Izrada Multiplayer igre

Murk, Matija

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

Matija Murk

IZRADA MULTIPLAYER IGRE

ZAVRŠNI RAD

Čakovec, rujan 2023.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

Matija Murk

IZRADA MULTIPLAYER IGRE

CREATING A MULTIPLAYER GAME

ZAVRŠNI RAD

Mentor:

Nenad Breslauer, v. pred.

Čakovec, rujan 2023.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
ODBOR ZA ZAVRŠNI RAD

Čakovec, 22. veljače 2022.

Polje: **2.09 Računarstvo**

ZAVRŠNI ZADATAK br. 2021-RAC-R-36

Pristupnik: **Matija Murk (0313023729)**
Studij: Redoviti preddiplomski stručni studij Računarstvo
Smjer: Programsko inženjerstvo

Zadatak: **Izrada Multiplayer igre**

Opis zadatka:

Ideja je napraviti Multiplayer igru u kojoj igrači prolaze kroz nasumično odabrane levele, pobjeđuje onaj koji zadnji ostaje. Potrebno je izraditi scenu te sve potrebne elemente za igru, koristiti mogućnosti koje pruža podsustav za osvjetljenje scene, koristiti simulaciju fizikalnih svojstava, kontrolu kamere, game meni i zvučne efekte. Koristiti platformu Unreal, programski jezik C++ te dodatne programske alate. Završni rad mora sadržavati sažetak, sadržaj i uvod, nakon čega slijedi poglavlje u kojem je potrebno navesti i pojasniti osnovne ciljeve rada te očekivani rezultat. U narednom poglavlju potrebno je opisati primijenjene postupke, alate i metode. Poglavlje koje slijedi obrađivat će postignute rezultate nakon čega slijedi poglavlja u kojem se kritički raspravlja o primijenjenim metodama i postupcima te se u narednom poglavlju iznose glavni zaključci rada. Rad se završava poglavljima s popisom literature te prilozima.

Zadatak uručen pristupniku: 22. veljače 2022.
Rok za predaju rada: 20. rujna 2022.

Mentor:

Nenad Breslauer, v. pred.

Predsjednik povjerenstva za završni ispit:

ZAHVALA

Želim zahvaliti svom dragom mentoru na usmjeravanju i vođenju tijekom pisanja ovog rada čime mi je omogućio da kvalitetno razložim temu i sročim je u strukturiranu cjelinu.

Matija Murk

Sažetak

Ovaj završni rad bavi se tematikom izrade videoigre u Unreal Engineu i povezivanjem igrača preko servera kako bi mogli igrati zajedno, na različitim računalima. U radu je opisano programsko okruženje koje je pritom korišteno.

Današnje videoigre postaju sve složenije. Potrebno je imati znanje iz više razvojnih okolina i programa da bi se igra mogla samostalno napraviti. Rad opisuje izradu igre za više igrača (engl. *multiplayer*) što mu daje dodatnu notu složenosti. Prednosti i mane izrade takve vrste igre također su opisane u ovom završnom radu.

Rad se, nadalje, bavi poviješću videoigara i razvojem istih tijekom vremena. Opisani su isto tako i moderni programi za kreiranje videoigara te njihove mogućnosti. Usto, detaljno je prikazana izrada računalnih igara kao složen proces koji, osim samog programiranja, uključuje i dizajn. Rad se bavi i postavljanjem Unreal Engine izvornog koda s Visual Studijom te kreiranjem projekta uz njegovu pomoć. Raščlanjen je tijek procesa kojim se kreira C++ klasa u samom Unreal Engineu.

Predstavljeno je konfiguriranje Amazon GameLift platforme, kao i integriranje iste u Unreal Engine. Bit će riječi i o povezivanju igre sa serverom te o tome koja je baza podataka izabrana za korištenje u ovom projektu. Na kraju se objašnjava proces izrade same igre, npr. kreiranje razina i prepreka za igrača u Unreal Engineu. Opisan je i proces prijave igrača na vlastiti račun za igru prilikom njezina pokretanja.

Cilj ovog završnog rada jest napraviti igru za više igrača, koji će biti povezani preko vanjskog servera. Igra će automatski spajati igrače sličnih vještina i s najbližom mogućom lokacijom kako bi se smanjilo zastajkivanje tijekom igre te kako bi se pružilo najugodnije moguće iskustvo igranja. U igri će igrači prolaziti kroz prethodno izrađene razine, no svaki put kada igra počinje ispočetka, one se biraju nasumično iz skupa određenih razina. Cilj svake razine jest doći do cilja prije drugih igrača. Koristit će se Amazon GameLift serveri.

Ključne riječi*: Videoigra, Unreal Engine, C++, razvoj videoigara, multiplayer, GameLift, Grafika, efekti*

SADRŽAJ

1. UVOD 7
2. POČETAK VIDEOIGARA KAO MEDIJA 8
3. RAZVOJ RAČUNALNIH VIDEOIGARA 9
3.1. Stadiji razvoja videoigara 9
3.2. Dizajn videoigara 10
3.2.1 Različite vrste igara s obzirom na broj igrača i njihovu ulogu 11
4. RAZVOJNE OKOLINE 12
4.1. Unreal Engine 12
4.2. Visual Studio 13
4.3. Amazon Web Service 13
5. PREUZIMANJE I INSTALIRANJE UNREAL ENGINE IZVORNOG KODA 14
5.1. Postavljanje Amazon GameLift Server SDK-a 17
5.2. Kreiranje C++ klase u Unreal Engineu 19
5.3. Klasa unutar Visual Studija 21
5.3.1. Header datoteke 21
5.3.2. Implementation datoteke 23
5.3.3. Klasa za povezivanje igre sa serverom 27
6. POSTAVLJANJE SERVERA PREKO AWS GAMELIFT KONZOLE 32
6.1. GameLift konzola 33
6.2. Matchmaking rule set 35
6.3. Baza podataka DynamoDB 37
6.4. Tablice 37
7. POSTAVLJANJE I KREIRANJE IGRE U UNREAL ENGINEU 39
7.1. Kreiranje razina i prepreka 39
7.2. Registracija i prijava igrača 41
7.3. Glavni izbornik 43

7.4. Dva igrača su se uspješno spojila.....45
8. ZAKLJUČAK 47
LITERATURA.....48
POPIS KODOVA.....49
POPIS SLIKA 50

1. UVOD

Igra za više igrača (engl. *Multiplayer game*) preko interneta namijenjena je ljudima koji svoje vještine žele omjeriti u srazu s drugim igračima, koji mogu biti na svojem računalu na potpuno drugoj lokaciji. Za razliku od igara u kojima igra samo jedan igrač protiv umjetne inteligencije ili samog svijeta igre, ovdje se dva ili više igrača mogu staviti u istu igru te se tako, na neki način, dobije društveno iskustvo. No, treba krenuti ispočetka i uopće razumjeti kako se igre stvaraju. Temelj svake igre je njezin program koji je pokreće. Dok veliki studiji koriste vlastite programe poput Electronic Arts (Frostbite), CD Projekt Red (REDEngine), neovisni studiji većinom se oslanjaju na javne programe poput Unitya, Unreal Enginea i Godota zbog toga što su uglavnom besplatni ili je barem neka verzija istih besplatna. Kada se odabire program koji će se koristiti, ako se bira jedan od javnih, treba obratiti pozornost na prednosti i nedostatke svakog. Kod Unitya prednost je što je najrasprostranjeniji program kod malih razvijača (engl. *developer)* pa ima najviše dostupnih izvora za učenje (npr. YouTube) i najopširniju dokumentaciju. Također, izvrstan je u kreiranju 2D igara. Godot je najmlađi program koji je tek nedavno dobio na popularnosti i koristi prilagođen jezik za pisanje koda, što znači da ima najmanje dostupnih izvora za učenje i veoma malo dokumentacije. Najveću je prednost u ovom slučaju dobio Unreal Engine, koji nema toliko izvora za učenje kao Unity, ali ima prilično dobru dokumentaciju i koristi C++ (Unity koristi C#), koji daje visoke performanse kod velikog broja izračuna, što je bitno prilikom kreiranja igara za više igrača. Također, uz malo truda može se dobiti izvrsna 3D grafika. Još jedna prednost je to što se može koristiti i vizualno skriptiranje pomoću Nacrta (engl. *Blueprints*). Iako to koristi malo više resursa nego čisti C++, ipak može osjetno ubrzati rad na nekoj značajki.

2. POČETAK VIDEOIGARA KAO MEDIJA

Videigre i tehnologija vezana uz njih u posljednjih su nekoliko godina napravile velik skok izlaskom Unreal Enginea 5, koji se koristi čak i u filmskoj industriji (npr. serija The Mandalorian). Nove tehnike osvjetljenja i *raytracinga* u stvarnom vremenu omogućili su da igre izgledaju bolje no ikada dosad i sličnije stvarnom životu.

Teško je povjerovati da je sve počelo 1972. [1] s dvije daske i lopticom, aludirajući, naravno, na igru Pong, kojom je i započela velika zainteresiranost ljudi za videoigre. Inspirirana legendarnom arkadnom igrom Pinball [1], Pong je oduševio mnoge svojom jednostavnim, ali zaraznim *game loopom*. Također, može se reći da je to prva igra za više igrača jer su, da bi igra imala smisla, u početku bila potrebna dva igrača. Šest godina kasnije izašla je i igra Spacewar, a 1980 Pac-Man [2].

Iako su te igre bile inovativne i zabavne, ono što ih je uistinu učinilo popularnima jest izum video arkade. Taj je koncept postojao i ranije, ali u mehaničkom smislu, npr. u prije spomenutom Pinballu, no digitalna verzija arkada počela je 1970-ih i bila je osobito popularna među mladima [2].

Početkom 1980-ih videoigrama je rasla popularnost što je dovelo do nastanka Nintendovog *Super Mario Bros*, 1985 godine[2]. Također, narednih je godina izašlo još nekoliko igara koje su se izuzetno dobro prodavale.

Tada su se pojavila mala, ali za to vrijeme snažna osobna računala ili PC (engl. *Personal Computer)*. To je omogućilo da igre dođu u kućanstvo, što je opet povećalo popularnost i potražnju za videoigrama. Tih su godina nastali neki od klasika, npr. *Wolfenstein 3D*, *Sonic the Hedgehog, Mortal Kombat* itd.

Kako se industrija videoigara razvijala i njezina popularnost rasla, tvrtke poput Sonya i Nintenda razvijale su vlastite konzole. Godine 2000. nastala je danas najprodavanija konzola ikad, Sonyev *PlayStation 2* [2]. U razdoblju koje slijedi dominira razvijanje sve bolje grafike i igara sa sve dubljim mehanikama.

3. RAZVOJ RAČUNALNIH VIDEOIGARA

U videoigrama prevladava naglasak na izgradnji zanosne igrivosti i prožimajućeg iskustva. Iako se danas izrada videoigara smatra umjetnošću, koja je subjektivna, ipak postoje neke smjernice i pravila koja pomažu da programer postigne upravo ono što je zamislio i predoči to igračima u obliku igre. Bilo to nekom inovativnom mehanikom ili jedinstvenom grafikom, poželjno je da svaka igra ima neki svoj *hook*, koji će igru učiniti prepoznatljivom.

3.1. Stadiji razvoja videoigara

Stvaranje igre prolazi kroz sedam važnih faza. Ukratko rečeno, to su:

- Početna ideja ili koncept
- Opseg
- Predproizvodnja
- Proizvodnja
- Testiranje
- Izbacivanje
- *Post* proizvodnja

Ukupan uspjeh projekta razvoja igre ovisi o svakoj fazi. Iako su koraci načelno sukcesivni, ponekad može doći do regresije u koracima.

