Izrada računalne igre u razvojnom okruženju Unreal Engine 4

Drozina, Adrian

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:111061

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-12**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

Adrian Drozina

Izrada računalne igre u razvojnom okruženju Unreal Engine 4

Završni rad

Pula, rujan, 2016. godine

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

Adrian Drozina

Izrada računalne igre u razvojnom okruženju Unreal Engine 4

Završni rad

JMBAG: 0303046442, redoviti student

Studijski smjer: Informatika

Predmet: Napredne tehnike programiranja

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan, 2016. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Adrian Drozina, kandidat za prvostupnika informatike, ovime izjavljujem da je ovaj Završni 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 Završnog 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, Adrian Drozina dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Izrada računalne igre u razvojnom okruženju Unreal Engine 4 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.

U Puli, _____ (datum)

Potpis

_____

Sadržaj

1. Uvod ... 1

2. Biranje Game Engine-a ... 2

3. Unreal Engine 4 ... 5

3.1. Osnovne klase Unreal Engine-a 4 ... 5

3.1.1. UObject ... 5

2.1.2. AActor ... 5

2.1.3. UWorld ... 5

3.1.4. UActorComponent ... 5

3.2. Osnovni tipovi podataka u Unreal Engine-u 4 ... 6

3.3. Blueprint sustav ... 6

3.3.1. Načini komunikacije između dva Blueprint-a ... 6

4. QP_Tabletop ... 8

4.1. Osnovne klase ... 9

4.1.1. Container ... 9

4.1.2. QPT_Actor ... 9

4.1.3. Component ... 9

4.1.4. GameRule ... 10

4.1.5. ContainerComponent ... 10

4.2. Lanac inicijalizacije ... 12

4.2.1. Template ... 16

4.2.2. CommonRule ... 16

4.2.3. Spawner ... 16

4.2.4. QPT_Actor ... 17

4.2.5. Manager ... 17

4.2.6. Component ... 18

4.3. Blueprint klase ... 18

4.3.1. QPT_Actor-i ... 18

4.3.2. Dice Controller ... 24

4.3.3. Spawner-i ... 28

4.3.4. MultiDiceDemo ... 29

4.3.5. DiceWithBoxTemplate ... 34

4.3.6. DiceTemplate ... 36

5. Zaključak ... 38

Literatura 39
Popis Tablica 40
Popis programskog koda 40
Popis slika 41
Sažetak 42
Summary 43

1. Uvod

QPT_Tabletop je aplikacija dizajnirana za kreiranje društvenih igara u Unreal Engine-u 4. Prije samog početka kreiranja aplikacije, trebalo je odgovoriti na sljedeća dva pitanja: na koji način kreirati aplikaciju koja je ujedno fleksibilna i lagana za korištenje, te u kojem Game Engine-u će se kreirat.

Kako bi aplikacija bila fleksibilna, mora imati mnogo funkcionalnosti i parametara pomoću kojih korisnik ima veću slobodu pri kreiranju svoje društvene igre. No kako broj funkcionalnosti i parametara raste, tako interface postaje sve kompliciraniji, te je ujedno i teži za korištenje. Kako bi se taj problem riješio, aplikacija je podijeljena na nekoliko različitih klasa, koje zajedno predstavljaju igru u cjelini. Većina tih klasa predstavlja neki koncept, dok samo jedna od njih predstavlja specifična pravila. Ta klasa koristi druge klase kao komponente, te im daje kontekst, no druge klase ne ovise o njoj. Upravo ta modularnost dopušta korisniku da se koncentrira na određene dijelove igre, te efektivno rješava problem na način da ga podijeli na više manjih dijelova. Osim toga, postoje i unaprijed kreirane klase koje se mogu koristiti za kreiranje željenih igara, te se mogu koristiti kao reference ili dokumentacija, kako bi korisnik saznao kako kreirati druge klase. Također, sve te klase sadrže brojne parametre koji sadrže početne vrijednosti definirane na način da su spremne za korištenje uz minimalnu modifikaciju. Kako bi se dodatno olakšao rad u aplikaciji, cijeli proces inicijalizacije svih klasa je automatiziran, te se sve potrebne reference automatski spremaju. Samim time korisnik ne mora brinuti o inicijalizaciji pojedinih klasa, što mu dopušta da se koncentrira na kreiranje željene igre.

Biranje Game Engine-a u kojem će se aplikacija kreirat je kompleksan problem. Važno je da aplikacija bude dostupna mnogim developerima, no ukoliko Engine u kojem je kreirana ne podržava potrebne funkcionalnosti, tada niti aplikacija neće sadržavati sve potrebne funkcionalnosti. No ukoliko Engine posjeduje sve potrebne funkcionalnosti, no previše je skup, tada mnogi indi developeri neće bit u stanju ju koristiti. Uzimajući cijenu, naknade, potrebne funkcionalnosti i dostupnu dokumentaciju u obzir, odabran je Unreal Engine 4 kao razvojna platforma.

Svrha ovog rada je prikazati rad aplikacije QP_Tabletop. Kako bi se to postiglo, prikazat će se, korak po korak, cijeli proces kreiranja jedne jednostavne demo igre. Među ostalim, prikazat će se na koji način rade sve osnovne klase aplikacije, koje su njihove specifične svrhe, te na koji način se one inicijaliziraju. Tada će se detaljno prikazati određene

klase koje nasljeđuju od njih, te na koji način se razne komponente mogu kombinirati kako bi se kreirale drugačije igre.

U poglavlju 2 se prikazuju razni Game Engine-i koji su bili kandidati za kreiranje QP_Tabletop aplikacije, te parametri po kojima su se ocijenili.

U poglavlju 3 objašnjavat će se osnovni pojmovi Unreal Engine-a koji su potrebni za razumijevanje rada QP_Tabletop aplikacije. Među tim pojmovima spadaju osnovne klase poput UObject i AActor, te se pojašnjavaju osnovni tipovi podataka. Nakon toga će se pojasniti što je to Blueprint sustav, koja je njegova svrha, te tri glavna načina komuniciranja između dvije klase Blueprint-a. To su direktna komunikacija, Blueprint interface i Event Dispatcher.

U poglavlju 4 objašnjavat će se rad aplikacije QP_Tabletop na način da se kreira jednostavna demo igra u kojoj igrač baca 4 kockice, te se njihova suma ispisuje. Prvo će se pojasnit osnovne klase te aplikacije, uključujući i lanac inicijalizacije, koji pokazuje interakciju između tih klasa prilikom kreiranja igre. Tada se objašnjava rad unaprijed kreiranih klasa koje će se koristiti za kreiranje demo igre, te proces spajanja komponenti kako bi se igra uspješno stvorila. Na kraju će se dio funkcionalnosti koji se često koristi u raznim igrama premjestiti u novu klasu tipa Template, kako bi se mogla ponovo upotrebljavati, te, kao primjer, će se kreirati ista demo igra koristeći tu novu Template klasu.

2. Biranje Game Engine-a

Pošto QP_Tabletop aplikaciji nisu potrebne napredne grafičke funkcije, kompleksno upravljanje animacijama, te ostali napredi sustavi, pri biranju Game Engine-a u kojem će se kreirat, te funkcionalnosti se ne uzimaju u obzir. Pošto aplikacija mora bit dostupna, među ostalim, i indi developerima, te ujedno mora biti lagana za korištenje, parametri koji se uzimaju u obzir su cijena, uključujući mjesečno plaćanje i naknade, lakoća korištenja, te dostupnost dokumentacije. Stoga su tri glavna kandidata bila Unity Engine, Cryengine i Unreal Engine 4.

Unity Engine je Engine koji je besplatan za sve timove developera koji zarađuju manje od 100.000$ godišnje, te nije potrebno plaćati naknade od dobivenog prihoda. No ukoliko je godišnji prihod veći od 100.000$, potrebno je nabaviti plus verziju, koja se plača 35$ mjesečno za godišnji prihod do 200.000$. Preko toga potrebno je nabaviti pro verziju koja košta 125$ mjesečno. Unity Engine također sadrži takozvane Analytics Points (analitičke

bodove), koji se koriste za kreiranje koda. Svaki Event unutar Engine-a košta određeni broj analitičkih bodova, te ukoliko se pređe limit, ne mogu se dodavati novi Event-i. Svaka verzija mjesečnog plaćanja daje korisniku određen broj analitičkih bodova po projektu. Tako osnovna verzija ima 1000 analitičkih bodova, Plus verzija ima 2000 analitičkih bodova, te Pro verzija ima 5000 analitičkih bodova. Sljedeća tablica prikazuje mjesečnu cijenu, maksimalan dopušten godišnji prihod te broj analitičkih bodova za svaku verziju.

	Personal	Plus	Pro
Mjesečna cijena	Besplatno	35$	125$
Maksimalan dopušten godišnji prihod	100.000$	200.000$	Neograničeno
Broj analitičkih bodova	1000	2000	5000

Tablica 1. Usporedba različitih verzija plaćanja za Unity Engine.

Izvor: „Usporedba raznih planova…“, 2015.

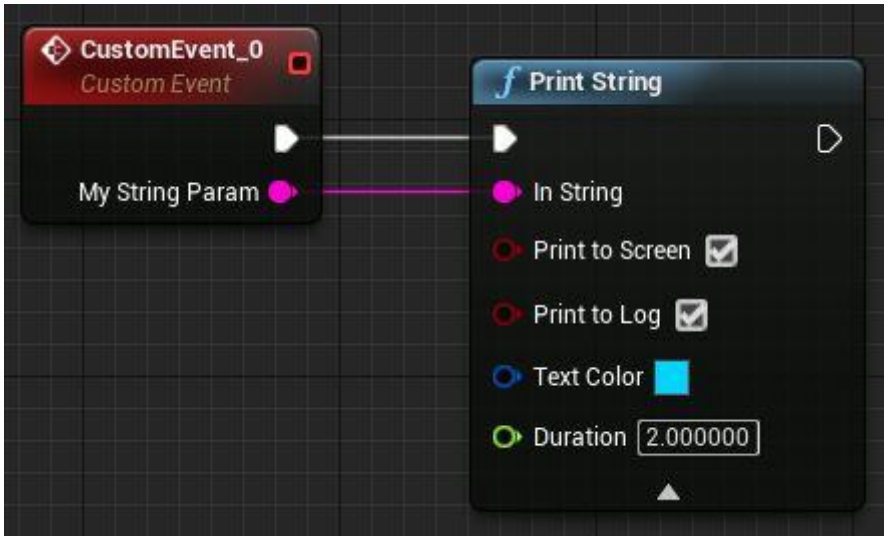
Korištenje Cryengine-a zahtjeva licencu koja mjesečno košta 9.90$. Source kod je dostupan, te je napisan u C++ i LUA jeziku. Kao što je prikazano na sljedećoj slici, Engine sadrži vrlo napredne grafičke sposobnosti.

Slika 1. Primjer naprednih grafičkih sposobnosti Cryengine-a

Izvor: „Prikaz grafičkih mogućnosti“, 2015.

Veliki nedostatak Cryengine-a je taj što korisnik nema pristup čitavoj dokumentaciji. Uz dodatno plaćanje od 50$ mjesečno, korisnici dobivaju pristup takozvanom Insider-u, tj. dijelu foruma na kojem mogu dobiti pomoć i dodatnu dokumentaciju za rješavanje svojih problema.

Unreal Engine 4 je besplatni Game Engine, te je korisnik dužan platiti 5% prihoda, ukoliko je u kvartalu zaradio preko 3.000$ („If you love…“, 2015). Pisan je u C++ jeziku, te je kod Open Source. Podržava velik broj platformi, te nove verzije dolaze po mjesečnoj bazi. Sadrži velik broj dokumentacije, koji varira od samog koda, primjera korištenja, video dokumentacije, te gotovih projekata. Također sadrži Blueprint sustav, koji omogućuje korisniku kreiranje cijelih igara isključivo putem vizualnog skriptiranja. Na sljedećoj slici je prikazan jednostavan kod u Blueprint sustavu.



Slika 2. Jednostavan kod u Blueprint sustavu koji ispisuje String.

Izvor: „Custom Events“, 2015.

Uzimajući u obzir te parametre, Unity Engine nije pogodan za kreiranje QP_Tabletop aplikacije, pošto svaki Event u projektu zahtjeva određeni broj analitičkih bodova, što može spriječit korisnike u kreiranju svoje igre, ili ih natjerati da koriste neku drugu verziju, te stoga plaćaju određenu mjesečnu svotu. Cryengine sadrži najmanje dokumentacije od ta tri Engine-a, te pošto je potreban dodatni mjesečni trošak kako bi se olakšalo njegovo korištenje, također nije pogodan za kreiranje aplikacije. Unreal Engine 4 je besplatan, te korisnik nema trošak ako nema profit veći od 3.000$ mjesečno. To dopušta korisniku da ne žuri s radom u aplikaciji, nego da kreira svojim tempom, što olakšava rad. Također Blueprint sustav uvelike pomaže u brzom kreiranju igara, pošto dopušta korisniku lagano kreiranje

vizualnog koda uz kratko vrijeme kompajliranja. Stoga je odabran upravo Unreal Engine 4 kao Game Engine u kojem će se kreirati QP_Tabletop aplikacija.

3. Unreal Engine 4

Unreal Engine 4 je profesionalni AAA Game Engine, kreiran od tvrtke Epic Games. Cijeli Engine, sa C++ source kodom, je dostupan besplatno, te je potrebno platiti 5% prihoda od prodaje igre napravljene u njemu. U vrijeme pisanja ovog rada, posljednja verzija je 4.13.0, koja je ujedno i verzija korištena za kreiranje aplikacije QP_Tabletop.

3.1. Osnovne klase Unreal Engine-a 4

3.1.1. UObject

Najosnovnija klasa koji Editor prepoznaje je UObject, stoga sve druge klase nasljeđuju od nje. Također svaka klasa, s iznimkom AActor-a, ima predznak U („Objects“, 2015).

2.1.2. AActor

AActor (ukratko Actor) je UObject koji se može nalaziti u svijetu (klasi UWorld) pošto podržava 3D transformaciju (lokacija, rotacija i veličina). Svaki Actor ima predznak A, podržava hijerarhiju komponenti (klasa UComponent), te njegovu transformaciju određuje upravo transformacija glavne komponente zvane Root. Actor također ima pristup funkciji Tick(), koja ažurira njegovo stanje, te se može koristiti za kod koji se treba odvijati svaki frame. Proces kreiranja Actora se zove Spawn-anje („Actors“, 2015).

2.1.3. UWorld

UWorld (ukratko World ili Svijet) je klasa koja predstavlja prostor u kojem se igra odvija. Većina klasa koje se nalaze u njemu su Actor-i, te je on također zadužen za njihovo Spawn-anje („UWorld“, 2015).

3.1.4. UActorComponent

UActorComponent je klasa koja predstavlja komponente koje se dodaju Actor-u, kako bi utjecala na njihovo ponašanje. Kao i Actor-i, UActorComponent sadrži funkciju koja služi za ažuriranje svakog frame-a, zvanu TickComponent().

USceneComponent je pod klasa UActorComponent-a, no dijeli neke sličnosti sa Actor-om. Točnije, sadrži transformaciju, te se može spajati ili biti spojena s drugim

instancama iste klase, no ne može samostalno postojati. Mora se nalaziti unutar Actor-a („Components“, 2015).

3.2. Osnovni tipovi podataka u Unreal Engine-u 4

Osim osnovnih tipova podataka, poput integer-a, float-a, bool-a i sličnih, Unreal Engine 4 sadrži dodatne tipove podataka u obliku unaprijed definiranih struktura (sve strukture imaju predznak F). Najčešće korištene strukture kao tipovi podataka su FVector, koji sadrži tri float vrijednosti, koje predstavljaju vektor u 3D prostoru sa X, Y i Z osima, FRotator, koji se također sastoji od 3 float vrijednosti, no one predstavljaju rotaciju u Euklidovoj geometriji, tj. rotaciju oko tri osi, X osi (Roll), Y osi (Pitch), te Z osi (Yaw). Posljednji važan tip podataka je transformacija, predstavljena u strukturi FTransform. Sastoji se od dva FVector-a i jednog FRotator-a, koji predstavljaju lokaciju, rotaciju i veličinu.

U Blueprint sustavu, svi tipovi podataka su obojani određenom bojom. Integer je svijetlo zelena, float je tamno zelena, FVector je žut, FRotator je svijetlo ljubičast, FTransform je narančast, te pokazivači na instance klasa su plave boje („Blueprint Variables“, 2015).

3.3. Blueprint sustav

Blueprint sustav je sustav koji omogućuje korisniku vizualno skriptiranje kako bi mogao stvorit kompleksnu igru bez pisanja koda. Jedan Blueprint predstavlja jednu klasu. Kompajliranje Blueprint koda je puno brže od c++ koda, što omogućuje korisniku puno veću brzinu kreiranja sadržaja („Blueprint Overview“, 2015).

3.3.1. Načini komunikacije između dva Blueprint-a

3.3.1.1. Direktna komunikacija

Najosnovniji način komunikacije dvaju Blueprint-a je direktna komunikacija. Ona se postiže na način da jedna klasa komunicira s drugom koristeći njenu referencu. („Direct Blueprint Communications“, 2015).

