# Upotreba Unreal Engine-a u razvoju 3D računalne igre

**Ćurić, Dino**



**SVEUČILIŠTE JOSIPA JURJA STROSMSMAYERA U OSIJEKU**
**ELEKTROTEHNIČKI FAKULTET**

**Sveučilišni studij**

# Upotreba Unreal Engine-a u razvoju 3D računalne igre

**Diplomski rad**

**Dino Ćurić**

**Osijek,2016**

# Sadržaj

# 1. UVOD

Početak razvoja računalne grafike datira još od 1950. godine kada je napravljeno računalo Whirlwind. To je prvo računalo koje je moglo u stvarnom vremenu prikazivati jednostavnu grafiku. Samu riječ, Računalna grafika, ili RG, upotrijebio je William Fetter, Boeing 1960. godine. Jedan od najpoznatijih prikaza u ranijoj povijesti računalne grafike je bila upravo ljudska figura koja je nazvana "*Boeing Man*", a koju je sam Fatter nazvao "Prvi Čovijek". Od tada dogodio se, a i dalje se događa i odvija, veliki napredak na ovome području. Prvi veći napredak može se vidjeti već 1962. godine kada je Ivan Sutherland stvorio program *Sketchpad* pomoću kojega je bilo moguće interaktivno stvarati grafički sadržaj. Još neka poznatija računala koja su vrijedna spomena su TX-2 iz 1959.godine, pa onda tijekom 1970 tih i 1980tih kada su računala postala sve moćnija, a s time i računalna grafika, Commodore Amiga i macintosh. 3D računalna grafika, što je ujedno i jedno od najvažnijih tema vezana za ovaj rad, postala je moguća krajem 1980. godine kada su se pojavila CGI(*Computer-generated imagery)* računala. Ona su kasnije korištena u animaciji kratkih filmova u Pixaru. Grafika kao takva je jedna od 5 ključnih elemenata multimedijske tehnologije. U samim računalnim igrama postala je poznata 1990-tih. godina kada je izdana jedna od prvih cjelovitih 3D igara, Queke. U drugom poglavlju ovoga rada objašnjene su osnove 3D računalne grafike i metode koje se koriste kao što su sjenčanje(*Shading*), mapiranje, *anti-aliasing* i slično. Sljedeće poglavlje obuhvaća osnove animacije općenito, a posebno je objašnjena i računalna animacija. U četvrtom poglavlju analizirana je fizika same 3D igre, njezin utjecaj i izvedba. Nakon toga opisan je Unreal Engine i njegov razvoj i na kraju izvedba same igre u UDK( *Unreal development kit*).

## 1.1. Zadatak diplomskog rada

Za izradu 3D računalne igre potrebna su određena programska okruženja i zahtjevi kako bi se ona realizirala na najbolji mogući način sukladno sa vještinama onoga tko ju radi. Zahtjev ovoga rada je opisati razvojno okruženje za izradu igara UDK(*Unreal Development Kit*) te načine rada u *Unreal Engine*-u. Također je potrebno opisati osnove 3D računalne grafike, fizike, animacije. U praktičnom dijelu rada potrebno je napraviti 3D računalnu igru.

# 2. 3D RAČUNALA GRAFIKA

Računalna grafika je umjetnost crtanja slike na zaslonima računala uz pomoć programa. To uključuje proračune, izradu i manipulaciju podacima. Drugim riječima, možemo reći da je računalna grafika alat za renderiranje, stvaranje i manipulaciju slika. Izraz računalna grafika uključuje gotovo sve što nije tekst i zvuk, a što je stvoreno pomoću računala. Danas gotovo sva računala koriste neku vrstu grafike i korisnicima je omogućeno korištenje ikona i slika umjesto same tipkovnice.

Termin računalna grafika ima nekoliko značenja:

- reprezentacija i manipulacija slikovnih podataka pomoću računala
- razne tehnologije koje se koriste pri izradi i manipulaciji takvih slikovnih podataka
- slike proizvedene na taj način
- potkategorija računalne znanosti koja proučava metode digitalnog sintetiziranja i manipuliranja vizualnim sadržajem

Postoje dvije vrste računalne grafike:

- rasterska grafika – gdje je svaki piksel posebno definiran
- vektorska grafika – gdje se upotrebljavaju matematičke formule za crtanje linija i oblika.

Upotrebom vektora dobiju se oštre grafike i često manji računalni dokumenti, ali kada su vektorske grafike komplicirane, tada je potrebno puno više vremena da se učitaju i mogu imati veće računalne dokumente nego rasterski ekvivalenti. Računalna grafika je grafika stvorena pomoću računala, ili generalno gledajući, to je reprezentacija manipulacije slikovnih podataka pomoću računala. [1] U 2D sistemima, koriste se samo dvije koordinate, X i Y, ali u 3D-u dodaje se još jedna dodatna, Z koordinata. 3D grafičke tehnike i njihova primjena su od temeljne važnosti za zabavu , igre i industriju čije je oblikovanje računalno potpomognuto. To je područje koje se nastavlja istraživati u znanstvenoj vizualizaciji. Nadalje, 3D grafičke komponente su sada dio gotovo svakog osobnog računala, iako je tradicionalno namijenjeno grafički zahtjevnijim programima kao što su igre , oni se sve više koristi od strane drugih aplikacija.

3D grafika se stvara pomoću geometrijskih oblika, poligona ili NURBS (*Non-uniform rational beizer spline*) krivulja. Koristi se za izradu trodimenzionalnih objekata i scena za upotrebu u različitim medijima poput filma, televizije, igara i vizualnih efekata. Za razliku od 2D grafike, 3D grafika koristi trodimenzionalni prikaz geometrijskih podataka koje se pohranjuju na računalo kako bi obavili izračune i renderirali 2D slike. Kao i 2D grafika, 3D grafika se oslanja na iste algoritme. Razlika među njima može biti zbunjujuća budući da 2D aplikacija može

koristiti 3D tehnike za postizanje efekata kao što je osvjetljenje, a 3D koristi 2D za tehnike renderiranja.

Proces kreiranja 3D računalnih grafika se može redoslijedom podijeliti na 3 osnovne faze:

1. **3D modeliranje koje opisuje proces formiranja oblika objekta** - 3D modeliranje je proces kreiranja matematičke reprezentacije nekog trodimenzionalnog objekta. Ono što nastane naziva se 3D modelom. Kroz proces 3D renderiranja, može se dobiti 2D slika 3D modela iz jedne perspektive ili kao alternativa, 3D model se može iskoristiti kao resurs u real-time grafičkoj simulaciji.

2. **Izgled i animacija koja opisuje kretanje i položaj objekata unutar scene**- Računalna animacija je umijeće kreiranja pokretnih slika upotrebom računala. To je potkategorija računalne grafike i animacije. Ponekad je meta animacije računalo samo, ali ponekad je meta i drugi medij, kao film. Također se za nju koristi termin CGI (*Computer-generated imagery or computer-generated imaging*) posebno kada se koristi u filmu. Kako bi se stvorila iluzija pokreta, slika je prikazana na zaslonu računala i onda brzo zamijenjena novom slikom koja je slična prethodnoj, ali neznatno pomaknuta. Ova tehnika je identična iluziji pokreta na televiziji i pokretnim slikama.

3. **3d renderiranje koje stvara sliku objekta** - Odnosi se na proces dodavanja realizam na računalnu grafiku dodavanjem trodimenzionalnih kvaliteta kao što su sjene i varijacije boja i sjena. Jedna od tehnika za prikazivanje grafike naziva se *ray tracing* . Druga vrsta renderiranja je *scanline* renderiranje, koje rendera sliku jednu vertikalnu liniji po jednu umjesto objekt-po-objekt kao kod *ray tracing*-a. Općenito, *scanline* renderanje ne daje tako dobre rezultate kao *ray tracing*, ali se koristi kada kvaliteta svakog pojedinog kadra nije toliko važna.

## 2.1. Sjenčanje (*Shading*)

U računalnoj grafici, sjenčanje se odnosi na proces mijenjanja boje nekog objekta/površina/poligona u 3D sceni , a na temelju svog kuta do svjetla i njegove udaljenosti od svjetla stvora foto-realističan učinak. Sjenčanje se izvodi tijekom procesa renderiranja u programu koji se zove shader. Brz i jednostavan način za renderiranje objekt s poligonskom

površinom je sjenčanje sa konstantnim intenzitetom, koje se naziva *flat shading*. U ovom postupku, jedan intenzitet se izračunava za svaki poligon. Sve točke na površini poligona tada su prikazani u istoj vrijednosti intenziteta. Stalno sjenčanje može biti korisno za brzo prikazivanje općeg izgleda zakrivljene površine. Uz *flat shading* postoji kao njegov kontrast i *smooth shading*. Kod njega se boja mijenja od piksela do piksela. Pretpostavlja da su površine zakrivljene i koristi interpolacijske tehnike za izračun vrijednosti piksela između vrhova poligona.[2] Postoje dvije vrste *smooth shading*-a :

1. ***Gouraud shading*** - Ovakva intenzitet-interpolacijska shema, razvijena od strane *Gouraud*-a naziva se *Gouraud* sjenčanje, rendera površinu poligona tako da linearno interpolira vrijednosti intenziteta preko površine. Vrijednosti intenziteta za svaki pojedini poligon se podudaraju sa vrijednostima susjednih poligona uz zajedničke rubove, čime se eliminira diskontinuitet intenziteta koje se može dogoditi u *flat shading*-u. Osnovni princip iza metode je izračunati površinske normale na vrhovima poligona u računalnom 3D modelu. Te normale su onda prosjek za sve poligone koji se sastaju u svaku točku. Zatim se radi izračuni rasvjete za proizvodnju intenziteta boja na vrhovima. Te se vrijednosti boje onda interpoliraju uz rubove poligona. Za dovršetak sjenčanja, slika se popunjava linijama nacrtanim preko slike koja interpolira između prijašnje izračunatih intenziteta rubova. *Gouraud* sjenčanje zahtjeva mnogo manji intenzitet rada procesora od *phong* sjenčanje, ali ne izračunava sve željene svjetlosne efekte tako točno. Na primjer, bijela točka na površini jabuke (reflektirajući vrhunac) je vrlo ovisna o normalama u tom mjestu. Ako vrhovi modela nisu u tom mjestu , njihove boje su pomiješane preko njega , što onda uzrokuje njegovo kompletni nestanak. Ovaj problem je još očitije kada je izvor svjetlosti premješten, pomicanjem vrhunca preko jednog vrha. Koristeći *gouraud* sjenčanje reflektirajući vrhunac će se misteriozno pojaviti i rasti u intenzitetu kako svjetlost kreće prema položaju refleksija od promatrača preko tjemena . Željeni rezultat bi bio vidjeti vrhunac kako se kreće glatko, a ne da izblijedi između vrhova.[3]

2. ***Phong shading*** - *Phong* sjenčanje odnosi se na interpolacijsku tehniku za sjenčanje površine u 3D računalnoj grafici . Također se naziva i *phong* interpolacija. Naime, to interpolira površine normala preko rasterski poligona i izračunava piksele boje na temelju interpoliranih normala i reflektirajućeg modela. *Phong* sjenčanje može se odnositi i na specifičnu kombinaciji *phong* interpolacije i reflektirajućeg modela *phong*. Reflektirajući model je lokalni model osvjetljenja kojeg je osmislio Bui Tuong Phong i može proizvesti

određeni stupanj realizma u trodimenzionalnim objektima , kombinirajući tri elementa - difuzna, reflektirajuća i ambijentalna osvjetljenje za svaku razmatranu točku na površini. Ono ima nekoliko pretpostavki - sva svjetla su točke , samo geometrija površine se uzima u obzir, postoje samo lokalna modeliranje difuzno i reflektirajućih svjetala, reflektirajuća boja je ista kao i boja svijetla , a ambijentalno svijetlo je globalni konstanta. Kao metoda renderiranja, *phong* sjenčanje može se smatrati poboljšanom verzijom *gouraud* sjenčanja koji omogućuje bolju sliku stvarnosti aproksimirajući *phong* model sjenčanja. [4][5]

## 2.2. Mapiranje

Mapiranja je grafički dizajnerski proces u kojem je dvodimenzionalna (2 -D) površina, pod nazivom mapa tekstura, "omotan oko" trodimenzionalnog ( 3 - D) objekta. 3- D objekta stekne površinsku teksturu sličnu onoj iz 2-D površine. Mapiranje je elektronski ekvivalent nanošenja tapeta, boje ili furnira na pravi objekt. U 3D grafici, ono predstavlja digitalni prikaz površine objekta. Osim dvodimenzionalnih osobina, kao što su boja i svjetlinu, tekstura je također kodirana trodimenzionalnim svojstvima, kao što su koliko je objekt transparentan i reflektirajuć. Jednom kada je tekstura definirana, ona se može omotati okolo bilo kakvog 3-D objekta. To se zove mapiranja. Dobro definirane teksture su vrlo važne za renderiranje realne 3D slike. Postoji više vrsta mapiranja:

