipa Bool. Ako je element sa tim indexom isitnit, prijašnja razina je prijeđena i vraća istinu.

6.4.2 C++ Mehanike

C++ implementacija ima klase mehanika sa istim ulogama kao i klase u Blueprint implementaciji. Unreal 4 razvojno okruženje ima nekoliko baznih klasa koje spadaju u kategoriju klasa za razvoj mehanika računalnih igara. Najčešće korištena klasa u ovoj implementaciji računalne igre je Actor klasa. Ona je jedna od baznih klasa koja služi za izradu jednostavnih mehanika u igri, kao na primjer mehanike za paljenje svjetla ili otvaranje vrata. Kao i prijašnje klase, C++ klasa zauzima ulogu roditelja, a Blueprint klasa ulogu djeteta. Actor klasa sastavljena je od komponenata koje se u C++ razvoju inicijaliziraju u C++ klasi, a manipuliraju, ako je to potrebno, u Blueprint uređivaču komponenata.

6.4.2.1 LiftBase Klasa

LiftBase je jedna od klasa ove igre koja je bazirana na Actor klasi. LiftBase klasa sastavljena je od nekoliko jednostavnih funkcija: funkcije koja se poziva u slučaju kontakta dizala s igračem i funkcija za kalkulaciju nove pozicije dizala. C++ klase imaju dvije datoteke, header datoteka i implementacijska datoteka (u daljnjem tekstu cpp). Header datoteka sadrži inicijalizaciju

funkcija i get i set funkcije. Na slici 44 prikazana je inicijalizacija klasa komponenata. Komponente se također inicijaliziraju u header datoteci, a one koje su inicijalizirane su SceeneComponent, StaticMeshComponent i UBoxComponent. SceneComponent služi kao centar scene i svaki Actor ima Scene komponentu, MeshComponent je 3D model dizala, odnosno 3D model na koji lik igrača može stati, i posljednje, UBoxComponent je komponenta koja ima ugrađen događaj u slučaju kontakta sa likom igrača. Taj događaj služi kao okidač za pokretanje dizala. Iznad svake klase definiran je UProperty sa specifikatorima koji mogu biti VisibleAnywhere i BlueprintReadOnly. Ovo je potrebno kako bi se komponenta mogla koristiti u Blueprint klasi, te kako bi se zadani model promijenio sa 3D modelom lifta i kako bi meta specifikator AllowPrivateAccess mogao pristupiti ovoj komponenti van klase i u drugoj datoteci, iako je komponenta po zadanim postavkama privatna.

```
// inicijalizacija klasa okomponenata
UPROPERTY(VisibleAnywhere,BlueprintReadOnly, Category="LiftBase", Meta = (AllowPrivateAccess = "true"))
class USceneComponent* RootSceneComponent = nullptr;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "LiftBase", Meta = (AllowPrivateAccess = "true"))
class UStaticMeshComponent* LiftMesh = nullptr;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "LiftBase", Meta = (AllowPrivateAccess = "true"))
class UBoxComponent* TriggerBox = nullptr;
```

Slika 44. Prikaz inicijalizacije komponenti klase u C++ kodu

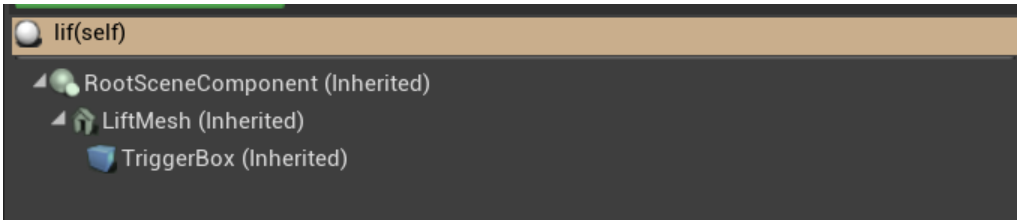
Nakon inicijalizacije komponenti u header datoteci, komponente se koriste u cpp datoteci. U cpp datoteci dodaju se odnosi i određuje hijerarhija komponenata. SceneComponent postavlja se kao korijen modela klase. Osim toga, za CoxComponent se dodaje dinamička funkcija koja se poziva u slučaju kolizija sa drugim modelom. Kod prikazan na slici 45 u zadnjoj liniji dodaje se okidač u slučaju kolizije.

```
//Cpp file postavljanje komponenata koji su prije inicjalizirani
RootSceneComponent = CreateDefaultSubobject<USceneComponent>(TEXT("RootSceneComponent"));
SetRootComponent(RootSceneComponent);

LiftMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("LiftMesh"));
LiftMesh->SetupAttachment(RootSceneComponent);

TriggerBox = CreateDefaultSubobject<UBoxComponent>(TEXT("TriggerBox"));
TriggerBox->SetupAttachment(LiftMesh);
TriggerBox->OnComponentBeginOverlap.AddDynamic(this, &ALiftBase::OnLiftTriggerd);
```

Slika 45. Prikaz dodavanja komponenata u LiftBase klasu

Nakon inicijalizacije komponenti klasa i postavljanja C++ klase kao roditelja Blueprint klase, komponente se pojavljuju kao komponente Actora. Nakon toga, kao što je prije

postavljeno na Cpp datoteci u LiftBase klasi, StaticMesh komponenta je roditelj Box komponente i dijete Scene komponente. Definiranje hijerarhije je prikazana na slici 46.



Slika 46. Prikaz hijerarhije komponenata

LiftBase klasa, osim komponenti, ima i nekoliko funkcija i varijabli. Varijable StartingHeight, EndingHeight i MoveHeight su varijable tipa Float koje se koriste za kalkulaciju i smjer u kojem se lift pomiče. StartingHeight je varijabla koja se postavlja na početku igre, na poziciji gdje je postavljena instanca klase. Budući da se radi o varijabli tipa Float, a ne tipa vector, ne može spremati poziciju na sve tri dimenzije (X, Y i Z), već samo numeričku vrijednost pozicije na osi Z. MoveHeight je varijabla koja se upisuje za svaku instancu posebno. U Uproperty je dodan EditAnywhere specifikator. On omogućuje uređivanje na instanci objekta. EndingHeight se kalkulira zbrajajući vrijednost StartingHeight i MoveHeight, te se kalkulacijom dobije nova Z os pozicija. Ostale osi ostaju iste jer se MoveHeight varijabla pomiče samo na Z osi.

LiftBase klasa također sadrži funkciju UpdateLift. Funkcija služi kao događaj i svaki put kada se okida kontakt TriggerBox događaj, okida se i ovaj događaj. Okidanje događaja u slučaju kolizije i događaja za pomicanje platforme prikazani su na slici 47. Nakon što se taj događaj pozove u Blueprintu pokreće se SetActorLocation čvor koji pomiče platformu od točke StartingHeight do EndingHeight. Nakon što je akcija završena, ponovno se pozove ista funkcija, ali suprotne lokacije.

```
//funkcija koja se poziva kada igrač stane na lift
void ALiftBase::OnLiftTriggerd(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, U
    AFaksCPlusPlusCharacter* sideScrollerChar = Cast<AFaksCPlusPlusCharacter>(OtherActor);
    if (sideScrollerChar != nullptr) {
        //prije stvoreni događaj se poziva ovdje
        UpdateLift();
    }
}
```

Slika 47. Prikaz okidač događaja UpdateLift u LiftBase funkciji

6.4.2.2 TurretBase Klasa

TurretBase se kao i LiftBase klasa bazira na Actor klasi, međutim, za razliku od nje, u igri služi kao protivnik igraču. Funkcija joj je da cilja i puca na igrača. Sastoji se od 5 komponenti, od kojih su tri StaticMesh komponente, jedan boxComponent komponenta i jedna arrow komponenta. Inicijalizacija komponenata je esencijalno ista kao i kod LiftBase klase, ali za razliku od nje ima tri StaticMesh komponente i jednu Arrow komponentu. Arrow komponenta je unutar igre nevidljiva igraču i za razliku od StaticMesh komponente ima ugrađenu funkciju koja kao output ima rotaciju Arrow komponente (strijele).

```
//postavljanje varijabla za ciljanje
UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FVector MyCharacter;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FVector TurretLocation;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FRotator MyRotator;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FVector ITD;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FRotator RotatorPlatform;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FRotator RotatorCannon;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FTransform ArrowSocketTransform;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FRotator ResetRotatorPlatform;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
FRotator ResetRotatorCannon;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
float Distance;

UPROPERTY(BlueprintReadOnly, category = "TurretBase")
float MaxDistance = 900;
```

Slika 48. Prikaz specifikatora uz inicijalizaciju varijabli u C++ kodu

Kao i kod LiftBase klase, C++ klasa služi za inicijalizaciju komponenti, varijabli i kalkulaciju istih. Blueprint dio klase provodi rotaciju sa varijablama i rotatorom koja je naslijeđena iz C++ klase. TurretBase ima tri StaticMesh komponente: komponenta baze MeshTurretBase koja služi kao baza, te su na njoj pozicionirane komponenta platforme MeshTurretPlatform koja se rotira oko Z osi i komponenta topa MeshTurretCannon koja se rotira oko Y osi. Komponente platforme i topa se istovremeno rotiraju kako bi animacija rotacije izgledala neprimjetno i kako se ne bi preklapali modeli TurretBase klase.