3.3.1.2. Blueprint interface

Blueprint interface je sustav koji se bazira na polimorfizmu, te dopušta različitim klasama da komuniciraju preko funkcija s jednakim deklaracijama, ali različitim definicijama. Kao što su Stroustrup (2014) i Jeff Langr i Ottinger (2011) objasnili, apstrakcija uvelike

povećava fleksibilnost aplikacije, te dopušta generičan kod. Koristeći apstraktnu klasu s pure virtual funkcijama kao roditelja, dopušta se da dijete samo kreira svoju definiciju klase. Na taj način više različitih klasa mogu međusobno komunicirati preko istog interface-a, no svaka može sadržavati drugačiju implementaciju.

Na primjer, klasa Lampa i klasa Auto sadrže funkcionalnost koje im dopušta da se upale, no ne sadrže zajedničkog roditelja koji ima tu funkcionalnost. Tada se koristi interface Upali, koji se pridodaje klasi Lampa i klasi Auto. Nakon toga se u svakoj od tih klasa definira događaj koji se dešava prilikom pozivanja funkcije Upali(). Auto pali motor, dok Lampa zatvara strujni krug, te pali svijetlo.

Blueprint interface je sustav baziran na Interface-u. Dopušta nepovezanim klasama da međusobno komuniciraju, bez potrebe da korisnik unaprijed provjerava da li klasa sadrži željeni inteface, iako ima tu mogućnost. Ukoliko klasa ne sadrži interface, tada poziv ne uspije, ne pozivajući greške („Blueprint Interface“, 2015).

3.3.1.3 Event Dispatcher

Event Dispatcher je sustav baziran na dizajnu uzoraka promatrač (Design Pattern Observer), kao što su objasnili Gamma, Helm, Johnson i Vlissides (1994), te je jedan od jedan-naprema-više načina komuniciranja. Koristi se na način da jedna klasa pošalje notifikaciju u slučaju nekog događaja. Druge klase mogu reagirati na tu notifikaciju na način da ju povežu s nekom svojom funkcijom (notifikacija i funkcija moraju imati iste tipove argumenata). Taj proces se zove Bind-anje. Ukoliko prva klasa pošalje notifikaciju, tada sve funkcije koje su Bind-ane na nju se pozivaju. Prednost Event Dispatcher-a je u tome što dopušta klasama da pokreću funkcije drugih klasa bez da znaju koje su to klase. Samim time su odlične za modularnost aplikacija, pošto ne ovise u funkcijama koje su Binda-ane, a samim time o drugim klasama. Ukoliko niti jedna funkcija nije Bind-ana na notifikaciju, te se ona pošalje, neće se ništa dogoditi (ne javlja se greška).

Primjer korištenja Event Dispatcher-a je kada korisnik kreira klasu Poluga, koja prilikom interakcije otvara neka vrata. No pošto Poluga može utjecat na druge klase, te ne bi trebala biti direktno povezana sa vratima, tada samo pošalje notifikaciju da je poluga povučena. Vrata mogu Bind-at funkciju Otvori() sa tom notifikacijom, te će se ona otvoriti kad se ona povuče. Na taj način se Poluga može koristiti za razne interakcije bez da ovisi o klasi na koju utječe („Event Dispatchers“, 2015).

4. QP_Tabletop

Aplikacija QP_Tabletop, skraćeno od Quick Prototype Tabletop, je alat koji služi za brzo prototipiranje društvenih igara, koji zahtjeva minimalno znanje i napor. To se postiže na način da korisnik ne mora znati unutarnji rad aplikacije kako bi ju mogao koristiti, te pošto koristi Blueprint sustav, nije potrebno da zna i pisanje koda.

Važno je razlikovati dva pojma, korisnik i igrač. Korisnik je osoba koja koristi aplikaciju kako bi kreirala željenu društvenu igru, dok je igrač osoba koja igra tu igru. Korisnik može igrati igru kako bi provjerio da li ona pravilno radi, no u pravilu igrača ne zanima kako ona radi, dok god radi kako treba.

Svaka društvena igra kreirana u QP_Tabletop aplikaciji sadrži komponente koje se nalaze unutar jedne klase, te koriste tu klasu kao referentnu točku u Svijetu. Samim time u jednom trenutku može postojati više različitih igara u istome Svijetu bez da one međusobno utječu jedna na drugu. Time se ubrzava rad korisnika, jer ne mora kreirati novi svijet za svaku instancu igre, te je u stanju pokretati više igara istodobno, kako bi ih mogao lakše usporediti.

Kako bi se prikazao rad aplikacije, kreirat će se jednostavna demo igra, u kojoj se klikom miša na dugme bacaju četiri kockice. Suma brojeva, koja se dobije nakon bacanja, će se ispisati na vrh dugmeta. Daljnjim pritiskom na dugme, kockice se ponovo bacaju te se ispisuje nova suma.

Prije početka kreiranja demo igre, potrebno je pobliže pojasniti klase koje se nalazu u aplikaciji.

Sve klase u QP_Tabletop aplikaciji su podijeljene u dvije grupe: osnovne klase i Blueprint klase. Osnovne klase predstavljaju Framework aplikacije, te su pisane u C++ kodu. Ne sadržavaju nikakav kod specifičan za društvene igre, te služe za automatsku inicijalizaciju i spremanje referenci, kako bi se korisnik mogao koncentrirati na kreiranje same igre.

Blueprint klase su klase kreirane u Blueprint sustavu, te nasljeđuju od osnovnih klasa. One sadrže kod za kreiranje same igre. Većina tih klasa sadrži generičan kod koji predstavlja neki koncept koji se često pojavljuje u različitim društvenim igrama, poput figurica, kockica, nasumične izmjene igrača prilikom igranja igre (igra na poteze), praćenje bodova igrača i sl.

4.1. Osnovne klase

QP_Tabletop aplikacija se sastoji od pet osnovnih klasa napisanih u C++ programskom jeziku, koje zajedno čine Framework alata. Te osnovne klase su QPT_ContainerBase (ukratko Container), QPT_ActorBase (ukratko QPT_Actor), QPT_ComponentBase (ukratko Component), QPT_GameRuleBase (ukratko GameRule), te QPT_ContainerComponentBase (ukratko ContainerComponent), koja ima djecu QPT_CommonRuleBase (ukratko CommonRule), QPT_ManagerBase (ukratko Manager), QPT_SpawnerBase (ukratko Spawner) i QPT_TemplateBase (ukratko Template). Sve te klase imaju prefiks QPT_ i sufiks Base, što označava da su to apstraktne klase koje ne bi trebale postojati unutar Svijeta. Njihova svrha je podržavanje rada Blueprint klasa na način da automatiziraju njihovu inicijalizaciju, te spremaju sve potrebne vrijednosti i reference. Time olakšavaju rad korisniku, te ih korisnik može smatrati crnim kutijama, pošto mu nije potrebno detaljno znanje njihova rada kako bi radio u aplikaciji.

4.1.1. Container

QPT_ContainerBase (ukratko Container) je klasa koja nasljeđuje od Actor klase. On je specifičan po tome što on, kao ni Blueprint klase koje nasljeđuju od njega, ne sadrži kod koji utječe na pravila igre. On predstavlja kutiju koja sadrži sve komponente koje zajedno čine igru (te samim time sadrži sve reference na te klase). Također, njegova transformacija ima direktan utjecaj na transformaciju drugih klasa koje nasljeđuju od Actor-a. Na taj način korisnik ne mora brinuti o apsolutnoj transformaciji u Svijetu, jer se ona automatski izračunava. Njegova posljednja funkcija je ta što on započinje lanac inicijalizacije, koji inicijalizira sve druge klase, te efektivno započinje igru. Pošto Container nema direktan utjecaj na pravila igre, tada se može koristiti za kod koji je neovisan o samoj igri, poput puštanja pozadinskog zvuka i sl.

4.1.2. QPT_Actor

QPT_ActorBase (ukratko QPT_Actor) je klasa koja nasljeđuje od Actor klase, te predstavlja fizičku komponentu igre u prostoru, tj. figuricu, kockicu, ploču, polje na ploči, kartu itd. Skup svih QPT_Actor klasa predstavlja igru u cjelini onako kako ju igrač doživljava, stoga su one najčešće i medij preko koje igrač može utjecati na igru (pomičući figuricu, bacajući kockicu, okrećući kartu i sl.).

4.1.3. Component

QPT_ComponentBase (ukratko Component) je klasa koja nasljeđuje od UActorComponent klase. Za razliku od ostalih klasa zasnivanih na UActorComponent-u,

ona se ne pridružuje Container-u, nego QPT_Actor-u. Primarna funkcija joj je da PT_Actor-u pridodaje neki tip podataka, koji igri daje kontekst. Na primjer, ukoliko postoji ploča sa 5X5 polja na kojoj se nalaze figurice, potrebno je znat na kojoj poziciji se nalazi figurica. Možemo kreirati Component klasu koja sadrži strukturu podataka Lokacija, koji se sastoji od dva integera. Instanca koja se nalazi na polju predstavlja poziciju tog polja na ploči, dok instanca koja se nalazi na figurici predstavlja lokaciju te figurice. Uspoređivajući vrijednosti lokacije figurice i lokacije polja, možemo saznati na kojem polju se nalazi figurica.

4.1.4. GameRule

QPT_GameRuleBase (ukratko GameRule) je klasa koja nasljeđuje od UActorComponenent klase. Ona predstavlja samo pravila igre, ne i njene komponente. Na primjer, u GameRule klasi je definirano šta se dešava ukoliko igrač okrene kartu, dok druge klase brinu o tome što je karta, te kako se ona okreče.

GameRule klasa koristi direktnu komunikaciju sa drugim klasama, pošto svaka igra zna koje komponente su joj potrebne. No obrnuta situacija je kompliciranija. Karta ne zna u kojim sve igrama pripada, stoga ne može koristiti reference za direktnu komunikaciju s GameRule klasama. Stoga se većina komunikacija s drugih klasa prema GameRule klasi odvija preko Event Dispatcher-a, koji dopušta da klase ne moraju unaprijed znati s kime komuniciraju. Ta dva svojstva uvelike pomažu modularnosti aplikacije.

Još jedna funkcija GameRule klase je ta što ona sadržava listu inicijalizacije, tj. listu Spawner-a, Manager-a, CommonRule-a i Template-a koji su joj potrebni za kreiranje željene igre. Sve te klase se kreiraju i inicijaliziraju preko GameRule klase.

4.1.5. ContainerComponent

QPT_ContainerComponentBase (ukratko ContainerComponent) je klasa koja nasljeđuje od klase UActorComponent. Ona služi kao roditelj za klase QPT_ManagerBase, QPT_SpawnerBase, QPT_CommonRuleBase, te QPT_TemplateBase. Te klase nasljeđuju funkcije OnCreated(), OnInitialized() i PostInitialized(), koje javljaju da je klasa prošla određeni stupanj inicijalizacije. Pridodaje se klasi Container.

-QPT_ManagerBase

QPT_ManagerBase (ukratko Manager) je klasa koja nasljeđuje od ContainerComponent klase. Ona služi za filtriranje kroz Component instance koje se nalaze na QPT_Actor instanci. Može vratit referencu na komponentu ili na samog QPT_Actor-a. Na primjer, koristeći prijašnju alegoriju sa Component klasom, korisnik bi trebao iterirati kroz

sve instance kako bi pronašao željeni par, te preko njih izvukao reference na željene QPT_Actore. Manager to radi automatski.

-QPT_SpawnerBase

QPT_SpawnerBase (ukratko Spawner) je klasa koja nasljeđuje od ContainerComponent klase. Njena funkcija je kreiranje (Spawn-anje) instance QPT_Actor-a. Pošto je uvijek potrebno kreirati neke instance QPT_Actor-a na samom početku igre, kao pomoć korisniku, Spawner ih automatski kreira tijekom inicijalizacije po zadanim parametrima. Ukoliko korisnik ne želi tu funkcionalnost, tada može staviti vrijednost bool varijable bSpawnOnInitialization na laž.

Transformacija QPT_Actora koju Spawner kreira ovisi o varijabli tipa FTransform TransformOffset. On automatski izračunava relativnu transformaciju, tako da ukoliko korisnik navede da QPT_Actor mora imati lokaciju 100.0, 0.0, 0.0, tada će se nalaziti 100 cm dalje od Container-a na X osi, te uvijek će se kreirati na toj udaljenosti neovisno o njegovoj lokaciji. Ukoliko korisnik želi kreirati instancu QPT_Actor-a na apsolutnoj transformaciji u svijetu, tada stavlja vrijednost bool varijable bUseContainerTransform na laž.

Ukoliko korisnik želi kreirati Spawner-a koji kreira instance QPT_Actor-a po određenim pravilima, na primjer, da u određenom prostoru nasumično kreira određen broj instanci, kod za to stavlja u virtualnu funkciju CalculateSpawnTransform, koja generira željene transformacije, te ih sprema u niz SpawnTransforms. Tada za svaki element u tom nizu kreira jednu instancu na toj transformaciji. To znači da je korisniku potrebna samo funkcija CalculateSpawnTransforms() da generira svoje vlastite klase Spawner-a sa željenim ponašanjem. Sve reference na instance koje Spawner kreira se spremaju, te se mogu čitati u bilo kojem trenutku.

-QPT_CommonRuleBase

QPT_CommonRuleBase (ukratko CommonRule) je klasa koja nasljeđuje od ContainerComponent klase, te predstavlja pravila koja se često pojavljuju u raznim društvenim igrama, a nisu vezana za druge klase, poput spremanja i računanja bodova igrača, upravljanja mišem i sl. Osim toga, mogu služit kao pomoć pri korištenju više QPT_Actor-a, npr., U slučaju kada je QPT_Actor karta, tada CommonRule može uzeti funkciju špila, te pratiti kartu kao dio cjeline.

-QPT_TemplateBase

QPT_TemplateBase (ukratko Template) je klasa koja nasljeđuje od ContainerComponent klase, te služi za olakšavanje i pojednostavljivanje inicijalizacije igre. Na primjer, umjesto da korisnik kreira špil karata, svakoj ručno postavi broj i simbol na sve karte, te svaku kartu doda u špil, Template može to učiniti za njega, olakšavajući mu posao. Kako bi to postigao, Template može dodavati klase na listu za inicijalizaciju, te zatim utjecati na instance tih klasa prilikom inicijalizacije.

4.2. Lanac inicijalizacije

Lanac inicijalizacije je niz funkcija koje se pozivaju po određenom redoslijedu kako bi se sve klase uspješno inicijalizirale. Pokreće se samo jednom, i to na samom početku igre, nakon čega se sve klase kreiraju. Dijelovi lanca se mogu ponoviti kasnije, kao na primjer kada se usred igre kreira novi QPT_Actor, te je i njega potrebno inicijalizirati. Lanac inicijalizacije direktno ovisi o GameRule klasi, i o njenoj listi za inicijalizaciju.

Neke funkcije imaju sufiks _Implementation. On je potreban kako bi Editor znao da funkcija ima implementaciju i definiciju u C++ kodu, no može se pozivati i modificirati u Blueprint sustavu.

Lanac inicijalizacije započinje pozivanjem funkcije Initialize() u klasi Container, te završava kada Container poziva funkciju PostInitialize().

Nakon provjere da svijet postoji i da je zadana GameRule klasa za kreiranje validna, ona se kreira i registrira, kako bi ju Engine prepoznao. Zatim poziva OnCreated() funkciju unutar GameRule klase koja kao argument prima referencu na Container, u koju se sprema. Zatim se poziva SetGameRuleParameters(), koja služi kako bi Container mogao poslati informacije GameRule klasi prije nego li se inicijalizira. Tada GameRule poziva PreInitialize(). U toj funkciji se mogu dodati dodatne klase u listi za inicijalizaciju, ukoliko je to potrebno. Na primjer, u slučaju da igra zahtjeva da svaki igrač ima vlastiti špil karata, a broj igrača može varirati, tada se u ovoj funkciji čita broj igrača, te se za svakog dodaje Template koji dodaje špil karata. Nakon što se ta funkcija izvrši, GameRule poziva OnInitialized(), nakon čega Container javlja da je inicijalizacija završila pozivajući PostInitialize(). Cijeli redoslijed pozivanja funkcija je prikazan u sljedećem kodu.

```
void AQPT_ContainerBase::Initialize()
{
	if (!GetWorld()) return;
	if (!GameRuleClassToCreate) return;
	UQPT_GameRuleBase* NewGameRule = NewObject<UQPT_GameRuleBase>(this,
GameRuleClassToCreate);
	if (!NewGameRule) return;
```

```
	GameRule = NewGameRule;
	NewGameRule->RegisterComponent();
	NewGameRule->OnCreated(this);
	SetGameRuleParameters();
	NewGameRule->PreInitialize();
	NewGameRule->OnInitialized();
	PostInitialize();
}
```

Programski kod 1. Funkcija Initialize() u klasi Container.