- **Mapiranje visine (*Hight mapping*)** - U računalnoj grafici, *heightmap* ili *heightfield* je rasterska slika koja se koristi za pohranu vrijednosti, kao što su podaci o površinskim povišenjima, za prikaz u 3D računalnoj grafici. *Heightmap* se može koristiti u mapiranju izbočina za izračunavanje gdje će 3D podaci stvoriti sjenu na materijalu, u mapiranju pomaka za premještanje stvarnih geometrijski položaja točke preko teksturirane površine ili terenu gdje se *heightmap* pretvara u 3D mreže. *Heightmap* sadrži jedan kanal protumačen kao udaljenosti od pomaka ili "visine" od "poda" površine, a ponekad i vizualizirana kao paleta od sivih tonova slike, sa crnom bojom koja predstavlja minimalnu visinu i bijelom bojom koja predstavlja maksimalnu visinu. Kada je karta renderirana, dizajner može odrediti iznos pomaka za svaku jedinicu po visini kanala, što odgovara "kontrastu" slike. *Heightmaps* mogu biti pohranjeni samo u postojećim sivim tonovima slikovnih formata, sa ili bez specijaliziranih meta-podataka ili u specijaliziranim formatima datoteka kao što su *Daylon Leveller*, *GenesisIV* i *Terragen* dokumenti. [6]

- **Mapiranje izbočina (*Bump mapping*) -** Mapiranje izbočina je tehnika u računalnoj grafici koja napravi to da već renderirana površina izgleda realnije simulirajući male pomake na površinu. Međutim, za razliku kod mapiranja pomaka, geometrija površina nije promijenjena. Umjesto toga samo je normala površine modificirana kao da je površina bila pomaknuta. Modificirana normala površine se tada rabi za izračune svjetlosti ( upotrebom, na primjer, modela refleksije *phong* ) dajući izgled detalja umjesto glatku površinu. Mapiranje izbočina je puno brže i troši manje resursa za istu razinu detalja u odnosu na mapiranje pomaka jer geometrija ostaje nepromijenjen. Osnovna ograničenja mapiranja izbočina je da remeti samo normalu površine bez promjene same podlogu. Siluete i sjene stoga ostaju nepromijenjene , što je posebno uočljivo za veće simulirane pomake. Ovo ograničenje može biti prevladano tehnikama uključujući mapiranje pomaka gdje se neravnine zapravo nanose na površinu ili pomoću iso-površine .[7]

- **Mapiranje pomaka (*Displacement mapping*) -** Mapiranje pomaka je alternativna tehnika računalne grafike za razliku od mapiranja izbočina, normalnog mapiranja, koristi teksture ili *hightmap* da izazove efekt gdje je stvaran geometrijski položaj točke preko teksturne površine pomaknut, često uzduž normale lokalne površine, prema vrijednosti koju funkcija tekstura procjenjuje u svakoj točki na površini. To daje površina veliki osjećaj dubine i detalja, omogućujući samo-okluziju, samo-sjenčanje i siluete; S druge strane, to je najskuplja tehnika ove klase zbog velike količine dodatnih geometrija. Algoritmi mapiranja pomaka uzimaju uzorke točaka i pomiču ih okomitu do normale makro-strukturne površine sa udaljenostima dobivene iz *hightmap*-a. Uzorci točaka mogu biti ili vrhovi originala ili vrhovi teselacijske mreže (mapiranje pomaka vrh po vrh) ili točke koje odgovaraju centrima teksturnih piksela (mapiranje pomaka piksel po piksle).
  U slučaju mapiranja pomaka vrh po vrh modificirana geometrija prolazi kroz renderiranje. Međutim, u mapiranju pomaka piksel po piksel, detalji površine se dodaju kada se počne odvijati teksturiranje boja. Godinama je mapiranje pomaka bilo posebnost najnovijih sustava za renderanje kao sto su *PhotoRealistic*, *RenderMan*, dok su aplikacije stvarnog vremena kao *OpenGL* i *DirectX* počele tek nedavno koristiti ovaj način mapiranja. Jedan od razloga za to je da je izvorna provedba mapiranja pomaka zahtijevala prilagodljivo korištenje mozaika na površini kako bi se dobio dovoljan broj mikro poligona čija je veličina usklađena veličini piksela na zaslonu.

- **Normalno mapiranje (*Normal mapping*) -** U 3D računalnoj grafici, normalno mapiranje ili Dot3 mapiranje izbočina, se koristi za lažiranje osvjetljenja izbočina i udubljenja - implementacija mapiranja izbočina. Koristi se za dodavanje pojedinosti bez upotrebe više poligona. Uobičajena upotreba ove tehnike je uvelike poboljšati izgled i detalje niskog poligonskog modela stvaranjem normalne mape iz visokog poligonskog modela ili mape visine. Normalne mape obično se pohranjuju kao redovne RGB slike u kojima RGB komponente odgovaraju X, Y i Z koordinatama normalne površine. Interaktivno normalno mapiranje prvobitno je bilo moguće samo na *PixelFlow*-u, paralelnom renderizacijskom stroju koji je izgrađen na Sveučilištu Sjeverne Karoline u Chapel Hillu. Kasnije je omogućena izvedba normalnih mapiranja na vrhunskim CGI radnim stanicama pomoću više-prolaznog renderiranja i *framebuffer* operacija[8] ili na jeftinijem PC hardveru s nekim trikovima koristeći paletirane teksture. Međutim, pojavom alata za sjenčanje u osobnim računalima i igraćim konzolama, normalno mapiranje postalo je široko korišteno u komercijalnim video igrama počevši krajem 2003. godine. Popularnost normalnog mapiranja za prikazivanje u stvarnom vremenu posljedica je dobre kvalitete u odnosu na zahtjeve obrade drugih metode koji proizvode slične učinke.

## 2.3. Anti-aliasing

Jedan od problema koji se javlja kada je u pitanju 3D računalna grafika je upravo aliasing. To je proces u kojim glatke krivulje i druge linije postaju nazubljeni, jer rezolucija grafičke kartice ili same datoteke nije dovoljna da predstavlja glatke krivulje. *Smoothing* i *anti-aliasing* su tehnike koje mogu smanjiti učinak *aliasing* efekta. U računalnoj grafici, *anti-aliasing* je softverska tehnika za smanjivanje i izglađivanje stepeničastih prikaz na linijama koja bi trebala biti glatka. Ti stepeničasti prikazi se pojavljuju jer izlazni uređaj, monitor ili pisač, nema dovoljno visoku rezoluciju za prezentaciju glatke linije. *Anti-aliasing* smanjuje istaknutost stepeničastog prikaza tako da okružuje stepenice sa srednjom nijansom sive boje ili boje (za uređaje u boji). Iako to smanjuje nazubljenost linija, također ih čini nejasnima. *Anti-aliasing* se ponekad naziva *oversampling*.

- ***Supersampling antia-liasing* (SSAA ili FSAA) -** algoritamska *anti-aliasing* tehnika uključuje uzorkovanje sadržaja svakog piksela na više lokacija, što znači da se boja izračunava na više mjesta unutar područja obuhvaćenog pikselima. Rezultati tog rukovanja se kombiniraju kako bi se utvrdila konačna boja piksela. Ti uzorci su u osnovi

dodatni pikseli, koji se koriste za povećanje efektivne rezolucije slike koja se prikazuje. Ako rub objekta padne djelomično unutar područja piksela, njegova boje i boje drugog objekta koji djelomično ispunjava "prostor" od piksela mogu se koristiti za izračunavanje konačne boje. Rezultat je glađi prijelaz iz jedne linije piksela na drugu liniju piksela uz rubove objekata, gdje je *aliasing* najočitiji. *Supersampling* je *anti-aliasing* tehnika koja je jednostavno "*brute-force*" pristup i koristi se u Nvidia GeForce GPU i drugim modernim grafičkim procesorima. Grafički procesor koji koristi *supersampling* rendera sliku ekrana na mnogo višoj razlučivosti od trenutnog načina prikaza, a zatim skalira i filtrira slike na konačnu rezoluciju prije nego što su poslane na zaslon. Različite metode postoje za izvođenje ove operacije, ali svaka zahtijeva grafički procesor da rendera što više dodatnih piksela po potrebi za metodu *supersampling*. Osim toga, jer grafički procesor rendera više stvarnih piksela nego što će se prikazati, on mora skalirati i filtrirati te piksela do vrijednosti rezolucije za konačni prikaz. To skaliranje i filtriranje može dodatno smanjiti performanse. Stupanj skaliranja u određenom *supersampling* modu često se poistovjećuje s omjerom piksela u neprilagođenoj slici sa brojem piksela u konačnoj, skaliranoj verziji izlaza. Na primjer, 2x *supersampling* piše dva puta više piksela na video među-spremnik nego što bi to bilo potrebna bez *antialiasing*-a. 4x, piše četiri puta više piksela. *Supersampling* uzrokuje znatan pad performansi, mjereno prema *frame rate-u.* Ako grafički procesor rendera četiri puta više piksela, onda će *frame rate* biti četvrtina onoga što je u standardnom načinu prikaza. [9]

- ***Multisample anti-aliasing* (MSAA) -** U *multisample anti-aliasing*-u, ako je bilo koji od višestrukih uzoraka lokacije u pikselu pokriven trokutom koji se rendera, izračun sjenčanja se mora obaviti za taj trokut. Međutim ovaj izračun samo treba obaviti jednom za cijeli piksel, bez obzira koliki je dio pozicije uzoraka pokriven; rezultat izračuna sjenčanja jednostavno se primjenjuje na sve relevantne lokacije sa više uzoraka. U posebnom slučaju gdje samo jedan trokut pokriva svaki dio lokacije sa više uzoraka unutar piksela, obavlja se samo jedan izračun siječanja, a ti pikseli su malo skuplji (i rezultat se ne razlikuje) u odnosu na ne-*anti-aliased* slike. To vrijedi za sredinu trokuta, gdje *aliasing* ne smatra da postoji problem. U ekstremnom slučaju, gdje je svaki od višestrukih uzorka lokacije pokrivena drugim trokutom, izračunavanje sjenčanja obavit će se za svaki položaj i rezultate zatim spojiti kako bi se dobio konačni piksela, a rezultat i računalni trošak su isti kao u ekvivalentnoj *supersampled* slici. Izračun sjenčanje nije jedini postupak koji se mora obaviti na određenom piksela; multisampling

implementacije mogu različito uzorkovati neke druge operacije kao što su vidljivost na različitim razinama uzorkovanja. [10]

- ***Fast Approximate Anti-Aliasing*** **(FXAA)** - algoritam *anti-aliasing* koji je stvorio Timothy Lottes iz NVIDIA-e. Glavna prednost ove tehnike u odnosu na konvencionalne *anti-aliasing* tehnike je da ne zahtijeva velike količine računalne snage. To postiže izglađivanjem nazubljenih rubova prema načinu na koji se pojavljuju na zaslonu kao pikseli, umjesto da analiziraju sam 3D model kao u konvencionalnim, anti-aliasing tehnikama. Osim toga, ona smanjuje rubove u svim pikselima na zaslonu, uključujući one unutar alfa-uklopljenih tekstura i one koje proizlaze iz efekata piksela sjenčanja, koji su prethodno bili imuni na učinke *multisample anti-aliasing* (MSAA). Nedostaci su da teksture možda neće izgledati oštre ako su uključene u otkrivanje rubova i moraju se primijeniti prije prikazivanja elemenata HUD-a u igri, kako i ne bi utjecali na njih.[11][12] FXAA metoda je toliko dobra u odnosu na ostale *anti-aliasing* metode da se mogu i zanemariti ukoliko je u nekoj 3-D igri dostupna.