TurretBase klasa sadrži tri glavna događaja. U cpp datoteci ove klase nalaze se funkcije koje se pozivaju kada je igrač u dometu i kada nije. U slučaju kada je igrač u dometu u Blueprint

datoteci okidaju se događaji AimAtTarget i FireTurret, te se objekt TurretBase klase rotira prema igraču i puca. Osim toga, u slučaju da je igrač u dometu, također se kalkulira smjer u kojem se nalazi igrač i u Blueprint klasu se šalje varijabla s podacima o rotacija potrebnoj za ciljanje na igrača.

```
AimAtTarget();


MeshTurreCannon->GetComponentLocation().operator-(MyCharacter);
ITD = GetActorTransform().InverseTransformVectorNoScale(MeshTurreCannon->GetComponentLocation().operator-(MyCharacter));
MyRotator = FRotationMatrix::MakeFromX(ITD).Rotator();
RotatorPlatform = MeshTurretPlatform->GetRelativeRotation();
RotatorCannon = MeshTurreCannon->GetRelativeRotation();


FireTurret();

ArrowSocketTransform = ArrowTurret->GetSocketTransform("None", RTS_World);
```

Slika 49. Prikaz kalkulacije rotacije prema igraču

U slučaju da igrač nije u dometu okida se događaj ResetRange koji vraća model na početno stanje. Ukoliko je prije toga bio pozvan AimAtTarget događaj, StaticMesh komponente se vraćaju u početno stanje modela. Također se kalkulira varijabla rotacije koja je potrebna za povratak iz trenutnog stanja u početno.

```
ResetRange();

RotatorPlatform = MeshTurretPlatform->GetRelativeRotation();
RotatorCannon = MeshTurreCannon->GetRelativeRotation();
FRotator ResetRotatorZero (0, 0, 0);
ResetRotatorPlatform = FMath::RInterpTo(RotatorPlatform, ResetRotatorZero, GetWorld()->GetDeltaSeconds(), 50);

ResetRotatorCannon = FMath::RInterpTo(RotatorCannon, ResetRotatorZero, GetWorld()->GetDeltaSeconds(), 50);
```

Slika 50. Primjer vraćanja rotacije na početno stanje

Za kalkulaciju rotacije koristi se nekoliko integriranih funkcija rotacije i lokacije iz polazne Actor klase. Rotacije se spremaju u FRotator varijable posebno za platformu i top. Funkcije koje su korištene za kalkulaciju rotacije su:

1. GetComponentLocation i GetActorLocation, koje se koriste za pronalaženje trenutnih lokacija komponenata ove klase i poziciju igrača koja je spremljena u MyCharacter varijabli.
2. GetActorTransform, koja također služi za pronalazak lokacije klase, ali osim toga i rotacije i veličine.
3. Operator funkcija, koja oduzima dvije varijable tipa FVector i vraća podatak o njihovoj udaljenosti.

4. InverseTransformVectorNoScale, koja primjenjuje inverznu transformaciju bez da povećava ili smanjuje unešeni podatak koji je tipa Fvector.
5. MakeFromX funkcija, koja je dio FRotationMatrix klase. U slučaju ovog projekta koristi se zajedno sa Rotator funkcijom kako bi se kalkulirala rotacija na X osi bez specificiranja vrijednosti Y i Z osi.
6. GetRelativeRotation funkcija, koja vraća rotaciju komponente u odnosu na svog roditelja.
7. FInterpTo funkcija, koja vraća podatke tipa FRotator i pokušava napraviti rotaciju među 2 FRotation varijable na način da animacija izgleda fluidno i prirodno.

6.4.2.3 ProjectileBase Klasa

ProjectileBase je C++ klasa tipa Actor koja na početku igre nema ni jednu svoju instancu unutar igre. Kada je igrač u dometu objekta TurretBase klase, poziva se i okida Fire događaj, te se poziva SpawnActor funkcija koja u svijetu igre stvara objekt ProjecBase klase i ispušta ga prema trenutačnoj poziciji igrača. Nakon kontakta sa igračem, ProjectileBase poziva funkciju HealthDrop iz klase igrača, te mu umanjuje Health varijablu (zdravlje).

ProjectileBase klasa ima 5 komponenata. Prva je StaticMesh komponenta, koja služi za vizualni prikaz igraču kako i gdje se kreće objekt. Zatim, BoxComponent koja se okida u kontaktu sa igračem kako bi prepoznala ako je igrač pogođen i da li mu je potrebno oduzeti život. ProjectileMovment komponenta koja je UE4 komponenta za Actor klasu i namijenjena je isključivo za izradu projektila kao i u ovom slučaju. Ova komponenta uzima u obzir fizičke zakonitosti gravitacije, brzine, otpora i slično, i primjenjuje ih na objekt klase u svijetu igre, te određuje putanju kretanja objekta. Zadnja komponenta je RotatingMovement koja dodaje rotaciju projektilu dok se nalazi u zraku.

6.4.2.4 ObstacleBase Klasa

ObstacleBase je C++ klasa tipa Actor i roditelj SpikeObstical Blueprint klase. SpikeObstical se ponaša slično kao i LiftBase klasa. Cilj ove klase je kretati se između dvije točke na Y i na Z osi. U slučaju kontakta sa igračem oduzima mu se dio života.

```
FRotator NewRotation = FRotator(PitchValue, YawValue, RollValue);

FQuat QuatRotation = FQuat(NewRotation);

AddActorLocalRotation(QuatRotation, false, 0, ETeleportType::None);
```

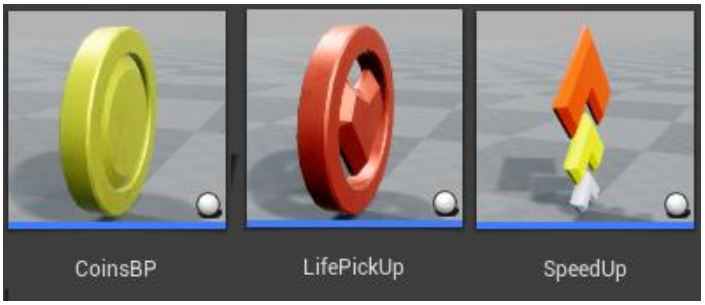
Slika 51. Prikaz funkcije za lokalnu rotaciju

Klasa ObstacleBase kroz instancu objekta prima varijable koje imaju spremljene podatke o završnoj lokaciji. Varijabla MovePoint kao i kod LiftBase klase prima Float vrijednost i ima EditAnywhere specifikator, te se on, osim kod ove varijable, koristi i kod varijabli PitchValue, YawValue, RollValue, koje služe za dodavanje rotacije ObstacleBase objektu. Nakon inicijalizacije i unošenja varijabli kroz instancu, dodaje se rotacija koristeći AddActorLocalRotation funkciju koja prima podatak tipa FQuat. FQuat je tip varijable koja može predstaviti rotaciju oko određene osi u 3D prostoru.

Kao i kod LiftBase klase, ObstacleBase klasa ima BoxComponent koja u slučaju kolizija okida funkciju iz C++ klase lika igrača. Također, zajedničko im je kretanje između dvije točke koje se odvija jednako kao i kod prije razrađene LiftBase klase.

6.4.2.5 CoinActor Klasa

CoinActor klasa je u početku razvoja bila korištena samo za razvoj logike u podlozi sakupljanja i spremanja novca u igri. Osim toga, CoinActor se koristi i kao bazna klasa za druge predmete koji se skupljaju (*eng. pick up item*). Osim novca, u klase koje nasljeđuju CoinActor klasu spadaju i objekti za povišenje PlayerFuel varijable i za podizanje života igrača. Klase koje nasljeđuju CoinActor klasu prikazani su na slici 52. Klasa ima lokalnu rotaciju oko Z osi.



Slika 52. Prikaz djece CoinActor klasa

CoinActor klasa, iako ima drugačiju ulogu, ima nekoliko istih funkcija i komponenata kao i ProjectileBase klasa. Komponenta USceneComponent koristi se kao korijen objekta, komponenta BoxComponent koristi se za prepoznavanje kolizije, dok se komponenta StaticMesh koristi za vizualni prikaz objekta. Funkcije za prepoznavanje kolizije sa igračem i uništavanje komponente nakon kolizije također su jednake kao i kod klase projektila.

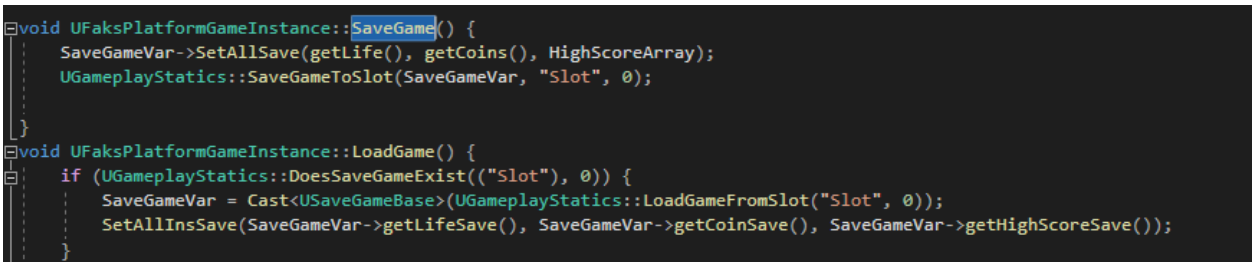
```
void ACoinsActor::OnCoinTriggerd(UPrimitiveComponent* OverlappedComponent, AActor* OtherAc
    AFaksCPlusPlusCharacter* sideScrollerChar = Cast<AFaksCPlusPlusCharacter>(OtherActor);
    if (sideScrollerChar != nullptr) {
        //add 1 to side scroler and destoy actor
        if(MeshType == 1){
            sideScrollerChar->Setcoinsa();
        }
        else if(MeshType == 2){
            sideScrollerChar->addLife();
        }
        else if(MeshType == 3) {
            sideScrollerChar->addFuel();
        }
        K2_DestroyActor();

    }
}
```

Slika 53. Prikaz okidanja funkcija iz SideScrollerCharater klase

U svrhu dodavanja više uloga i zadataka ovoj klasi, dodana je i varijabla MeshType. To je varijabla tipa Int koja za svako dijete klase CoinActor ima drugu vrijednost. U slučaju kolizije, ovisno o tome koja je vrijednost postavljena kao MeshType, poziva se druga funkcija iz klase SideScrollerCharacter ili klase lika igrača. CoinBP je dijete klase CoinActor i u slučaju kolizije poziva funkciju za dodavanje jednog novčića u varijablu novac koja se nalazi u klasi igrača. LifePickUp je također jedna od klasa koja je dijete klase CoinActor i u slučaju kolizije dodaje jedan život igraču. Zadnja je SpeedUp klasa koja u slučaju kolizije vraća dio PlayerFuel varijable ako je vrijednost varijable manje od maksimalne.