Prva klasa koja se inicijalizira je Template, jer onda dodaje druge klase na listu za inicijalizaciju. Iz istog razloga se u toj petlji ne koristi ForEachLoop, jer Template klasa može dodati drugu Template klasu na listu za inicijalizaciju, mijenjajući njenu veličinu prilikom iteracije, što može dovesti do neželjenih problema. Nakon što se za svaku Template klasu poziva CreateTemplate() funkcija, poziva se funkcija OnAllTemplatesInitialized() koja javlja da su se svi Template-i inicijalizirali. Na sličan način se za svaki CommonRule poziva CreateCommonRule() funkcija, te se zatim javlja da su sve klase kreirane i inicijalizirane pozivajući funkciju OnAllCommonRulesInitialized(). Isto se poziva i za Spawner-e i Manager-e. (Spawner-i se kreiraju prije Manager-a kako bi Componet-e imale QPT_Actor-e kojima se mogu pridružit). Sljedeći programski kod prikazuje funkciju OnInitialized() u klasi GameRule.

```
void UQPT_GameRuleBase::OnInitialized_Implementation()
{
	for (int32 i = 0; i < TemplatesToCreate.Num(); ++i)
	{
		TSubclassOf<UQPT_TemplateBase> CurrentTemplateClass =
TemplatesToCreate[i];
		CreateTemplate(CurrentTemplateClass);
	}
	OnAllTemplatesInitialized();

	for (TSubclassOf<UQPT_CommonRuleBase> CurrentCommonRuleClass :
CommonRulesToCreate)
	{
		CreateCommonRule(CurrentCommonRuleClass);
	}
	OnAllCommonRulesInitialized();

	for (TSubclassOf<UQPT_SpawnerBase> CurrentSpawnerClass :
SpawnersToCreate)
	{
		CreateSpawner(CurrentSpawnerClass);
	}
	OnAllSpawnersInitialized();
```

```
	for (TSubclassOf<UQPT_ManagerBase> CurrentManagerClass :
ManagersToCreate)
	{
		CreateManager(CurrentManagerClass);
	}
	OnAllManagersInitialized();
}
```

Programski kod 2. Funkcija OnInitialized() u klasi GameRule.

Funkcija CreateTemplate() kreira instancu zadane Template klase, registrira ju, te ju sprema. Zatim taj Template poziva OnCreated() funkciju koja kao argument prima referencu instance GameRule klase, te ju sprema. Zatim GameRule poziva funkciju OnSetTemplateParameters() koja prima argument upravo kreiranog Template-a. U toj funkciji GameRule postavlja potrebne parametre kako bi Template radio kako korisnik želi, na primjer, ukoliko Template kreira špil karata, tu se postavlja koliko džokera će bit u špilu, te taj Template dodaje nove klase u listu za inicijalizaciju, poput Spawnera koji će kreirati karte. Zatim Template poziva OnInitialized() funkciju u kojoj se inicijalizira. GameRule zatim javlja da je Template inicijaliziran pozivajući funkcije OnInitializeTemplate() i PostInitializeTemplate(). Sljedeći kod predstavlja funkciju CreateTemplate() u klasi GameRule.

```
void UQPT_GameRuleBase::CreateTemplate(TSubclassOf<UQPT_TemplateBase>
TemplateClass)
{
	UQPT_TemplateBase* NewTemplate =
NewObject<UQPT_TemplateBase>(GetContainer(), TemplateClass);
	if (!NewTemplate) return;

	NewTemplate->RegisterComponent();
	Templates.Add(NewTemplate);
	NewTemplate->OnCreated(this);
	OnCreateTemplate(NewTemplate);
	OnSetTemplateParameters(NewTemplate);
	NewTemplate->OnInitialized();
	OnInitializeTemplate(NewTemplate);
	PostInitializedTemplate(NewTemplate);
}
```

Programski kod 3. Funkcija CreateTemplate() u klasi GameRule.

Create funkcije su slične i za ostale klase. Programski kodovi 4, 5 i 6 predstavljaju funkcije CreateCommonRule(), CreateSpawner() i CreateManager() u klasi GameRule.

```
void UQPT_GameRuleBase::CreateCommonRule(TSubclassOf<UQPT_CommonRuleBase>
CommonRuleClass)
{
```

```
	UQPT_CommonRuleBase* NewCommonRule =
NewObject<UQPT_CommonRuleBase>(GetContainer(), CommonRuleClass);
	if (!NewCommonRule) return;

	NewCommonRule->RegisterComponent();
	CommonRules.Add(NewCommonRule);
	NewCommonRule->OnCreated(this);
	OnCreateCommonRule(NewCommonRule);
	OnSetCommonRuleParameters(NewCommonRule);
	NewCommonRule->OnInitialized();
	OnInitializeCommonRule(NewCommonRule);
	PostInitializedCommonRule(NewCommonRule);
}
```

Programski kod 4. Funkcija CreateCommonRule() u klasi GameRule.

```
void UQPT_GameRuleBase::CreateSpawner(TSubclassOf<UQPT_SpawnerBase>
SpawnerClass)
{
	UQPT_SpawnerBase* NewSpawner =
NewObject<UQPT_SpawnerBase>(GetContainer(), SpawnerClass);
	if (!NewSpawner) return;

	NewSpawner->RegisterComponent();
	Spawners.Add(NewSpawner);
	NewSpawner->OnCreated(this);
	OnCreateSpawner(NewSpawner);
	OnSetSpawnerParameters(NewSpawner);
	NewSpawner->OnInitialized();
	OnInitializeSpawner(NewSpawner);
	PostInitializedSpawner(NewSpawner);
}
```

Programski kod 5. Funkcija CreateSpawner() u klasi GameRule.

```
void UQPT_GameRuleBase::CreateManager(TSubclassOf<UQPT_ManagerBase>
ManagerClass)
{
	UQPT_ManagerBase* NewManager =
NewObject<UQPT_ManagerBase>(GetContainer(), ManagerClass);
	if (!NewManager) return;
	NewManager->RegisterComponent();
	Managers.Add(NewManager);
	NewManager->OnCreated(this);
	OnCreateManager(NewManager);
	OnSetManagerParameters(NewManager);
	NewManager->OnInitialized();
	OnInitializeManager(NewManager);
	PostInitializedManager(NewManager);
}
```

Programski kod 6. Funkcija CreateManager() u klasi GameRule.

Iako sve četiri klase imaju istog roditelja, pa su se te četiri funkcije mogle spojiti u jednu, korisniku je preglednije kada može pozivati zasebne funkcije za svaku klasu.

4.2.1. Template

Template sadrži praznu OnInitialized() funkciju. U njoj korisnik sam dodaje željeni kod. Template također označuje sve instance klase koje je dodao na listu za inicijalizaciju, stavljajući vrijednost bool varijable bCreatedByTemplate na istinu. Tako korisnik može razlikovati između klasa koje je stvorio GameRule, i koje su naknadno dodane putem Template-a. Template također sadrži slične funkcije kao i GameRule, tj. SetParameters i PostInitialized, koji dopuštaju klasi da mijenja parametre drugih klasa. One se odvijaju prije nego što GameRule pozove SetParameters i PostInitialized, tako da korisnik može promijenit vrijednosti i u GameRule klasi.

4.2.2. CommonRule

CommonRule nema nikakav kod koji se odvija prilikom inicijalizacije, pošto se najčešće koristi nakon inicijalizacije. Ukoliko se koristi tijekom inicijalizacije, najčešće Template instanca poziva potrebne funkcije.

4.2.3. Spawner

Spawner prilikom inicijalizacije pozivanja OnInitialized() funkcije gleda vrijednost bool varijable bSpawnOnInitialization, te ako je lažna, inicijalizacija završava. Ukoliko je točna, poziva se funkcija SpawnQPTActors().

CalculateSpawnTransforms() je funkcija u kojoj korisnik dodaje kod koji nadopunjuje niz SpawnTransforms, koji se koristi kako bi se kreirali QPT_Actor-i. Prilikom svakog pozivanja te funkcije, niz se prazni prije nego se pokreće kod kojeg je korisnik dodao, kako prijašnje vrijednosti ne bi utjecale na nove vrijednosti. Ukoliko je bool varijabla bUseContainerTransform istinita, tada se vrijednost transformacije mijenja kako bi bila relativna transformaciji Container-a. Zatim se kreira QPT_Actor sa vrijednošću iz niza, te se sprema. Novi QPT_Actor poziva OnCreated(), te prima Spawne-ra kao argument. Nakon toga Spawner javlja da je QPT_Actor kreiran pozivajući QPT_ActorSpawned(), te zatim GameRule poziva OnQPT_ActorCreated() kako bi javio da je kreiran. Zatim se svi kreirani QPT_Actor-i vraćaju. Programski kod 7 prikazuje funkciju SpawnQPTActors().

```
TArray<AQPT_ActorBase*> UQPT_SpawnerBase::SpawnQPTActors()
{
	CalculateSpawnTransform();
	UQPT_GameRuleBase* CurrentGameRule = GetGameRule();
	if (!CurrentGameRule) return TArray<AQPT_ActorBase*>();
```

```
	UWorld* world = CurrentGameRule->GetWorld();
	if (!world) return TArray<AQPT_ActorBase*>();

	TArray<AQPT_ActorBase*> NewActors;

	for (FTransform CurrentTransform : SpawnTransforms)
	{
		if (bUseContainerTransform) { CurrentTransform =
OffsetTransform(GetContainerTransform(), CurrentTransform); }

		AQPT_ActorBase* NewQPTActor = world-
>SpawnActor<AQPT_ActorBase>(ClassToSpawn, CurrentTransform);
		if (NewQPTActor)
		{
			QPTActors.Add(NewQPTActor);
			NewQPTActor->OnCreated(this);
			QPTActorSpawned(NewQPTActor);
			CurrentGameRule->OnQPTActorCreated(NewQPTActor);
			NewActors.Add(NewQPTActor);
		}

	}
	return NewActors;
}
```

Programski kod 7. Funkcija SpawnQPTActors() u klasi Spawner.

4.2.4. QPT_Actor

OnCreated() funkcija QPT_Actor-a služi za spremanje reference Spawner-a koji ga je kreirao. U nju korisnik može dodati dodatni kod koji se pokreće prilikom inicijalizacije.

4.2.5. Manager

Poput Spawner-a, I manager ima bool varijablu bCreateOnInitialization, koja ako je točna, poziva funkciju CreateComponents.

CreateComponents() prolazi kroz sav niz QPT_Actor-a, te za svakog stvori jednu instancu Component-e te ju dodaje tim QPT_Actor-ima. Zatim Component poziva OnCreated(), te prima reference Managera i QPT_Actor-a kojem pripada kao argumente, te vrati sve stvorene Component-e. Sljedeći programski kod prikazuje funkciju CreateComponents() u klasi Manager.

```
TArray<UQPT_ComponentBase*>
UQPT_ManagerBase::CreateComponents(TArray<AQPT_ActorBase*> QPTActors,
TSubclassOf<UQPT_ComponentBase> LocalClassToCreate)
{
	TArray<UQPT_ComponentBase*> CreatedComponents;
	for (AQPT_ActorBase* CurrentActor : QPTActors)
```

```
	{
		UQPT_ComponentBase* NewComponent =
NewObject<UQPT_ComponentBase>(CurrentActor, LocalClassToCreate);
		if (NewComponent)
		{
			QPT_Components.Add(NewComponent);
			NewComponent->OnCreated(this, CurrentActor);
		}
	}
	return CreatedComponents;
}
```

Programski kod 8. Funkcija CreateComponents() u klasi Manager.

4.2.6. Component

Na poziv OnCreated() funkcije, Component spremi reference na Managera koji ga je stvorio i na QPT_Actora kojemu pripada. Korisnik tu može dodati kod za koji želi da se pokrene prilikom kreiranja Component-e.

4.3. Blueprint klase

Pošto su glavne funkcionalnosti osnovnih klasa objašnjene, te je pojašnjen i lanac inicijalizacije, možemo pobliže objašnjavati rad određenih Blueprint klasa potrebnih za kreiranje demo igre.

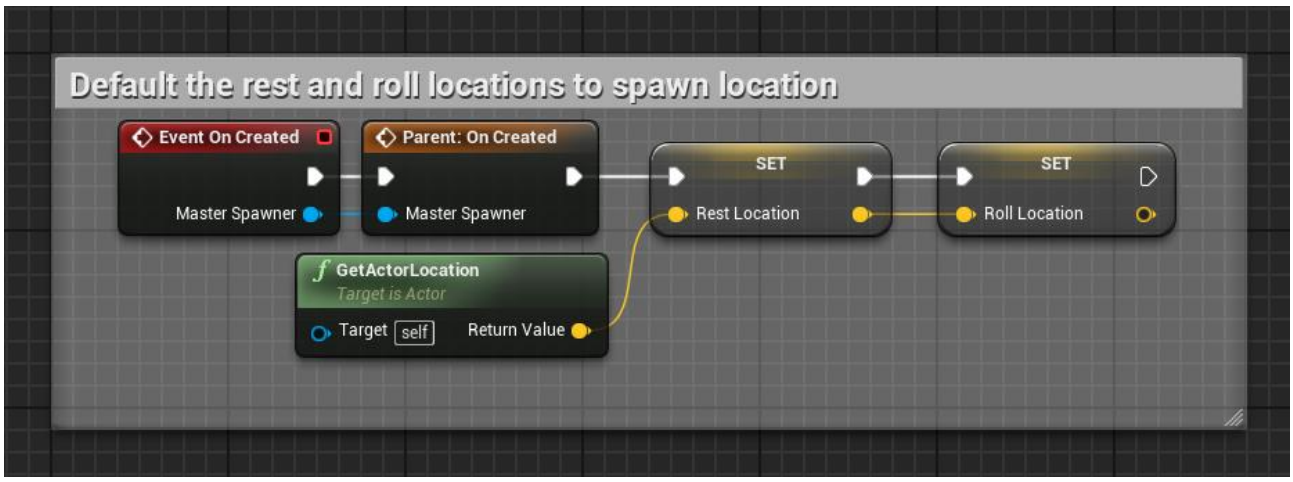
4.3.1. QPT_Actor-i

Prvo ćemo pogledati sve QPT_Actor-e koji su nam potrebni za realizaciju zadane demo igre. Potrebne su kockice, potrebno je dugme, te neka vrsta kutije koja će držati kockice u sebi kako se ne bi previše udaljile.

4.3.1.1. Dice

Dice je klasa koja nasljeđuje od QPT_Actor klase te predstavlja kockicu. Njena glavna funkcija je Roll, koja „baca“ kockicu, te javlja koji broj je na vrhu nakon što se zaustavi. No sadrži i mnoge druge parametre koji automatiziraju određene procese, poput bacanja kockice s određenog mjesta (lokacija spremljena u FVector varijabli RollLocation), vraćanje kockice na određeno mjesto nakon što se zaustavi (lokacija spremljena u FVector varijabli RestLocation) i slično.

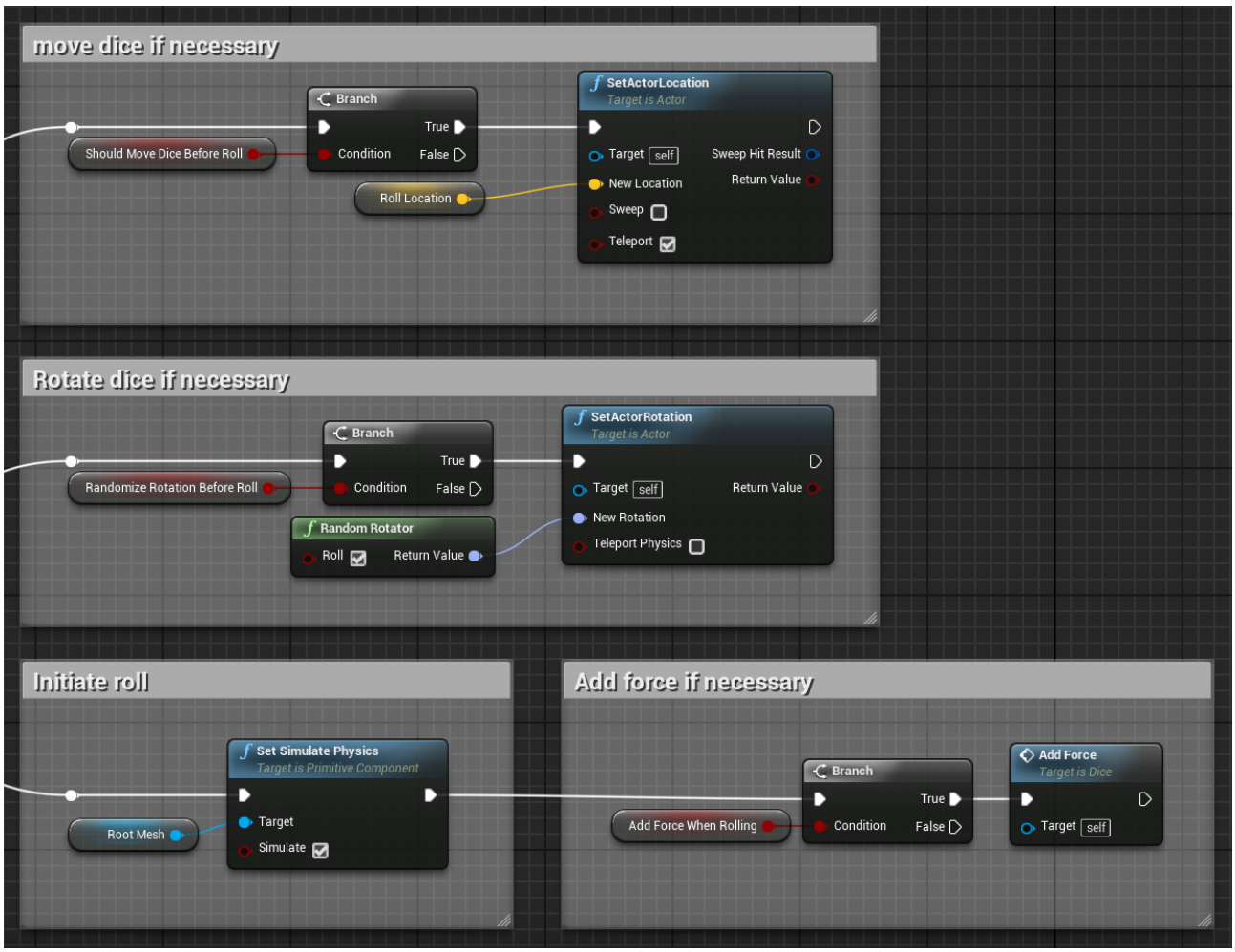
Kada se kockica kreira, automatski se dohvaća njena lokacija, koja se sprema u varijable RestLocation i RollLocation. Na sljedećoj slici je prikazan kod funkcije OnCreated() u Blueprint-u.