Primjerice, faze 1-7, mogu izgledati ovako:

1. Kroz brainstorming dolazite do novih mehanizama i koncepata igre. Provodite temeljitu studiju o tržištu (početni koncept).
2. Vi dajete proračun, veličinu tima, redoslijed i druge pojedinosti, zajedno s financiranjem (opseg).
3. Predprodukcija uključuje usklađivanje izvornog koncepta s novom stvarnošću.
4. U fazi proizvodnje vi stvarate priču ili faze igre. Proizvodite grafiku i druge resurse potrebne za dizajn igre. Logika igre programirana je pomoću programa poput C++ ili Java skripte.

5. Igra se zatim testira i dorađuje kako bi se osiguralo pridržavanje strogih smjernica za izvrsnost igranja i estetsku privlačnost.
6. Igra se zatim objavljuje na raznim platformama ili sustavima.
7. Konačno, nastavljate podržavati igru izdavanjem zakrpa i nadogradnji sadržaja nakon izdavanja (post produkcija).

3.2. Dizajn videoigara

Igre su iskustva samo za igrače, a ti isti igrači dobrovoljno prihvaćaju pravila i ograničenja igre da bi je igrali.

Jedan od najvažnijih trenutaka u igri jest poziv na igru:

- U društvenoj igri, poziv na igranje predstavlja društveni dio igre - igrači pozivaju jedni druge na zajedničku igru
- U digitalnim igrama proces je tehnički - tipka za pokretanje, ulaz u zaslon
- Cilj nekih igara može biti i poboljšanje igranja – Guitar Hero Controller

Uz pozivnicu na igru potrebno je odgovoriti na sljedeća pitanja:

- Koliko je igrača potrebno za igru?
- Koji je najveći broj igrača u igri?
- Imaju li igrači različite uloge?
- Hoće li igrači surađivati, natjecati se jedni s drugima ili oboje?

Igra za jednog igrača bitno se razlikuje od igre za dva, četiri ili 10 000 igrača.

Većina igara ima istu ulogu za sve igrače:

- Šah, Monopol - jedna uloga za sve igrače
- Mastermind - jedan igrač stvara kod/kombinaciju, dok ga drugi pokušava pronaći

RPG igre imaju različit broj uloga, koje igrači mogu birati - izbor uloge ima značajan utjecaj na osnovne sposobnosti odabranog lika [4].

3.2.1 Različite vrste igara s obzirom na broj igrača i njihovu ulogu

U igrama kao što su Pac-Man ili pasijans jedan igrač natječe se protiv same igre. Ovakve igre imaju više zagonetki ili prepreka. Više igrača protiv igre (npr. Rulet) igre su u kojima nema interakcije i međudjevolanja među igračima. Takav tip igre odličan je za one koji nisu previše kompetetivni i vole igru kao takvu, ali i njezin društveni dio.

Igrač protiv igrača (npr. Mortal Kombat) tip je igre u kojoj se dva igrača natječu direktno.

Ovo su samo neki primjeri vrsta igara s obzirom na broj igrača i njihovu ulogu. U stvarnom svijetu postoje deseci različitih kombinacija prema kojima se igre mogu definirati [4].

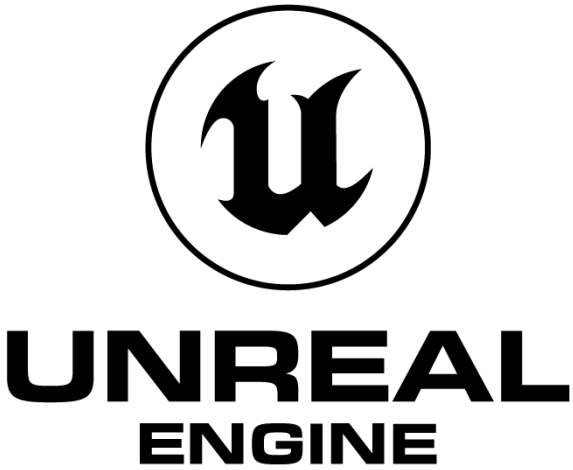
4. RAZVOJNE OKOLINE

Videoigra za više igrača preko interneta, u kojoj dolazi do povezivanja više igrača sa zasebnih računala i koji mogu biti bilo gdje na svijetu, omogućuje igri da bude pristupna većoj bazi kompetitivnih korisnika nego, recimo, lokalna kooperativna igra gdje mogu igrati samo igrači na istom računalu.

Igra će biti izrađena koristeći primarno Unreal Engine, Visual Studio, Blender i Amazon Web Service (AWS) platformu. Unreal Engine, specifično Unreal Engine 4, bit će korišten za povezivanje svakog dijela izrade igre u jednu cjelinu, a Visual Studio za pisanje potrebnog koda, koji igri omogućuje funkcionalnost. AWS će biti korišten za pružanje usluge servera koja je ključan dio ovog projekta.

4.1. Unreal Engine

3D računalni grafički program za igre pod nazivom Unreal Engine (UE) kreiran je od strane Epic Gamesa i debitirao je u igri pucačine u prvom licu Unreal 1998. godine. U početku je stvoren za PC igre pucačine u prvom licu, ali je od tada prilagođen za korištenje i u mnogim drugim žanrovima igara, kao i u drugim industrijama, ponajviše u filmskoj i televizijskoj industriji. Unreal Engine, temeljen na C++-u, vrlo je prenosiv i podržava razne platforme za računala, mobilne uređaje, konzole i virtualnu stvarnost.



Slika 1 Unreal Engine logo

Izvor: https://www.unrealengine.com/en-US/branding

4.2. Visual Studio

Microsoftov Visual Studio integrirano je razvojno okruženje (IDE). Računalni programi, uključujući web stranice, web aplikacije, online usluge i mobilne aplikacije, razvijaju se pomoću njega. Visual Studio koristi Microsoftove platforme za razvoj softvera, uključujući Windows Store, Windows Presentation Foundation, Windows API i Windows Forms. C, C++, C++/CLI, Visual Basic.NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS samo su neki od ugrađenih jezika. Putem dodataka može se podržati više jezika, uključujući Python, Ruby, Node.js i M, između ostalih.

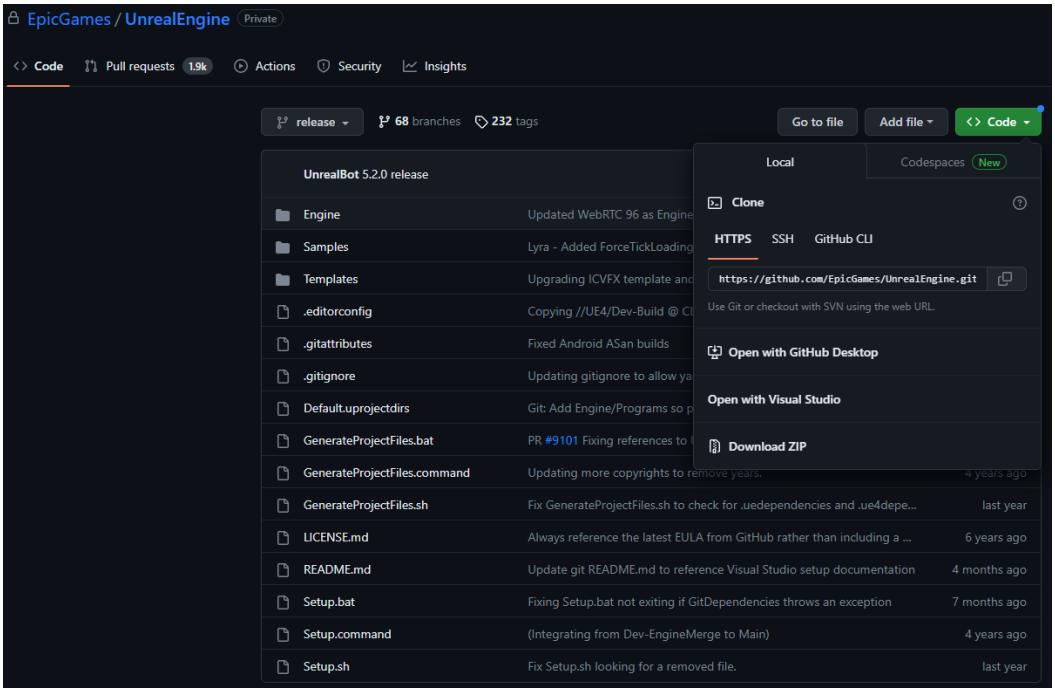
4.3. Amazon Web Service

Podružnica Amazona pod nazivom Amazon Web Services, Inc. (AWS) nudi platforme i API-je za računalstvo u oblaku s ograničenim korištenjem računala na zahtjev ljudima, tvrtkama i vladama.

Korisnici koji koriste AWS usluge dobivaju ih putem globalne mreže AWS farmi poslužitelja. „*Pay-as-you-go*“ pristup naplati temelji se na potrošnji, kao i na hardveru, operativnim sustavima, softveru i mrežnim karakteristikama po izboru pretplatnika koji moraju biti dostupni, redundantni, sigurni i nuditi niz alternativnih usluga. Pretplatnici imaju mogućnost plaćanja za jedno fizičko ili virtualno AWS računalo, klaster bilo kojeg, ili oboje.

5. PREUZIMANJE I INSTALIRANJE UNREAL ENGINE IZVORNOG KODA

Za ovaj će se projekt koristiti izvorni kod Unreal Enginea zbog toga što „obična“ verzija (ona koju se može instalirati preko Epic Games Launchera) nema mogućnost pakiranja servera, što je potrebno kada se radi na *online multiplayer* igri. Izvorni se kod može preuzeti preko GitHub stranice, no da bi se moglo pristupiti samom izvornom kodu, mora se imati GitHub račun povezan s Unreal Engine računom jer je repozitorij definiran kao privatan na GitHubu.

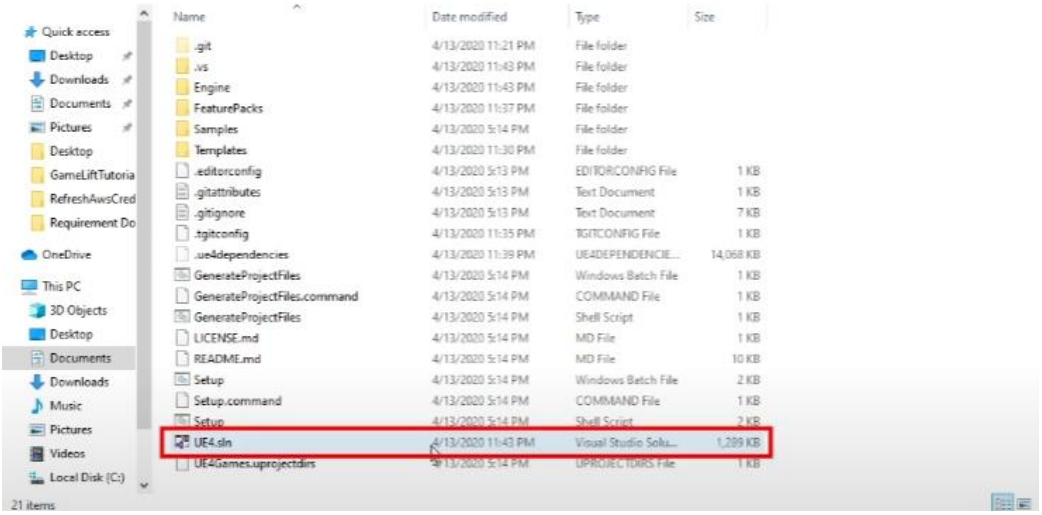


Slika 2 Preuzimanje Unreal Enginea

Izvor: Autor

Nakon što se projekt preuzme mora se pokrenuti nekoliko *batch* datoteka, koje se preuzimaju zajedno s Unreal Engineom. Prva na redu je Setup.bat, koja instalira zavisnosti za Unreal Engine ovisno o tome na kojoj je platformi instaliran, u ovom slučaju to je Windows.