6.4.2.6 SaveGameBase Klasa

SaveGameBase je klasa u kojoj se spremaju podaci, koji se naknadno spremaju u bazu podataka računalne igre. U Unreal razvojnom okruženju spremanje podataka ne odvija se direktno u SaveGameBase klasi, već se podaci spremaju u klasi SaveGameBase ali funkcije za dugoročno spremanje se nalaze u klasi instance igre.



```
void UFaksPlatformGameInstance::SaveGame() {
    SaveGameVar->SetAllSave(getLife(), getCoins(), HighScoreArray);
    UGameplayStatics::SaveGameToSlot(SaveGameVar, "Slot", 0);

}
void UFaksPlatformGameInstance::LoadGame() {
    if (UGameplayStatics::DoesSaveGameExist(("Slot"), 0)) {
        SaveGameVar = Cast<USaveGameBase>(UGameplayStatics::LoadGameFromSlot("Slot", 0));
        SetAllInsSave(SaveGameVar->getLifeSave(), SaveGameVar->getCoinSave(), SaveGameVar->getHighScoreSave());
    }
```

Slika 54. Prikaz funkcije za spremanje i učitavanje podataka

Prilikom spremanje trenutnog stanja igre nakon izrade klase potrebno je spremiti ažurirano stanje varijabli u jedan od objekata klase. U slučaju ove igre sprema se broj života, novac,

prijeđene razine i vrijeme u kojem je igrač prešao pojedine razinu. Za igre otvorenoga svijeta mogle bi se spremiti još i varijable poput FVectora sa lokacijom igrača u trenutku spremanja, varijable sa imenom i količinom alta, oružja, hrane i slično, te dijelove mape svijeta koje je već otkrio i istražio. Nakon što su podaci spremljeni u objekt klase za spremanje podataka, pohranjuju se u bazu podataka pomoću funkcije SaveGameToSlot. U ovu funkciju mora se unijeti objekt klase sa varijablama, utor (*eng. save slot*) i Indeks. Punjenje klase i pozivanje funkcije radi se u SaveGame funkciji koja se kasnije poziva pritiskom na gumb u izborniku opcija i izborniku pauze. Izbornik sa opcijama pod kategorijom koja spada u spremanje podataka, osim spremanja, ima i gumb za očistit Slot i izbrisati spremljene podatke, te vratiti spremljene varijable na početno stanje.

Funkcija LoadGame je jedna od funkcija u klasi instance igre koja na pritisak gumba učita spremljene podatke i vraća igru i varijable u stanje u kojem su bili u trenutku posljednjeg spremanja. Prije učitavanja, funkcija DoesSavedGameExist provjerava ima li spremljenih podataka, te vraća varijablu tipa Bool sa podatkom o postojanju ili izostanku istih. Nakon učitavanja iz baze podataka, nove varijable se pohranjuju u varijable instance igre.

6.5 Opis modela i ostalih vizualnih komponenata konačne računalne igre

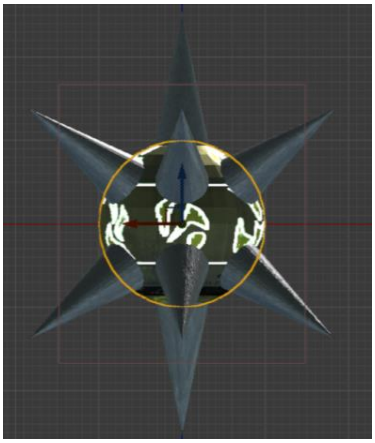
U ovom ćemo se poglavlju osvrnuti na opis vizualnog izgleda razina i komponenata s kojima igrač može uči u interakciju tijekom igranja igre. Za razliku od prijašnjih opisa komponenata u ovom će djelu biti opisane komponente iz perspektive igrača nakon završetka izrade konačne igre.

Primarna je uloga izrade igre bila usporedba dva programska jezika, zbog toga vizualni izgled nije bio u fokusu izrade iste. Tijekom razvoja fokus je bio na razvoju, dokumentaciji i opisu programskog dijela igre. Bez obzira na to, kao i veći broj drugih igara, i ova računalna igra ima vizualne komponente kroz koje je moguća interakcija sa igračem.

Već je prethodno spomenuto da je naša računalna igra bočnog prikaza, tj. igrač iz bočne perspektive vidi svog lika u igri. Perspektiva igre od strane igrača prikazana je na slikama broj 61 i 62.

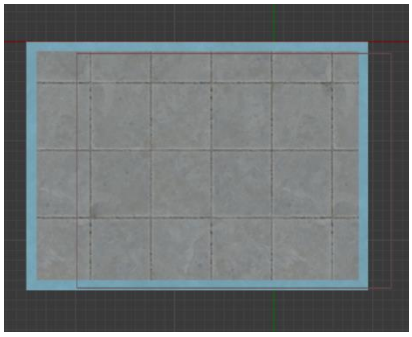
6.5.1 Izgled i uloge prepreka

Igra ima nekoliko prepreka ili komponenata koji se mogu podijeliti na dvije glavne kategorije: prijateljske i neprijateljske. Prijateljske komponente ne napadaju igrača i imaju različitu svrhu ovisno o vrsti, dok su neprijateljske prepreke objekti koji mogu napadati i smanjiti zdravlje igrača.



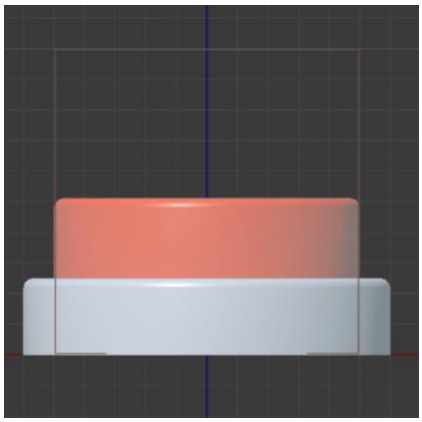
Slika 55. Objekt SpikeObstacle klase

U prijateljske komponente u ovoj igri spadaju objekti klase LiftBase, WinCondition, Coin, LifeUp i SpeedUp. LiftBase je platforma koja se pomiče ako lik igrača stane na nju i prenosi lika igrača na određenu poziciju. U igri se razlikuju dvije LiftBase klase, prva koja se kreće horizontalno i druga koja se kreće vertikalno. Ova klasa omogućuje igraču prelaženje prepreka poput statičkih objekata koji su postavljeni po cijeloj razini. Objekt klase LiftBase prikazana je na slici 57. WinCondition u igri ima oblik gumba koji se nalazi na kraju razine i pritiskom na gumb igrač pobjeđuje razinu. Ova klasa se poziva u slučaju kontakta lika igrača sa objektom. Coin, LifeUp i Speed Up su klase koje u slučaju kontakta sa igračem daju igraču određene bonuse. Na objekte ovih klasa osvrnuli smo se ranije u tekstu, te su vidljivi na slici 52. Coin u igri izgleda kao novčić, kao što je prikazano na slici 52, i u slučaju kontakta sa igračem



Slika 56.Objekt LiftBase klase

podiže ukupan broj novčića s kojima igrač raspolaže. Kada igrač skupi 100 novčića dobiva jedan dodatan život. Objekt LifeUp klase ima sličan izgled u igri kao i Coin klasa i u slučaju kontakta daje igraču jedan dodatan život. SpeedUp klasa u slučaju kontakta sa igračem povećava mu razinu stamine ili kondicije.



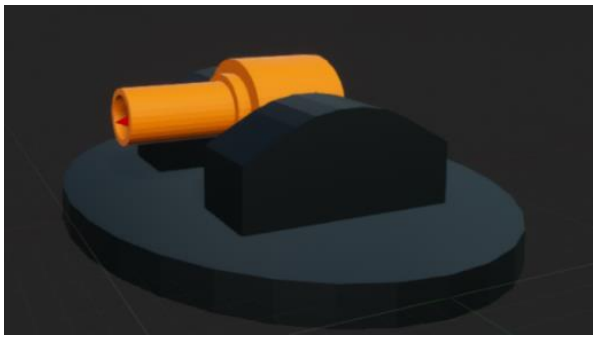
Slika 57. Objekt WinCondition klase

Neprijateljske klase u ovoj igri su klase TurretBase, SpinObstacle, i SpikeObstacle. TurretBase klasa je klasa koja ima oblik topa, što je prikazano na slici 59. Top je neprijateljska klasa koja cilja prema liku igrača kada je u dometu i u kombinaciji sa ProjectileBase klasom puca projektil prema igraču. Ako projektil ima kontakt sa igračem, zdravlje igrača se smanji. Ako zdravlje igrača padne na nulu ili manje, igrač se vraća na početak razine i život mu se smanji za jedan. Zadatak igrača je da izbjegava prepreke projektila. SpinObstacle je neprijateljska klasa čiji je objekt sastavljen od četiri statičke komponente koje se vrte oko jedne točke. Objekt klase je prikazan na slici 60. Ova prepreka prisiljava igrača da se pažljivo kreće kroz razinu i izbjegava ju. U slučaju kontakta igrač gubi zdravlje. Vizualni izgled klase SpikeObstacke je prikazana na slici 57. Slično kao i LiftBase klase, ova klasa se kreće između dvije točke koje mogu biti horizontalno ili vertikalno pozicionirane. Jednako kao i za SpinObstacle, objekt ove klase prisiljava igrača da ju izbjegava, jer u slučaju kontakta, odnosno kolizije, zdravlje igrača pada.

Slika 58. Objekt SpinObstacle klase

6.5.2 Izgled razine

Računalna igra je podijeljena na 4 razine. Svaka razina se razlikuje od prijašnje i ima svoju temu i ulogu u igri. Razine se također razlikuju po stupnju težine (*eng. difficulty level*). Sa svakom razinom raste kompleksnost i broj prepreka u razini.



Slika 59. Objekt TurretBase klase

Prva razina je vizualno jednostavna, te nema veliki broj prepreka i komponenata koje sprječavaju igrača da dođe do pobjede. Sastavljena je od 109 glumaca. Od neprijateljskih komponenata ova razina ima tri objekta klase SpikeObstacle, četiri objekta klase SpinObstacle i jedan objekt klase TurretBase. U njoj se nalazi najmanji broj neprijateljskih komponenata što

ju čini najlakšom razinom igrice i uloga joj je upoznati igrača sa objektima u igri. Iz tog razloga ova klasa sadrži sve neprijateljske i prijateljske objekte, ali u manjem broju.