Slika 3. OnCreated() funkcija u klasi Dice

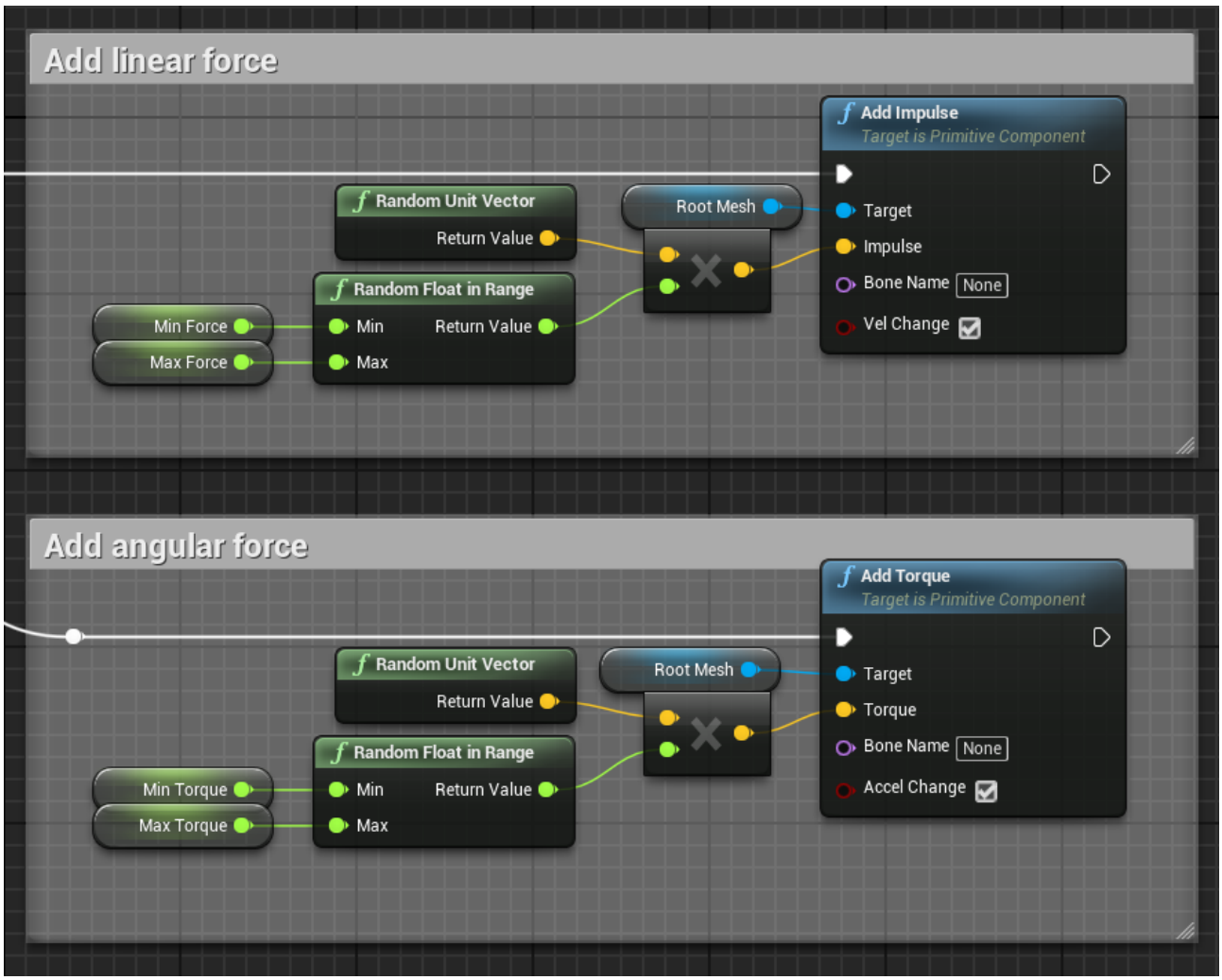
RestLocation označava lokaciju na koju se kockica vraća nakon što se zaustavi, a RollLocation označava lokaciju s koje se kockica baca. Korisnik može mijenjati te vrijednosti u bilo kojem trenutku, no na ovaj način postoji konzistentna kontrola bez dodatnih izmjena.

Funkcija Roll() započinje tako da prvo pogleda vrijednost bool varijable ShouldMoveDiceBeforeRoll, koja, ako je istinita, premješta kockicu na vrijednost varijable RollLocation. Zatim pregledava vrijednost bool varijable RandomizeRotationBeforeRoll, te ako je istinita, nasumično ju zarotira. Nakon toga započinje simulacija fizike, nakon čega se pregledava vrijednost bool varijable AddForceWhenRolling, koja ukoliko je istinita poziva funkciju AddForce(). Na sljedećoj slici je prikazana funkcija Roll() u Blueprint klasi Dice.



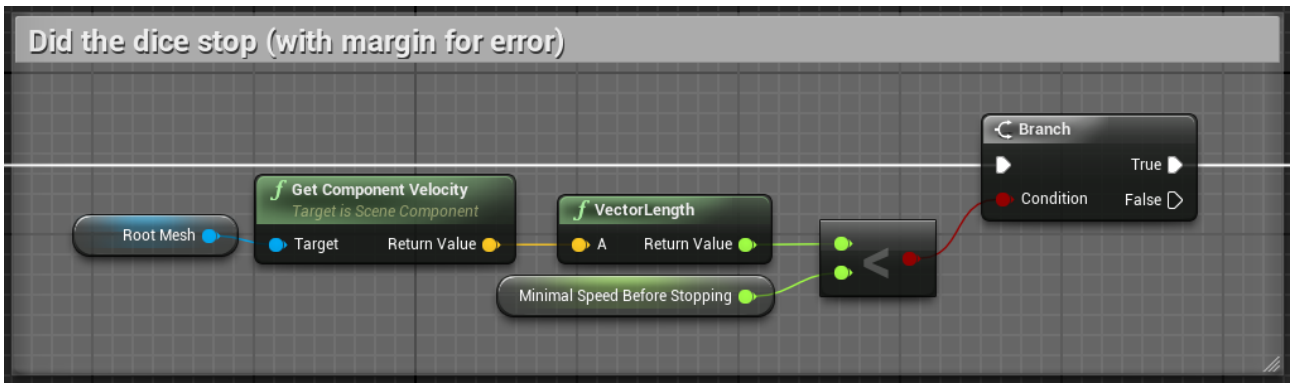
Slika 4. Roll() funkcija u klasi Dice.

Funkcija AddForce() čita vrijednosti float varijabli MinForce i MaxForce, te uzima nasumičan broj koji se nalazi između njih. Zatim dohvaća vektor duljine 1, nasumičnog smjera, te ga množi s dobivenim brojem kako bi se dobio smjer i intenzitet sile koja utječe na kockicu. Na isti način se kockica zarotira, no ovaj put koristeći varijable MinTorque i MaxTorque. Na slici 5 je prikazan kod funkcije AddForce() u Blueprint klasi Dice.



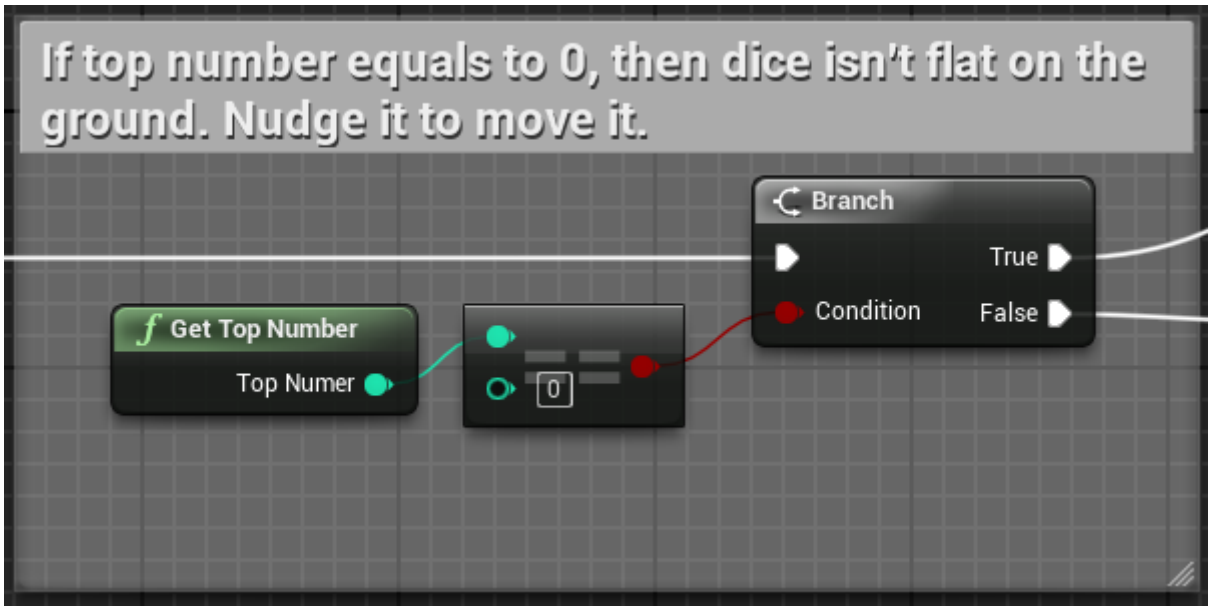
Slika 5. AddForce() funkcija u klasi Dice.

Zatim se gleda da li se kockica zaustavila, tako da se u Tick() funkciji konstantno uzima njena brzina, te se uspoređuje s vrijednošću float varijable MinimalSpeedBeforeStopping (početna vrijednost je 0.05 cm/s). Ukoliko je brzina manja, možemo smatrati da se kockica zaustavila, u protivnom kockica se još kreće, te nastavljamo s usporedbom dok se ne zaustavi. Taj dio koda je prikazan na sljedećoj slici.



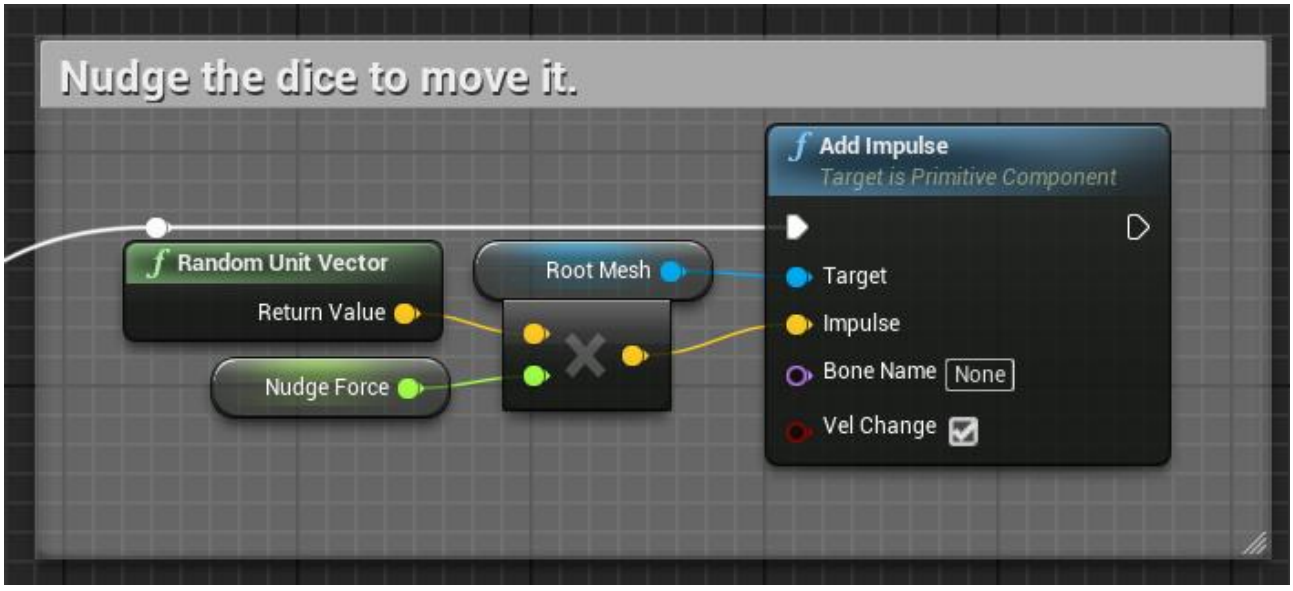
Slika 6. Provjera brzine u Tick() funkciji unutar klase Dice.

Ukoliko se kockica zaustavila, poziva se funkcija GetTopNumber(), koja vraća broj na vrhu kockice. Ukoliko je kockica nakošena, te je pod kutom većim od vrijednosti float varijable AllowedOffset (početna vrijednost je 20 stupnjeva), funkcija vraća nulu. Na sljedećoj slici vidimo dio koda koji provjerava da li je kockica nakošena.



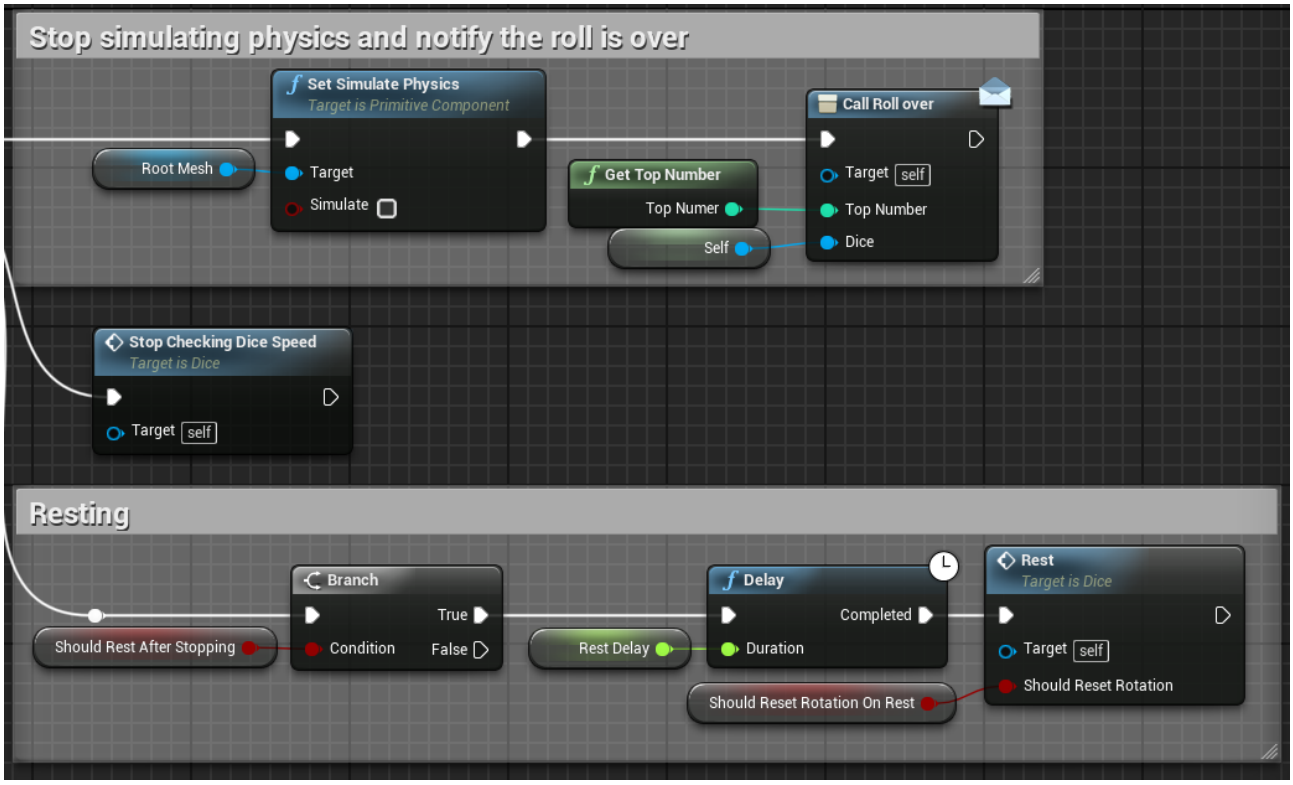
Slika 7. Provjera nakošenosti kockice u Tick() funkciji unutar klase Dice.

Ukoliko GetTopNumber() funkcija vraća nulu, tada je potrebno pomaknuti kockicu (Nudge). To se čini na isti način kao i u AddForce() funkciji, no ovaj put se intenzitet sile uzima iz vrijednosti float varijable NudgeForce, s početnom vrijednošću od 200, te se ponovo pregledava brzina kockice. Na sljedećoj slici se nalazi dio koda koji pomiče kockicu ukoliko je ona nakošena.



Slika 8. Dio koda u Tick() funkciji unutar Dice klase koji ju pomiče ukoliko je ona nakošena.