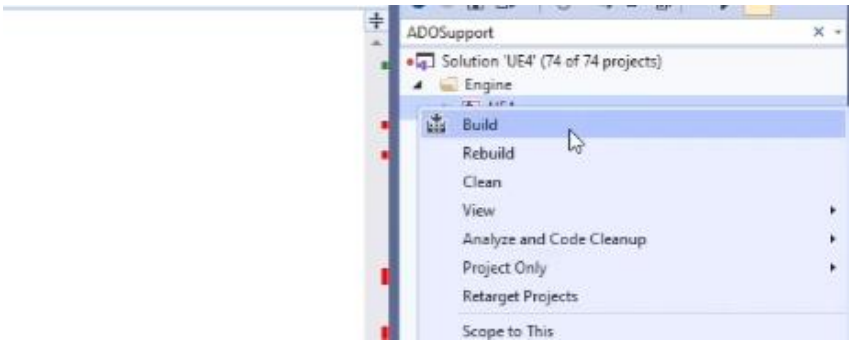
Zatim se moraju generirati projektne datoteke. Da bi se to omogućilo, mora se pokrenuti još jedna *batch* datoteka, GenerateProjectFiles.bat. To omogućuje da se može napraviti novi projekt, koji će koristiti instaliranu verziju Visual Studija i Unreal Enginea. Sljedeća se pokreće UE4.sln *solution* datoteka, koja se otvara u Visual Studio programu.



Slika 3 Otvaranje Visual Studio projekta

Izvor: Autor

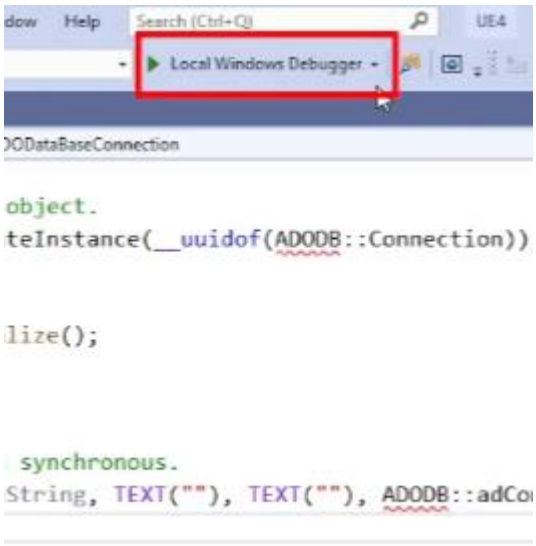
Kada se Visual Studio pokrene, u njemu se mora izgraditi (engl. *build*) Unreal Engine projekt. Kako bi se to učinilo, potrebno je pritisnuti desnu tipku miša na UE4 projekt zatim izabrati *Build* u izborniku koji se otvorio.



Slika 4 Građenje Unreal Enginea

Izvor: Autor

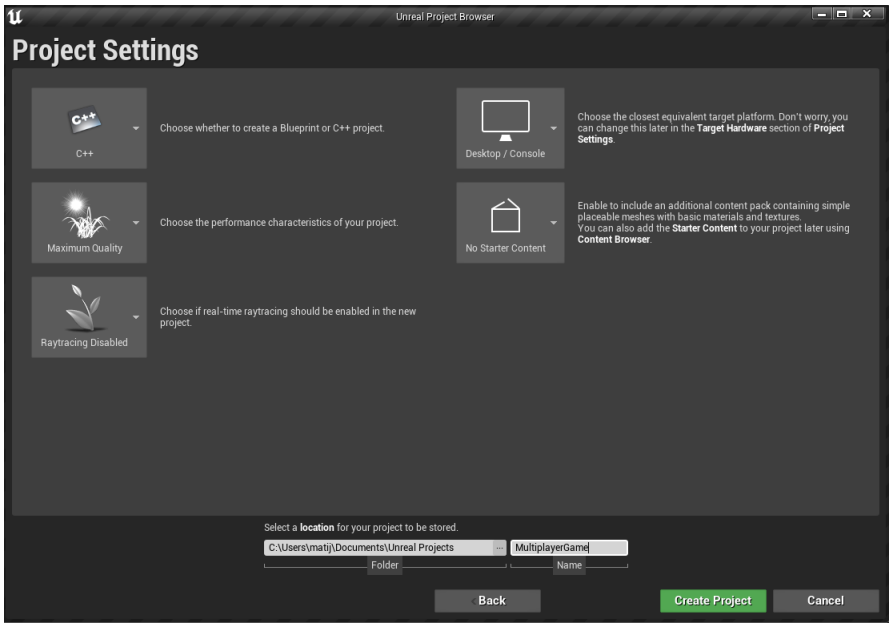
Nakon što se projekt izgradio, može se pokrenuti Unreal Engine pritiskom na tipku *Local Windows Debugger*.



Slika 5 Pokretanje aplikacije

Izvor: Autor

Kod otvaranja Unreal Enginea dolazi i do kreiranja projekta, koji će se koristiti za izradu *multiplayer* igre. Otvara se prozor u kojem se mogu promijeniti razne postavke. Za ovaj projekt koristi se C++ pa će se to i odabrati. Sve ostale postavke potrebno je ostaviti na zadanom statusu i može se započeti s kreiranjem novog projekta.

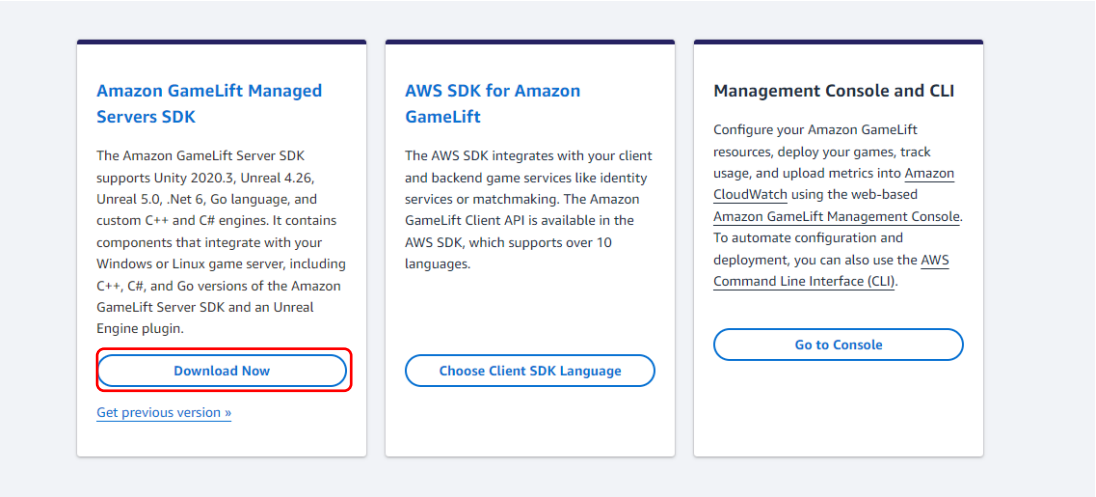


Slika 6 Kreiranje projekta unutar Unreala

Izvor: Autor

5.1. Postavljanje Amazon GameLift Server SDK-a

Kako bi se igra mogla povezivati s ostalim igračima, treba postaviti server. U ovom slučaju koristit će se Amazonov GameLift Server. Potrebno je preuzeti SDK s Amazonove Internet stranice. SDK omogućuje da GameLift Service komunicira sa serverom. Naprimjer, GameLift Service će reći serveru kada se kreira novi meč i povezat će igrače koji odgovaraju kriterijima. U suprotnom smjeru server će obavijestiti GameLift Service kada igrač napusti meč, kada igra završi ili kada meč treba nove igrače. Ta komunikacija odvijat će se preko GameLift Server SDK-a.

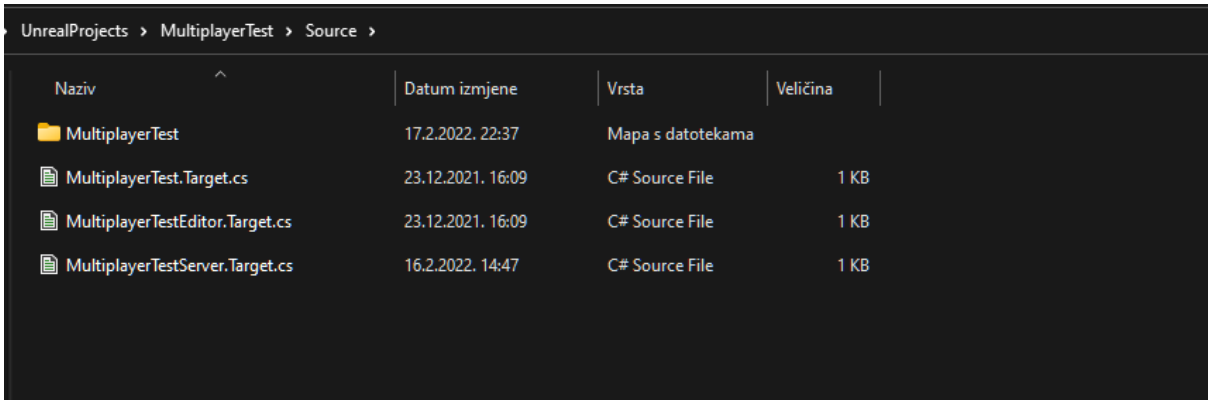


Slika 7 Preuzimanje Amazon GameLift SDK-a

Izvor: Autor

Skinuti SDK stavlja se u projekt kao *plugin*. To znači da se preuzeti sadržaj mora kopirati u *Plugins* mapu, koja se nalazi u mapi projekta. Ako mapa *Plugins* ne postoji, mora se ručno kreirati. No, to nije dovoljno za postavljanje. Potrebno je kreirati i izvornu datoteku servera, a da bi se to učinilo, jednostavno se kopira [ImeProjekta]Editor.Target.cs datoteka i preimenuje u [ImeProjekta]Server.Target.cs.

.

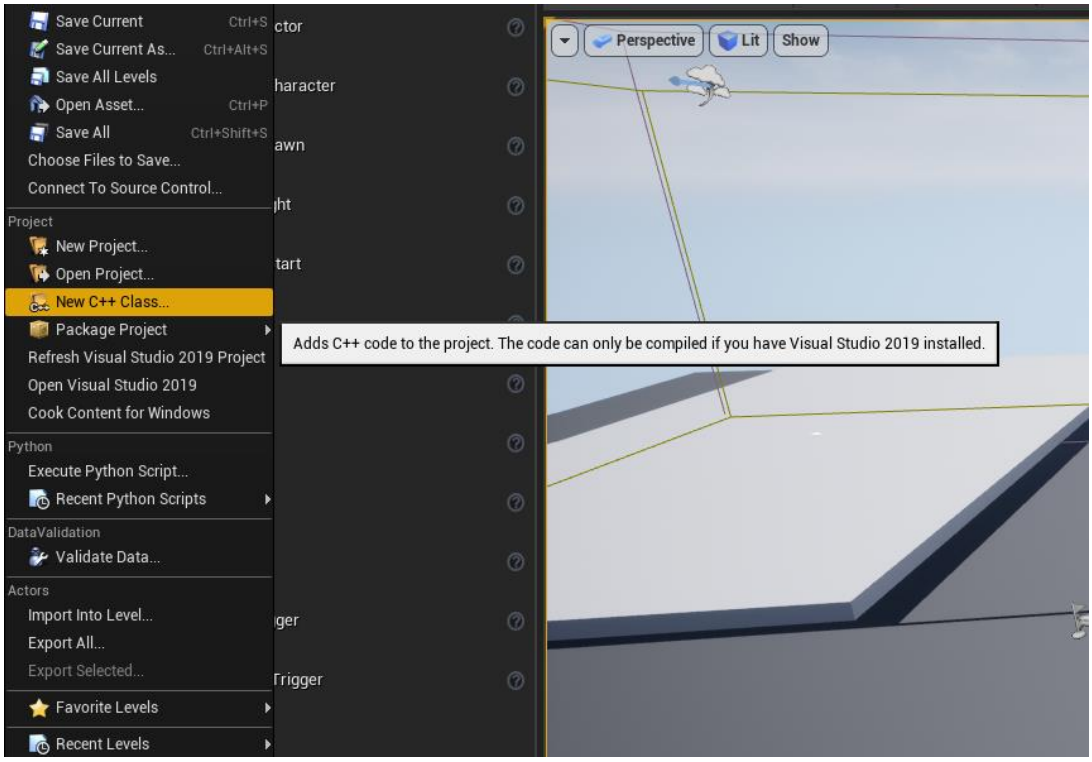


Slika 8 Datoteke za izgradnju projekta kao servera i datoteke za klijenta

Izvor: Autor

5.2. Kreiranje C++ klase u Unreal Engineu

Sada se može otvoriti projekt u Unreal Engineu jer je potrebno napraviti nekoliko klasa, koje će se koristiti prilikom kreiranja igre. Kako bi se kreirala klasa unutar Unreal Enginea, prate se sljedeći koraci: Otvara se *File* izbornik i odabire se *New C++ Class…*