- ***Temporal anti-aliasing*** **(TXAA)** - TXAA je nova film-*anti-aliasing* tehnika koja je dizajnirana posebno za smanjenje vremenskog pomaka (treperenje koje se vidi pri kretanju pri igranju igara). Ova je tehnologija mješavina vremenskih filtara, hardverskog *anti-aliasing*-a i prilagođenog CG filmskog stila kojemu se *anti-aliasing* protivi. Za filtriranje bilo kojeg piksela na zaslonu, TXAA koristi doprinos uzoraka, i unutar i izvan piksela, zajedno s uzorcima iz prethodnih kadrova, kako bi omogućio najvišu kvalitetu filtriranja. TXAA je poboljšala prostorno filtriranje preko standardnih 2xMSAA i 4xMSAA. Na primjer, na ogradi ili lišća i u pokretu, TXAA počinje pristupiti i ponekad premašuje kvalitetu drugih visokokvalitetnih profesionalnih *anti-aliasing* algoritama. TXAA kombinira sirovu snagu MSAA s sofisticiranim filtrima sličnih onima koji se primjenjuju u CG filmovima kako bi se dobila glatkija slika što je daleko bolje od ostalih sličnih tehnika. Ovisno o vrsti zasjenjivanja koja se provodi u danoj igri, učinak performansi TXAA može biti malo drugačiji. Za razliku od metoda poput FXAA koja pokušava maksimizirati performanse na štetu kvalitete, TXAA pokušava maksimizirati kvalitetu na račun učinkovitosti.[13]

# 3. ANIMACIJA

Prebacivanje iz 2D-a u 3D, s razlikom da se u 2D animaciji efekt perspektive kreira umjetnički, ali u 3D-u predmeti se modeliraju u internom 3D prikazu unutar računala, a zatim se 'ispaljuju' iz izabranih kutova, baš kao u stvarnom životu, prije nego što se renderiraju u 2D *bitmap* okvire. Primjena računalne animacije kao način postizanja inače nemogućeg u uobičajeno snimljenim filmovima je dovelo do pojma "kompjuterski generiranog prikaza" (CGI), iako je pojam postalo teško razlikovati od računalne animacije zato što se sada njegovo korištenje odnosi na 3D filmove koji su u potpunosti animirane. Računalna animacija uključuje modeliranje, generiranje pokreta, a zatim se dodaje površina, a na kraju renderiranje. Površine su programirane da se protežu i automatski savijaju kao odgovor na kretanja u *wire frame* modelu, a konačni prikaz pretvara takve pokrete u *bitmap* slike. Animacija je proces dovođenja beživotne lutke u život kroz upotrebu pokreta. Mnogi ljudi brkaju efekte i teksture visoke rezolucije s animacijom, ali u stvari gibanje kao u stvarnosti može se izraditi pomoću najjednostavnijih modela. Danas se obično pogrešno misli da računala stvaraju animacije. No računalo nije ništa više od vrlo skupog i kompliciranog alat za crtanje, kao što je olovka alat za crtanje. Izbori koje računalo čini kod interpolacije pokreta su gotovo uvijek nespretni ili neprivlačni, jer računalo ne može znati što je animator pokušavao izraziti. Čak ako se stvori dovoljno složen sustav fizike da se točno oponaša stvarni svijet, krajnji rezultat ne bi bio toliko utjecajan, jer je značajan dio umijeća animacije upravo umjetnički izbor koji animator čini, a koje računalo nije u stanju prepoznati.

Računalna animacija je proces koji se koristi za generiranje animiranih slika. Općenitije pojam "kompjutorski generirane slike" (CGI) obuhvaća statičke scene i dinamičke slike, dok se računalne animacije odnose samo na pokretne slike. Moderna računalna animacija obično koristi 3D računalnu grafiku, iako se 2D računalna grafika još uvijek koristi za stilsku, nisku širinu pojasa i brže realno prikazivanje. Ponekad, cilj animacije je računalo, no ponekad i film. Za 3D animaciju, objekti (modeli) izrađuju se na monitoru računala (modelirani), a 3D slike su opremljene virtualnim kosturima. Za animacije 2D figura koriste se zasebni objekti (ilustracije) i zasebni transparentni slojevi sa ili bez tog virtualnog kostura. Potom animatori pokreću udove, oči, usta, odjeću, itd. figure na ključnim okvirima(*key frames)*. Razlike u izgledu između ključnih okvira automatski se izračunavaju pomoću računala u procesu koji se naziva *tweening* ili *morphing*. Konačno, animacija je izvedena. Svi okviri moraju biti izvedeni nakon što je modeliranje završeno. Za 2D vektorske animacije, proces renderiranja je ključni proces ilustracije okvira, a *tweened* okviri se prikazuju po potrebi. Za unaprijed snimljene prezentacije,

prikazani okviri se prenose u drugi format ili medij, kao što je digitalni videozapis. Okviri se također mogu prikazati u stvarnom vremenu, budući da su prikazani krajnjim korisnicima. Animacije niske širine pojasa koje se prenose putem interneta (na primjer, Adobe Flash, X3D) često upotrebljavaju softver na računalu krajnjeg korisnika za prikazivanje u stvarnom vremenu kao alternativu za *streaming* ili unaprijed učitane animacije velike propusnosti. U većini 3D računalno animacijskih sustava, animator stvara pojednostavljen prikaz anatomije lika, što je analogno slici kostura ili štapa.[14] Položaj svakog segmenta skeletnog modela određen je animacijskim varijablama, ili *Avars* skraćeno. U ljudskim i životinjskim likovima mnogi dijelovi skeletnog modela odgovaraju stvarnim kostima, ali skeletna animacija također se koristi za animiranje drugih stvari s ekspresijama lica (iako postoje druge metode za animaciju lica). [15] Računalo obično ne rendera skeletni model izravno ali koristi skeletni model za izračunavanje točnog položaja i orijentacije tog određenog lika, koji se na kraju pojavljuje u slici. Tako promjenom vrijednosti *Avara* tijekom vremena animator stvara kretanje tako što se lik pomiče iz okvira u okvir.

Postoji nekoliko metoda za stvaranje *Avars* vrijednosti kako bi se dobio realan pokret. Tradicionalno, animatori izravno manipuliraju *Avars-ima*.[16] Umjesto da postave *Avars* za svaki okvir, oni obično postavljaju *Avars* na strateške točke (okviri) u vremenu i puste da računalo interpolira ili *tween-a* između njih u procesu koji se zove *keyframing*. *Keyframing* stavlja kontrolu u ruke animatora i ima korijene u ručno izrađenoj tradicionalnoj animaciji. Nasuprot tome, noviji postupak zvan snimanje pokreta(*motion capture*) koristi snimke uživo.[17] Kada se računalna animacija upravlja pokretnim snimanjem, pravi izvođač djeluje na sceni kao da su likovi animirani. Njegovo kretanje zabilježeno je na računalu pomoću video kamera i oznaka, a ta se izvedba tada primjenjuje na animirani lik. Svaka metoda ima svoje prednosti, a od 2007. igre i filmovi koriste jednu ili obje od tih metoda u produkcijama. Animacija *keyframing-om* može proizvesti kretnje koje bi bile teško ili nemoguće odglumiti, a snimanje pokreta može reproducirati suptilnosti određenog glumca. [18]

# 4. FIZIKA U RAČUNALNIM IGRAMA

Fizika računalne animacije ili fizika igre uključuje uvođenje zakona fizike u simulaciju ili *game engine*, posebno u 3D računalnoj grafici, u svrhu izrade takvih efekata da budu stvarniji za promatrača. Tipično, simulacija fizike ja samo približna stvarnoj fizici, a izračun se izvodi pomoću diskretnih vrijednosti.

Postoje dva centralna tipa simulacije fizike :

- **Simulacija krutih tijela *(rigid body simulator*) -** u ovoj simulaciji objekti su grupirani u kategorije na temelju toga kako bi trebali komunicirati i oni su manje intezivni sto se tiće performansi.[19]
- **Simulacija mekih tijela (*soft body simulator*) -** fizika mekih tijela podrazumijeva simulaciju individualnih dijelova svakog objekta tako da ga prikaže na što realističniji način.[19]

U većini računalnih igara, brzina procesora i igrivost su važnije od točnost simulacije. To dovodi do dizajna *physics engine* koji rezultate daje u stvarnom vremenu, ali replicira fiziku stvarnoga svijeta samo za jednostavne slučajeve i obično s nekom aproksimacijom. U sve više slučajeva, simulacija je usmjerena prema pružanju "perceptualne ispravnosti" aproksimacije nego prave simulacije. Animacija likova temeljenih na fizici u prošlosti je koristila samo dinamiku krutih tijela, jer su brže i lakše za izračunati, ali moderne igre i filmovi se počinju koristiti fizikom mekih tijela. Fizika mekih tijela također se koriste za efekte čestica, tekućina i tkanina. Neki oblici ograničene simulacije dinamike fluida su ponekad ponuđeni da simuliraju vodu i druge tekućina, kao kretanje vatre i eksplozije kroz zrak. Postoji nekoliko elemenata koji čine komponente simulacije fizike uključujući *physics engine*, programerski kod koji se koristi kako bi predočio Newtonovu fiziku u okolišu, te otkrivanje sudara, koje se koristi za rješavanje problema određivanja kada bilo koja dva ili više objekata se u okolišu sudare.

- **Physics Engine** - *physics engine* je računalni softver koji pruža približnu simulaciju određenih fizičkih sustava, kao što su simulacije krutih tijela (uključujući otkrivanje sudara), simulaciju mekih tijela i dinamika fluida, korištena u domeni računalne grafike, video igara i filma. Njihove glavne koristi su u video igrama (obično kao posrednički sloj), gdje su simulacije u stvarnom vremenu. Pojam se ponekad koristi općenitije za opisivanje bilo kojeg softverskog sustava za simulaciju fizikalnih pojava, poput znanstvene simulacije visokih performansi. Općenito postoje dvije klase *physics engine-a*: u stvarnom vremenu(*real-time*) i visoke preciznosti. Visoko precizni *physics engine*-i

zahtijevaju više procesorske snage za izračunavanje vrlo precizne fizike i obično ih koriste znanstvenici i računalni animirani filmovi. *Physics engine-i* u stvarnom vremenu - upotrebljavani u videoigrama i drugim oblicima interaktivnog računanja - koriste pojednostavljene izračune i smanjenu točnost kako bi postigla veća učinkovitost da igra reagira odgovarajućom brzinom za igranje. Primarna ograničenja realizma *physics engine-a* je preciznost brojeva koji predstavljaju položaj i sile koje djeluju na objekte. Kada je preciznost preniska, pogreške zaokruživanja utječu na rezultate i male promjene koje nisu modelirane u simulaciji mogu drastično promijeniti predviđene rezultate; Simulirani objekti se mogu ponašati neočekivano ili doći na pogrešno mjesto. Pogreške su povezane u situacijama u kojima se dva objekta spojena zajedno kreću slobodno sa preciznosti koja je veća od onoga što *physics engine* može izračunati. To može dovesti do neprirodne energije nakupljanja u objektu zbog grešaka zaokruživanja koje se počinju nasilno tresti i na kraju razbiju objekte. Svaka vrsta fizike objekta koji se slobodno kreće može pokazati ovaj problem, ali je osobito sklona utjecati na lančane veze pod visokim naprezanjem i predmetima na kotačima s aktivnim fizičkim ležajnim površinama. Veća preciznost smanjuje pogreške položaja / snage, ali po cijeni veće snage CPU-a potrebne za izračune. [20]

- ***Game Engine*** - *game engine* je softver dizajniran za stvaranje i razvoj video igara. Programeri ih koriste za stvaranje igara za konzole, mobilne uređaje i osobna računala. Glavna funkcionalnost koju obično pruža *game engine* uključuje renderer za 2D ili 3D grafiku, *physics engine* ili detekciju sudara (i sudara), zvuk, skriptiranje, animaciju, umjetnu inteligenciju, umrežavanje, *streaming*, upravljanje memorijom, navođenje, podrška za lokalizaciju, grafikon scene i može uključivati video podršku za video isječke. Proces razvoja igara često štedi, u velikoj mjeri, ponovnim korištenjem/prilagodbom istog *game engine-a* za stvaranje različitih igara[21] ili olakšavanjem distribucije igara na više platformi. U mnogim slučajevima *game engine*-i nude niz alata za vizualni razvoj, uz nove korisne komponente softvera. Ti su alati općenito dostupni u integriranom razvojnom okruženju kako bi omogućili pojednostavljen i brz razvoj igara na način koji se temelji na podacima. Programeri igara pokušavaju izmisliti nešto novo razvijanjem robusnih softver okruženja koji uključuju mnoge elemente koje razvojni programer igre možda treba izgraditi. Većina *game engine* okruženja pružaju objekte koji olakšavaju razvoj, kao što su grafika, zvuk, fizika i funkcije AI. Ovi *game engine*-i ponekad se nazivaju *middleware* jer, kao i poslovni smisao tog pojma, pružaju fleksibilnu i višekratnu softversku platformu koja pruža sve potrebne osnovne funkcije, odmah u

startu, za razvoj igre, istodobno smanjivši troškove , složenosti i vrijeme plasiranja na tržište - svi ključni čimbenici u vrlo konkurentnoj industriji videoigara.[22]

# 5. UNREAL ENGINE

Unreal Engine je *game engine* razvijen od strane Epic Games-a, prvi put prikazan u 1998.g. u FPS(*first person shooter*) igri Unreal. Iako se prvenstveno razvio za pucačine iz prvog lica, uspješno se upotrebljava u različitim žanrovima, uključujući MMORPG(m*assive multyplayer online role playing game*) i druge RPG-ove(*role playing game)*. Sa svojim kodom napisanima u C++, Unreal Engine ima visok stupanj prenosivosti i alat je koji danas koriste mnogi programeri igara. [23]

## 5.1. Unreal Engine 1

Godine 1998. u igri Unreal, predstavljena je prva generacija Unreal Engine koji je integrirao renderiranje, otkrivanja sudara, AI, vidljivosti, umrežavanja, skriptiranja i upravljanja datotečnim sustavom u jedan kompletan *engine*. Unreal Engine 1 je osigurao napredni raster softver i hardverski ubrzano renderiranje pomoću Glide API-ja, posebno razvijen za 3dfx GPU. Ažuriran je za OpenGL i Direct3D. Izlazak Unreal Tournament-a obilježilo je veliki napredak u mrežnim performansama i podršci za Direct3D i OpenGL. *Engine* je postao popularan zbog toga što je modularne arhitekture i uključivanja skriptnog jezika zvanog UnrealScript, što je olakšalo programiranje. Od samog početka, *engine* je dizajniran na način da se može proširiti i poboljšati tijekom sljedećih generacija igara. [23]