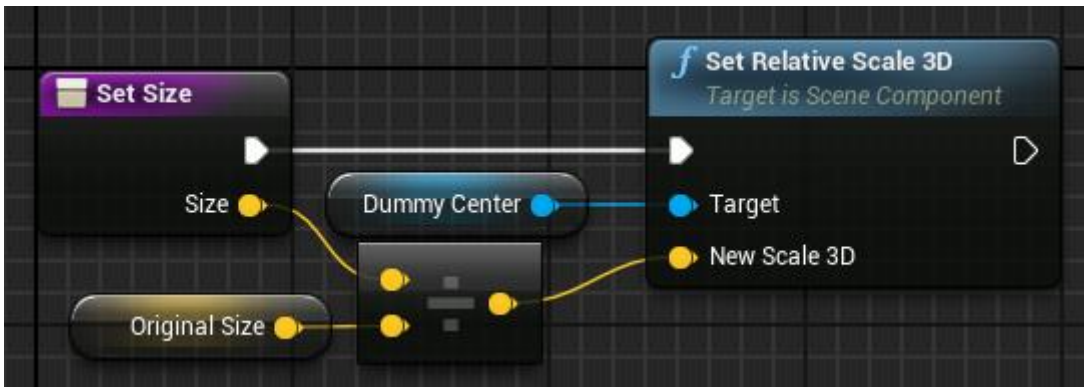
Ukoliko funkcija GetTopNumber() ne vrati nulu, tada se kockica uspješno zaustavila, te prestaje simulacija fizike. Dobiveni broj se šalje preko Event Dispatcher-a, te se poziva funkcija StopCheckingDiceSpeed() koja prekida pregled brzine kockice. Nakon toga, ukoliko je vrijednost bool varijable ShouldRestAfterStopping istinita, tada se pričeka vrijeme specificirano vrijednošću float varijable RestDelay (s početnom vrijednošću od jedne sekunde), kako bi igrač vidio koji broj je na vrhu kockice. Tada se poziva funkcija Rest(), koja kao argument prima bool varijablu ShouldResetRotationOnRest, te ukoliko je točna, rotira kockicu na način da je dobiveni broj na vrhu i da ne bude nakošena. Tada ju premješta na lokaciju spremljenu u varijabli RestLocation, te preko Event Dispatcher-a šalje notifikaciju da se vratila na zadanu lokaciju. Ta funkcionalnost je prikazana na sljedećoj slici.



Slika 9. Dio koda u klasi Dice koji se odvija nakon što se ona uspješno zaustavila.

4.3.1.2. DiceBox

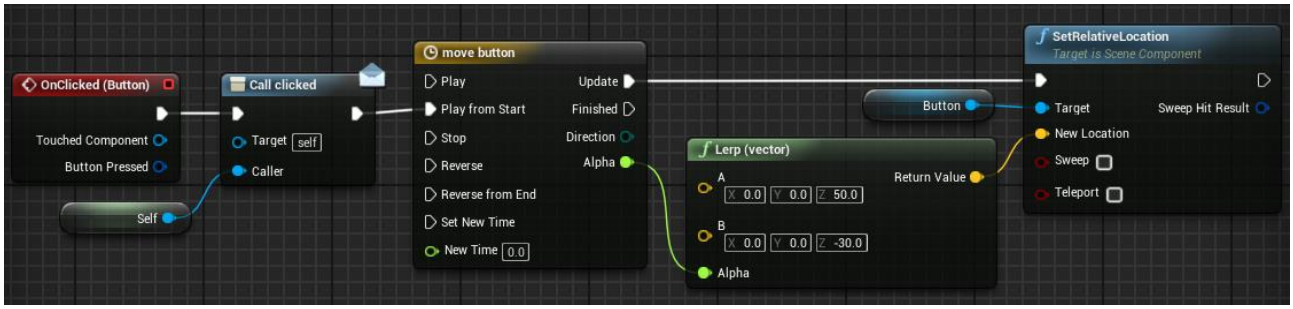
DiceBox je klasa koja nasljeđuje od QPT_Actor klase. Njena svrha je da zadržava kockice u sebi kako se ne bi odvojile od igre. Na sljedećoj slici je prikazana funkcija SetSize(), koja prima argument tipa Vector, te pomoću nje mijenja svoju veličinu.



Slika 10. Funkcija SetSize() u klasi DiceBox.

4.3.1.3. Button

Button je klasa koja nasljeđuje od QPT_Actor klase. Ona predstavlja dugme koje se može pritisnuti, te na sebi sadrži tekst koji se može promijeniti pozivajući SetText() funkciju s argumentom tipa FText. Kao što je prikazano na sljedećoj slici, ukoliko se mišem klikne na dugme, ono šalje notifikaciju preko Event Dispatcher-a. Također se pomiče, kako bi igrač imao vizualnu informaciju koja govori da je uspješno pritisnuto.



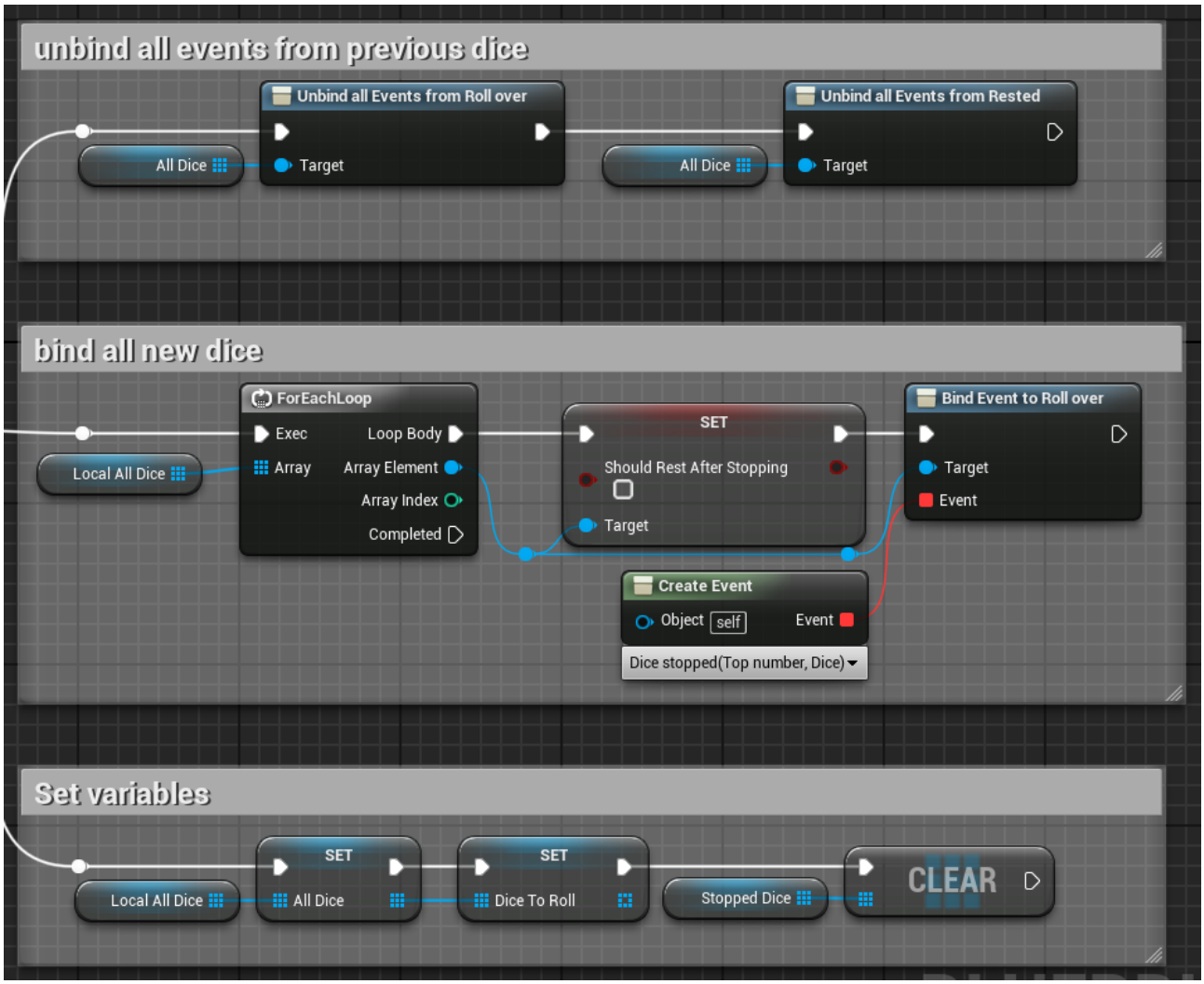
Slika 11. Kod za slanje notifikacije i pomicanje dugmeta u klasi Button.

4.3.2. Dice Controller

Kako bi olakšali kontrolu nad više kockica istovremeno, koristit ćemo novu klasu, Dice Controller. Ona nasljeđuje od klase CommonRule, te sadrži dva niza koje sadrže reference na kockice, niz AllDice i niz DiceToRoll. AllDice niz sadržava sve instance na koje Dice Controller može utjecati, dok DiceToRoll sadrži reference samo na one instance na koje korisnik želi pozvati funkciju Roll(). To znači da iako Dice Controller sadrži reference na sve instance klase Dice, ne mora ih sve koristiti. To je korisno kod igara u kojima igrač bira koliko kockica želi bacit.

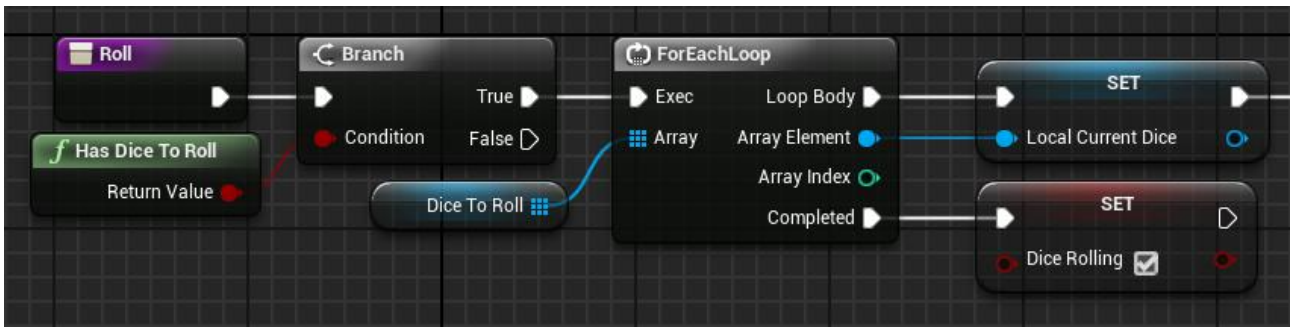
Funkcija SetAllDice() prima niz instanci klase Dice kao argument, koje zatim sprema u lokalnu varijablu LocalAllDice, svaku instancu spremljenu u nizu AllDice Unbind-a (prekida

vezu Event Dispatcher-a), kako neka kockica koja nije u LocalAllDice nizu ne bi pozvala funkciju unutar Dice Controller-a. Tada svakoj instanci tog niza mijenja vrijednost ShouldRestAfterStopping na laž, kako bi Dice Conroller omogučio da se sve kokice istodobno vrate na RestLocation lokaciju. Također se notifikacija RollOver Bind-a na funkciju DiceStopped(), te se popunjava niz AllDice i DiceToRoll. Na slici 12 je prikazan dio koda iz funkcije SetAllDice().

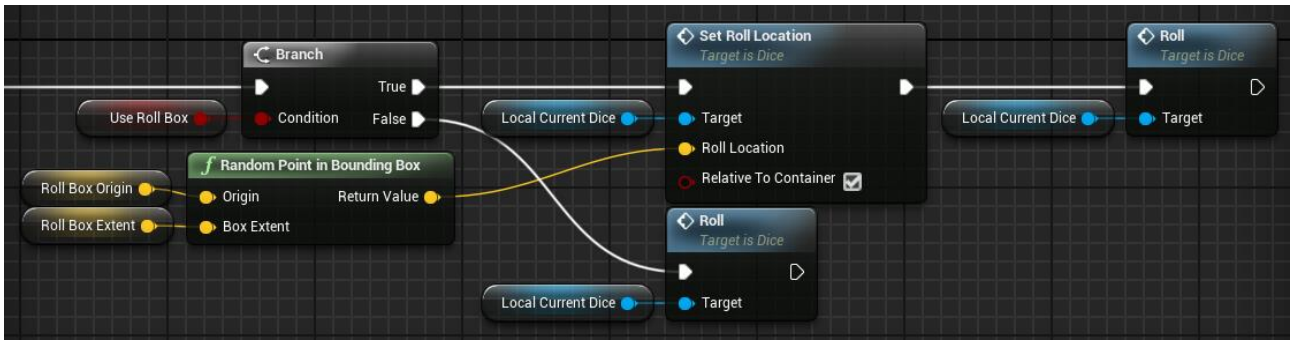


Slika 12. Dio koda funkcije SetAllDice() u klasi Dice Controller.

Funkcija Roll() poziva funkciju HasDiceToRoll(), koja vraća istinu ukoliko niz DiceToRoll nije prazan, te ako je istinit, gleda da li je vrijednost bool varijable UseRollBox jednaka istini, te ako je, uzima nasumičnu lokaciju iz kocke zadane njenom lokacijom i veličinom, te tu točku postavlja kao RollLocation te kocke. Nakon toga kocka poziva funkciju Roll(). Na slikama 13 i 14 je prikazana funkcija Roll() Dice Controller klase.

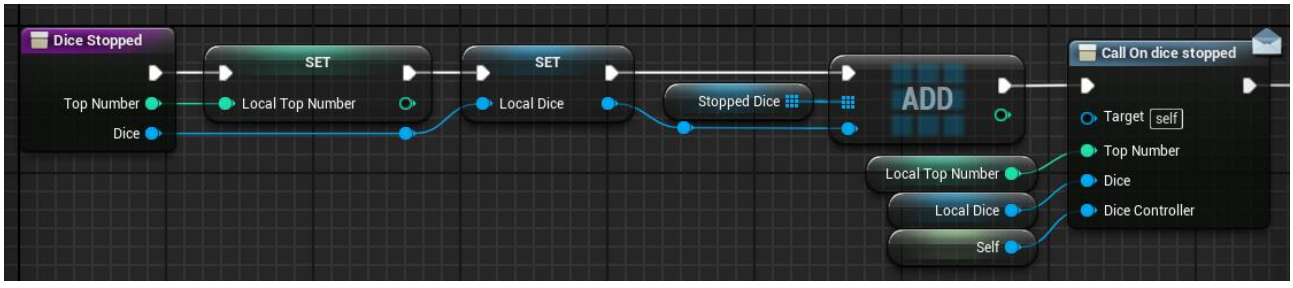


Slika 13. Prvi dio funkcije Roll() u klasi Dice Controller.



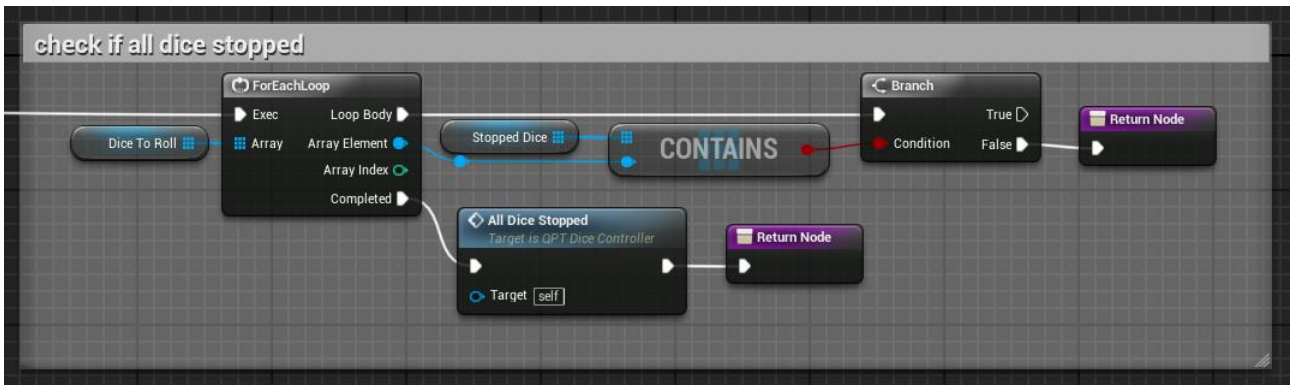
Slika 14. Drugi dio funkcije Roll() u klasi Dice Controller.

Kada se kockica zaustavi, tada ona preko Event Dispatcher-a poziva funkciju DiceStopped() u klasi Dice Controller. Ona Dodaje tu instancu u niz s nazivom StoppedDice, koji sadrži kockice koje su se zaustavile, te tada prosljeđuje notifikaciju da je ona stala. Taj dio koda se vidi na slici 15.



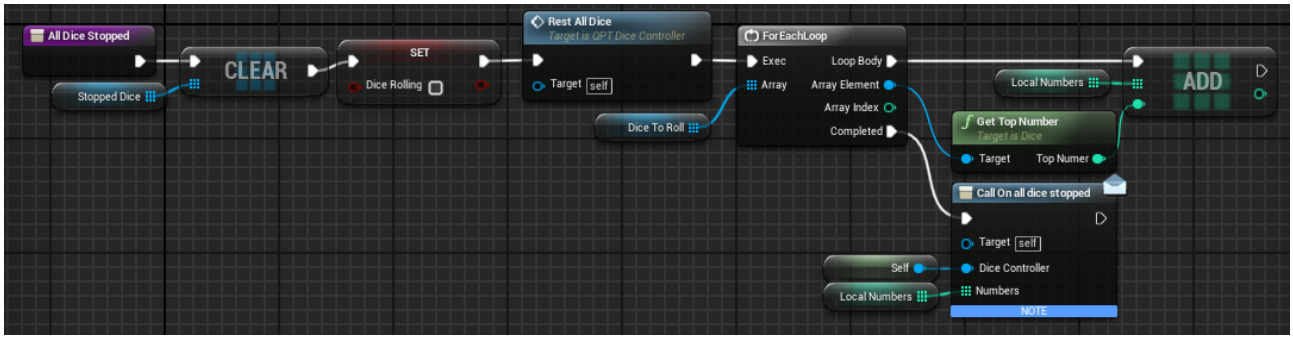
Slika 15. Dio koda u funkciji DiceStopped(), unutar klase Dice Controller, koji šalje notifikaciju OnDiceStopped.