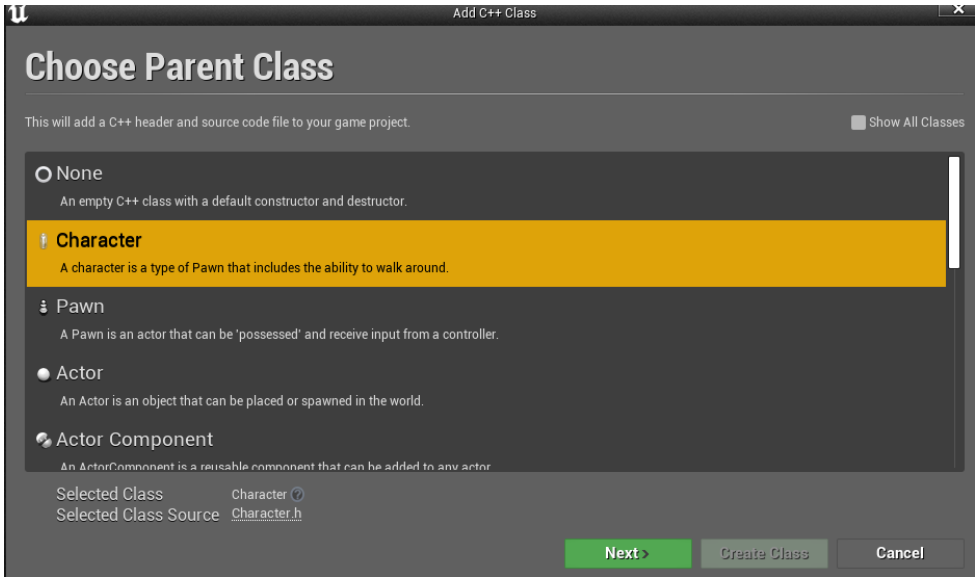


Slika 9 Gumb za odabir kreiranja klase u Unrealu

Izvor: Autor

Otvara se novi prozor, u njemu se odabire roditeljska klasa koja dolazi sa samim Unrealom ili se odabere *None* ako se ne želi koristiti roditeljska klasa. U ovom će se slučaju, kao primjer, koristiti kreiranje klase lika kojim će igrač upravljati, a zvat će se *MultiplayerGameCharacter*.

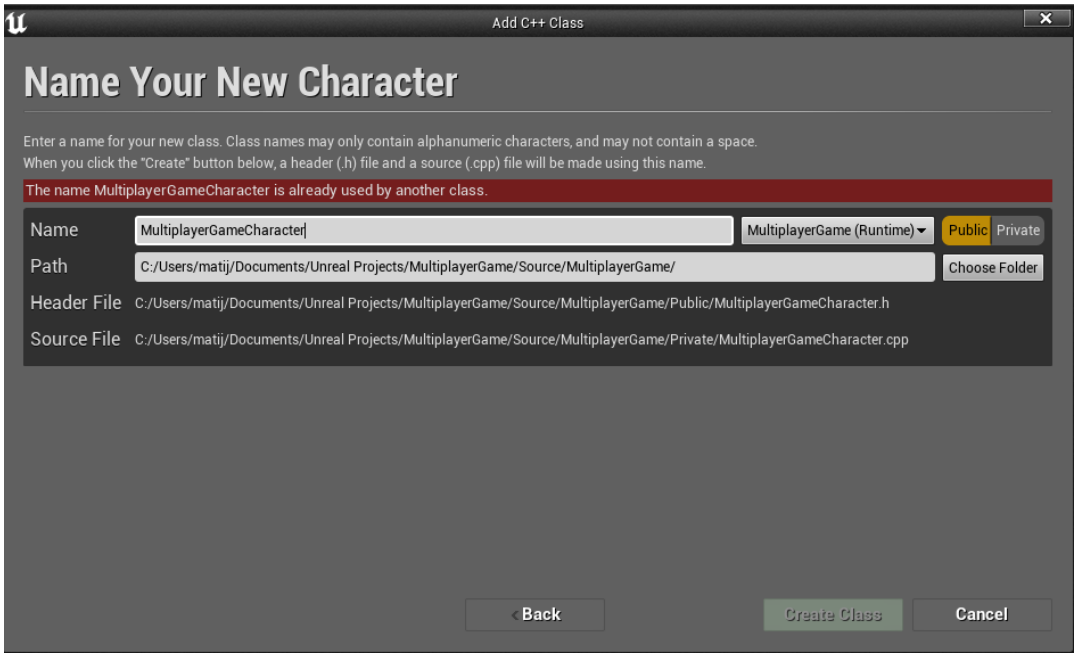
Odabire se roditeljska klasa *Character* koja već ima ugrađene komponente poput kretanja, kolizije i ulaznih informacija pomoću kojih će igrač upravljati svojim likom.



Slika 10 Meni prilikom kreiranja klase - biranje roditeljske klase

Izvor: Autor

Zatim se daje ime klase i pritisne gumb za kreiranje klase. Kada se klasa kreira, može se otvoriti u Visual Studiju.



Slika 11 Imenovanje klase

Izvor: Autor

5.3. Klasa unutar Visual Studija

Prilikom kreiranja klase unutar Unreal Enginea, generiraju se dvije datoteke koje se mogu uređivati. Jedna nazvana *header file* s ekstenzijom .h i jedna *implementation file* .cpp.

5.3.1. Header datoteke

Header datoteke pohranjuju unaprijed definirane funkcije. Sadrži definicije funkcija koje se mogu uključiti ili uvesti pomoću direktive pretprocesora #include. Ova direktiva pretprocesora govori prevoditelju da se datoteka zaglavlja mora obraditi prije kompilacije. Naprimjer, datoteka zaglavlja <iostream> u C++ sadrži definiciju ulazno-izlaznih funkcija [5].

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "MultiplayerGameCharacter.generated.h"

class UMultiplayerMovementComponent;

UCLASS(config=Game)
class MULTIPLAYERGAME_API AMultiplayerGameCharacter : public
ACharacter
{
	GENERATED_BODY()


	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category =
Camera, meta = (AllowPrivateAccess = "true"))
	class USpringArmComponent* CameraBoom;


	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category =
Camera, meta = (AllowPrivateAccess = "true"))
	class UCameraComponent* FollowCamera;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category =
Player, meta = (AllowPrivateAccess = "true"))
	class UTextRenderComponent* PlayerName;
public:
	AMultiplayerGameCharacter(const class FObjectInitializer&
ObjectInitializer);

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
Category=Camera)
	float BaseTurnRate;
```

```
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category =
Multiplayer)
	FString TeamName;


	UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
Category=Camera)
	float BaseLookUpRate;

protected:


	void OnResetVR();


	void MoveForward(float Value);


	void MoveRight(float Value);

	void TurnAtRate(float Rate);

	void LookUpAtRate(float Rate);


	void TouchStarted(ETouchIndex::Type FingerIndex, FVector
Location);


	void TouchStopped(ETouchIndex::Type FingerIndex, FVector
Location);

	void ReturnToMainMenu();

protected:

	virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

public:

	FORCEINLINE class USpringArmComponent* GetCameraBoom() const
{ return CameraBoom; }

	FORCEINLINE class UCameraComponent* GetFollowCamera() const {
return FollowCamera; }

	UFUNCTION(BlueprintCallable, Category = "Movement")
		UMultiplayerMovementComponent*
GetMultiplayerMovementComponent() const;

private:
```

```
	virtual void OnRep_PlayerState() override;

};
```

Kod 1 Header datoteka za klasu igrača

Izvor: Autor

5.3.2. Implementation datoteke

Kao što ime sugerira (Implementation (hrv. implementacija)), ove datoteke implementiraju funkcije neke klase. Takav način programiranja koristi se u C++ primarno zbog urednosti koda, tako da je implementacijski kod skriven u header datoteci, a dizajn je vidljiv u implementacijskoj datoteci.

```
#include "MultiplayerGameCharacter.h"
#include "HeadMountedDisplayFunctionLibrary.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "Components/InputComponent.h"
#include "MultiplayerMovementComponent.h"
#include "GameFramework/Controller.h"
#include "GameFramework/SpringArmComponent.h"
#include "MultiplayerGamePlayerState.h"
#include "Net/UnrealNetwork.h"
#include "Kismet/GameplayStatics.h"

//////////////////////////////////////////////////////////////////
////////
// AMultiplayerGameCharacter

AMultiplayerGameCharacter::AMultiplayerGameCharacter(const class
FObjectInitializer& ObjectInitializer) :
	Super(ObjectInitializer.SetDefaultSubobjectClass<UMultiplayer
MovementComponent>(ACharacter::CharacterMovementComponentName))
{


	GetCapsuleComponent()->InitCapsuleSize(35.f, 90.0f);
	BaseTurnRate = 45.f;
	BaseLookUpRate = 45.f;
	bUseControllerRotationPitch = false;
	bUseControllerRotationYaw = false;
	bUseControllerRotationRoll = false;

	GetCharacterMovement()->bOrientRotationToMovement = true;
	GetCharacterMovement()->RotationRate = FRotator(0.0f, 540.0f,
0.0f);
	GetCharacterMovement()->JumpZVelocity = 450.f;
	GetCharacterMovement()->AirControl = 0.2f;
```

```
	CameraBoom = 
CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
	CameraBoom->SetupAttachment(GetMesh());
	CameraBoom->TargetArmLength = 450.0f;
	CameraBoom->bUsePawnControlRotation = false;

	FollowCamera = 
CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
	FollowCamera->SetupAttachment(CameraBoom, 
USpringArmComponent::SocketName);
	FollowCamera->bUsePawnControlRotation = false;
}

// Input

void AMultiplayerGameCharacter::SetupPlayerInputComponent(class 
UInputComponent* PlayerInputComponent)
{

	check(PlayerInputComponent);
	PlayerInputComponent->BindAction("Jump", IE_Pressed, this, 
&ACharacter::Jump);
	PlayerInputComponent->BindAction("Jump", IE_Released, this, 
&ACharacter::StopJumping);

	PlayerInputComponent->BindAxis("MoveForward", this, 
&AMultiplayerGameCharacter::MoveForward);
	PlayerInputComponent->BindAxis("MoveRight", this, 
&AMultiplayerGameCharacter::MoveRight);

	PlayerInputComponent->BindAxis("Turn", this, 
&APawn::AddControllerYawInput);
	PlayerInputComponent->BindAxis("TurnRate", this, 
&AMultiplayerGameCharacter::TurnAtRate);
	PlayerInputComponent->BindAxis("LookUp", this, 
&APawn::AddControllerPitchInput);
	PlayerInputComponent->BindAxis("LookUpRate", this, 
&AMultiplayerGameCharacter::LookUpAtRate);