## 5.2. Unreal Engine 2

Druga verzija objavljena je 2002. godine s *America's army*, besplatnom pucačinom za više igrača koje je stvorila američka vojska i koju financira američka vlada. Ova generacija je u potpunosti prepisala kod jezgre i renderiranje *engine*-a. Osim toga, sadržavao je i UnrealEd 2, urednik nivoa, koji je debitirao s prethodnom generacijom *engine-a*, a kasnije ga je naslijedio UnrealEd 3, zajedno s Karma physics SDK. Ovaj fizikalni *engine* pokretao je fiziku u Unreal Tournament 2003 i Unreal Championshipu. Ostali elementi *engine-a* također su ažurirani, uz poboljšane alate, kao i podrška za GameCube i Xbox. Podrška za PlayStation 2 konzolu prethodno je dodana u UE1. Treće strane koje su željele koristiti daljnje revizije Unreal Enginea morale su raditi svoje vlastite verzije tijekom čitave generacije. UE2.5, ažuriranje originalne verzije UE2, poboljšana performansa renderinga i dodana fizika vozila, uređivač sustava čestica za UnrealEd i 64-bitna podrška u Unreal Tournamentu 2004. Specijalizirana verzija UE2.5 nazvanog UE2X

korištena je za Unreal Championship2: The Liandri Conflict na originalnoj Xbox platformi. Sadrži optimizacije specifične za tu konzolu. EAX 3.0 podržava i zvuk. [23]

## 5.3. Unreal Engine 3

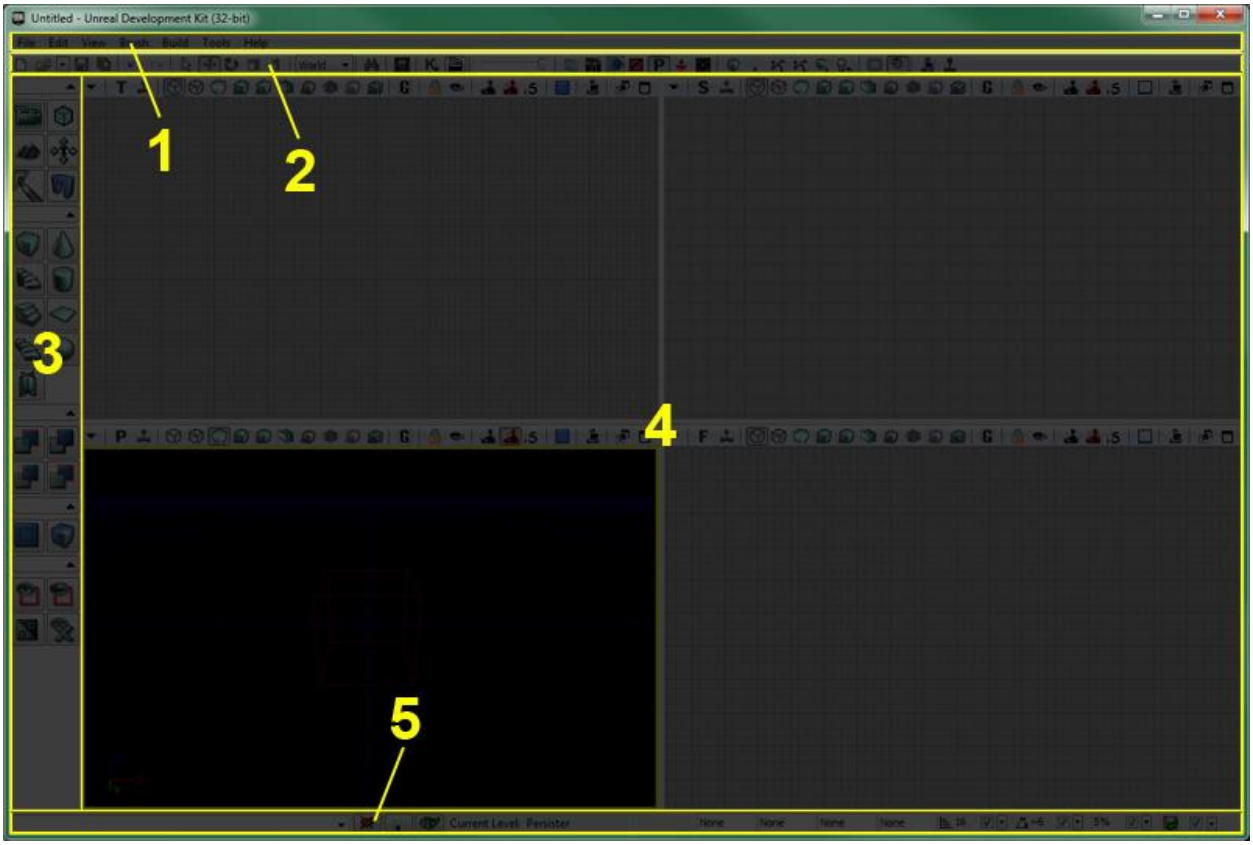
Prve slike Unreal Engine 3 predstavljene su 2004, u tom trenutku *engine* je već bio u razvoju 18 mjeseci. Za razliku od Unreal Engine 2, koji je još uvijek podržavao *pipeline* s fiksnim funkcijama, Unreal Engine 3 je dizajniran za iskorištavanje potpuno programabilnog hardvera za sjenčanje. Svi izračuni rasvjete činili su se po pikselu, umjesto po vrhovima. Unreal Engine 3 podržava gama-ispravni visoko-dinamički raspon renderera. U početku, Unreal Engine 3 podržavao je samo platforme sustava Windows, PlayStation 3 i Xbox 360, a iOS i Android dodani su kasnije u 2010. godini. OS X podrška dodana je 2011. godine. Njegova renderer podržava tehnike kao HDRR(*high dynamic range rendering)*, rasvjeta po pikselu i dinamičke sjene. U listopadu 2011. *engine* je nadograđen kako bi podržao Adobe Flash Player 11 putem Stage 3D hardverskih ubrzanih API-ja. Epic je koristio ovu verziju *engine* za svoje igre. Agresivno licenciranje ove iteracije dobilo je veliku podršku mnogih istaknutih korisnika licence. Epic je najavio da Unreal Engine 3 radi na Windows 8 i Windows RT. Tijekom cijelog životnog ciklusa UE3-a uključena su značajnija ažuriranja uključujući globalno rješenje osvjetljenja, poboljšana destruktivna okruženja, meka dinamika tijela, velika simulacija gužve, funkcionalnost iPod Toucha, integriranje Steamworks, real-time globalno osvjetljenje. Iako je Unreal Engine 3 bio prilično otvoren za suradnike, sposobnost objavljivanja i prodaje igara napravljenih putem UE3 bila je ograničena na licencirane osobe *engine-a*. Međutim, u studenom 2009. Epic je objavio besplatnu verziju SDK-a UE3, zvan Unreal Development Kit (UDK), koji je dostupan široj javnosti. Izdanje UDK u prosincu 2010. dodalo je podršku za stvaranje iOS igara i aplikacija. [23]

# 6. UDK(UNREAL DEVELOPMENT KI) I IZVEDBA IGRE

Igra za ovaj rad je kompletno izrađen u programskom okruženju UDK. Nije korišten niti jedan drugi program kako bi se ona realizirala pošto UDK nudi sve potrebne alate za izvedbu cjelokupne igre. U početku je bilo potrebno odrediti neke od elemenata same igre kao i tip igre koja će se praviti, dizajnirati nivoe, samog igrača i botove[1]. Kao tip igre odabran je FPS(*first person shooter*) koji se sastoji od dva mod-a, *DeathMatch* i *Capture The Flag*.

## 6.2. Dizajn mapa/nivoa

Cjelokupne mape izrađene su pomoću UDK editor-a koji je jezgra samog UDK-a za kreiranje svjetova i nivoa. Sučelje samog editora sastoji se od nekoliko dijelova.




Sl.6.1. Sučelje UDK editora

[1] U video igrama, bot je vrsta ekspertnog sustava umjetne inteligencije koji igra video igru umjesto čovjeka. Mogu uključivati analizu mape i osnovne strategije. Bot pokušava imitirati kako bi čovjek igrao igru.

(Sl. 6.1.)

1. *Menu bar* - standardni meni koji pruža veliku mogućnost korištenja alata i akcija kako bi se omogućila izvedba zamišljene igre.

2. *Toolbar* - kao i u većini aplikacija, grupa komandi koja pruža brz pristup često korištenim alatima i akcijama

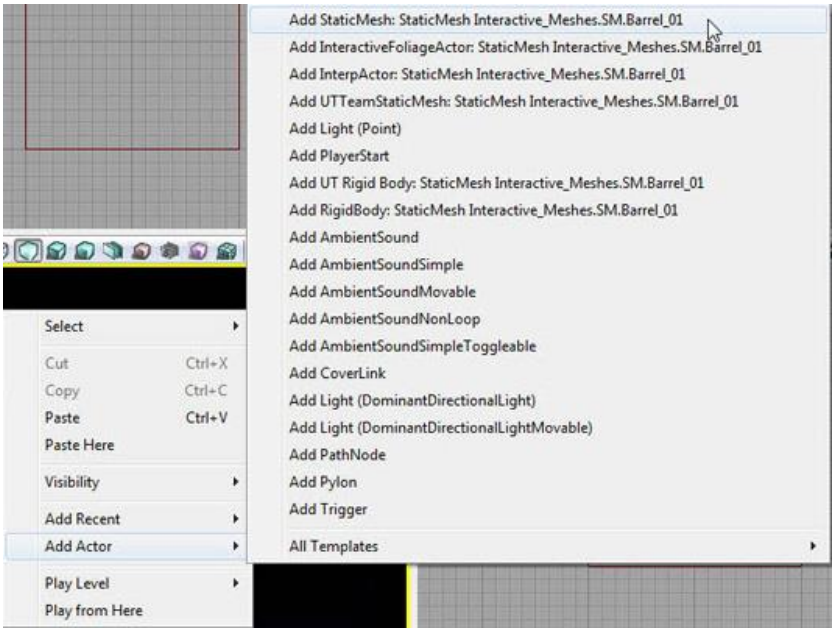
3.*Toolbox* - set alata koji se koriste za kontrolu nad mod-om u koje je trenutni *level-editor*. Koristi se za mijenjanje *builder-brush-a,* kreiranje nove BS geometrije i volumena, kontrola nad osvjetljenjem i akterima u određenim *Viewport-ovima.*

4. *Viewports* - vrste pogleda na sam nivo.

5. *Status Bar*[24]

Samo kreiranje nivoa može se svesti na nekoliko osnovnih zadataka: stavljanje aktera, označavanje aktera, transformacija aktera, modificiranje aktera. Drugim riječima, kako bi se kreirao nivo, akteri moraju biti stavljeni na samu mapu i onda ce njihove karakteristike određivati kako ce se oni ponašati. Svaka mapa dolazi kao prazna ploča. Postoji nekoliko načina na koji se akteri mogu ubaciti u mapu. Ovdje su opisani oni koji su korišteni u radu.

Određene vrste *asset-a* možete odabrati u *Content Browser-u,* a zatim dodijeliti novoj instanci odgovarajuće vrste *asset-a* tako da desnom tipkom miša kliknete na jedan od prikaza i odaberete "*Add Actor*>". Na to će se prikazati izbornik za ispis s popisom mogućih vrsta aktera za dodavanje pomoću odabrane opcije.[24] (Sl.6.2.).