U razine su obično lik igrača kreće horizontalno, jedina iznimka je druga razina u kojoj se lik igrača kreće vertikalno. Iz tog razloga, kako bi liku bilo izazovnije kretanje, ova razina ima najveći broj platformi i objekata iz LiftBase klase. Osim toga, ova razina ima i jednu neprijateljsku komponentu, tj. objekte klase TurretBase. Druga razina ukupno ima pet objekata klase TurretBase i dvadeset i jedan objekata klase LiftBase. Uloga ove razine je upoznavanje igrača sa dizalima i tornjevima, te je glavni izazov igraču precizno kretanje po platformama i objektima LiftBase klase i istovremeno izbjegavanja projektila koje tornjevi pucaju. Snimka zaslona jednog dijela ove razine prikazana je na slici 61.

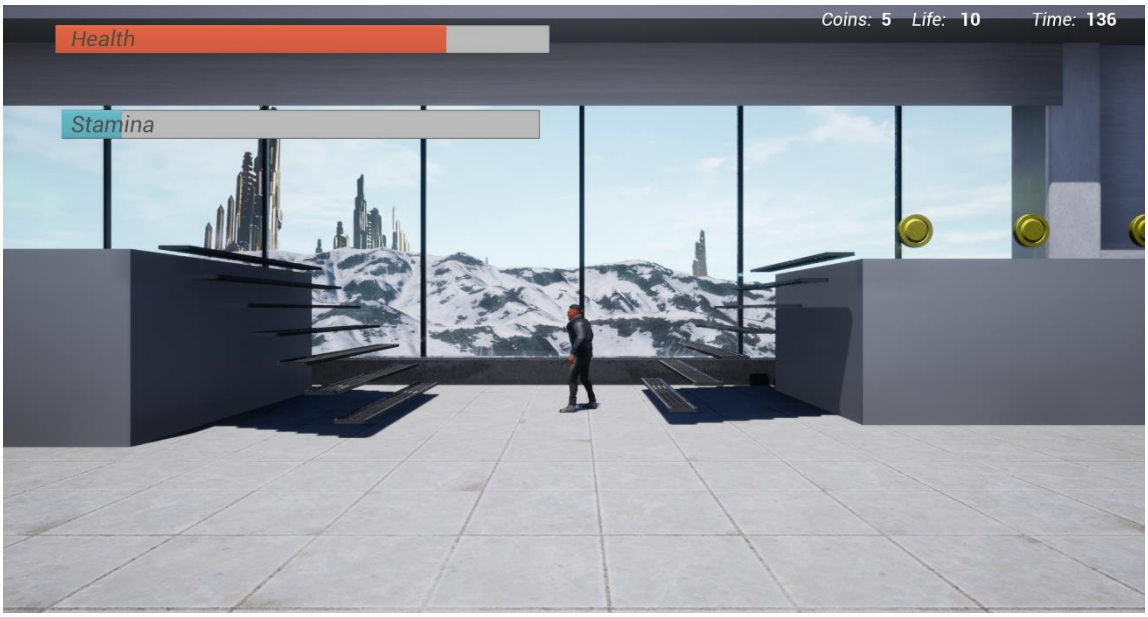
Za razliku od druge razine koja ima veliki broj objekata klase LiftBase i TurretBase, treća razina nema niti jedan objekt LiftBase klase i ima samo četiri objekata TurretBase klase, ali zato ima najveći broj objekata klase SpikeObstacle, ukupno njih 13. Tema i uloga ove razine je upoznavanje igrača sa objektima SpikeObstacle klase.

Četvrta razina kombinira sve prijašnje teme i prepreke. Ova klasa ima najveći broj objekata i najveći broj kombinacija istih. Razina ima osam objekata SpikeObstacle klase, deset objekata SpinObstacle klase, četrnaest objekata TurretBase klase i osam objekata LiftBase klase. Na ovoj razini igrač integrira prethodno stečeno iskustvo. Na slici 60 je prikazan jedan dio četvrte razine.

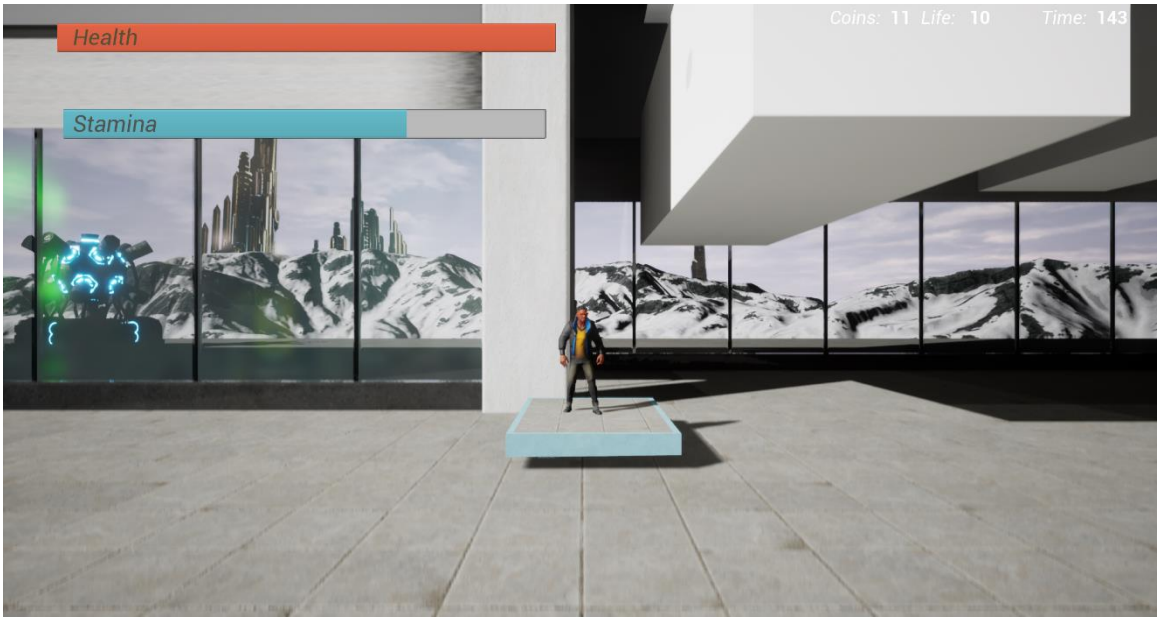
Slika 60. Snimka zaslona prve razine

Slika 61. Snimka zaslona druge razine

Slika 62. Prikaz izgleda druge razine



Slika 63. Snimka zaslona treće razine



Slika 64. Snimka zaslona četvrte razine

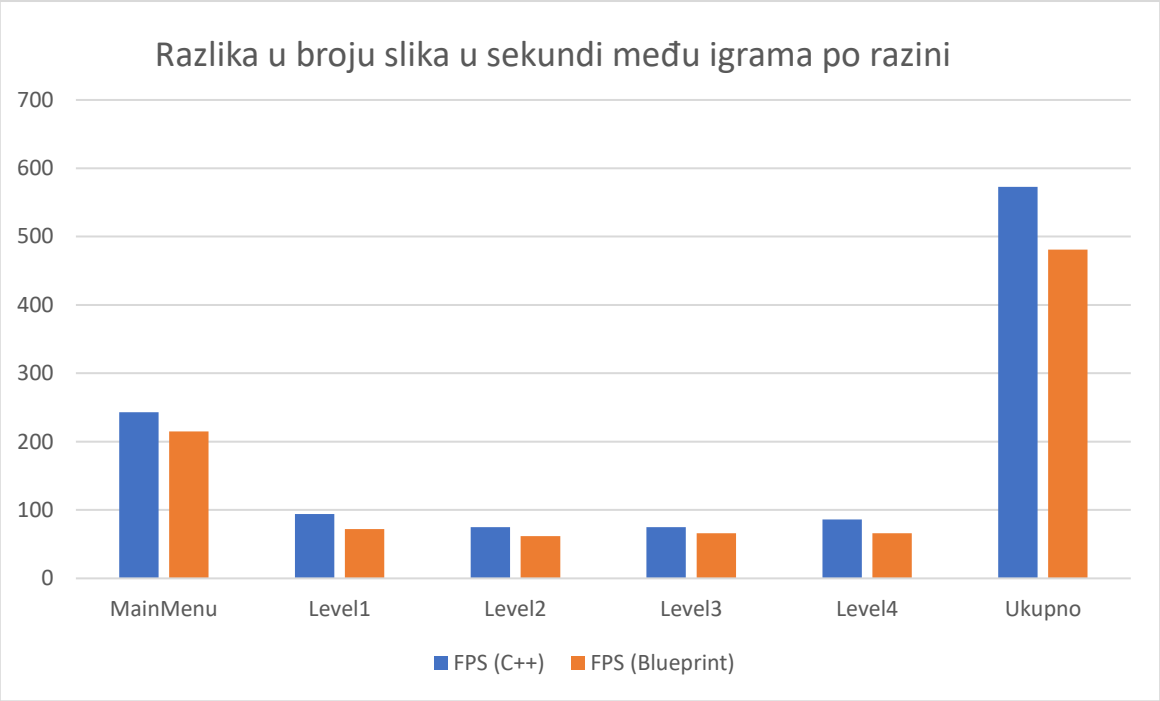
7 RAZLIKE, PREDNOSTI I NEDOSTACI C++ I BLUEPRINT PROGRAMSKIH JEZIKA

Tijekom izrade ovog rada i implementacije igara, vidi se kako oba programska jezika imaju svoje prednosti i uloge. Blueprint visual scripting jezik baziran je na C++ programskom jeziku, te samim time nasljeđuje mnogo njegovih nedostataka, ali i prednosti. Također, Blueprint nasljeđuje neke od prednosti i nedostataka ostalih vizualnih programskih jezika, a oni će detaljnije biti analizirani kroz izradu naše računalne igre u poglavljima 8.1 i 8.2. Uz to dodatno su analizirane i performanse igara, a u nastavku će biti prikazani i rezultati testiranja računalnih igara. Prilikom testiranja mjeren je broj slika u sekundi (ili u daljnjem tekstu FPS - *eng. Frames per second*). Dobiveni podaci biti će prikazani tablično i grafički. Pokazalo se da su rezultati većim dijelom potvrdili očekivane pretpostavke. Slijedi prikaz i analiza prikupljenih podataka.