	PlayerInputComponent->BindTouch(IE_Pressed, this, 
&AMultiplayerGameCharacter::TouchStarted);
	PlayerInputComponent->BindTouch(IE_Released, this, 
&AMultiplayerGameCharacter::TouchStopped);


	PlayerInputComponent->BindAction("ResetVR", IE_Pressed, this, 
&AMultiplayerGameCharacter::OnResetVR);
	PlayerInputComponent->BindAction("ReturnToMainMenu", 
IE_Pressed, this, &AMultiplayerGameCharacter::ReturnToMainMenu);
}
```

```
void AMultiplayerGameCharacter::OnResetVR()
{
	UHeadMountedDisplayFunctionLibrary::ResetOrientationAndPositi
on();
}

void AMultiplayerGameCharacter::TouchStarted(ETouchIndex::Type
FingerIndex, FVector Location)
{
		Jump();
}

void AMultiplayerGameCharacter::TouchStopped(ETouchIndex::Type
FingerIndex, FVector Location)
{
		StopJumping();
}

void AMultiplayerGameCharacter::TurnAtRate(float Rate)
{

	AddControllerYawInput(Rate * (BaseTurnRate/2) * GetWorld()-
>GetDeltaSeconds());
}

void AMultiplayerGameCharacter::LookUpAtRate(float Rate)
{
	AddControllerPitchInput(Rate * (BaseLookUpRate/2) *
GetWorld()->GetDeltaSeconds());
}

void AMultiplayerGameCharacter::MoveForward(float Value)
{
	if ((Controller != nullptr) && (Value != 0.0f))
	{
		const FRotator Rotation = Controller-
>GetControlRotation();
		const FRotator YawRotation(0, Rotation.Yaw, 0);


		const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
		AddMovementInput(Direction, Value);
	}
}

void AMultiplayerGameCharacter::MoveRight(float Value)
{
	if ((Controller != nullptr) && (Value != 0.0f))
	{
```

```
            const FRotator Rotation = Controller-
>GetControlRotation();
            const FRotator YawRotation(0, Rotation.Yaw, 0);


            const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

            AddMovementInput(Direction, Value);
      }
}
      void AMultiplayerGameCharacter::ReturnToMainMenu() {
            FString LevelName = "MainMenuMap";

            UGameplayStatics::OpenLevel(GetWorld(),
FName(*LevelName), false, "");
      }
      void AMultiplayerGameCharacter::OnRep_PlayerState() {
            Super::OnRep_PlayerState();

            APlayerState* OwningPlayerState = GetPlayerState();
            if (OwningPlayerState != nullptr) {
                  AMultiplayerGamePlayerState*
OwningMultiplayerGamePlayerState =
Cast<AMultiplayerGamePlayerState>(OwningPlayerState);
                  if (OwningMultiplayerGamePlayerState != nullptr) {
                        TeamName = OwningMultiplayerGamePlayerState-
>Team;


                        if (TeamName.Len() > 0) {
                              UMaterialInstanceDynamic*
OwningPlayerMaterial = UMaterialInstanceDynamic::Create(GetMesh()-
>GetMaterial(0), this);

                              if (TeamName.Equals("cowboys")) {
                                    OwningPlayerMaterial-
>SetVectorParameterValue("BodyColor", FLinearColor::Red);
                              }
                              else if (TeamName.Equals("aliens")) {
                                    OwningPlayerMaterial-
>SetVectorParameterValue("BodyColor", FLinearColor::Blue);
                              }

                              GetMesh()->SetMaterial(0,
OwningPlayerMaterial);
                        }
                  }
            }


}
```

```
	UMultiplayerMovementComponent*
AMultiplayerGameCharacter::GetMultiplayerMovementComponent() const
	{
		return
static_cast<UMultiplayerMovementComponent*>(GetCharacterMovement()
);
	}
```

Kod 2 Implementacijska datoteka s programom za igrača

Izvor: Autor

5.3.3. Klasa za povezivanje igre sa serverom

Igra u Unreal Engineu mora imati *Game Mode* klasu koja definira zadane vrijednosti poput klase igrača pri stvaranju u svijet, klase za HUD (*Heads Up Display)*, itd. S obzirom da se ovdje radi o igri za više igrača, u njoj će se također definirati i opisati funkcije za upravljanje inicijalizacijom igre te uništavanje procesa kada igra završi.

Također će imati *backfill* funkciju, koja omogućava da se dodatni igrači priključe igri prije nego ona počne i koja uklanja igrače kad im se prekine veza.

```
#pragma once

#include "CoreMinimal.h"
#include "GameLiftServerSDK.h"
#include "MultiplayerGameCharacter.h"
#include "GameFramework/GameModeBase.h"
#include "Runtime/Online/HTTP/Public/Http.h"
#include "MultiplayerGameGameMode.generated.h"

UENUM()
enum class EUpdateReason : uint8
{
	NO_UPDATE_RECEIVED,
	BACKFILL_INITIATED,
	MATCHMAKING_DATA_UPDATED,
	BACKFILL_FAILED,
	BACKFILL_TIMED_OUT,
	BACKFILL_CANCELLED,
	BACKFILL_COMPLETED
};

USTRUCT()
struct FStartGameSessionState
{
	GENERATED_BODY();
```

```
	UPROPERTY()
		bool Status;
	UPROPERTY()
		FString MatchmakingConfigurationArn;

	TMap<FString, Aws::GameLift::Server::Model::Player>
PlayerIdToPlayer;

	FStartGameSessionState() {
		Status = false;
	}
};

USTRUCT()
struct FUpdateGameSessionState {
	GENERATED_BODY();


	UPROPERTY()
		EUpdateReason Reason;

	TMap<FString, Aws::GameLift::Server::Model::Player>
PlayerIdToPlayer;

	FUpdateGameSessionState() {
		Reason = EUpdateReason::NO_UPDATE_RECEIVED;
	}
};

USTRUCT()
struct FProcessTerminateState {
	GENERATED_BODY();

	UPROPERTY()
		bool Status;

	long TerminationTime;

	FProcessTerminateState() {
		Status = false;
		TerminationTime = 0L;
	}


};

USTRUCT()
struct FHealthCheckState {
	GENERATED_BODY();

	UPROPERTY()
		bool Status;
```

```
	FHealthCheckState() {
		Status = false;
	}

};

UCLASS(minimalapi)
class AMultiplayerGameGameMode : public AGameModeBase
{
	GENERATED_BODY()

public:
	AMultiplayerGameGameMode();
	virtual void PreLogin(const FString& Options, const FString&
Address, const FUniqueNetIdRepl& UniqueId, FString& ErrorMessage)
override;

	virtual void Logout(AController* Exiting) override;

	AMultiplayerGameCharacter* Player;
public:
	UPROPERTY()
		FTimerHandle CountDownUntilGameOverHandle;

	UPROPERTY()
		FTimerHandle EndGameHandle;

	UPROPERTY()
		FTimerHandle PickAWinningTeamHandle;

	UPROPERTY()
		FTimerHandle HandleProcessTerminationHandle;

	UPROPERTY()
		FTimerHandle HandleGameSessionUpdateHandle;

	UPROPERTY()
		FTimerHandle SuspendBackfillHandle;

	UFUNCTION(BlueprintCallable,Category="MultiplayerGameMode")
		void PickAWinningTeam(FString WinningTeam);


protected:
	virtual void BeginPlay() override;



	virtual FString InitNewPlayer(APlayerController*
NewPlayerController, const FUniqueNetIdRepl& UniqueId, const
FString& Options, const FString& Portal) override;

private:
```

```
	FHttpModule* HttpModule;

	UPROPERTY()
		FStartGameSessionState StartGameSessionState;

	UPROPERTY()
		FUpdateGameSessionState UpdateGameSessionState;

	UPROPERTY()
		FProcessTerminateState ProcessTerminateState;

	UPROPERTY()
		FHealthCheckState HealthCheckState;

	UPROPERTY()
		FString ApiUrl;

	UPROPERTY()
		FString ServerPassword;

	UPROPERTY()
		int RemainingGameTime;

	UPROPERTY()
		bool GameSessionActivated;

	UPROPERTY()
		FString LatestBackfillTicketId;

	UPROPERTY()
		bool WaitingForPlayersToJoin;

	UPROPERTY()
		int TimeSpentWaitingForPlayersToJoin;

	TMap<FString, Aws::GameLift::Server::Model::Player>
ExpectedPlayers;

	UFUNCTION()
		void CountDownUntilGameOver();

	UFUNCTION()
		void EndGame();



	UFUNCTION()
		void HandleProcessTermination();

	UFUNCTION()
		void HandleGameSessionUpdate();

	UFUNCTION()
```

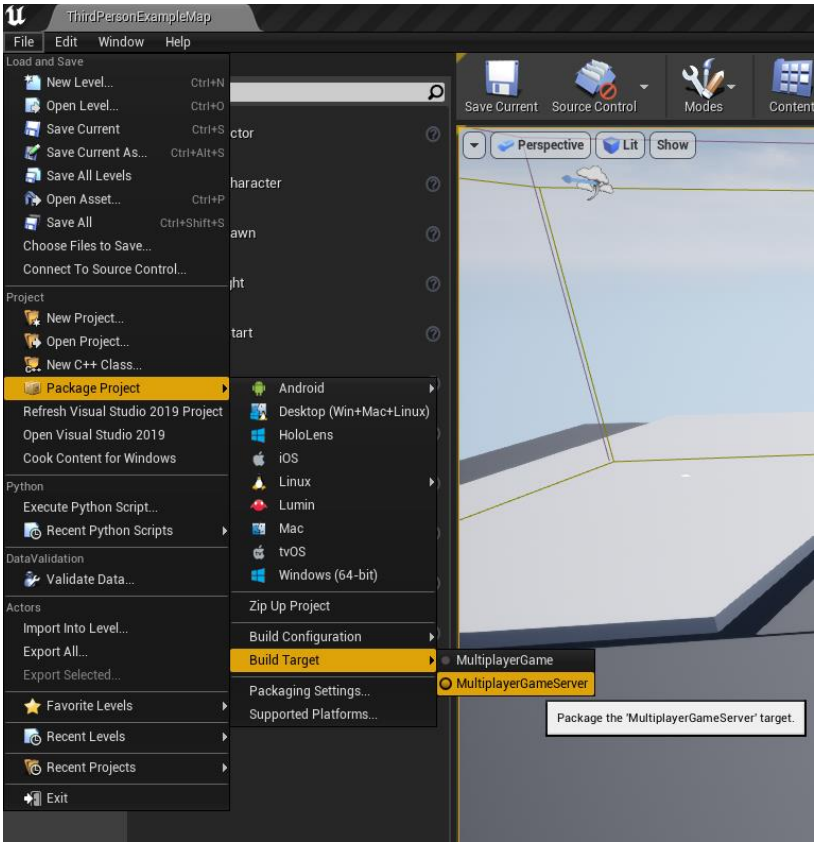
```
            void SuspendBackfill();