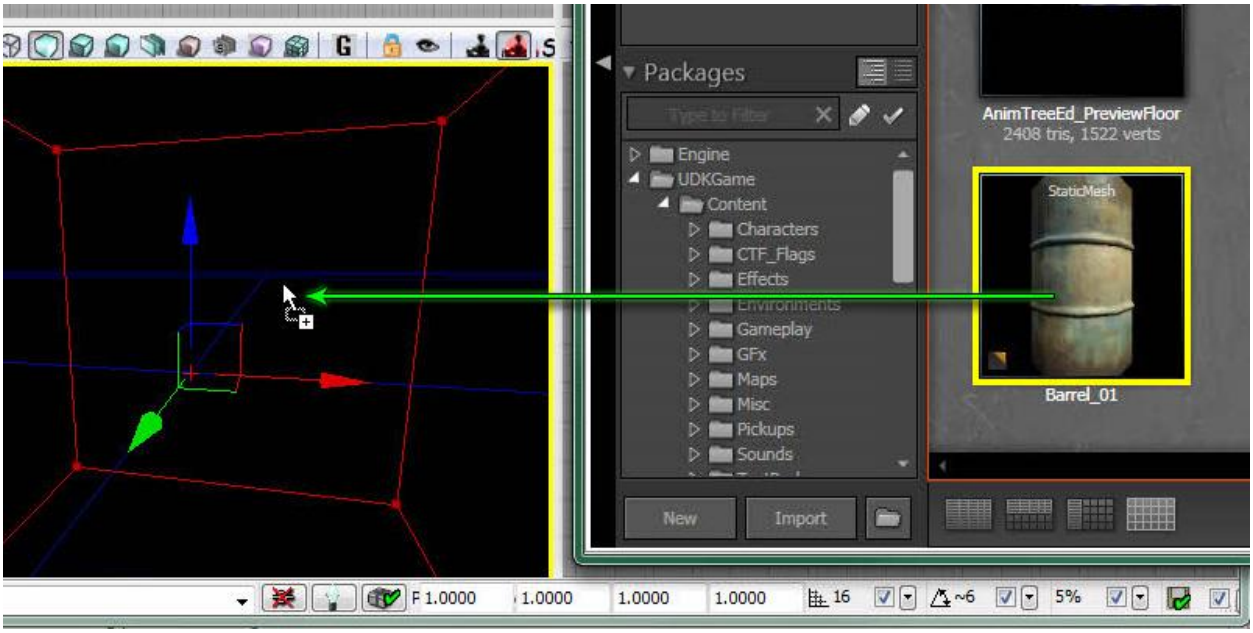



Sl.6.2. Dodavanje aktera

Vrste *asset-a* koje se mogu dodati iz *Content Browser-a su :*

- Static Meshes
- Skeletal Meshes
- Physics Assets
- Fractured Static Meshes
- Particle System
- Speed Trees
- Sound Cue
- Sound Wave
- Lens Flare

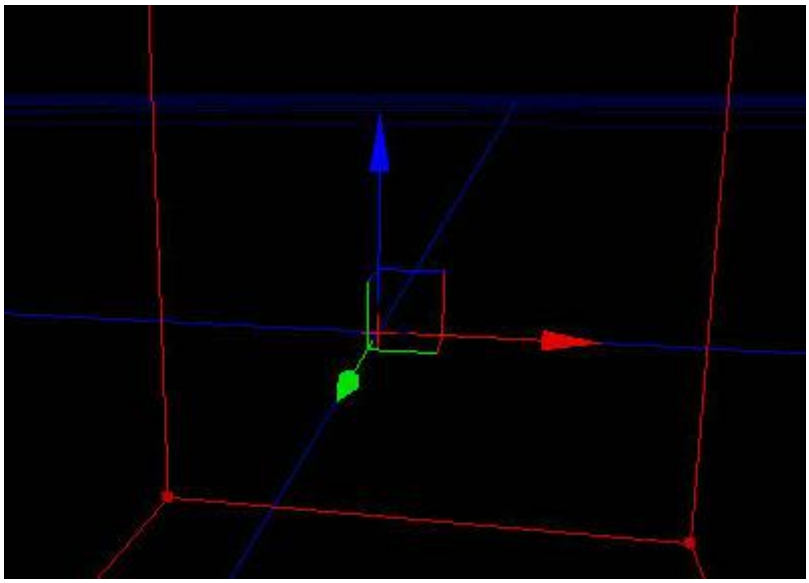
Osim što se mogu dodati određene vrste aktera iz *Content Browser-a* na mapu kroz *viewport* meni, one se mogu dodati i jednostavnim povlačenjem sadržaja iz *Content Browser-a* i stavljanjem na neke od lokacija unutra mape. Kada se to učini, kursor će se promijeniti tako da se zna da se vrsta *asset-a* može ispustiti na određeni *viewport.*[24] (Sl.6.3).




Sl.6.3. *Drag and drop* metoda dodavanja aktera iz *Contetn Browser-a*

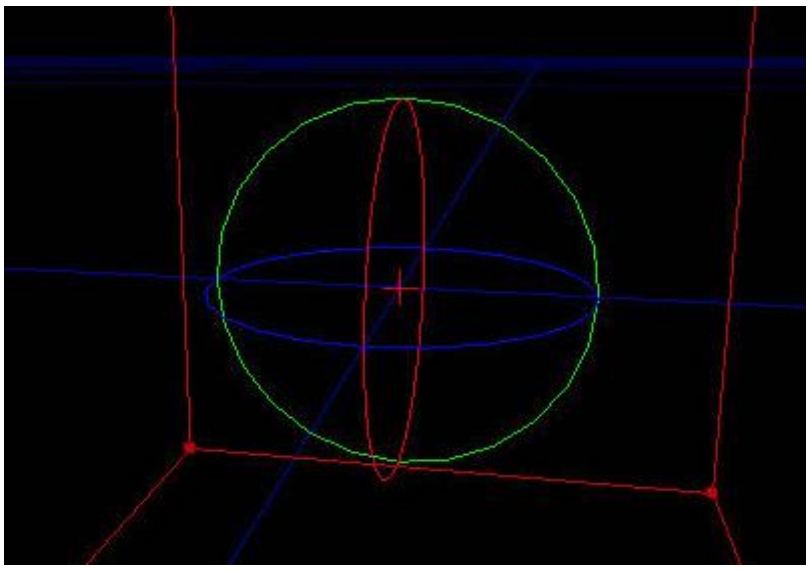
Za samo uređivanje mape korišteni su alati iz *Toolbar-a* i *toolbox-a.*

- *Translation widget* - sastoji se od skupa boja označenih strelica usmjerenih u pozitivnom smjeru svake osi. Svaka od tih strelica se može uhvatiti (klikom miša na nju) i povući kako bi se odabrani akter premjestio duž uhvaćene osi. kad je miš iznad jedne od osi, ona mijenja boju. (Sl.6.4)



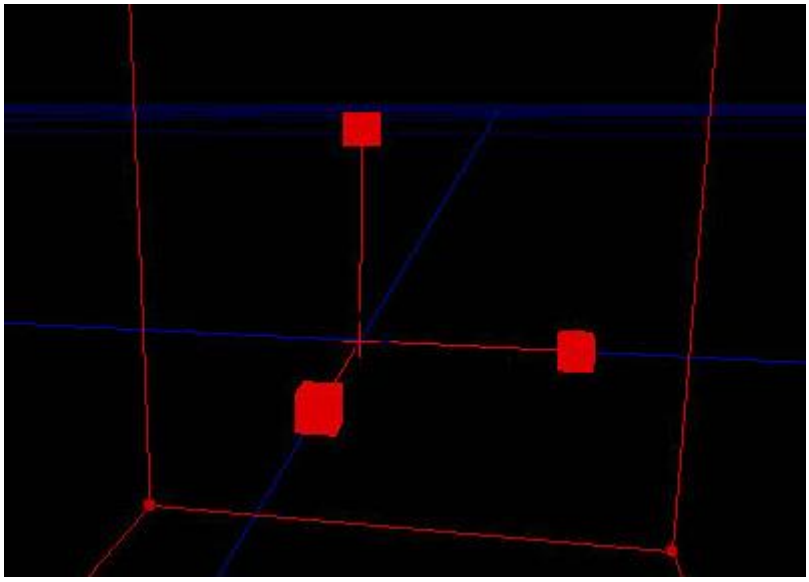
Sl.6.4. *Translation widget*

- *Rotation widget* - sastoji se od bojom označenih kružnica koje su pridružene svaka svojoj osi. Služi za rotaciju odabranog aktera oko željene osi. (Sl.6.5)



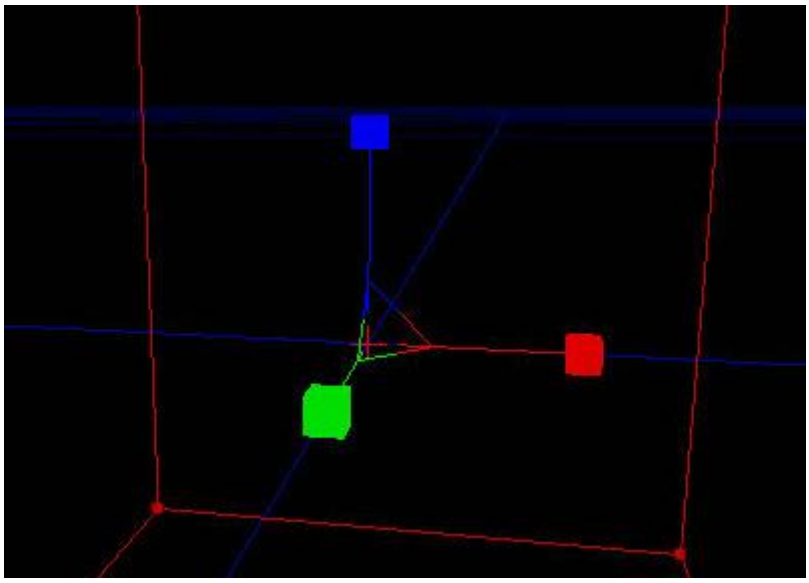
Sl.6.5. *Rotation widget*

- *Uniform Scale Widget* - sastoji se od tri ručke koje kad se uhvate i povuku povećavaju/smanjuju odabranog aktera u sva tri smjera istovremeno. (Sl.6.6)



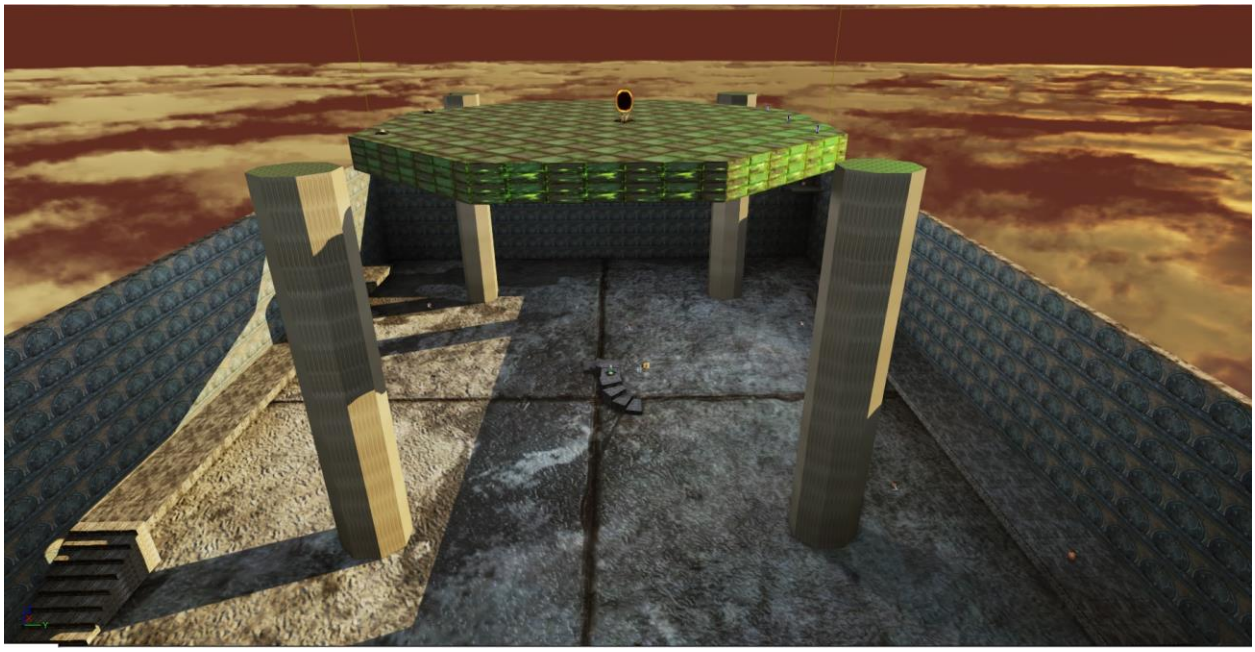
Sl.6.6. *Uniform scale widget*

- *Non-uniform Scale Widget* - skoro identičan kao i *Uniform Scale Widget* samo što se akter može smanjivati/povećavati po određenoj osi ovisno koja se odabere.[24] (Sl.6.7)