Nakon toga provjerava da li postoji još kockica koje se moraju zaustavit, te ukoliko ih nema, znači da su sve zaustavljene, te stoga poziva funkciju AllDiceStopped(), kao što je prikazano na sljedećoj slici.



Slika 16. Dio koda u funkciji DiceStopped(), unutar klase Dice Controller, koji provjerava da li su se sve kockice zaustavile.

Funkcija AllDiceStopped() prazni niz StoppedDice, te označava bool varijablu DiceRolling kao laž, kako bi korisnik mogao provjerit da li se kockice još vrte. Tada poziva funkciju RestAllDice(), te preko Event Dispatcher-a šalje notifikaciju da su se sve kockice zaustavile, te također šalje i sumu njihovih brojeva. Cijela AllDiceStopped() funkcija je prikazana na sljedećoj slici.



Slika 17. Funkcija AllDiceStopped() u klasi Dice Controller.

Funkcija RestAllDice() čeka određeno vrijeme (specificirano vrijednošću float varijable RestDelay s početnom vrijednošću od jedne sekunde) te zatim svim kockicama poziva funkciju Rest. Tada šalje notifikaciju preko Event Dispatcher-a koja javlja da su se sve kockice vratile na lokaciju definiranu njihovom varijablom RestLocation. Na sljedećoj slici je prikazana funkcija RestAllDice().

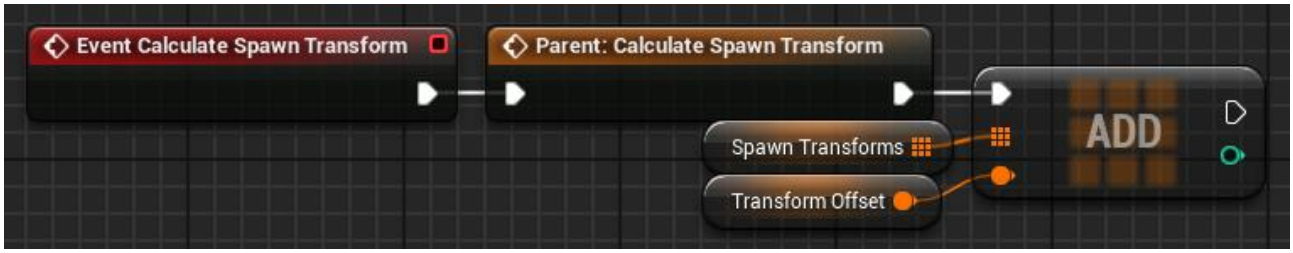
Slika 18. Funkcija RestAllDice() u klasi Dice Controller.

4.3.3. Spawner-i

Potrebno je kreirati instance klasa Dice, DiceBox i Button. Pošto nam je potrebna samo jedna instanca Button-a i DiceBox-a, za njih možemo koristiti Single Spawner klasu. Pošto su nam potrebne 4 kockice, tada ćemo koristiti Row Spawner klasu.

4.3.3.1. Single Spawner

Single Spawner je jednostavna klasa bazirana na Spawner-u, koja kreira samo jednu instancu QPT_Actor klase. Zbog toga ima vrlo jednostavnu CalculateSpawnTransform() funkciju, u kojoj niz SpawnTransforms dobiva samo vrijednost FTransform varijable TransformOffset, kao što je prikazano na sljedećoj slici.



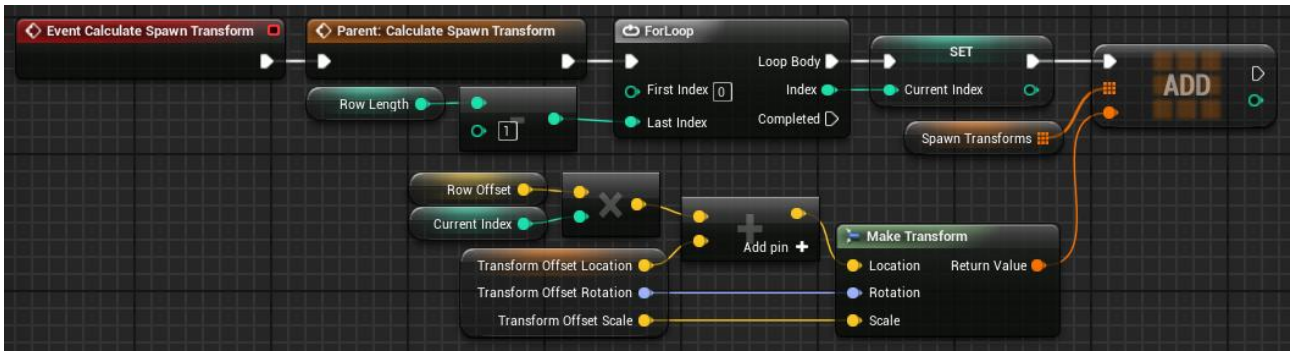
Slika 19. Funkcija CalculateSpawnTransform() u klasi Single Spawner.

Kako bi se olakšao rad korisniku, postoji i funkcija SpawnSingle(), prikazana na sljedećoj slici, koja kao argumente prima transformaciju i klasu koju treba kreirati, te vraća njenu referencu.

Slika 20. Funkcija SpawnSingle() unutar klase Single Spawner.

4.3.3.2. Row Spawner

Row Spawner je klasa koja kreira QPT_Actor-e na način da ih poslaže po zamišljenoj liniji. Koristi varijablu TransformOffset kao transformaciju prve instance kreirane klase, dok je svaka sljedeća udaljena za vrijednost FVector varijable RowOffset od prethodne. Integer varijabla RowLength diktira broj instanci koje mora kreirati. Na slici 21 je prikazana funkcija CalculateSpawnTransform() klase Row Spawner.

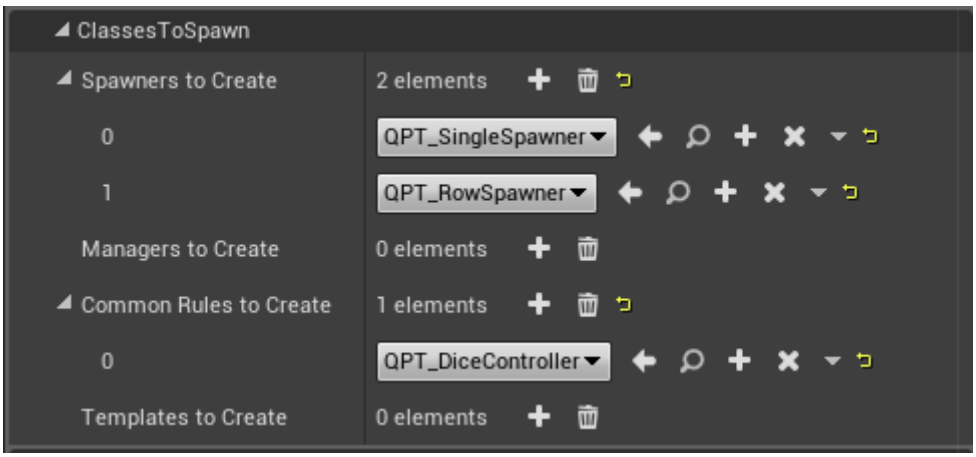


Slika 21. Funkcija CalculateSpawnTransform() unutar klase Row Spawner.

4.3.4. MultiDiceDemo

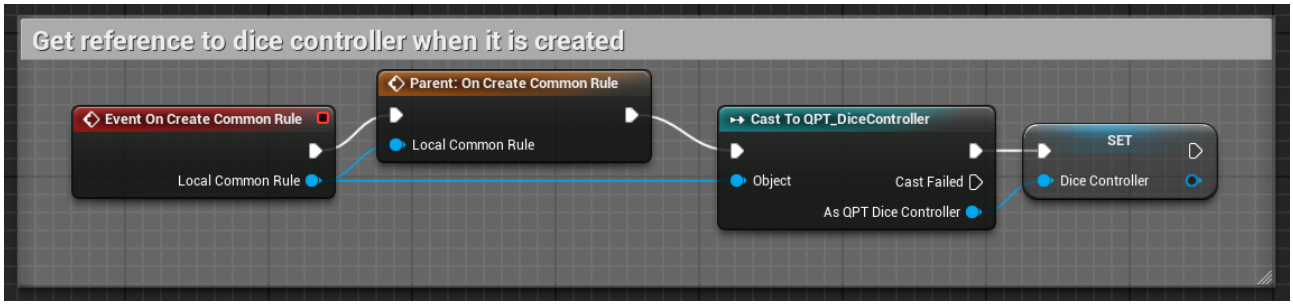
MultiDiceDemo je klasa kreirana u Blueprint sustavu, te nasljeđuje od GameRule klase. Ona sadrži pravila zadane Demo igre. Sve instance klase Dice se nalaze zatvorene unutar klase DiceBox, kako bi ostale na željenom mjestu. Kockice se bacaju pritiskom na dugme (klasa Button), na kojem, nakon što se kockice zaustave, ispiše njihova suma. Zatim se kockice vrate u prvotnu poziciju, te se mogu ponovo bacati pritiskom na dugme.

Pošto je potrebno kreirat jednu instancu klase DiceBox i jednu instancu klase Button, u listu za inicijalizaciju se pod Spawner-e dodaje Single Spawner. Pošto se kreira više kockica, točnije 4, tada možemo koristiti klasu Row Spawner. Za kontrolu više instanci kockica nam je potrebna jedna klasa Dice Controller, te ju dodajemo na listu za inicijalizaciju. Na sljedećoj slici je prikazana cijela lista za inicijalizaciju klase MultiDiceDemo.



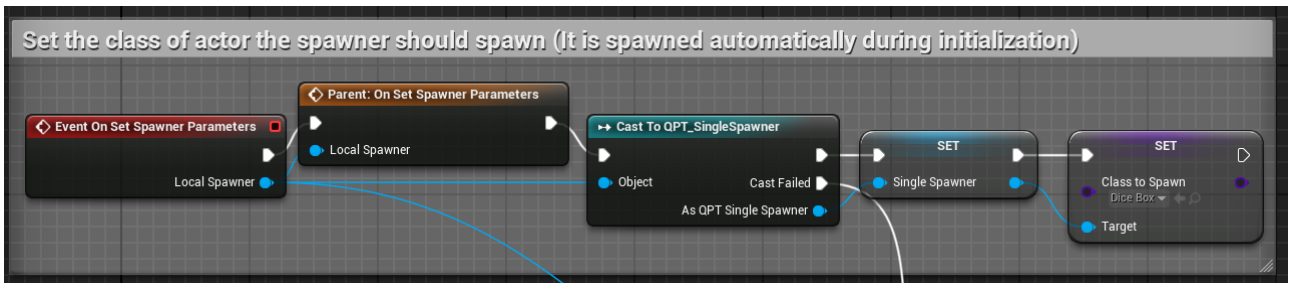
Slika 22. Lista za inicijalizaciju u klasi MultiDiceDemo.

Potrebno je dohvatiti sve reference kreiranih instanci, počevši s instancom DiceController. To činimo tako da override-amo funkciju OnCreateCommonRule(). Prvo pozivamo OnCreateCommonRule() funkciju roditelja, te zatim uzimamo referencu CommonRule instance te ju Cast-amo kako bi bili sigurni da se radi upravo o Dice Controller klasi, kako bi smo ju spremili. Na sljedećoj slici je prikazano spremanje Dice Controller instance u funkciji OnCreateCommonRule().



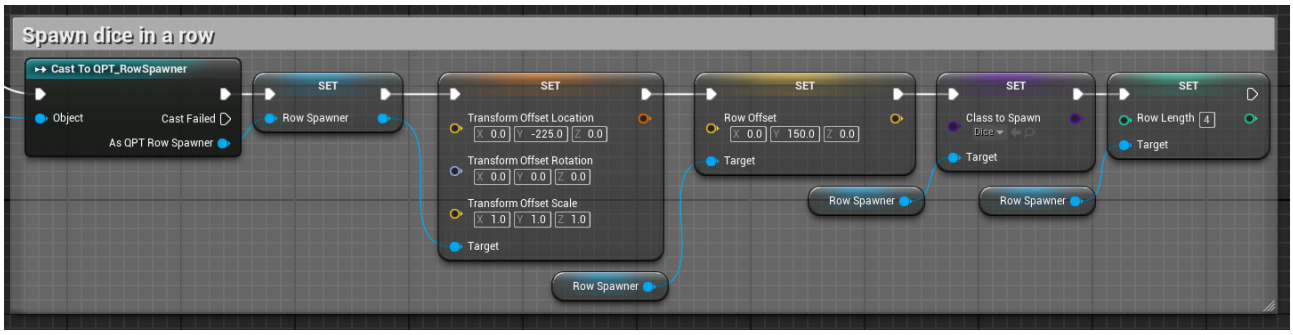
Slika 23. Funkcija OnCreateCommonRule() u klasi MultiDiceDemo.

Zatim moramo dobit reference na instance klase Spawner. No pošto im odmah moramo promijenit parametre, što činimo u funkciji SetSpawnerParameters(), možemo od tamo i dohvatit reference. Prvo pozivamo SetSpawnerParameters() funkciju roditelja, te instancu Cast-amo na SingleSpawner-a. Ukoliko ta instanca jest SingleSpawner, spremamo ju, te joj dajemo klasu DiceBox za kreiranje. Ukoliko ne damo nikakvu transformaciju, kreirat će se na lokaciji Container-a, što je u redu. Na slici 24 je prikazan dio funkcije koji se pokreće ukoliko funkcija OnSetSpawnerParameters() ima referencu na Single Spawner klasu kao argument.



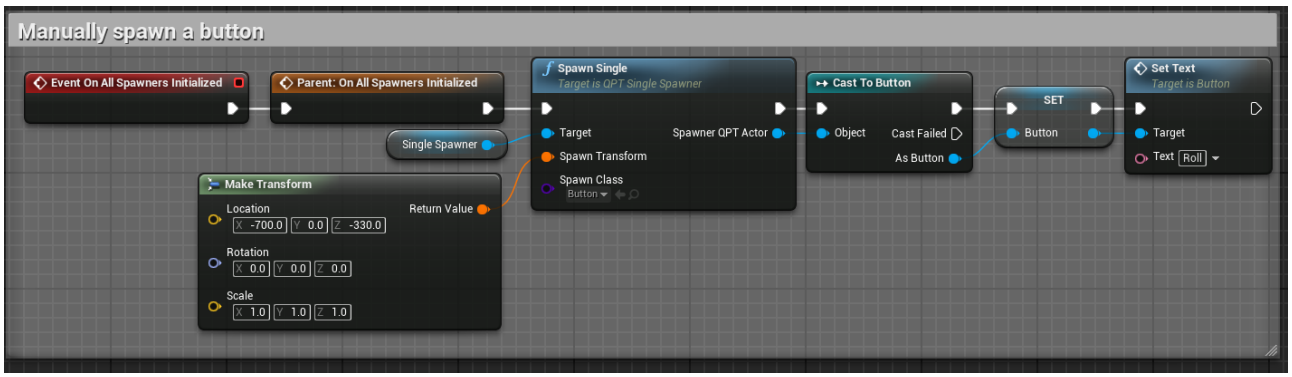
Slika 24. Mijenjanje parametara Single Spawner instance u funkciji OnSetSpawnerParameters(), unutar klase MultiDiceDemo.

Ukoliko Cast nije uspio, Castamo na Row Spawner-a, te ga spremamo. Zatim preko TransformOffset varijable namjestimo da se prva instanca klase Dice kreira 225 centimetara u smjeru negativne Y osi od lokacije Container-a. Varijablom RowOffset dajemo do znanja da svaka sljedeća instanca će biti udaljena za 150 centimetara po Y osi od prethodne. Varijabla ClassToSpawn određuje koju klasu će Spawner kreirat, u ovom slučaju klasu Dice. I na kraju integer varijabla RowLength koja određuje koliko instanci će se kreirati, u ovom slučaju 4. Na sljedećoj slici je prikazan dio funkcije koji se pokreće ukoliko je argument funkcije OnSetSpawnerParameters() instanca klase Row Spawner.



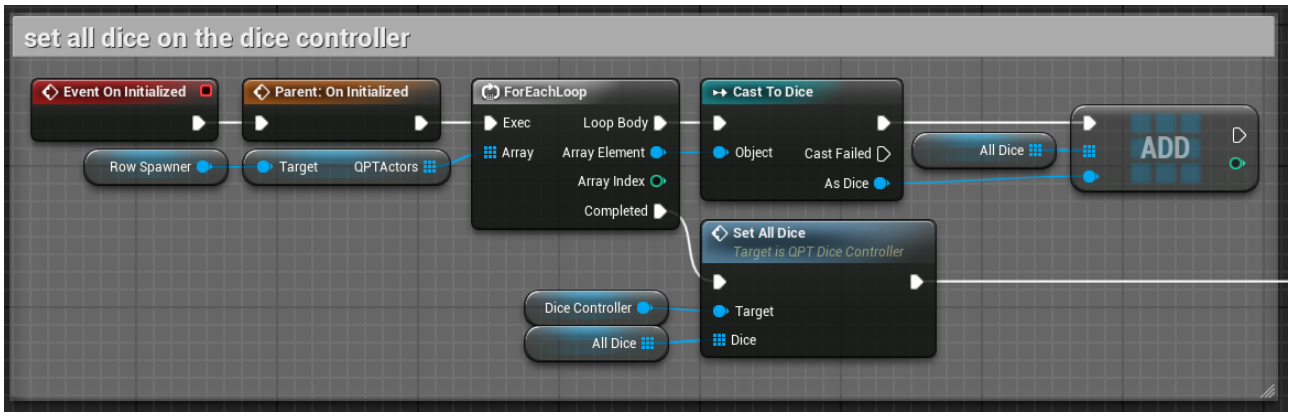
Slika 25. Mijenjanje parametara Row Spawner instance u funkciji OnSetSpawnerParameters(), unutar klase MultiDiceDemo.