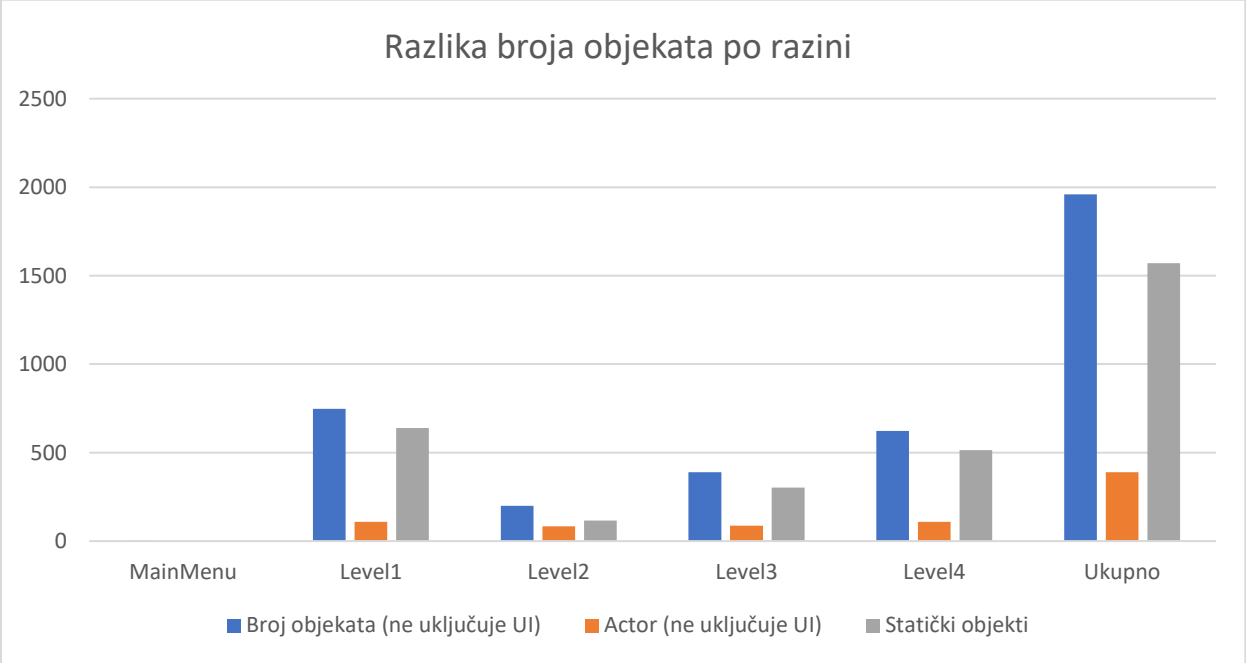
Performanse igara testirane su na računalu sa windows 10 operativnim sustavom, grafičkom karticom Radeon rx 480 i amd ryzen 5 2600 procesorom i 16gb ddr4 RAM-a. Prosječni broj slika u sekundi nakon 10 minuta igranja u Blueprint verziji igre je 85. +, dok je prosjek u C++ verziji igre 152 slike u sekundi. Razlike su također vidljive u svakoj razini zasebno. U glavnom izborniku Blueprint projekt imao je oko 215, a C++ projekt se kretao oko 243 slika u sekundi. U Blueprint verziji igre u prvoj razini FPS se kretao oko 72, dok se u C++ verziji igre na istoj razini kretao oko 94 FPS-a. C++ igra je u ovoj razini u prosjeku imala oko 1.3 puta više slika u sekundi od Bluperint verzije. Broj objekata u prvoj razini je 747, od čega samo njih 109 imaju skriptu sa funkcijama. Ostalih 638 objekata su statički objekti bez skripte. U drugoj razini Blueprint verzija ima prosjek od 62, dok C++ verzija ima 75 slika u sekundi. C++ verzija i u ovom slučaju radi brže od Blueprint verzije i ima oko 1.2 puta veći broj slika u sekundi od druge verzije igre. Broj objekata u ovoj razini je 199, od kojih su 83 objekti sa skriptom i mehanikama. Prosječni FPS treće razine je 75 u C++ verziji i 66 u Blueprint verziji. C++ verzija je 1.13 puta brža. Treća razina ima 390 glumaca ili objekta od kojih je 88 sa skriptom. U četvrtoj i posljednjoj razini C++ verzija ima 86, a Blueprint ima 66 slika u sekundi. Razlika u brzini je 1,3, te zadnja razina ima 623 objekta, od kojih je 109 sa skriptom. Navedeni podaci prikazani su u tablici 2.

Level	FPS (C++)	FPS (Blueprint)	Razlika	Broj objekata (ne uključuje UI)	Actor (ne uključuje UI)	Statički objekti
MainMenu	243	215	1,466667	1	1	0
Level1	94	72	1,305556	747	109	638
Level2	75	62	1,209677	199	83	116
Level3	75	66	1,136364	390	88	302
Level4	86	66	1,30303	623	109	514
Ukupno	550	416		1960	390	1570

Tablica 2. Prikaz prikupljenih podataka testiranjem gotovih igra



Graf 1. Broj slika u sekundi među igrama po razini

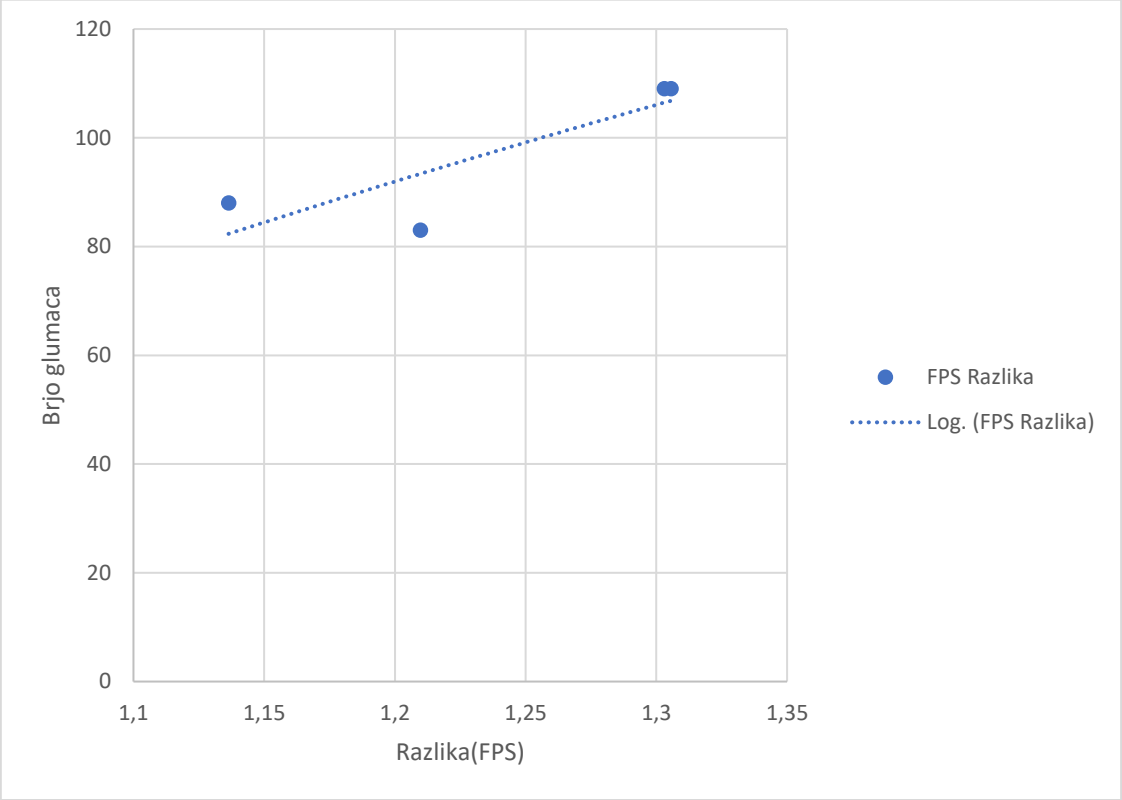


Graf 2. Broj objekata po razini

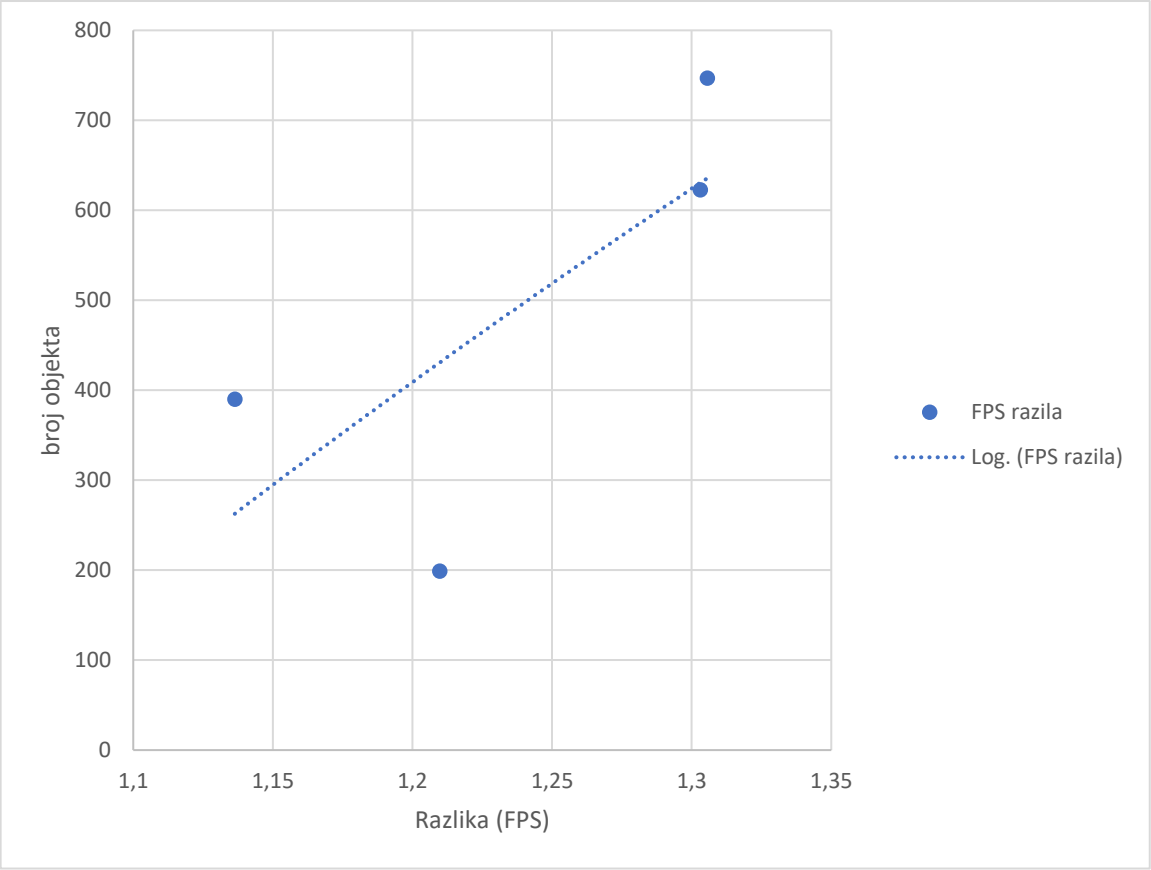
Graf 1 i 2 prikazuju dobivene razlike po razinama. Prvi graf prikazuje razliku među igrama u FPS-u ovisno o programskom jeziku u kojem je igra izrađena, te su razlike prikazane po razinama i kroz cijelu igru. Iz grafa je vidljivo kako se u igri izrađenoj u C++ programskom jeziku prikazuje generalno više slika u sekundi nego u igri izrađenoj u Blueprint programskom

jeziku. Osim toga u zadnjem stupcu prikazan je ukupan broj slika u sekundi kroz cijelu igru, kako bi se vidjela ukupna razlika u brizni rada programskih jezika.