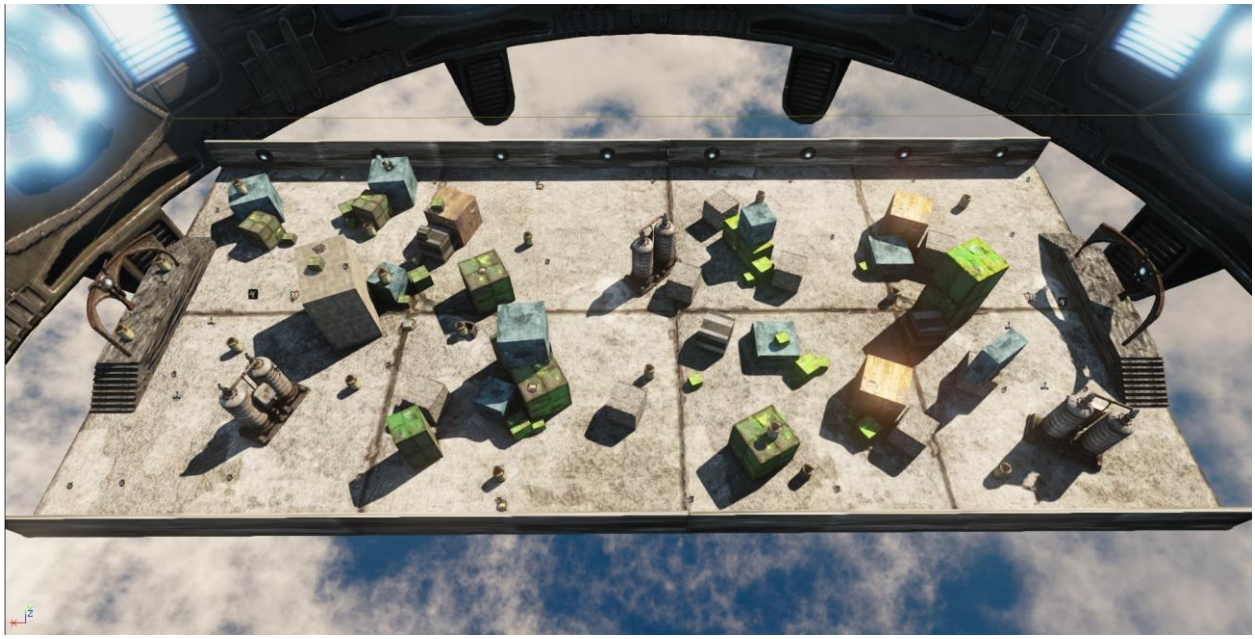


Sl.6.7. *Non-uniform Scale Widget*

Na kraju, kombinirajući gore navedene metode dobiva se krajnji rezultat izgleda određenog nivoa. (Sl.6.8) i (Sl.6.9)



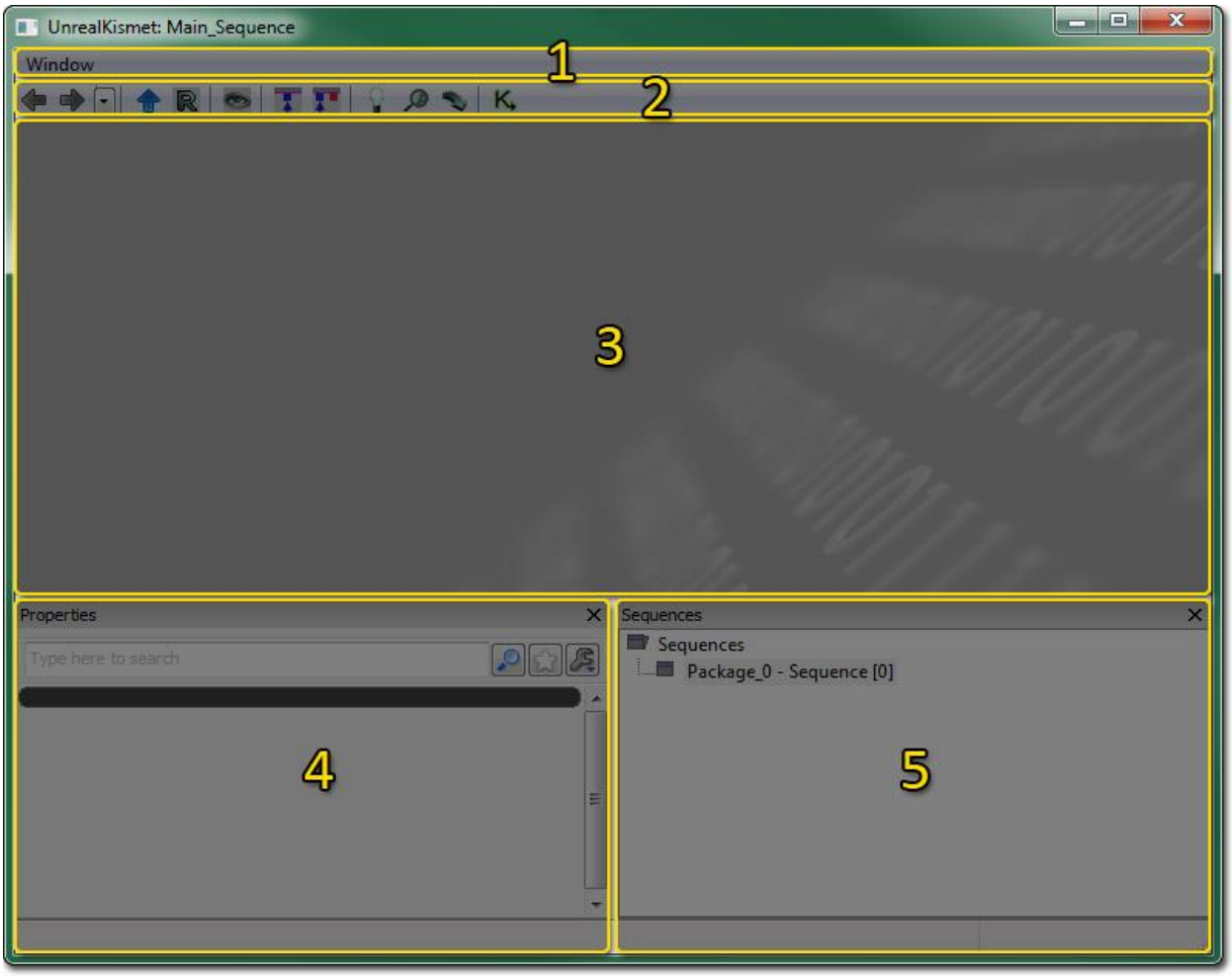
Sl.6.8. Izgled nivoa *DeathMatch*



Sl.6.9. Izgled nivo *Capture The Flag*

## 6.3. Kismet

Alat *UnrealKismet* je vrlo fleksibilan i moćan alat koji omogućuje skriptiranje složenih igrivosti u nivou. Djeluje tako što vam omogućuje povezivanje jednostavnih funkcionalnih sekvencijalnih objekata u obliku složenih sekvenci. Kismet je podijeljen u nekoliko područja: [24]

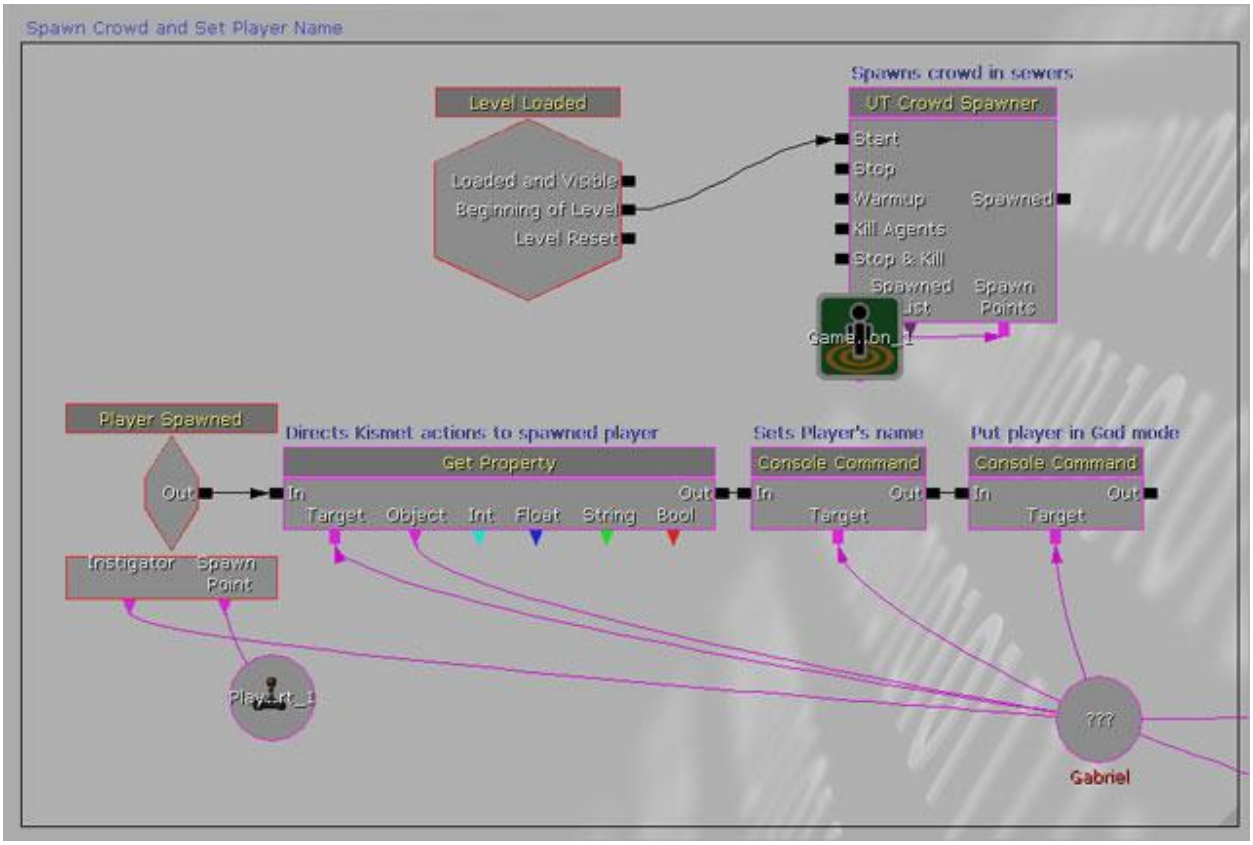



Sl.6.10. *Kismet*

(Sl.6.10.)

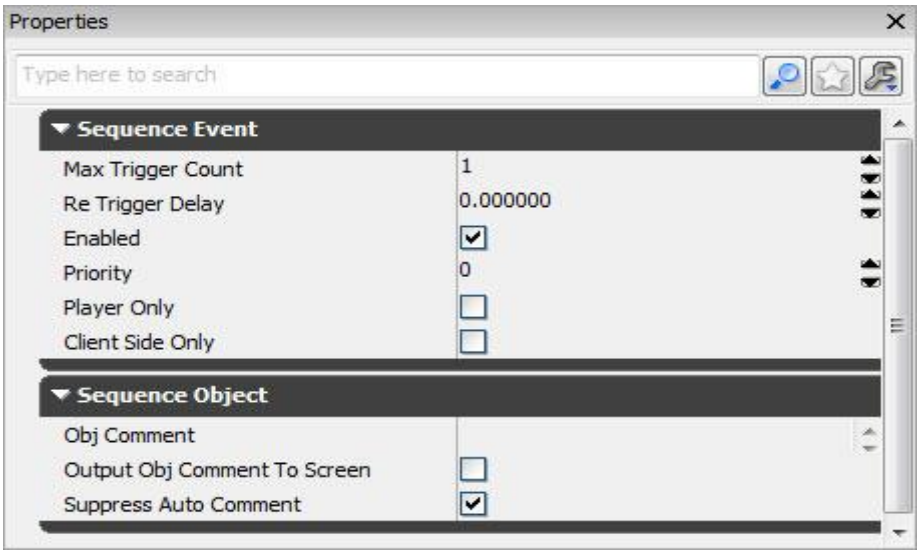
1. Menu bar
2. Tool bar
3. Graph pane
4. Properties Pane
5. Sequences Pane

Najbitniji dio je svakako *Graph pane* koji prikazuje akcije, uvijete, evente, varijable i pod-sekvence. (Sl.6.11)




Sl.6.11. *Graph Pane*

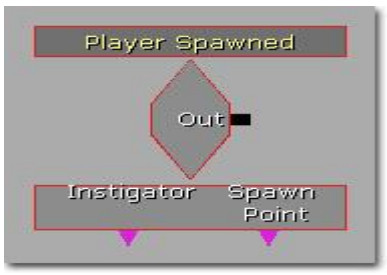
Nadalje, *Properties pane* prikazuje svojstva pripadajućem odabranom objektu u *Graph pane-u.* Svojstva se naravno mogu modificirati kako to programer želi i ukoliko je potrebno. (Sl.6.12)




Sl.6.12. *Properties pane*

Postoje 4 kategorije objekata koji se mogu staviti u sekvencu:

- Event - objekti koji kreiraju unos u sekvenci, eventualno od aktera u igri. (Sl.6.13)

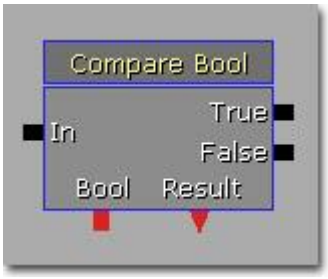



Sl.6.13. *Event*

- Action - To su objekti koji izvode akcije na akterima u nivou. Ulazi se nalaze na lijevoj strani, izlaz na desnoj ,a konekcije varijable na dnu. (Sl.6.14)


Sl.6.14. *Action*

- Condition - uvjeti koji ne utječi zapravo na nivo, već kontroliraju tijek sekvence. (Sl.6.15)