Sljedeće, potrebno je kreirati dugme. Za to koristimo Single Spawner instancu. U funkciji OnAllSpawnersInitialized() znamo da su svi Spawner-i inicijalizirani, stoga ih možemo koristiti. Prvo pozivamo OnAllSpawnersInitialized() funkciju roditelja, te pozivamo funkciju SpawnSingle() iz instance Single Spawner kako bi kreirali instancu klase Button. Kao argumente dajemo transformaciju s lokacijom -700.0, 0.0, -330.0, te kao klasu za kreiranje postavljamo Button. Dobivenu referencu spremamo, te joj mijenjamo tekst na riječ „Roll“ pomoću funkcije SetText(). Na Sljedećoj slici je prikazana funkcija OnAllSpawnersInitialized().



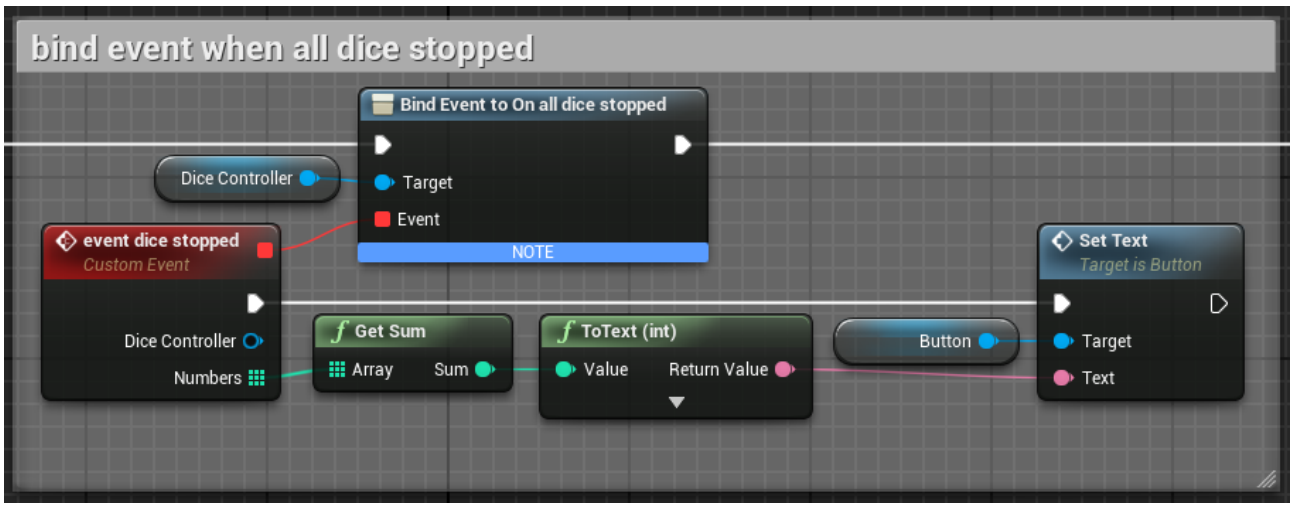
Slika 26. Funckija OnAllSpawnersInitialized() u klasi MultiDiceDemo.

Nakon što su se sve instance inicijalizirale, potrebno je dovršit inicijalizaciju demo igre. Za to koristimo funkciju OnInitialized(). Prvo pozivamo OnInitialized() funkciju roditelja, te zatim od Row Spawner reference uzimamo sve QPT_Actor-e koje je kreirao, stavimo ih u petlju, te ih spremamo u niz AllDice kao reference na klasu Dice. Nakon toga DiceController-u pozivamo funkciju SetAllDice(), i kao argument predajemo sve instance klase Dice. Taj dio funkcije je prikazan na sljedećoj slici.



Slika 27. Pozivanje funkcija SetAllDice() unutar funkcije OnInitialized() u klasi MultiDiceDemo.

Zatim, koristeći Event Dispatcher, svaki put kada Dice Controller javlja da su se sve kockice zaustavile, koristeći notifikaciju OnAllDiceStopped, pokrenut će se funkcija DiceStopped(), koja ispisuje sumu brojeva dobivenih iz kockica na dugme, kao što je prikazano na sljedećoj slici.



Slika 28. Ispisivanje sume kockica na dugme u funkciji DiceStopped() u klasi MultiDiceDemo.

Na kraju, još jednom pomoću Event Dispatcher-a Bind-amo notifikaciju Clicked, koja se šalje kada se mišem klikne na dugme, te ona poziva Roll() funkciju unutar klase Dice Controller, kao što je prikazano na sljedećoj slici.

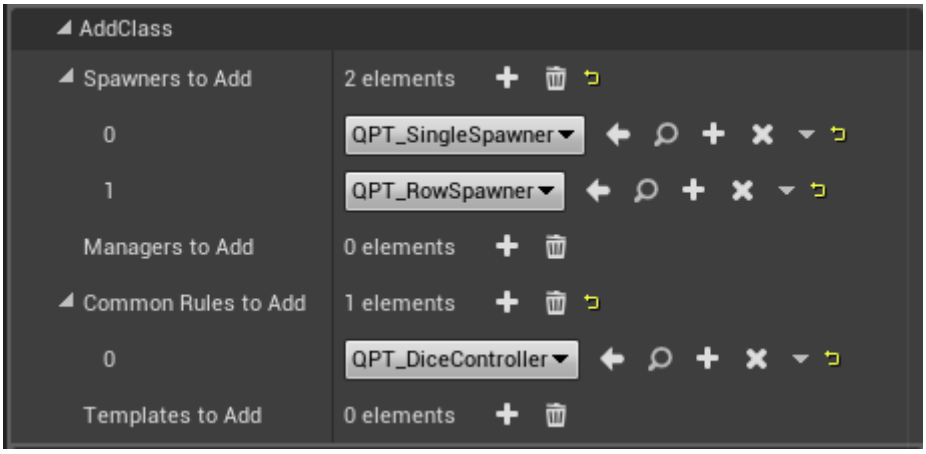
Slika 29. Pozivanje funkcije Roll() klase Dice Controller unutar klase MultiDiceDemo, kada igrač klikne mišem na dugme.

Time je dovršena demo igra. Pritiskom na dugme, bacaju se kockice. Kad se zaustave, vrate se na početno mjesto i na dugmetu se ispiše njihova suma. No za sve igre koje sadrže više kockica korisnik mora iznova koristiti isti kod, kako bi kreirao sve kockice,

predao ih Dice Controller instanci, te kreirao DiceBox instancu u kojem će se one nalazit. Kako bi se taj postupak automatizirao, može se koristiti Template.

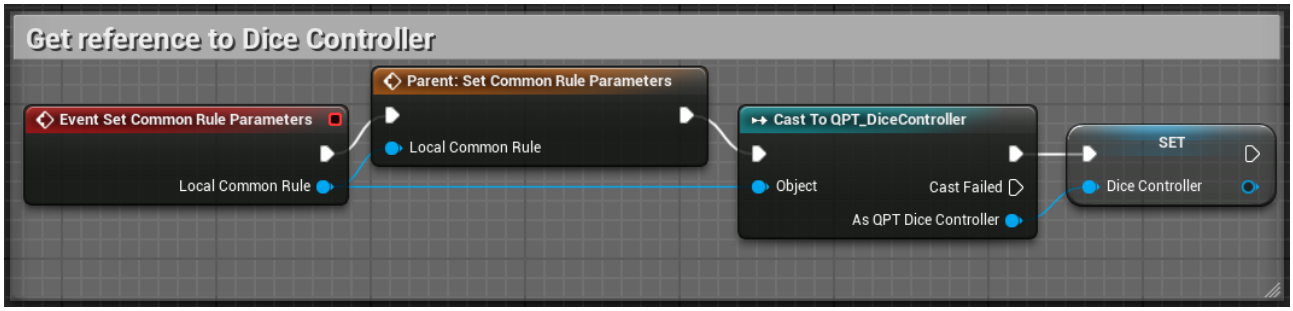
4.3.5. DiceWithBoxTemplate

DiceWithBoxTemplate nasljeđuje od klase Template, te služi za automatizaciju inicijalizacije igre, na način da kombinira često korištene klase zajedno, u ovom slučaju Dice, Dice Controller i DiceBox. Pošto ova klasa ima istu funkciju kao i MultiDiceDemo, kod će biti sličan. Samim time lista klase koje Template dodaje za inicijalizaciju sadrži jednu Single Spawner klasu, jednu Row Spawner klasu, te jednu Dice Controller klasu, kao što je prikazano na sljedećoj slici.



Slika 30. Lista klasa koje će se dodati listi za inicijalizaciju unutar klase DiceWithBoxTemplate.

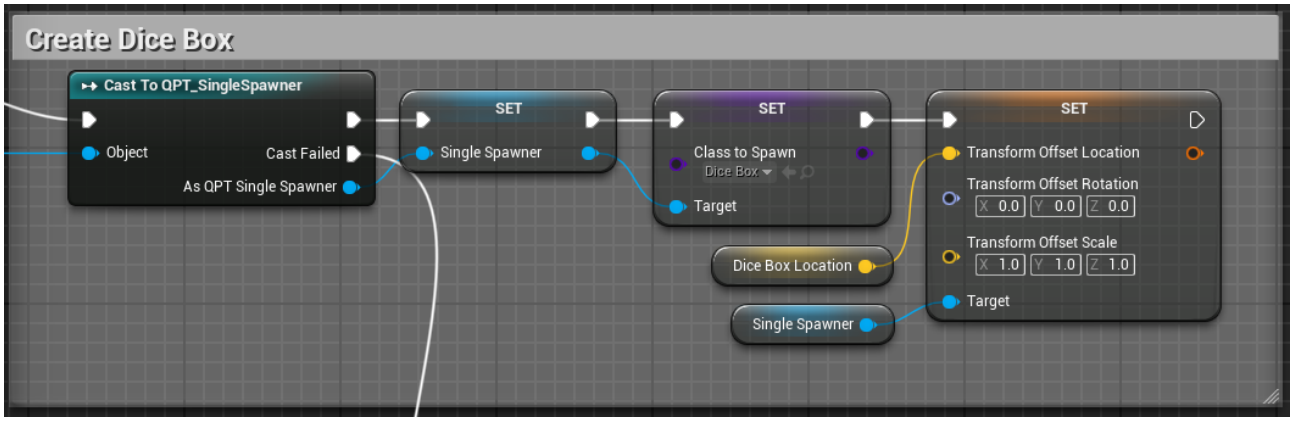
Prvo se sa SetCommonRuleParameters() funkcijom poziva SetCommonRuleParameters() funkcija roditelja, te se zatim sprema referenca Dice Controller-a. Funkcija SetCommonRuleParameters() je prikazana na sljedećoj slici.



Slika 31. Funckija SetCommonRuleParameters() u klasi DiceWithBoxTemplate.

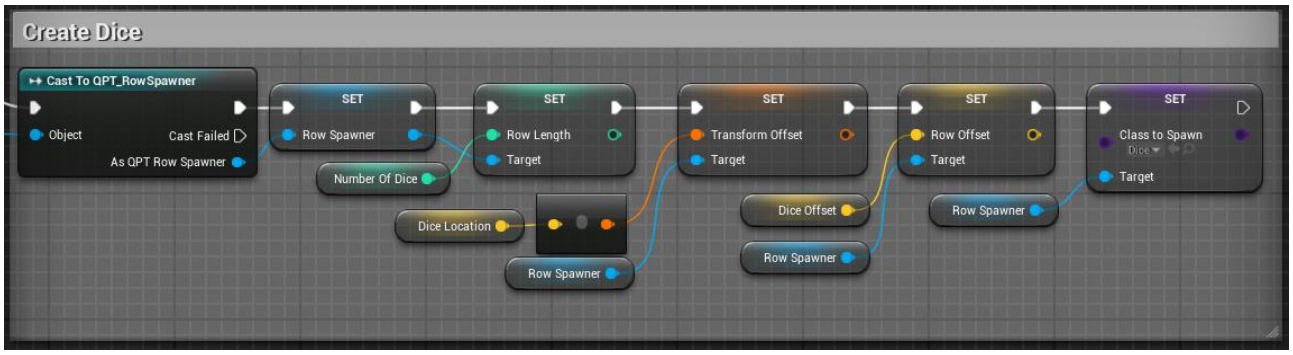
Zatim u funkciji SetSpawnerParameters(), nakon poziva funkcije roditelja, se sprema referenca instance Single Spawner, u kojoj se vrijednost varijable ClassToSpawn sprema

Dicebox, te se lokacija varijable TransformOffset mijenja na vrijednost varijable DiceBoxLocation. Funkcija SetSpawnerParameters(), koja se odnosi na instancu Single Spawner, je prikazana na sljedećoj slici.



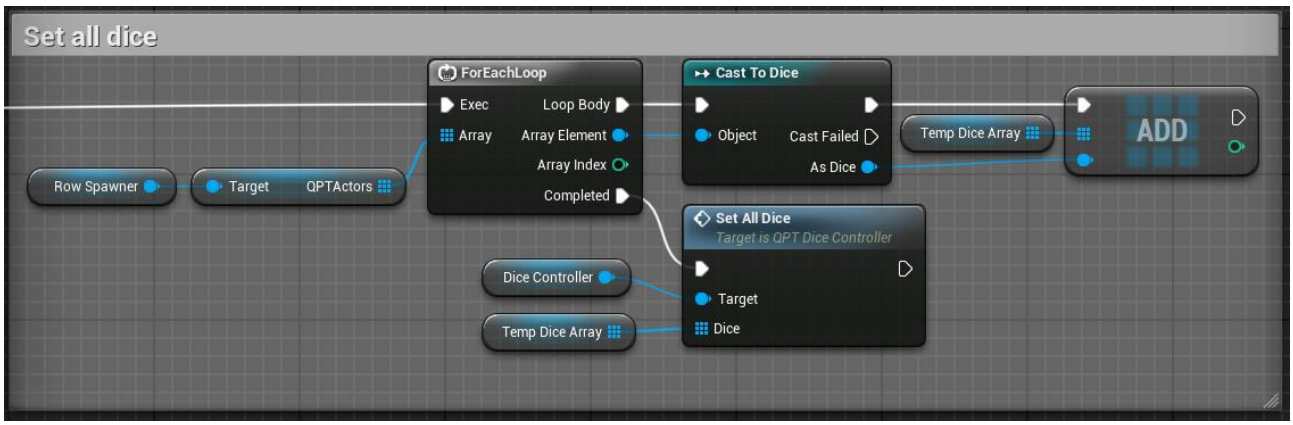
Slika 32. Parametri instance Single Spawner u funkciji SetSpawnerParameters() unutar klase DiceWithBoxTemplate.

Zatim se prelazi na Row Spawner instancu, koja se sprema. Njena varijabla RowLength se mijenja na vrijednost varijable NumberOfDice. Lokacija varijable TransformOffset se mijenja na vrijednost FVector varijable DiceLocation, te RowOffset poprima vrijednost varijable DiceOffset. Na kraju se u varijablu ClassToSpawn sprema klasa Dice. Taj kod je prikazan na slici 33.



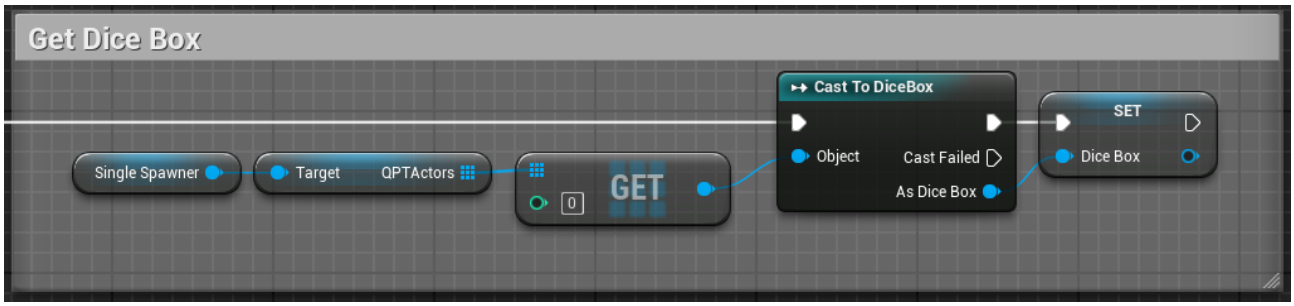
Slika 33. Parametri instance Row Spawner u funkciji SetSpawnerParameters() unutar klase DiceWithBoxTemplate.

Nakon što su se Spawner-i inicijalizirali, te nakon što se pozvala PostInitializeSpawner() funkcija roditelja, sve kreirane instance klase Dice se spremaju i predaju instanci Dice Controller kao argument funkcije SetAllDice(), kao što je prikazano na sljedećoj slici.