Drugi graf prikazuje razliku u broju objekata za svaku razinu zasebno, te ukupno kroz cijelu igru. Obzirom na to da je broj objekata identičan u obje igre nije potrebna posebna tablica za svaki programski jezik. U grafu je prikazan broj glumaca, statičkih objekata i sveukupnih objekata. Ako usporedimo broj slika u sekundi u razini MainManu sa brojem glumaca kod te iste razine (vidljivo na grafu 1 i 2), vidljivo je da postoji velika razlika u broju slika, iako razina nema niti jednog glumca. Razlika nastaje jer glavni izbornik nema statičke glumce, ali ima veći broj komponenata korisničkog sučelja od ostalih razina koje nisu uključene u ukupan broj objekata / glumaca. Iz tog razloga u grafu 4 i 5, gdje se prikazuje povezanost broja glumaca i objekata sa razlikom u brzini igara, nije uključena razina MainManu. Broj elementa korisničkog sučelja među razinama, ne uključujući glavni izbornik, je isti. Razlika u performansama u glavnom izborniku nastaje iz funkcija i procesa koji su povezani za rad korisničkog sučelja.

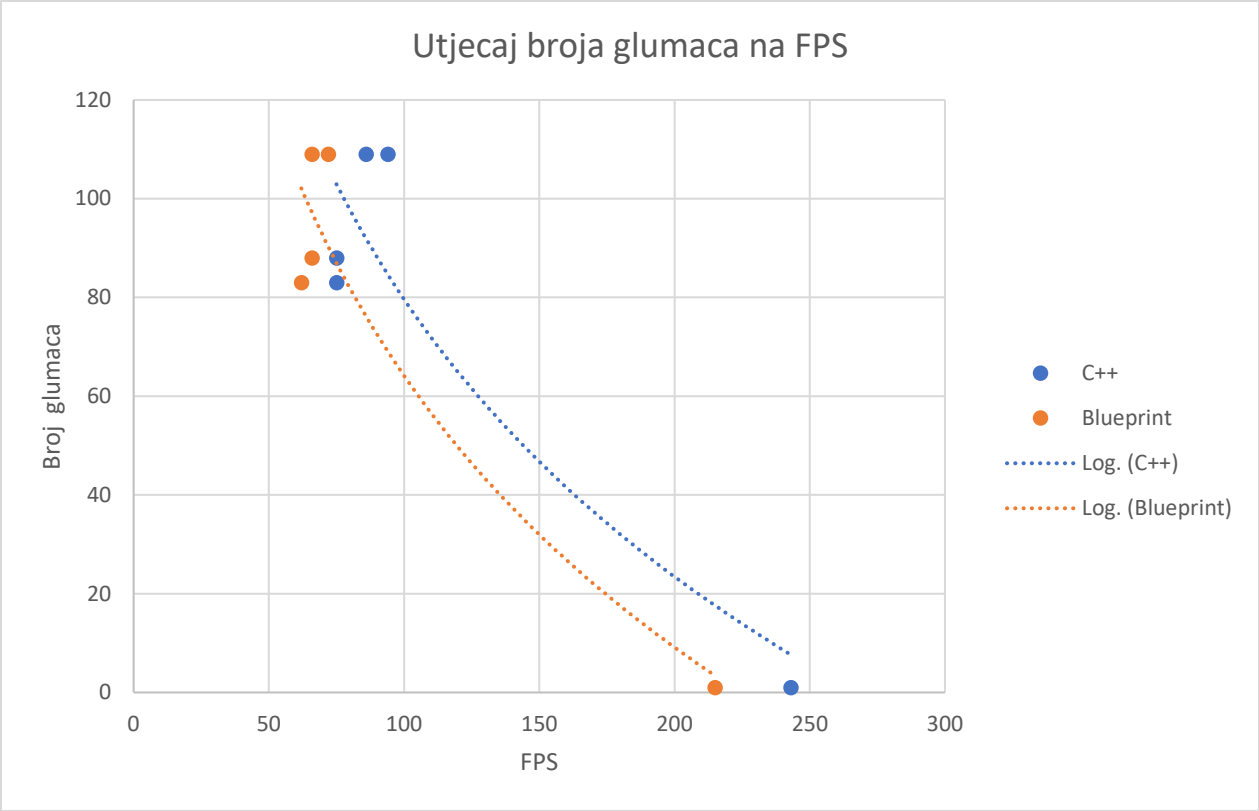


Graf 3. Utjecaj glumaca na razliku FPS-a među programskim jezicima



Graf 4. Utjecaj broja objekata na razliku FPS-a među programskim jezicima

Grafikoni 3 i 4 prikazuju razlike u FPS-u među igrama izrađenima u C++ i Blueprint programskom jeziku u usporedbi sa brojem objekata, odnosno prikazuju ima li broj glumaca/objekata utjecaj na razliku u FPS-u. Ovim grafikonima je također dodana logaritamska crta trenda. Povezanost je vidljiva na način da razlika u FPS-u među igrama raste sa većim brojem glumaca, odnosno razlika je vidljiva više u projektima sa mnogo glumaca. Ako se radi u projektima sa malim brojem glumaca, manje je vidljiva razlika u efikasnosti između C++ i Blueprint programskog jezika. Jednako vrijedi i za objekte. Ova informacija može korisniku Unreal razvojnog okruženja pomoći pri odabiru programskog jezika. Ako je u planu razvoj jednostavne računalne igre sa manjim brojem glumaca, Blueprint programski jezik je prihvatljiva opcija, iako C++ i u tim situacijama ima bolje performanse, no ako se radi o igri sa većim brojem glumaca, razlika među performansama postaje velika i C++ je u svakom slučaju bolja opcija. Ovi rezultati su u skladu s informacijom da su vizualni programski jezici kao Blueprint više orijentirani početnicima, iz razloga što početnici obično rade na manjim igrama, te je onda Blueprint, unatoč svojoj manjoj brzini rada, prihvatljiva opcija.



Graf 5. Prikaz slika u sekundi s obzirom na broj glumaca po programskom jeziku

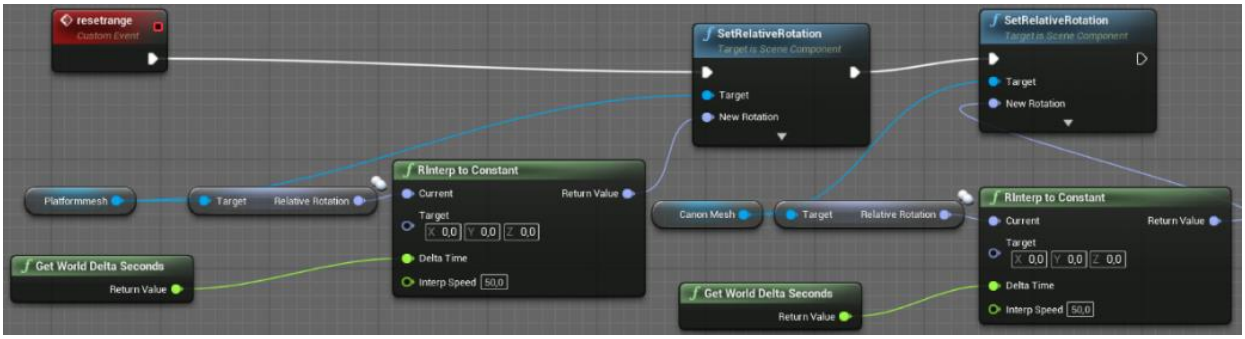
Graf 5 prikazuje utjecaj broja glumaca po razinama s obzirom na FPS. Ovom grafu dodana je logaritamska crta trenda iz koje je vidljivo kako FPS raste kada broj glumaca u razini pada. Jednadžba grafa trenda za C++ programski jezik je y = -81,02ln(x) + 452,74, a za Blueprint programski jezik je y = -79,38ln(x) + 429,7.

7.1 PREDNOSTI BLUEPRINT PROGRAMSKOG JEZIKA

Jedan od razloga zašto vizualni programski jezici postoje je kako bi omogućili dizajnerima i drugom ne programerskom osoblju u razvoju računalnih igara i drugih softvera jednostavan pristup programskom dijelu razvoja. Vizualni jezici mogu, osim kao programska skripta za dodavanje funkcionalnosti klasama, također služiti kao neka vrsta mentalne mape, odnosno njihova jednostavna funkcionalnost čini ih preglednijima. Jednako tako kod dodavanje jednostavnijih funkcija mogu biti jednako brzi kao tekstualni programski jezici. Međutim, vizualnih programski jezici imaju i neke nedostatke. Računalne igre sa kompleksnijim klasama mogu postati vrlo kompliciranе i teže čitljive. Osim toga, obično su više ograničeni od tekstualnih programskih jezika zbog toga što su bazirani i ograničeni određenim brojem čvorova.