Sl.6.15. *Condition*

- Variable - ovi objekti jednostavno pohranjuju informacije o određenoj vrsti za korištenje od strane *Event-a, Action-a ili Condition-a.*[24] (Sl.6.16)

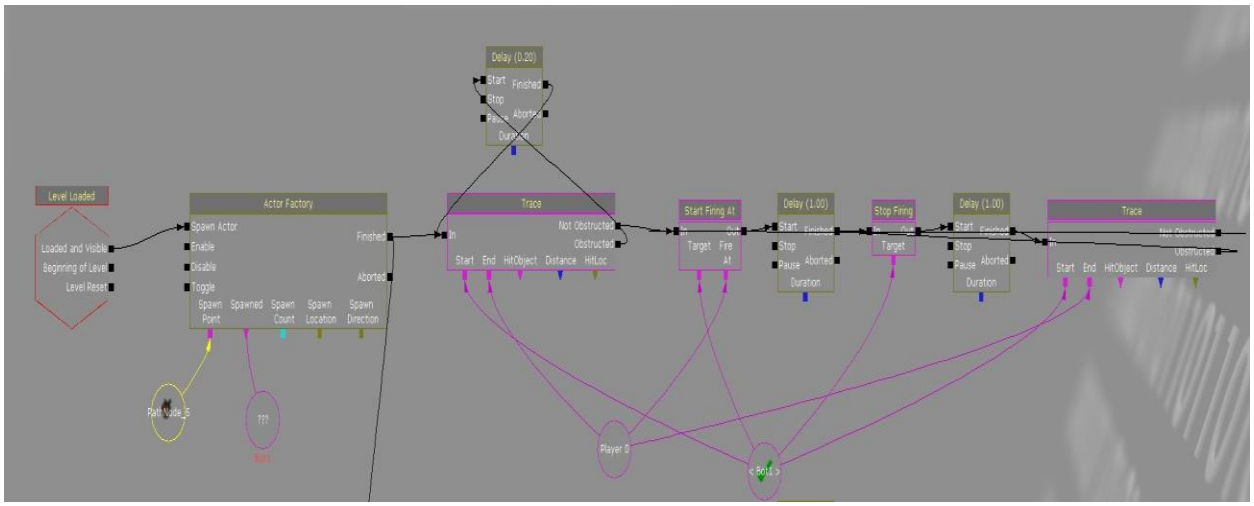

Sl.6.16. *Variable*

## 6.4. Izvedba logike i umjetne inteligencije botova

Sama logika igre i način na koji će se protivnici ponašati unutar igre izvedena je preko Kismet-a. Koristeći ranije već objašnjene objekte. Prije samoga početka kreiranja sekvenci unutar Kismeta u nivo je bilo potrebno dodatni određene objekte koji će kasnije kada ih se stavi u funkciju utjecati na krajnji rezultat igre. To se prije svega odnosi na *Pathnode-s,PlayerStart,Pickups Teleporters. Pathnode-s* koristimo kao točke u putanji kretanja botova, *PlayerStart* kao startnu poziciju igrača kada se nivo učita, *Pickup's i Teleporter's* kao objekte koje igrač i botovi mogu koristiti. Prema slici 6.17., kako bi igrač znao što mu je cilj unutar igre, kroz Kismet, je napravljena sekvenca objekata koja ispisuju na početku svakog nivo što je igraču činiti.

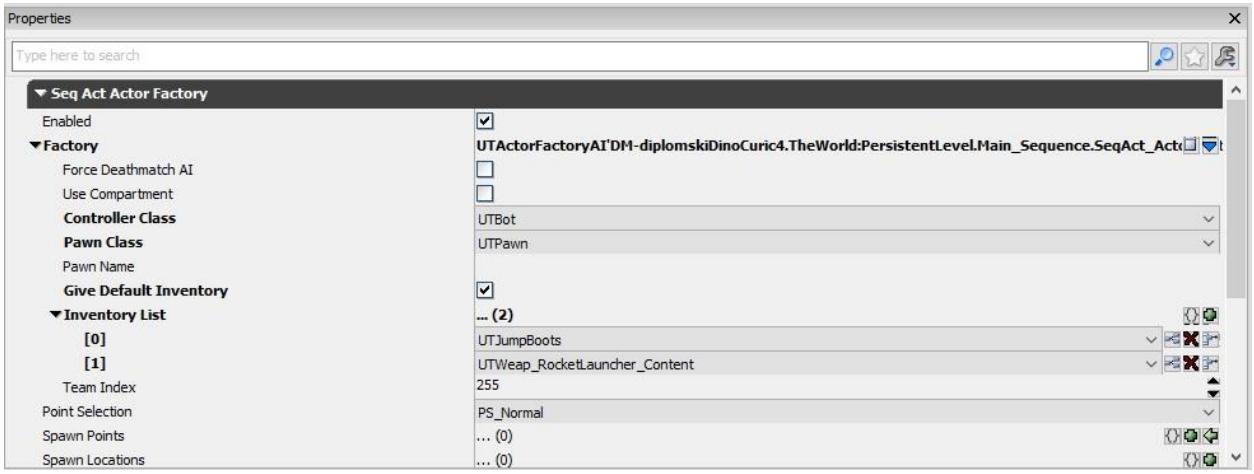

Sl.6.17. Sekvenca ispisa teksta na početku nivoa

Kao što se vidi iz slike Korišten je *Event- Level Loaded* i *Action- Play Announcement i Delay.* Svojstva akcije *Play Announcement* se moraju modificirati kako bi se tekst ispisao, što se vidi iz slike pod *Announcement Text*. Ova sekvenca na početku nivo ispisuje navedeni tekst kako bi igrač imao nekakvu sliku što bi trebalo raditi u nivou. Trajanje teksta produženo je sa akcijom *delay* cija su svojstva isto promijenjena što se vidi na samom objektu iz slike. Na slici 6.18. prikazan je niz objekata koji daju nekakvu umjetnu inteligenciju botovima koji se stvaraju na početku nivoa.




Sl.6.18. Sekvenca stvaranja bota

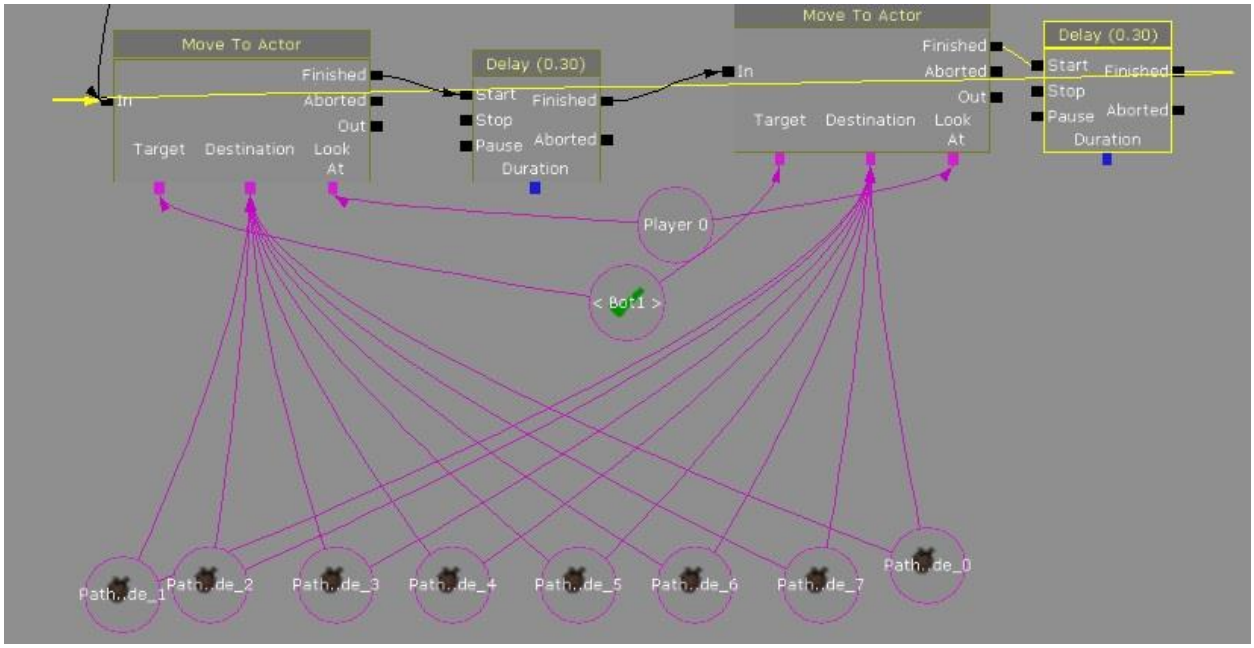
Objekti korišteni u ovaj sekvenci su : *Level loaded, Actor factory, trace, Start firing at, Stop firing, delay i varijable player i bot.* Ova sekvenca na početku nivo sa objektom *Level loaded* stvara bot-a čije se karakteristike i svojstva nalaze u objektu *Actor factory.* (Sl.6.19)




Sl.6.19. Svojstva *Actor factory-a*

Što je bitno podesiti u svojstvima je *Controller Class* na UTBot, gdje se stvara standardni bot iz baze UDK-a, nadalje *Pawn class* na UTPawn koji označava samo klasu bot-a. Podešeno je i koja

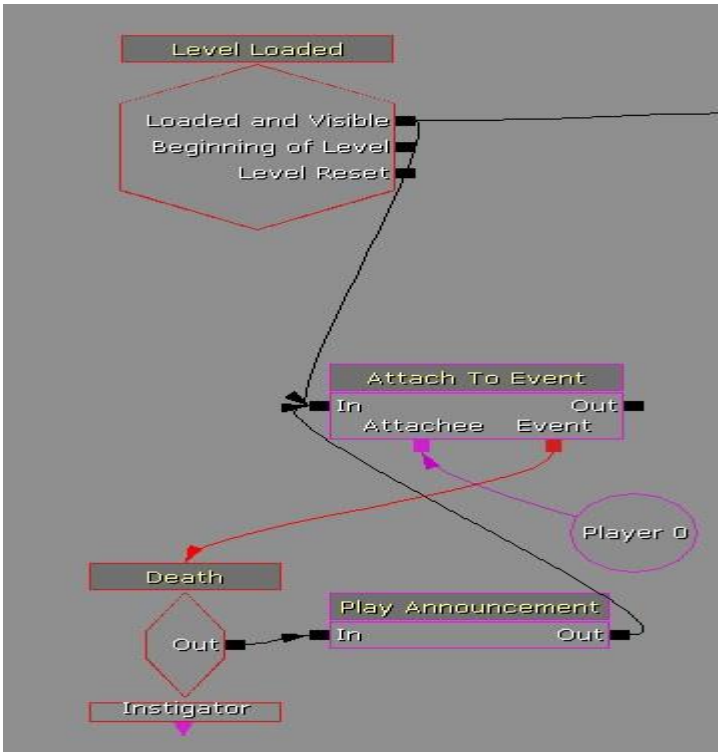
će sva oružja i ostale alate imati botovi. To je podešeno pod *Intentory List,* kako se vidi botovi će imati kao oružje *Rocket Launcher i Jump Boots.* Potrebno je još izdvojiti da opcija *Force Deathmatch AI* mora biti odznačena kako botovi ne bi ubijali sami sebe. Za varijable spojene na objekt *Actor factory* dodan je na *Spawn point, pathnode5* što označava gdje će se određeni bot stvoriti na početku nivo. I na Opciju *Spawned* povezana je varijabla bot-a da se zna na kojeg se bot-a sekvenca odnosi. Primarna zadaća ove sekvence je da bot u danom nivou nađe igrača i pokuša ga ubiti. To je postignuto objektima *Trace, Start firing at, Stop firing.* Oba objetka *Trace* su stavljena u petlju kako bi se događaj odvijao dok igrač postoji odnosno dok ga se ne ubije. Objektima *Delay* samu su stavljeni razmaci između ostalih objekata kako bi sekvenca radila što bolje. Naravno svaki od objekata je povezan sa odgovarajućom varijablom kako bi program prepoznao za koga je namijenjen određeni objekt. Za samo kretanje bot-ova i prepoznavanje igrača zaslužna je sljedeća petlja prikazana na slici 6.20..