Slika 34. Pozivanje funkcije SetAllDice() unutar funkcije PostInitializeSpawner() klase DiceWithBoxTemplate.

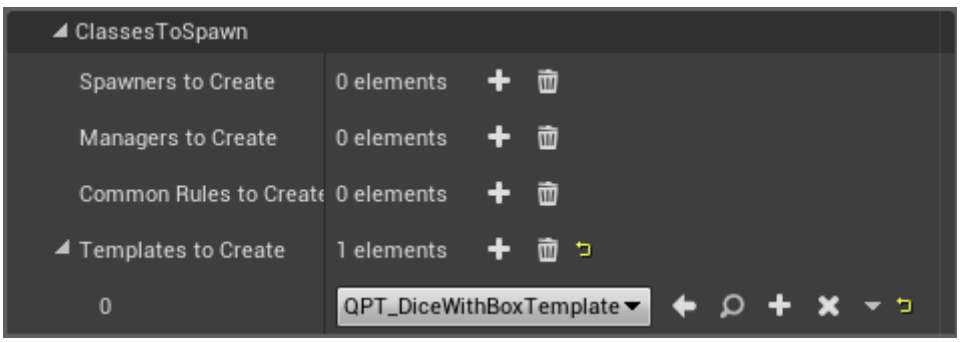
Na sljedećoj slici se prikazuje kako se iz Single Spawner instance sprema referenca na kreirani Dice Box.



Slika 35. Spremanje instance Dice Box dohvaćene iz instance Single Spawner unutar funkcije postInitializeSpawner() unutar klase DiceWithBoxTemplate.

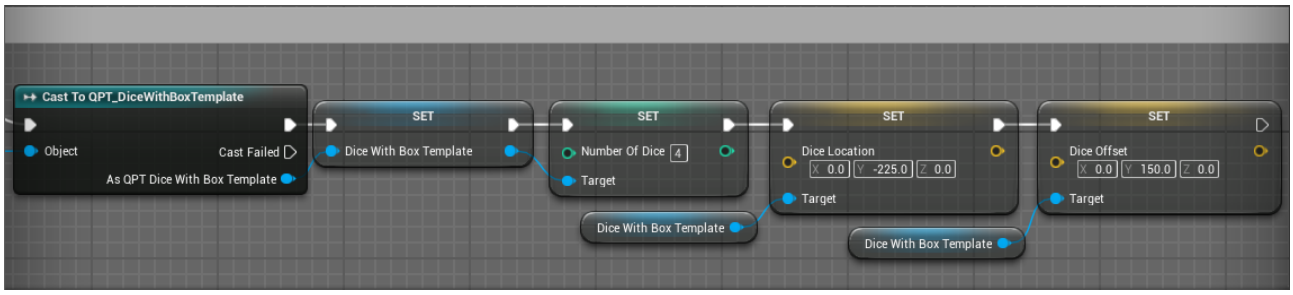
4.3.6. DiceTemplate

Kako bi prikazali efektivnost DiceWithBoxTemplate klase, kreirat ćemo novu klasu DiceTemplate baziranu na GameRule klasi. Imat će istu funkcionalnost kao i MultiDiceDemo klasa, no koristit će samo DiceWithBoxTemplate klasu u listi za inicijalizaciju. Na sljedećoj slici je prikazana cijela lista za inicijalizaciju.



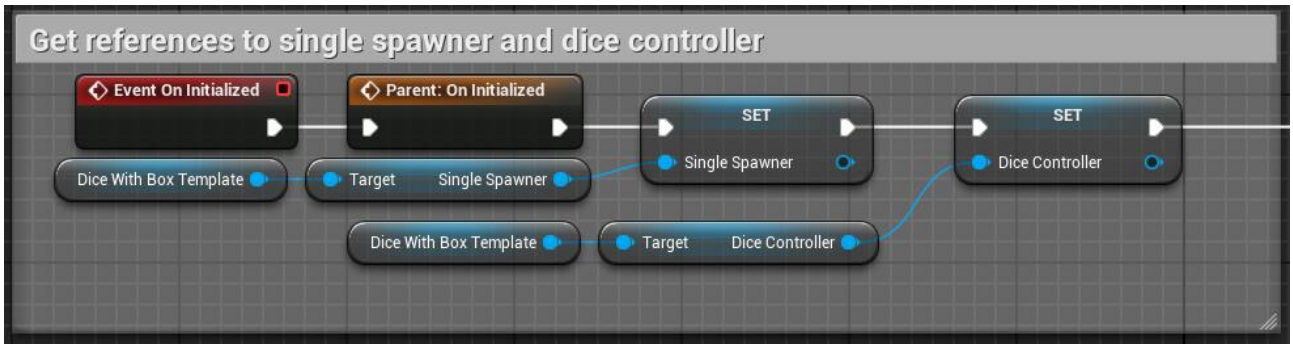
Slika 36. Lista za inicijalizaciju iz klase Dice Template.

Pošto je DiceWithBoxTemplate jedina klasa u listi za inicijalizaciju, tada pozivamo samo SetTemplateParameters() funkciju. Nakon poziva SetTemplateParameters() funkcije roditelja, sprema se referenca Template-a. Postavlja se broj željenih kockica na 4, preko varijable NumberOfDice. Varijabla DiceLocation označava lokaciju prve kockice u redu, te je namještena na vrijednost od 0.0, -225.0, 0.0., te je udaljenost između kockica 0.0, 150.0, 0.0, koja je spremljena u varijabli DiceOffset. Funkcija SetTemplateParameters() je prikazana na sljedećoj slici.



Slika 37. Postavljanje parametara instance DiceWithBoxTemplate u funkciji SetTemplateParameters() unutar klase Dice Template.

Time se nakon inicijalizacije kreiraju instance Dice, Dice Controller, te DiceBox, isto kao i u klasi MultiDiceDemo, no uz razliku što klasa DiceTemplate sadrži samo jednu klasu za inicijalizaciju, umjesto tri, te je dovoljno promijeniti vrijednosti troje varijabli. Kod kreiranja dugmeta, mijenjanja teksta na njemu, te bacanja kockica je identičan kao i u MultiDiceTemplate klasi. Ukoliko korisnik želi reference na instance koje su se koristile u Template-u, tada ih može dobit preko reference tog Template-a, kao što je prikazano na sljedećoj slici.



Slika 38. Dohvaćanje referenci iz DiceWithBoxTemplate instance preko funkcije OnInitialized() unutar klase DiceTemplate.

Koristeći DiceWithBoxTemplate klasu, uvelike smo olakšali korisniku kreiranje demo igre, te može tu istu klasu kasnije koristiti na drugim igrama.

5. Zaključak

Pomoću aplikacije QP_Tabletop moguće je kreirati društvene igre na lagan i brz način. Osnovne klase se brinu o inicijalizaciji i spremanju referenci, te se korisnik ne mora brinuti o tome. Blueprint sustav mu omogućava brzo i jednostavno stvaranje koda na vizualan način, te se ne mora brinuti o sintaksama. Modularnost klasa mu omogućava ponovno korištenje komponenti u drugim igrama, te klasa Template dodatno pojednostavljuje inicijalizaciju igre na način da sadržava kod koji automatski dodaje nove klase te im modificira parametre. Unaprijed gotove klase, poput Dice, Dice Controller, Card i sl., dodatno pojednostavljuju rad, te omogućuju korisniku da odmah može stvarati pravila igre, ne razmišljajući o komponentama. Ukoliko mu je potrebna nova klasa, može ju s lakoćom kreirati u Blueprint sustavu. Ukoliko korisnik ima sve klase koje su potrebne za kreiranje željene igre, nije mu potrebno detaljno znanje njihova rada, te mora samo definirati pravila igre u klasi GameRule.

QP_Tabletop aplikacija sadrži određena ograničenja. Najveće ograničenje aplikacije je inherentno, tj. služi isključivo za kreiranje društvenih igara, te nije korisno u drugim područjima. Pošto je rađena u Unreal Engine-u 4, korisnik ga mora koristiti kako bi koristio QP_Tabletop alat. Drugo ograničenje je to što Spawner-i nisu u stanju kreirati različite klase QPT_Actor-a u jednom pozivu funkcije SpawnQPTActors(), te svi ti QPT_Actori moraju bit iste klase. Za kreiranje QPT_Actor-e različitih klasa, potrebno je za svaku klasu pozivati funkciju SpawnQPTActors(). Još jedno ograničenje je to što korisnik mora pokrenuti igru, kako bi se sve komponente inicijalizirale, pošto se Actor-i ne mogu Spawn-ati prije pokretanja igre. Samim time korisnik mora pokrenuti igru svaki put nanovo kako bi provjerio da li je lokacija nekog QPT_Actora zadovoljavajuća. Ukoliko nije, mora izaći iz igre, kako bi ušao u kod te promijenio parametre.

Za buduću verziju aplikacije, plan je kreiranje novih Blueprint klasa koje olakšavaju kreiranje drugih igara, uključujući GameRule klasu Monopoli, koja predstavlja društvenu igru istog imena. Ona predstavlja igru s kompleksnim pravilima te mnogim konceptima koji se pojedinačno pojavljuju u drugim igrama, poput varirajućeg broja igrača, kontrole bodova pojedinačnog igrača (novca), pomicanje figurica po ploči, kontrola karata i sl. Buduća nadogradnja aplikacije će sadržavati kod koji, preko bool varijable, dopušta korisniku da kreira QPT_Actor-e kao komponente klase Container, što dopušta njihovu inicijalizaciju bez pokretanja igre. Samim time korisnik može vidjeti njihovu transformaciju dok mijenja kod.

Literatura

Gamma, Erich., Helm, Richard., Johnson, Ralph., Vlissides, John. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional; 1 edition.

Langr, J., Ottinger, T. (2011). Abstraction. *PragPub.* 21-26, <https://magazines.pragprog.com/2011/pragpub-2011-02.pdf> Datum zadnjeg pristupa: 17.09.2016

Stroustrup, Bjarne. (2014). *Programming: Principles and Practice Using C++ (2nd Edition).* Addison-Wesley Professional; 2 edition.

Unreal Engine 4 službena stanica. Dostupno na: https://www.unrealengine.com/blog/ue4-is-free (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o UObject klasama. Dostupno na: https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Objects/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o AActor klasama. Dostupno na: https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o UWorld klasi. Dostupno na: https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Engine/UWorld/index.html (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o klasi UActorComponent. Dostupno na: https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors/Components/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o varijablama i tipovima podataka. Dostupno na: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Variables/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o Blueprint sustavu. Dostupno na: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/Overview/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o direktnoj komunikaciji u Blueprint sustavu. Dostupno na: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/BlueprintComms/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o Blueprint Interface sustavu. Dostupno na: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/Interface/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o Event Dispatcher sustavu. Dostupno na: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/BP_HowTo/EventDispatcher/ (Datum zadnjeg pristupa: 15.09.2016.)

Unreal Engine 4 službena stranica, dokumentacija o Custom Events funkcionalnošću. Dostupno na: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Events/Custom/index.html (Datum zadnjeg pristupa: 19.09.2016)

Unity službena stranica, usporedba raznih planova plaćanja. Dostupno na: https://store.unity.com/ (Datum zadnjeg pristupa: 19.09.2016)

Cryengine službena stranica, prikaz grafičkih mogućnosti. Dostupno na: https://www.cryengine.com/features/visuals (Datum zadnjeg pristupa: 19.09.2016)

Popis Tablica

Tablica 1. Usporedba različitih verzija plaćanja za Unity Engine. 3

Popis programskog koda

Programski kod 1. Funkcija Initialize() u klasi Container. 13
Programski kod 2. Funkcija OnInitialized() u klasi GameRule. 14
Programski kod 3. Funkcija CreateTemplate() u klasi GameRule. 14
Programski kod 4. Funkcija CreateCommonRule() u klasi GameRule. 15
Programski kod 5. Funkcija CreateSpawner() u klasi GameRule. 15

Programski kod 6. Funkcija CreateManager() u klasi GameRule. 15
Programski kod 7. Funkcija SpawnQPTActors() u klasi Spawner.....17
Programski kod 8. Funkcija CreateComponents() u klasi Manager. 18

Popis slika

Slika 1. Primjer naprednih grafičkih sposobnosti Cryengine-a 3
Slika 2. Jednostavan kod u Blueprint sustavu koji ispisuje String. 4
Slika 3. OnCreated() funkcija u klasi Dice.....19
Slika 4. Roll() funkcija u klasi Dice. 20
Slika 5. AddForce() funkcija u klasi Dice. 21
Slika 6. Provjera brzina u Tick() funkciji unutar klase Dice.....21
Slika 7. Provjera nakošenosti kockice i Tick() funkciji unutar klase Dice.....22
Slika 8. Dio koda u Tick() funkciji unutar Dice klase koji ju pomiče ukoliko je ona nakošena. 22
Slika 9. Dio koda u klasi Dice koji se odvija nakon što se ona uspješno zaustavila. 23
Slika 10. Funkcija SetSize() u klasi DiceBox. 24
Slika 11. Kod za slanje notifikacije i pomicanje dugmeta u klasi Button. 24
Slika 12. Dio koda funkcije SetAllDice() u klasi Dice Controller. 25
Slika 13. Prvi dio funkcije Roll() u klasi Dice Controller. 26
Slika 14. Drugi dio funkcije Roll() u klasi Dice Controller. 26
Slika 15Dio koda u funkciji DiceStopped(), unutar klase Dice Controller, koji šalje notifikaciju OnDiceStopped. 26
Slika 16. Dio koda u funkciji DiceStopped(), unutar klase Dice Controller, koji provjerava da li su se sve kockice zaustavile. 27
Slika 17. Funkcija AllDiceStopped() u klasi Dice Controller. 27
Slika 18. Funkcija RestAllDice() u klasi Dice Controller. 28
Slika 19. Funkcija CalculateSpawnTransform() u klasi Single Spawner. 28
Slika 20. Funkcija SpawnSingle() unutar klase Single Spawner. 28
Slika 21. Funkcija CalculateSpawnTransform() unutar klase Row Spawner.....29
Slika 22. Lista za inicijalizaciju u klasi MultiDiceDemo. 30
Slika 23. Funkcija OnCreateCommonRule() u klasi MultiDiceDemo. 30
Slika 24. Mijenjanje parametara Single Spawner instance u funkciji OnSetSpawnerParameters(), unutar klase MultiDiceDemo. 31
Slika 25. Mijenjanje parametara Row Spawner instance u funkciji OnSetSpawnerParameters() unutar klase MultiDiceDemo. 31

Slika 26. Funckija OnAllSpawnersInitialized() u klasi MultiDiceDemo. 32
Slika 27. Pozivanje funkcija SetAllDice() unutar funkcije OnInitialized() u klasi MultiDiceDemo. 32
Slika 28. Ispisivanje sume kockica na dugme u funkciji DiceStopped() u klasi MultiDiceDemo. 33
Slika 29. Pozivanje funkcije Roll() klase Dice Controller unutar klase MultiDiceDemo, kada igrač klikne mišem na dugme. 33
Slika 30. Lista klasa koje će se dodati listi za inicijalizaciju unutar klase DiceWithBoxTemplate. 34
Slika 31. Funckija SetCommonRuleParameters() u klasi DiceWithBoxTemplate. 34
Slika 32. Parametri instance Single Spawner u funkciji SetSpawnerParameters() unutar klase DiceWithBoxTemplate. 35
Slika 33. Parametri instance Row Spawner u funkciji SetSpawnerParameters() unutar klase DiceWithBoxTemplate. 35
Slika 34. Pozivanje funkcije SetAllDice() unutar funkcije PostInitializeSpawner() klase DiceWithBoxTemplate. 36
Slika 35. Spremanje instance Dice Box dohvaćene iz instance Single Spawner unutar funkcije postInitializeSpawner() unutar klase DiceWithBoxTemplate. 36
Slika 36. Lista za inicijalizaciju iz klase Dice Template. 36
Slika 37. Postavljanje parametara instance DiceWithBoxTemplate u funkciji SetTemplateParameters() unutar klase Dice Template. 37
Slika 38. Dohvaćanje referenci iz DiceWithBoxTemplate instance preko funkcije OnInitialized() unutar klase DiceTemplate. 37

Sažetak

U ovom dijelu se prikazuje rad aplikacije QP_Tabletop, ukratko se opisuje Game Engine korišten za njenu kreaciju, odabran po određenim parametrima, te se postepeno opisuje proces kreiranja društvene igre u njoj. Također se detaljno prikazuju osnovne klase, njihove inicijalizacije te međusobne interakcije između tih klasa. Zatim se prikazuje rad unaprijed kreiranih Blueprint klasa, koje se koriste kao komponente druge klase koja predstavlja pravila željene igre. Tada se dio funkcionalnosti premješta u posebnu klasu, kako bi ju korisnik mogao koristiti u budućim društvenim igrama, neovisno o trenutnoj demo igri.

Ključne riječi

QP_Tabletop, QPT, Unreal Engine 4, društvene igre.

Summary

This paper shows how QP_Tabletop application works, shortly describes the Game Engine used in its creation, why it was chosen, and presents a step-by-step process of creating a tabletop game. It also describes all the base classes in detail, their initialization, and interactions with each other. Then the pre-made Blueprint classes are described. These pre-made classes act as components for another class that represents the rules of the game. After that, part of the functionality is moved to another special class, which allows the user to use it again in the future.

Keywords

QP_Tabletop, QPT, Unreal Engine 4, tabletop games.