Programiranje u Blueprint vizualnom programskom jeziku dijeli neke od prednosti i nedostataka sa ostalim vizualnim programskim jezicima. Prvenstveno, efektivnost, urednost i

čitljivost programskog koda uvelike ovisi o kompleksnosti koda. Jednostavniji programski kod može biti brzo dodan i izmijenjen i može ga se lako razumjeti. Kompleksnije funkcionalnosti ili klase sa više od jedne funkcionalnosti u Blueprint vizualnom skriptiranju, ako su podijeljene u funkcije i događaje, mogu postati jako neorganizirane. U nekim slučajevima jedna linija koda u C++ programskom jeziku može izraditi istu funkcionalnost kao i velika skupina čvorova i veza u Blueprint sustavu. Primjer ovoga u izrađenoj igri vidljiv je kod kalkulacije objekta TourettBase klase na kojoj poziciji se nalazi igrač u odnosu njega. Funkcionalnost je ista, ali programski kod je mnogo uredniji i pregledniji u C++ verziji, kao što je i vidljivo na slici 62 i 64. Detaljniji prikaz istog koda u Blueprint verziji vidljiv je na slici 63.

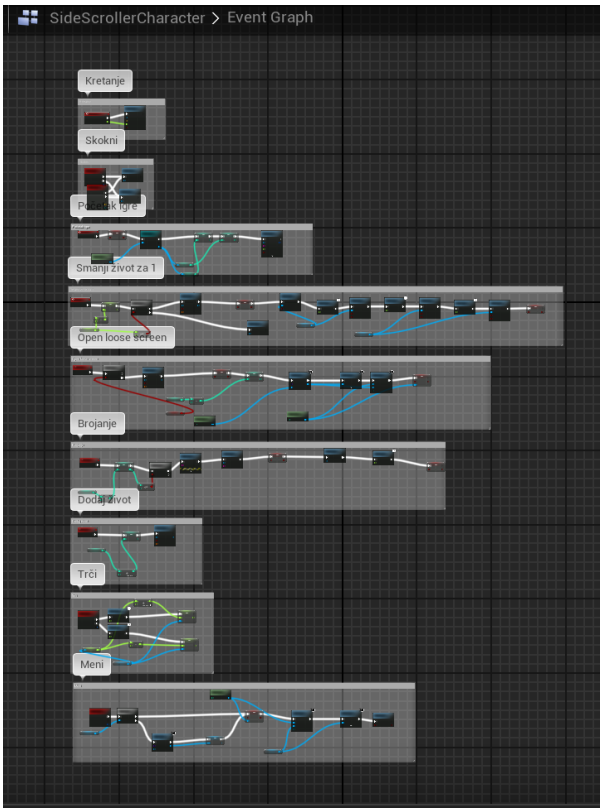


Slika 65. Prikaz kalkulacije rotacije TurretBase klase u Blueprint implementaciji igre

```
ResetRange();

RotatorPlatform = MeshTurretPlatform->GetRelativeRotation();
RotatorCannon = MeshTurreCannon->GetRelativeRotation();
FRotator ResetRotatorZero (0, 0, 0);
ResetRotatorPlatform = FMath::RInterpTo(RotatorPlatform, ResetRotatorZero, GetWorld()->GetDeltaSeconds(), 50);

ResetRotatorCannon = FMath::RInterpTo(RotatorCannon, ResetRotatorZero, GetWorld()->GetDeltaSeconds(), 50);
```

Slika 66. Prikaz kalkulacije rotacije TurretBase klase u C++ implementaciji igre

Obzirom na to da duže funkcije mogu biti vrlo kompleksne u Blueprint programskom jeziku svim korisnicima osim autoru, programski kod može biti velik, nepregledan, teško čitljiv i problematičan u slučajevima kada je potrebno izmijeniti dio skripte. Kompleksnost koda u Blueprint programskom jeziku prikazana je na slici 64.



Slika 67. Primjer kompleksnosti SideScrollerCharacter Blueprint klase

Jedna od većih prednosti Blueprint programskog jezika, posebno za početnike, je da on ima već ugrađene funkcije u čvorovima, te sam softver predlaže programeru čvorove. Ako se iz varijable ili komponente izvlači veza, Unreal razvojno okruženje automatski ponudi igraču moguće čvorove koji uzimaju tu komponentu ili varijablu kao input. Na primjer ako se iz varijable tipa vektor izvlači veza, softver će programeru ponuditi funkcije kao na primjer vector + vector ili break vector. Blueprint uređivač ograničava i sprječava igrača da odabere krivi čvor koji bi mogao rezultirati u pogrešci, tj. error-u. Također, čvorovi ne dopuštaju postavljanje inputa krivog tipa, što dodatno sprječava pogreške u programskom kodu. Na taj se način izgjegava izgradnja programskog koda sa greškama, tj. errorima.

Osim toga referenciranje komponenata i varijabli u Blueprintu moguće je već samim povlačenjem. Inicijalizacija funkcija također je jednostavnija iz perspektive početnika obzirom na to da se varijabla dodaje jednim pritiskom na tipku miša, te ju se povuče u prozor za uređivanje. Jednako tako, komponente se dodaju po principu povlačenja i ispuštanja (drag and drop) i hijerarhija po kojoj su one postavljene automatski se izrađuje. Na primjer, ako se radi sa komponentama u C++-u inicijalizacija započinje u header dokumentu i onda se u cpp datoteci ručno određuje hijerarhija komponenata. Dio ovog koda vidljiv je na slici 65. Ekvivalent u Blueprint sustavu radi se samo kroz povlačenje i ispuštanje komponenata.

```
RootSceneComponentTurret = CreateDefaultSubobject<USceneComponent>(TEXT("RootSceneComponentTurret"));
SetRootComponent(RootSceneComponentTurret);

MeshTurretBase = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshTurretBase"));
MeshTurretBase->SetupAttachment(RootSceneComponentTurret);

MeshTurretPlatform = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshTurretPlatform"));
MeshTurretPlatform->SetupAttachment(MeshTurretBase);

MeshTurreCannon = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshTurreCannon"));
MeshTurreCannon->SetupAttachment(MeshTurretPlatform);

ArrowTurret = CreateDefaultSubobject<UArrowComponent>(TEXT("ArrowTurret"));
ArrowTurret->SetupAttachment(MeshTurreCannon);
```

Slika 68. Primjer upravljanja komponentama u C++ kodu

Za korištenje drugih klasa u Blueprint sustavu potrebna je samo referenca na objekt klase. Nije potrebno upotrijebiti include komandu kao što je potrebno u C++-u. Dodavanje i referenciranje drugih klasa u Blueprint sustavu također je vrlo jednostavno. Nakon reference automatski se ponuđuju varijable, funkcije i događaji iz druge klase. Radi navedenih prednosti razvijanje računalne igre u vizualnom skriptnom jeziku je za početnike vrlo intuitivno i jednostavno.

7.2 PREDNOSTI C++ PROGRAMSKOG JEZIKA

C++ kao programski jezik unutar softvera Unreal 4 razvojnog okruženja ima svoje prednosti i nedostatke u usporedbi sa Blueprint programskim jezikom. Prva prednost C++-a je da ima bolje performanse. One se u računalnim igrama mjere uz pomoć FPS-a ili brojem uzastopnih slika prikazanih u sekundi. Pri snimanju filmova i animiranih filmova često se koriste kamere koje snimaju 24 slike u sekundi. U slučaju igara, performanse variraju značajno ovisno o igri, postavkama igre i platformi na kojoj se igra. Osim toga brzina i reaktivnost ovise i o veličini igre. Jedan od programera popularne igre Fortnite, Billy Bramer (2014), je u objavi na Unreal engine forumu također potvrdio kako nativni C++ nadmašuje performanse Blueprint sustava iz nekoliko razloga. Kao prvo, Unreal razvojno okruženje je izvorno programirano u C++-u i u pozadini C++ uvijek radi. Nadalje, Blueprint nasljeđuje C++ klasu, odnosno, funkcije koje poziva su originalno u C++-u. Direktno pozivanje funkcije u C++-u je brže nego dodavanje jednog dodatnog koraka i pozivanja funkcije u C++-u kroz Blueprint sustav.

U slučaju izrađene igre za ovaj rad slike u sekundi C++ igre i Blueprint igre značajno se razlikuju. U Blueprint igri u prvoj razini FPS se kretao oko 72 FPS-a , dok se C++ igra na istoj razini kretala oko 94 FPS-a. U glavnom izborniku Blueprint projekt imao je oko 215 slika u sekundi, a C++ projekt se kretao oko 243. Ostali rezultati su navedeni na početku sedmog poglavlja. Testiranje gotovih igara potvrdilo je početna očekivanja. Iako razlika u performansama varira ovisno o razini, C++ projekt je u prosjeku, na svakoj razini i u glavnom izborniku donosio bolje performanse od Blueprint projekta. Obzirom na to da je broj funkcija koje se odvijaju u pozadini glavnog izbornika manji, razlika u performansama projekta u glavnom izborniku je također manja od razlike u drugim razinama igre. U prvoj razini Blueprint

ima 1.3 puta manje slika u sekundi od C++-a, dok je u glavnom izborniku razlika 1.13 puta manje slika u sekundi. Glavni faktor koji utječe na pad slika u sekundi unutar glavnog izbornika su pozadinska glazba i komponente korisničkog sučelja. Funkcionalnosti u glavnom izborniku se pozivaju samo u trenutku pritiska na gumb i u trenutku otvaranja glavnog izbornika, odnosno glavni izbornik nema funkcionalnosti koja se odvijaju u pozadini osim u navedenim trenucima, dok razine imaju veliki broj provjera i funkcija koji se odvijaju tijekom cijele igre. Blueprint jezik je prilagođen jednostavnijim funkcijama poput funkcije u glavnom izborniku. Što je veći broj funkcija i provjera to je i veća razlika među performansama jednog i drugog programskog jezika.

Jedan od nedostataka Blueprint programskog jezika je da rastom broja funkcija i događaja u jednoj klasi, ta klasa postaje teže čitljiva i kaotična. To se događa i kod C++-a i ostalih tekstualnih programskih jezika, ali posebno je vidljivo u slučaju vizualnih programskih jezika. Urednost i čitljivost u razvoju računalnih igara i u bilo kojem razvoja programskog softvera veoma je bitan. U tom pogledu, C++ ima veliku prednost nad Blueprint sustavom kada je riječ o čitljivosti u većim projektima.