      FString CreateBackfillRequest(FString GameSessionArn, FString
MatchmakingConfigurationArn, TMap<FString,
Aws::GameLift::Server::Model::Player> Players);
      bool StopBackfillRequest(FString GameSessionArn, FString
MatchmakingConfigurationArn, FString TicketId);
      void OnRecordMatchResultResponseReceived(FHttpRequestPtr
Request, FHttpResponsePtr Response, bool bWasSuccessful);
};
```

Kod 3 Header datoteka za GameMode igre

Izvor: Autor

6. POSTAVLJANJE SERVERA PREKO AWS GAMELIFT KONZOLE

Da bi se server mogao pokrenuti preko AWS servera, mora se prvo izgraditi serverska izvršna datoteka u Unreal Engineu i učitati na AWS GameLift. Za izgradnju serverske datoteke, prilikom pakiranja projekta mora se odabrati cilj izgradnje (engl. *Build target*).



Slika 12 Postavljanje cilja izgradnje na serversku datoteku

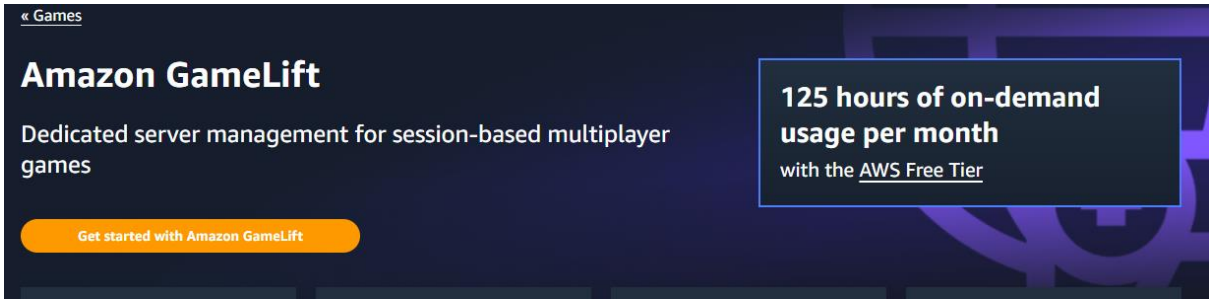
Izvor: Autor

Nakon što se datoteka izgradila(engl. *build)*, mora se učitati na AWS preko komandne linije. Za to se koristi komanda: aws gamelift upload-build --name MultiplayerGame --build-version 1.0.0 --build-root [path-to-server-build-directory] --operating-system WINDOWS_2016 --region [region-code]. Naravno, „[path-to-server-build-directory]“ se mijenja s putem prema

direktoriju gdje se nalazi izvršna datoteka servera, a „[region-code]“ s regijom koja se odabrala na AWS Gamelift konzoli.

6.1. GameLift konzola

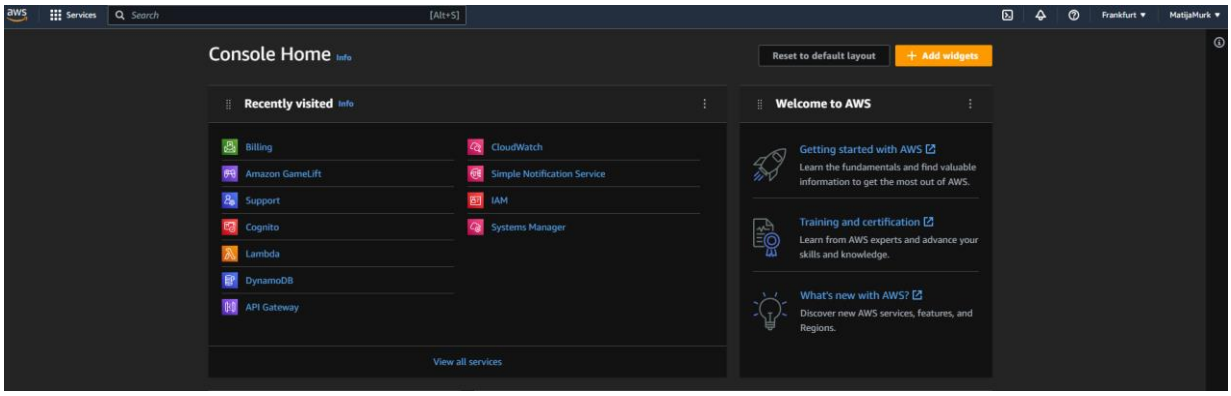
Kako bi se koristila Amazon GameLift konzola, mora se napraviti račun na stranici. Nakon njegova kreiranja, dobije se besplatnih 125 sati on-demand korištenja. To znači da će prvih 125 sati, kada je server upaljen, biti besplatno.



Slika 13 Amazon GameLift početna stranica

Izvor: Autor

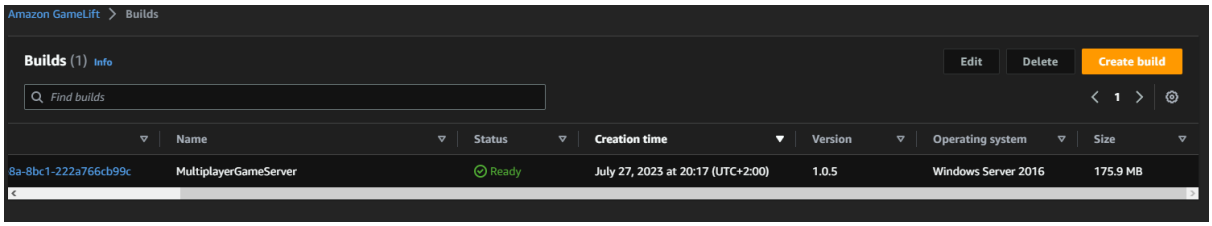
S kreiranim računom dobije se pristup raznim servisima, koji će biti korišteni prilikom konfiguracije za potrebe ovog projekta.



Slika 14 GameLift nadzorna ploča

Izvor: Autor

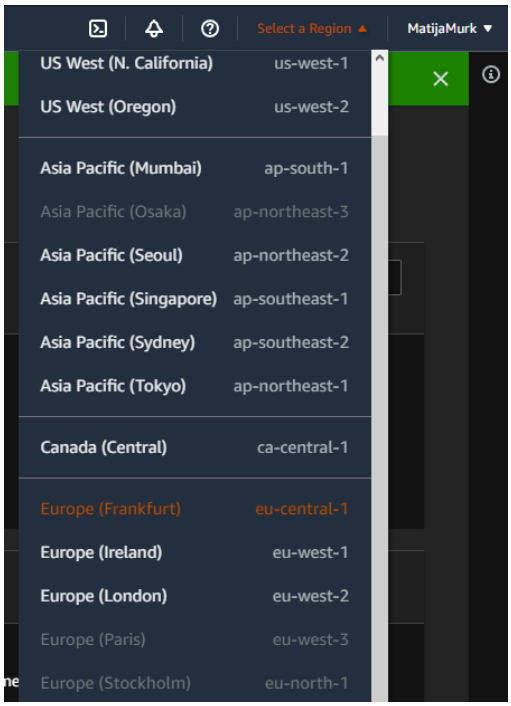
Za početak se može otvoriti nadzorna ploča gdje se vidi server koji se uploadao u prethodnom koraku.



Slika 15 Build servera spremnog za korištenje

Izvor: Autor

Nakon što je server spreman može se kreirati flota. *Fleet* ili flota je skup računalnih instanci u jednoj regiji koje pokreću jednu verziju poslužitelja za igre [6]. Preko flote će se kontrolirati je li server upaljen i koliko instanci je upaljeno. U ovom slučaju, regija se postavlja na eu-central-1 jer se nalazi u Frankfurtu, što je od ponuđenih lokacija najbliže Hrvatskoj.



Slika 16 Odabir regije gdje će se pokretati server

Izvor: Autor

6.2. Matchmaking rule set

Prije nego što se igra pokrene i više igrača spoji u istu sesiju, nužno je definirati neka pravila prema kojima će server spajati igrače. Za ovaj projekt koristit će se pravila koja se mogu naći u dokumentaciji Amazon GameLifta, ali će se broj igrača promijeniti na maksimalno 2 i minimalno 1 [7].

Pravila izgledaju ovako:

```
{
    "name": "aliens_vs_cowboys",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "skill",
        "type": "number",
        "default": 10
    }],
    "teams": [{
        "name": "cowboys",
        "maxPlayers": 2,
```