Događaji u svakoj Blueprint klasi mogu imati samo jedan čvor. Na primjer, bilo koja klasa ima samo jedan BeginPlay čvor i sve funkcije koji se moraju izvoditi u tom događaju moraju biti pozvane linearno. Svaka funkcija koja se mora odvijati na početku igre mora biti postavljena u linearni red. Veći projekti imaju veći broj funkcija koji se tijekom igranja igre moraju odvijati istovremeno ili paralelno ili kao posljedica istog okidača. One se u Blueprint sustavu postavljaju kao čvorovi jedan nakon drugoga. Problem nastaje kada klase i funkcije postaju nelinearne, te se granaju i dijele. Drugi razlog za lošiju čitljivost Blueprinta je da većina događaja ima samo jedan output. Ako klasa ima mnogo funkcija koje se pozivaju na jednom događaju, tada Blueprint postaje pre kompleksan na vizualnoj razini na tom događaju. Pronalaženje problema, uređivanje i ispravljanje programskog koda iz tog razloga u vizualnom programskom jeziku postaje veoma zahtjevno. Iako i kod C++ programskog jezika razumljivost koda pada u slučaju kada on sadrži više funkcija i varijabli, to je manje izraženo nego kod Blueprint sustava, što je njegova velika prednost.

8 ZAKLJUČAK

Unreal razvojno okruženje, za razliku od mnogih drugih okruženja, za razvoj računalnih igara ima dva programska jezika, C++ programski jezik i Blueprint programski jezik. Cilj ovog rada bila je usporedba Blueprint i C++ programskih jezika, te su u tu svrhu izrađene dvije jednake računalne igre u svakom programskom jeziku.

Mnoge prednosti jednog programskog jezika u usporedbi sa drugim mogu se svesti na usporedbu tradicionalnih, tj. tekstualnih programskih jezika sa vizualnim programskim jezicima. Vizualni programski jezici, u koje spada i Blueprint jezik, orijentirani su početnicima i dizajnerima ili drugim osobama koje rade na razvoju računalnih igara i trebaju pristupiti programskom kodu, a nisu programeri. Blueprint ima ugrađene provjere i mehanike koje sprječavaju izradu skripte sa velikim pogreškama uz koje igra ne može funkcionirati (*eng. game breaking error*). Performanse su lošije u usporedbi sa C++ programskim jezikom, posebno ako je cijeli projekt napravljen u Blueprint programskom jeziku. Izrada projekta u Blueprint sustavu iz tog je razloga i brža, posebno za jednostavne računalne igre. Također, inicijalizacija komponenata, ugrađenih funkcija i varijabli brža je nego inicijalizacija istih u C++ sustavu. Međutim, rezultati testiranja performansi igara su dokazali da je C++ brži, posebno u igrama sa velikim brojem klasa.

Zbog velike razlike programskih jezika tijekom razvoja računalne igre, ali i u finalnoj igri, Unreal razvojno okruženje ima mnogo potencijalnih korisnika i programera. Zahvaljujući vizualnim programskim jezicima potencijalni korisnici koji nisu upoznati sa programiranjem ili razvojem računalnih igara brže i jednostavnije mogu naučiti raditi u Unreal 4 razvojnom okruženju, nego što bi bilo moguće u mnogim drugim razvojnim okruženjima. Osim toga početnicima nije nužna direktna tranzicija iz Blueprint sustava u C++, nego je moguća djelomična tranzicija. Zbog dobre suradnje i velike kompatibilnosti oba programska jezika moguće je raditi u njima istovremeno što omogućuje početnicima jednostavniju tranziciju iz vizualnih u tekstualne programske jezike. Moguća je konzistentna tranzicija u kojoj se sve više koristi jedan programski jezik, pošto je postotak u kojemu se koristi bilo koji od programskih jezika ovisan o korisniku. Jednako tako programski jezici funkcioniraju na sličnoj bazi te su imena varijabli, funkcija i mnogih klasa ista što dodatno pojednostavljuje korisniku tranziciju iz jednog programskog jezika u drugi.

Iako Blueprint ima svoju ulogu kao programski jezik, iz rada je vidljivo da je C++ brži i bolji programski jezik za potencijalnog korisnika koji želi dugoročno raditi u ovom programskom okruženju i razvijati igru na kompleksnijoj razini. C++ brži je u izvođenju, ubrzava tijek razvoja igre i uredniji je, što je analizirano i potvrđeno kroz izrađeni projekt.

C++ programski jezik, osim što brže izvodi operacije i mehanike u većim projektima, ima tendenciju biti uredniji i jednostavniji u svome programskom kodu. Kroz testiranje performansi prikazano je da u cijeloj igri i u svakoj razini C++ verzija igre ima veći broj slika u sekundi od Blueprint verzije, te ta razlika postaje još jasnija u kompleksnijim igrama, što je prikazano i objašnjeno u 7. poglavlju i na popratnim grafikonima.

Osim kompletnog programiranja u C++ programskom jeziku, alternativa za razvoj računalne igre također je kombinacija oba programska jezika. Ovakav način rada može ubrzati razvoj računalne igre, ali istovremeno može smanjiti performanse izgrađene igre. U ovom načinu rada bitno je balansirati C++ i Blueprint jezik kako bi se ubrzao i pojednostavio tijek rada dodavanjem Blueprint koda, a da ga se istovremeno ne koristi previše kako bi se izbjegla prevelika razlika u performansama igre. Cilj ovog načina rada u Unrealu je kombinirati oba programska jezika kako bi se maksimalno iskoristile njihove prednosti. C++ radi sa varijablama, događajima, kalkulacijama, kompleksnijim funkcijama, nasljeđivanjem i razmjenom podataka među klasama, dok Blueprint radi sa jednostavnijim funkcijama i komponentama u objektima i korisničkom sučelju. Uz to Blueprint je također vrlo koristan za brzo prototopiranje mehanika. Mogućnosti u kojem omjeru kombinirati programske jezike ima mnogo i to najviše ovisi o tipu, veličini, platformi i ciljanoj publici igre. Obzirom na to da je C++ jezik brži i da su kompleksnije igre u tom programu efikasnije, cilj je programirati što više u C++ programskom jeziku i u ovom načinu rada. U ovoj varijaciji bitno je da je C++ klasa roditeljska klasa Blueprint klasi. Jednako tako bitno je znati da se nasljeđivanje varijabli i klasa može vršiti samo u jednome smjeru, sa C++ klase na Blueprint klasu, ali ne i iz Blueprint klase u C++ klasu.

Obzirom na to da se pokazalo da ima bolje performanse u finalnoj igri, a istovremeno je razvoj igre u ovom programskom jeziku brz i u mnogim slučajevima i jednostavan, C++ programski jezik je u odnosu na Blueprint programski jezik bolji, i sa dovoljno prakse, i brži način razvoja računalnih igara u Unreal 4 razvojnom okruženju.

9 LITERATURA

1. Boshernitsan, M. i Downes, M. (1998). *Visual Programming Languages: A Survey*. Preuzeto 20.12.2020. sa izvora https://www.researchgate.net/publication/2379982_Visual_Programming_Languages_A_Survey

2. Bramer, B. (2014). *Blueprint Usage Questions.* Preuzeto 15.12.2020. sa izvora https://forums.Unrealengine.com/Unreal-engine/events/3192-twitch-fortnite-developers-discussion-apr-17-2014/page2?3035-New-Twitch-livestream-with-Fortnite-developers-Thursday-April-17=&viewfull=1#post152196

3. Cassar, L. (2017). *Traditional vs visual programming; are they really that different?* Preuzeto 15.12.2020. sa izvora https://www.auraq.com/traditional-vs-visual-programming-are-they-really-that-different

4. Cordone, R. (2019). *Unreal Engine 4 Game Development Quick Start Guide: Programming professional 3D games with Unreal Engine 4.* Birmingham: Packt Publishing Ltd.

5. Epic games (2004-2021). *Unreal Engine 4 Documentation.* Preuzeto 20.12.2020 sa izvora https://docs.Unrealengine.com/en-US/Programming/Introduction/index.html

6. Epic games (2004-2021). *Blueprints Visual Scripting.* Preuzeto 20.12.2020 sa izvora https://docs.Unrealengine.com/en-US/Engine/Blueprints/index.html

7. Fabricatore, C. (2007). *Gameplay and game mechanics design: a key to quality in videogames.* Preuzeto 20.12.2020 sa izvora: http://eprints.hud.ac.uk/id/eprint/20927/

8. Jones, B. i Liberty, J. (2004). *Sams Teach Yourself C++ in 21 Days*. Indianapolis, Indiana: Sams Publishing

9. Kirch-Prinz, U. i Prinz, P. (2002). *A Complete Guide to Programming in C++, JONES AND BARTLETT PUBLISHERS. Sudbury Massachusetts:* Jones & Bartlett Publishers

10. Macklin, C. i Sharp, J. (2016). *Games, Design and Play: A detailed approach to iterative game design.* Boston: Addison-Wesley

11. Mcguire, M. i College W. (2009*). Programming language notes.* Preuzeto 20.12.2020 sa izvora https://www.researchgate.net/publication/228882836_Programming_Language_Notes

12. Nixon, D. (2017). *Unreal Engine 4 for Beginners.* West Palm Beach: Apress

13. Ousterhout, J. K. (1998*). Scripting: Higher-Level Programming for the 21st Century. Computer (Volume: 31, Issue: 3)* Sun MicroSystems Laboratories, Inc.

14. Romero, M. i Sewell, B. (2019). *Blueprints Visual Scripting for Unreal Engine: The faster way to build games using UE4 Blueprints, 2nd Edition*. *BIRMINGHAM – MUMBAI:* Packt Publishing

15. Sherif, W. i Whittle, S. (2016). *Unreal Engine 4 Scripting with C++ Cookbook 1st Edition BIRMINGHAM – MUMBAI:* Packt Publishing

16. Tykhomyrov, O. (2002). *Introducing to Object-Oriented Programming, Seventh College on Microprocessor-Based Real–Time Systems in Physics*

10 POPIS SLIKA

11 POPIS TABLICA

12 POPIS GRAFOVA