```
            "minPlayers": 1
        }, {
            "name": "aliens",
            "maxPlayers": 2,
            "minPlayers": 1
        }],
        "rules": [{
            "name": "FairTeamSkill",
            "description": "The average skill of players in each team
is within 10 points from the average skill of all players in the
match",
            "type": "distance",
            // get skill values for players in each team and average
separately to produce list of two numbers
            "measurements": [
"avg(teams[*].players.attributes[skill])" ],
            // get skill values for players in each team, flatten into
a single list, and average to produce an overall average
            "referenceValue":
"avg(flatten(teams[*].players.attributes[skill]))",
            "maxDistance": 10 // minDistance would achieve the
opposite result
        }, {
            "name": "EqualTeamSizes",
            "type": "comparison",
            "measurements": [ "count(teams[cowboys].players)" ],
            "referenceValue": "count(teams[aliens].players)",
            "operation": "=" // other operations: !=, <, <=, >, >=
        }],
        "expansions": [{
            "target": "rules[FairTeamSkill].maxDistance",
            "steps": [{
                "waitTimeSeconds": 5,
                "value": 50
            }, {
                "waitTimeSeconds": 15,
                "value": 100
            }]
        }]
}
```

Kod 4 Pravila igre

Izvor: Autor

Ono što je važno znati jest da se pravilima definira maksimalan i minimalan broj igrača, razina vještine i međusobna udaljenost igrača. Zatim se igrači uspoređuju i odabire se najbolji mogući spoj igrača.

6.3. Baza podataka DynamoDB

Za ovaj projekt koristit će se DynamoDB baza podataka zbog toga što je integrirana u GameLift konzolu, što će znatno olakšati upravljanje podacima. DynamoDB je hostirana NoSQL baza podataka. Ona pruža dosljednu izvedbu čak i dok se povećava. Kompaktan, jednostavan API podržava i osnovni pristup ključ-vrijednost i sofisticiranije uzorke upita [8].

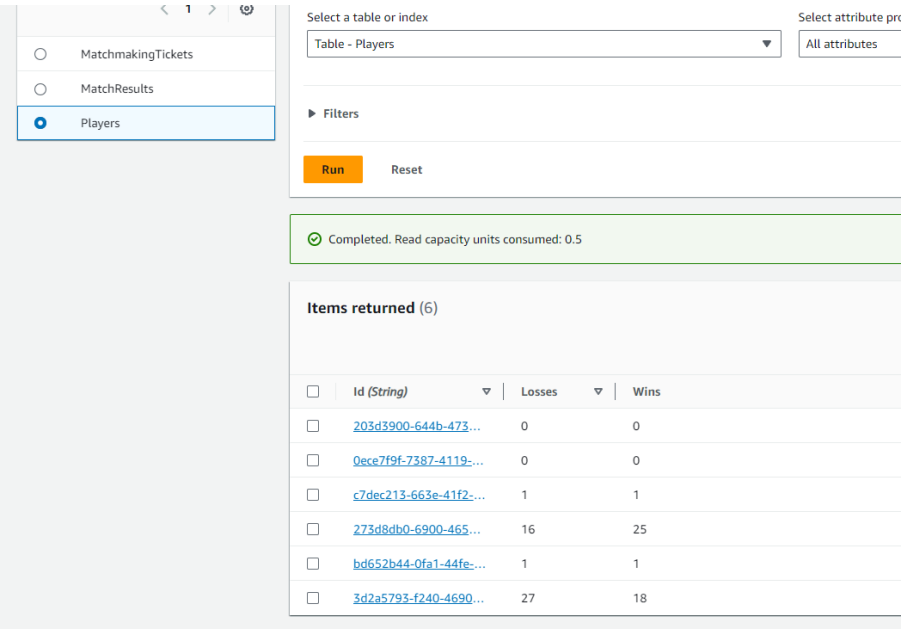
DynamoDB je izvrstan izbor za aplikacije s velikom količinom podataka i strogim ograničenjima latencije. Kako količina podataka raste, JOIN-ovi i složene SQL operacije mogu usporiti upite. AWS Lambda koristi se za pokretanje aplikacija bez poslužitelja [8]. Kao reakcija na okidače događaja, AWS Lambda pruža automatsko skaliranje. DynamoDB je dostupan putem HTTP API-ja i upravlja autentifikacijom i autorizacijom putem IAM uloga, što ga čini idealnim za razvoj aplikacija bez poslužitelja [8].

Skupovi podataka imaju dobro poznate uzorke lakog pristupa. DynamoDB-ovi jasni obrasci pristupa ključ-vrijednost čine ga brzim, pouzdanim rješenjem za stvaranje i posluživanje prijedloga korisnicima [8].

6.4. Tablice

Za ovaj su projekt potrebne tri tablice: jedna za spremanje korisnika koji se registriraju u igri, druga za spremanje svakog uparivanja igrača i treća za spremanje rezultata svake igre.

Kod tablice igrača kreiraju se tri stupca, ID igrača koji mu se dodjeljuje kod kreiranja novog računa, broj pobjeda i broj poraza.



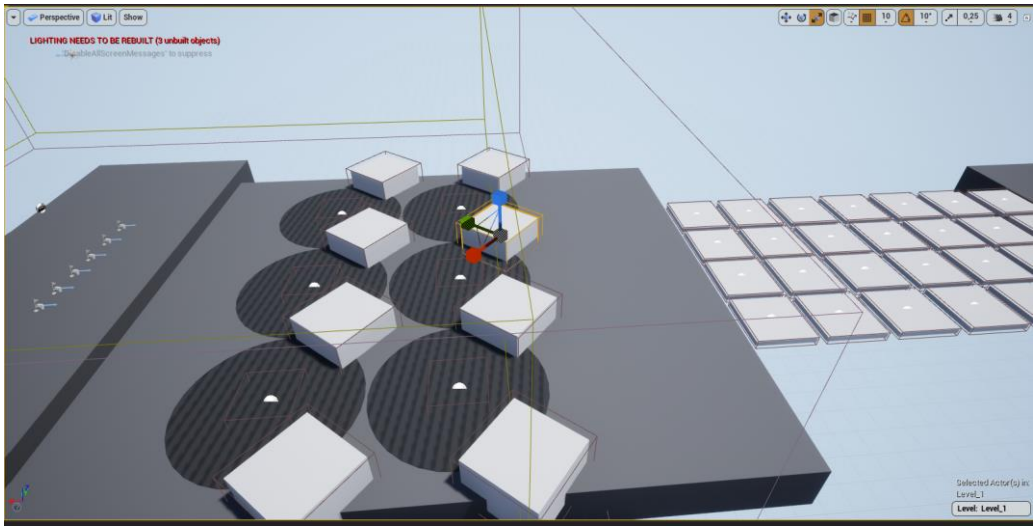
Slika 17 Tablica Players u DynamoDB bazi podataka

Izvor: Autor

7. POSTAVLJANJE I KREIRANJE IGRE U UNREAL ENGINEU

7.1. Kreiranje razina i prepreka

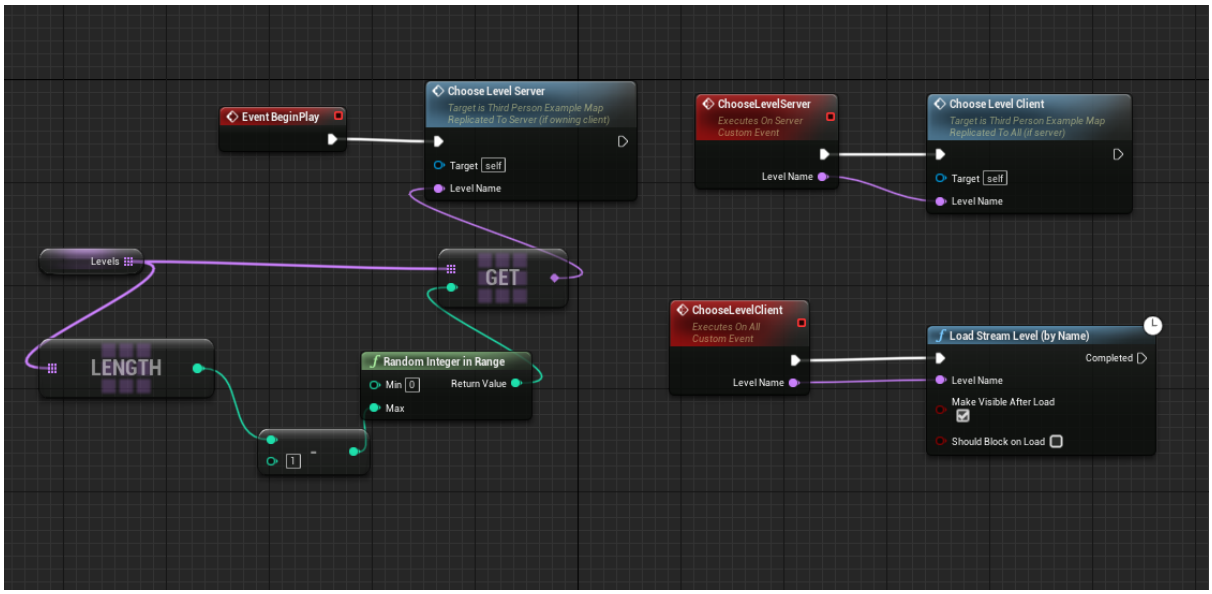
Kod ovog projekta razine će se nasumično odabrati kada se igra pokrene. Zbog toga, sve razine ustvari moraju biti podrazine, koje neće biti učitane tako dugo dok ih se ne pozove na učitavanje unutar koda. Pritom će se koristiti Unrealovi Nacrti (engl. *Blueprints*) da bi se ubrzao proces kreiranja razina i da bi se značajke mogle brže testirati.



Slika 18 Primjer razine 1

Izvor: Autor

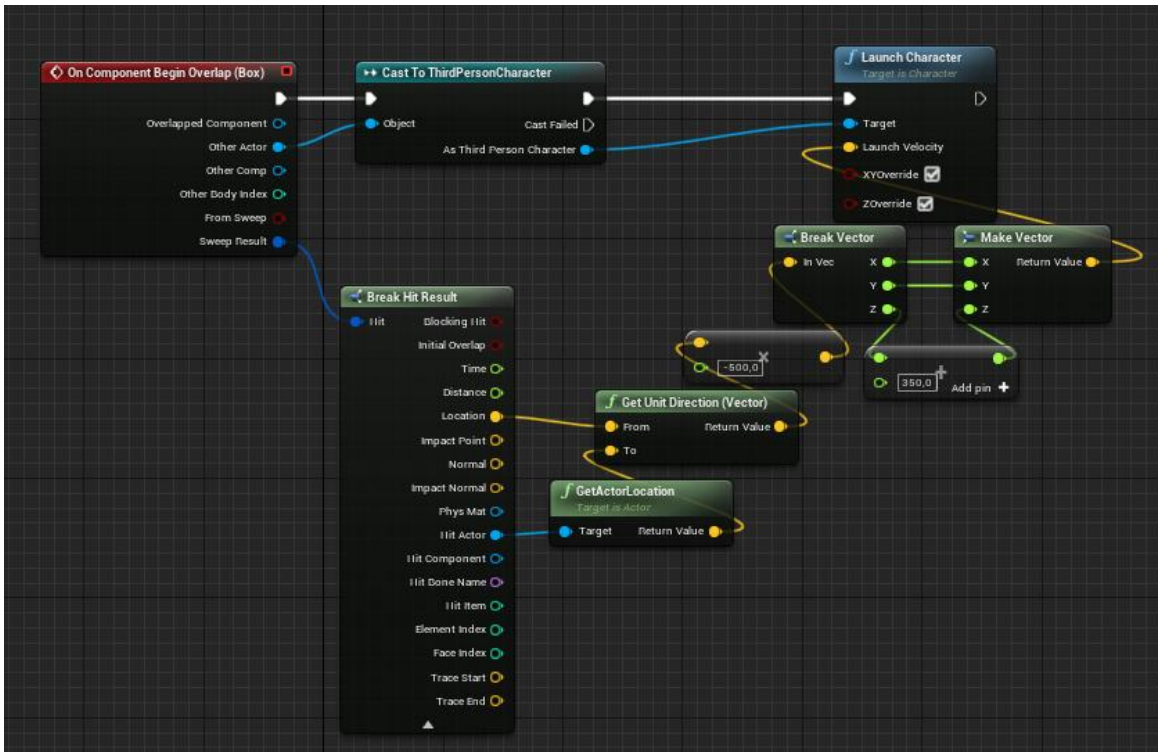
Razine se moraju nasumično odabrati unutar Razinskog Nacrta. S obzirom da se radi o igri s više igrača koja se igra preko servera, nije dovoljno pozvati funkciju samo jednom, već se mora pozvati jednom za server i jednom za klijenta. U suprotnom, dolazi do desinkronizacije, npr. igraču 1 se učita razina 2 dok se igraču 2 učita razina 1.



Slika 19 Nacrt koji nasumično bira razinu na kojoj će se igrati

Izvor: Autor

Prepreke će biti prilično jednostavne. Uglavnom se baziraju na ometanju kretanja igrača. Naprimjer, kocka koja ima „elastičnu“ površinu pa tako igrača, kada je dotakne, odbija u suprotni smjer.

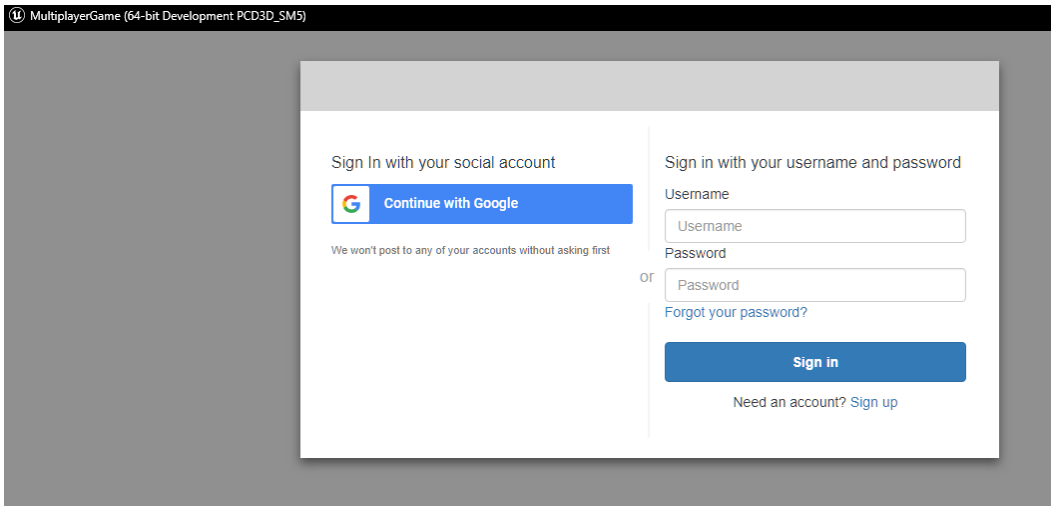


Slika 20 Prikaz koda elastične kocke

Izvor: Autor

7.2. Registracija i prijava igrača

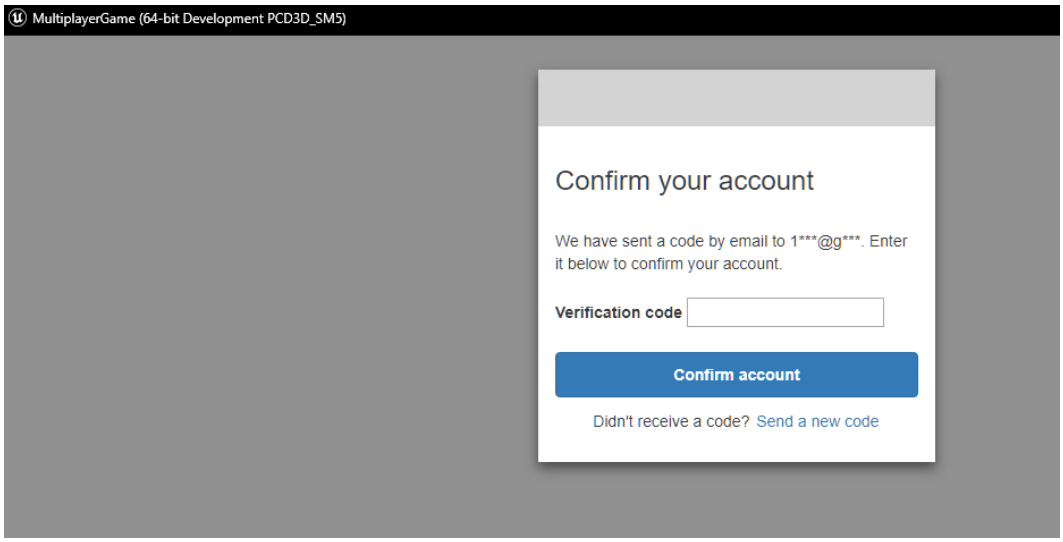
Kako bi se igrači mogli spojiti u istu igru i kako bi mogli imati svoju statistiku pobjeda i poraza, moraju se prijaviti na svoj račun ili kreirati novi ako prvi put pokreću igru. Ovo se rješava pomoću Unrealovog Widget sistema, gdje se može implementirati browser. On će otvoriti stranicu Amazon Cognita koji koristimo za prijavu i registraciju.



Slika 21 Početna stranica kod pokretanja igre

Izvor: Autor

Kada korisnik upiše tražene podatke i lozinku, šalje se poruka na adresu e-pošte korisnika gdje dobije jednokratni kod za potvrdu svog računa. Također, korisnik se može prijaviti svojim Google računom. U tom slučaju korisnik ne dobije verifikacijski kod, već se račun potvrđuje automatski.



Slika 22 Potvrđivanje računa nakon što se korisnik registrirao

Izvor: Autor

S time završava proces kreacije računa i korisnik se dodaje, s jedinstvenim ID-om, u tablicu Players u bazi podataka.

7.3. Glavni izbornik

Glavni izbornik je čvorište gdje pojedini igrač može vidjeti svoju statistiku u lijevom gornjem kutu i ping u desnom gornjem kutu. Također, igrač ima izbor pokretanja igre i pretraživanja dostupnih mečeva pritiskom na *Join Game* tipku ili izlazak iz igre pritiskom na *Exit Game* tipku.

Slika 23 Glavni izbornik igre

Izvor: Autor

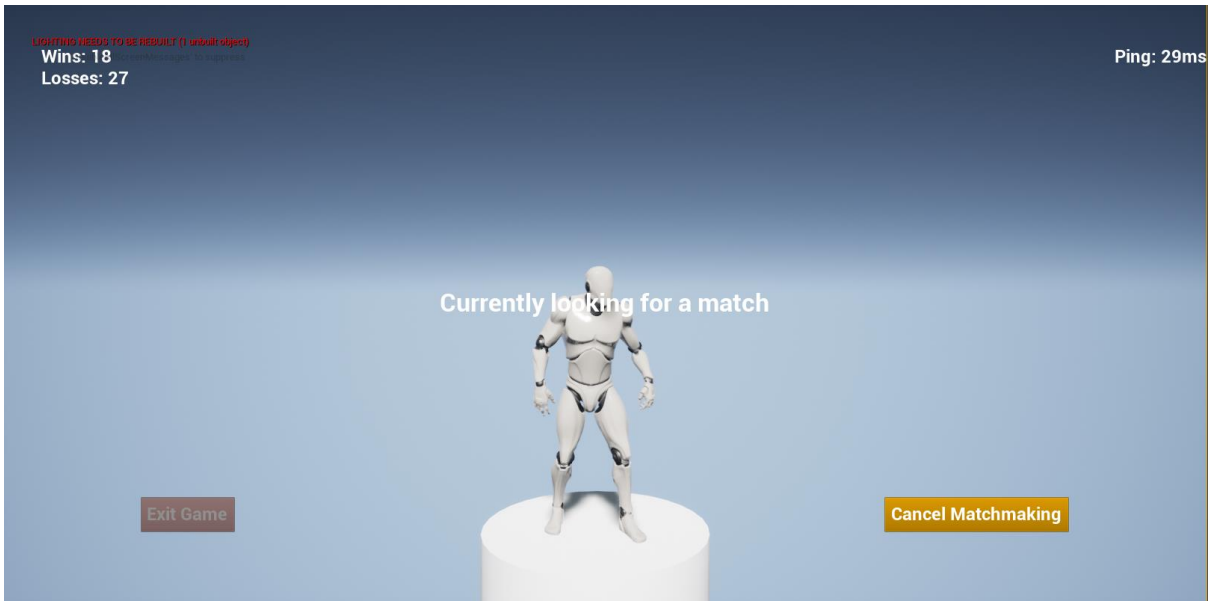
Ako igrač odluči pokrenuti pretraživanje dostupnih mečeva, šalje se njegov *ticketId* s njegovim podacima o vještini i lokaciji te zahtjevom za početak igre s drugim igračima.

```
void
UMainMenuWidget::OnStartMatchmakingResponseReceived(FHttpRequestPt
r Request, FHttpResponsePtr Response, bool bWasSuccessful) {
      if (bWasSuccessful) {
            TSharedPtr<FJsonObject> JsonObject;
            TSharedRef<TJsonReader<>> Reader =
TJsonReaderFactory<>::Create(Response->GetContentAsString());
```

```
            if (FJsonSerializer::Deserialize(Reader, JsonObject)) {
                  if (JsonObject->HasField("ticketId")) {
                        FString MatchmakingTicketId = JsonObject-
>GetStringField("ticketId");

                        UGameInstance* GameInstance =
GetGameInstance();
                        if (GameInstance != nullptr) {
                              UMultiplayerGameInstance*
MultiplayerGameInstance =
Cast<UMultiplayerGameInstance>(GameInstance);
                              if (MultiplayerGameInstance != nullptr)
{
                                    MultiplayerGameInstance-
>MatchmakingTicketId = MatchmakingTicketId;

                                    GetWorld()-
>GetTimerManager().SetTimer(PollMatchmakingHandle, this,
&UMainMenuWidget::PollMatchmaking, 1.0f, true, 1.0f);
                                    SearchingForGame = true;

                                    UTextBlock* ButtonTextBlock =
(UTextBlock*)MatchmakingButton->GetChildAt(0);
                                    ButtonTextBlock-
>SetText(FText::FromString("Cancel Matchmaking"));
                                    MatchmakingEvent-
>SetText(FText::FromString("Currently looking for a match"));
                              }
                        }
                  }
            }
      }
      MatchmakingButton->SetIsEnabled(true);
}
```

Kod 5 Programski kod kada igrač pritisne tipku za početak igre

Izvor: Autor



Slika 24 Pretraživanje slobodnih mečeva

Izvor: Autor

7.4. Dva igrača su se uspješno spojila

Ako se pozitivna povratna informacija vrati, odnosno ako je drugi igrač nađen, kreće njihovo spajanje u istu sesiju igre (engl. *Game session*). Svakom igraču nasumično se dodjeljuje njegov tim, a boja lika kojim igrač upravlja mijenja se ovisno o timu kojem je dodijeljen. U ovom slučaju, s obzirom da su moguća samo dva igrača jer je tako navedeno u pravilima igre zadanima na serveru, postoje samo dva tima: *aliens* i *cowboys*. Tim *aliens* poprima plavu boju, dok tim *cowboys* poprima crvenu boju. Nakon što su se timovi dodijelili igračima, nasumično se bira na kojoj razini će se igrati.

Slika 25 Dva igrača u istom meču

Izvor: Autor

Igrači imaju zadatak proći sve prepreke i doći do cilja na drugom kraju mape. Igrač koji prvi dođe do cilja proglašava se pobjednikom. Sesija igre tada se uništava i igrači se vraćaju na glavni izbornik.

8. ZAKLJUČAK

Izrada videoigara ni u kojem slučaju nije jednostavan posao. Iako u posljednjih nekoliko godina raste dostupnost softvera, još uvijek je taj zadatak vrlo složen za jednog čovjeka, pa čak i za mali tim ako treba napraviti zahtjevniju igru.

Kada se u izradu igre uključe i funkcije povezivanja na mrežu i igranje preko servera u stvarnom vremenu, zadatak se dodatno usložnjava. Stoga u većim timovima postoje posebni pod-timovi, koji rade samo na tim specifičnim zadacima. Naravno, to manjim razvijačima igara nije dostupno, pa jedan čovjek mora odraditi posao za troje. Iako to zvuči naporno, izrada videoigara je posao koji se isplati raditi. Ne postoji ni jedan drugi medij gdje bi bilo moguće prikazati toliko različitih umjetničkih, programerskih i matematičkih vještina u jednom proizvodu. Razvoj igara napreduje nevjerojatnom brzinom i ne mijenja samo svijet igara, već i filmske industrije. Primjerice, televizijska znanstvenofantastična serija Mandalorian koristi Unreal Engine, ali i svijet općenito. Tvrtka koja se bavi izradom grafičkih čipova, Nvidia, veliki je razvijač umjetne inteligencije, koja sve više osvaja svijet.

Cilj ovog rada bio je dokazati da jedan čovjek može napraviti igru za više igrača, iako to može biti dugi proces, koji ostavlja malo vremena za dizajn same igre. Razloženo je koje su važnosti prilikom spajanja dva igrača preko vanjskog servera, koja pravila se postavljaju kada se traže igrači za igru i koje su razlike u programiranju igra za jednog igrača ili za više njih.

LITERATURA

[1] Steven L. Kent The Ultimate History of Videogames, New York: Three Rivers Press; 2001.

[2] https://hub.yamaha.com/audio/gaming/a-brief-history-of-video-games/ (20.5.2023.)

[3] https://www.pulsecollege.com/what-is-game-development-everything-you-need-to-know/ (25.5.2023.)

[4] Nenad Breslauer, Formalni elementi dizajna, pptx

[5] https://www.simplilearn.com/tutorials/cpp-tutorial/guide-to-understand-cpp-header-files (26.5.2023.)

[6] https://aws.amazon.com/gamelift/faq/ (25.8.2023.)

[7] https://docs.aws.amazon.com/gamelift/latest/flexmatchguide/match-examples.html (27.8.2023.)

[8] https://www.dynamodbguide.com/what-is-dynamo-db/ (31.8.2023.)

POPIS KODOVA

Kod 1 Header datoteka za klasu igrača 23
Kod 2 Implementacijska datoteka s programom za igrača 27
Kod 3 Header datoteka za GameMode igre 31
Kod 4 Pravila igre 36
Kod 5 Programski kod kada igrač pritisne tipku za početak igre 44

POPIS SLIKA

Slika 1 Unreal Engine logo 12
Slika 2 Preuzimanje Unreal Enginea 14
Slika 3 Otvaranje Visual Studio projekta 15
Slika 4 Građenje Unreal Enginea 15
Slika 5 Pokretanje aplikacije 16
Slika 6 Kreiranje projekta unutar Unreala 17
Slika 7 Preuzimanje Amazon GameLift SDK-a 18
Slika 8 Datoteke za izgradnju projekta kao servera i datoteke za klijenta 18
Slika 9 Gumb za odabir kreiranja klase u Unrealu 19
Slika 10 Meni prilikom kreiranja klase - biranje roditeljske klase 20
Slika 11 Imenovanje klase 20
Slika 12 Postavljanje cilja izgradnje na serversku datoteku 32
Slika 13 Amazon GameLift početna stranica 33
Slika 14 GameLift nadzorna ploča 33
Slika 15 Build servera spremnog za korištenje 34
Slika 16 Odabir regije gdje će se pokretati server 35
Slika 17 Tablica Players u DynamoDB bazi podataka 38
Slika 18 Primjer razine 1 39
Slika 19 Nacrt koji nasumično bira razinu na kojoj će se igrati 40
Slika 20 Prikaz koda elastične kocke 41
Slika 21 Početna stranica kod pokretanja igre 42
Slika 22 Potvrđivanje računa nakon što se korisnik registrirao 42
Slika 23 Glavni izbornik igre 43
Slika 24 Pretraživanje slobodnih mečeva 45
Slika 25 Dva igrača u istom meču 46