Sl.6.20. Sekvenca kretanja bot-ova i prepoznavanja igrača

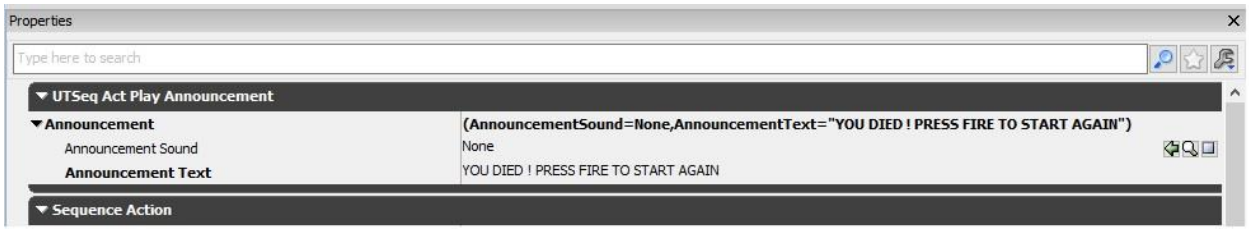
Korišteni objekti su *Move to Actor,delay,variable i pathnode's.* Kao što je već ranije spomenuto u nivou su postavljeni *pathnode-ovi,* koji služe bot-ovima za kretanje po nivou. Ova Sekvenca upravo to radi, bot će se kretati između zadanih *pathnode-ova* i tražiti igrača. Varijable objekata *Move to Actor*, *destination* povezana su sa *pathnode-ovima* kako bi se dobila gore navedena radnja, na *target* je povezena varijabla *bot* kako bi se znalo tko izvršava radnju i na *Look At* povezana je varijabla *player* kako bi bot znao koga treba tražiti. Objektima *delay* modificirana je sekvenca kako bi bolje radila. Ovakva sekvenca je napravljena 4 puta u samome nivou za 4 bot-a

koji se stvaraju na početku nivoa. Na početku samog nivoa dan je uvjet pomoću sekvence kada igrač umre da se ispiše odgovarajuća poruka i da do znanja igraću da se pritiskom tipke na mišu za pucanje vraća nazad u igru . (Sl.6.21.)



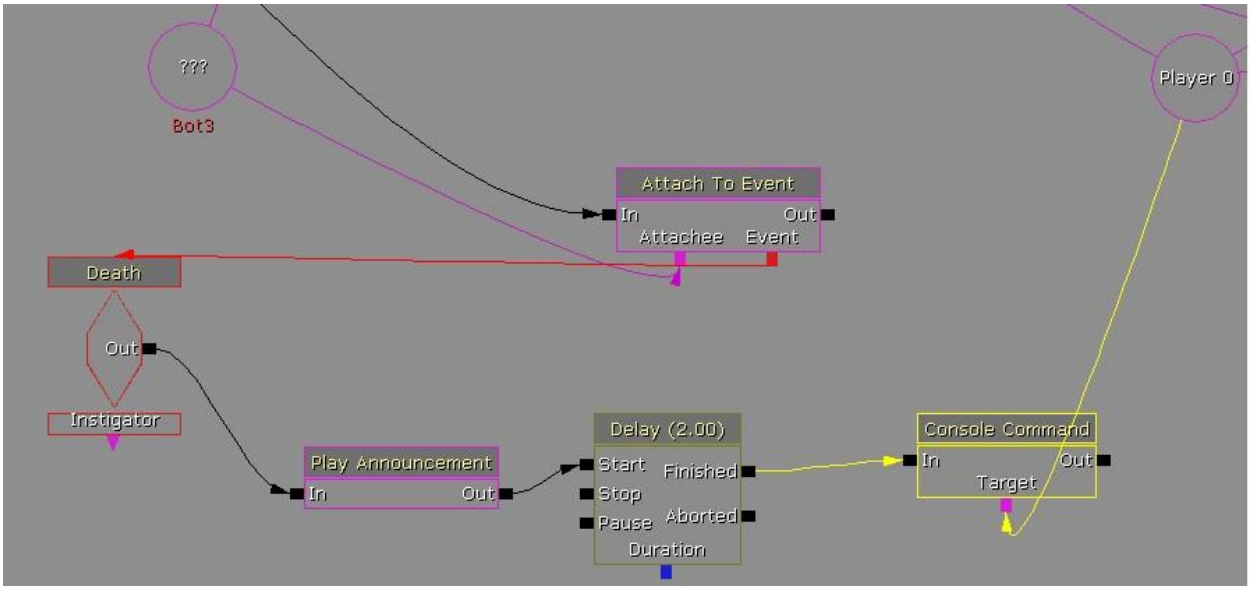

Sl.6.21. Sekvenca ispisa poruke nakon smrti igrača

Korišteni objekti su *Level Loaded, Attach To Event, Death, Play Announcement i variabla.* Na početku nivo kreira se event za igrača, pošto je opcija *Attachee* spojena na varijablu *player,* da ukoliko igrač umre, ispiše se poruka na ekranu što je modificirano u svojstvima objekta *Play Announcement* pod *Announcemetn Text.* (Sl.6.22.)




Sl.6.22. Svojstva objekta *Play announcement*

Na kraju je predviđeno da igra završava kada igrač ubije sve bot-ove na mapi što je prikaznao na slici 6.23..

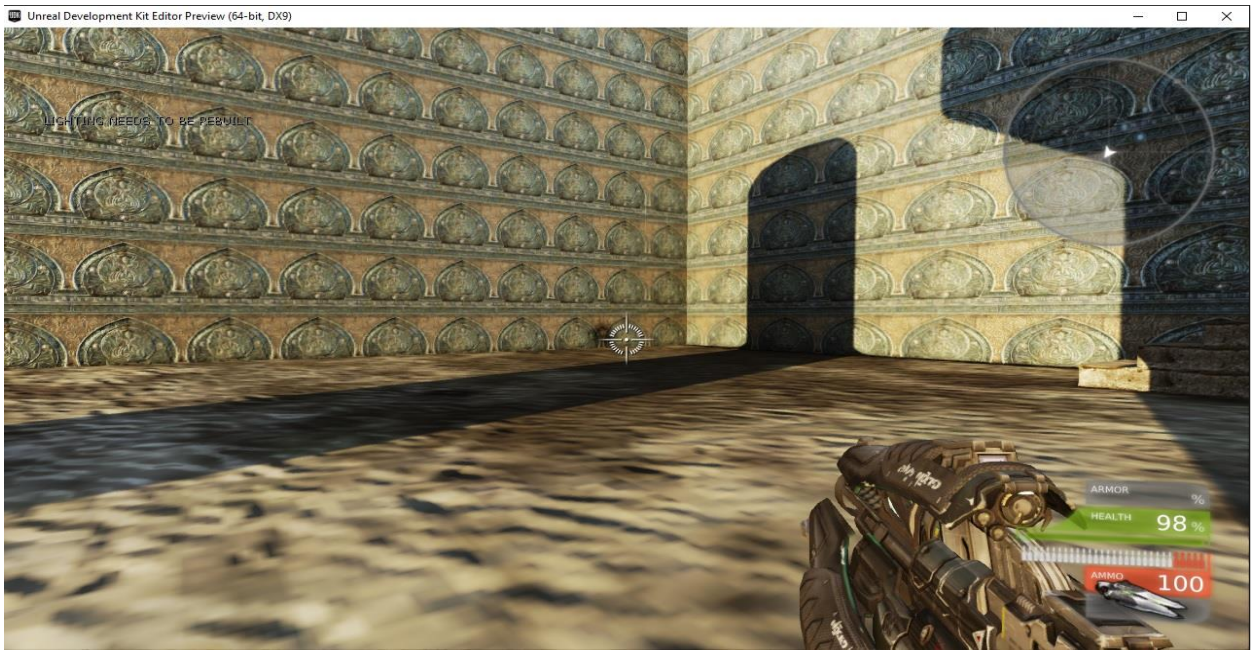



Sl.6.23. Sekvenca završetka nivoa

Kao i u prošloj petlji koriste se svi isti objekti samo je sada događaj vezan za smrt bot-ova, a ne za igrača. Jedina razlika u ovoj petlji je objekt *Consol command* koja poziva funkciju konzole *open* ako je ispunjen uvjet da su svi bot-ovi mrtvi i vrača igrača u izbornik što je vidljivo pod *Commands* - open UDKFrontEndMap.udk. (Sl.6.24.)

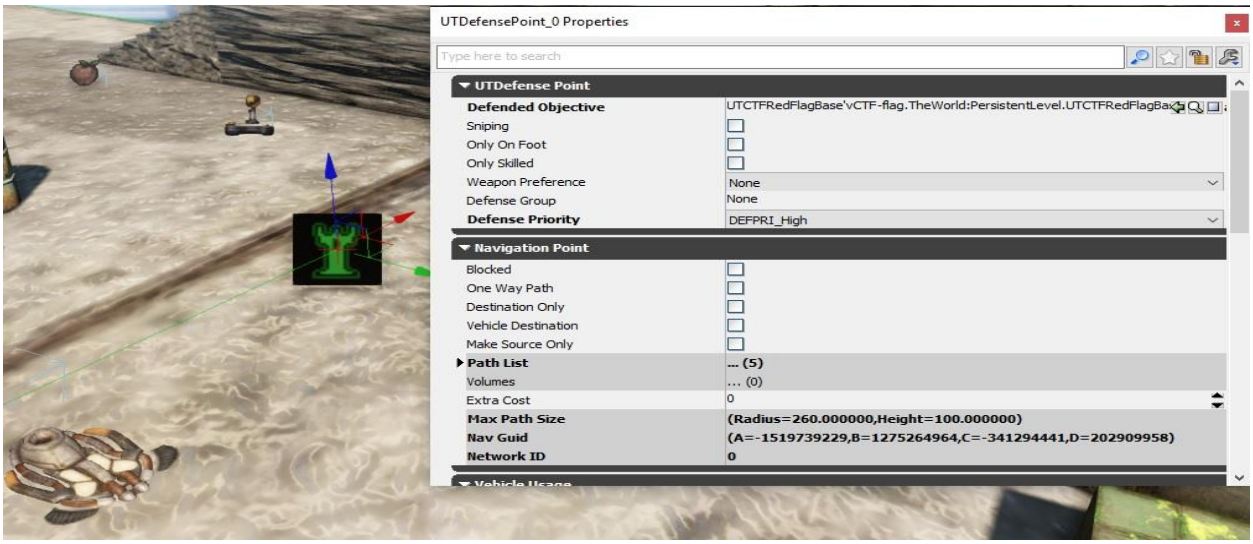

Sl.6.24. Svojstva objekta *Consol Comand*

Kao zadnji korak da bi igra funkcionirala na željeni način dolazi spremanje samog projekta koji igra veliku ulogu. Ovaj nivo je zamišljen da bude *DeathMatch,* stoga, kod spremanja samoga projekta potrebno ga je spremiti u obliku DM-nazivprojekta.udk. S time se postiže da sam program prepoznaje o kojem tipu igre se radi i dodjeljuje mu već predodređene komponente kao što su oružje, *Healthbar*, *Armorbar,* municija i radar. (Sl.6.25)



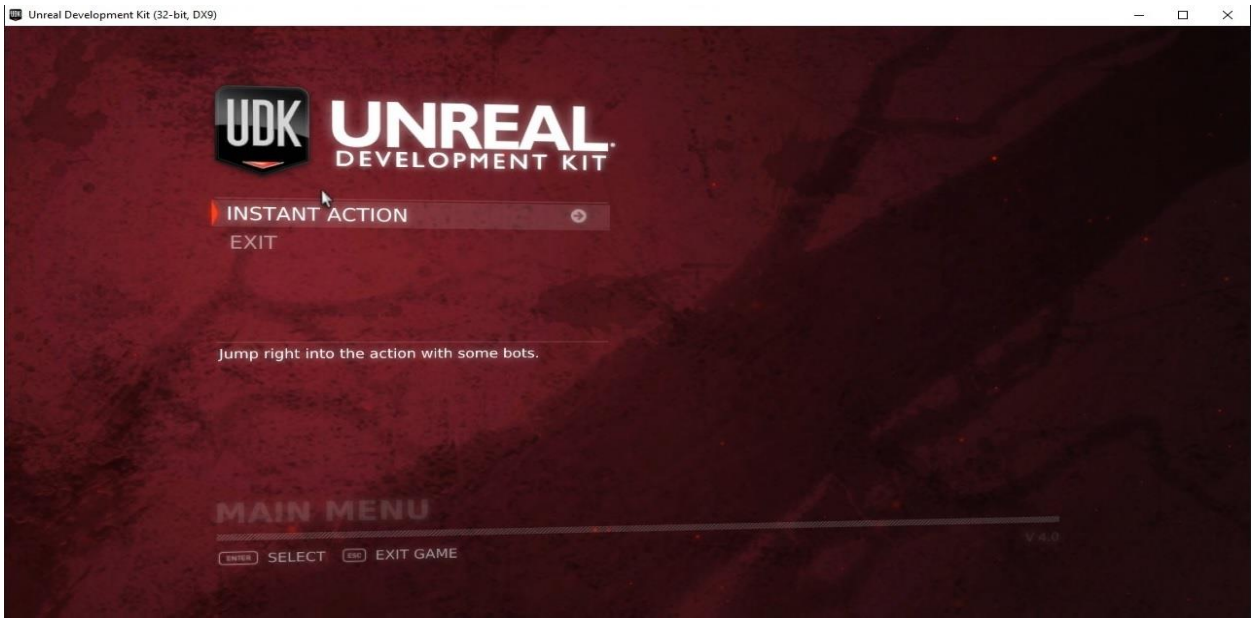

Sl.6.25. Izgled HUD-a

Za nivo u kojem je tip igre *Capture The Flag* projekt se sprema u obliku vCTF-nazviprojekta.udk. Program prepoznaje tip igre i isto tako mu dodjeljuje već predodređenu logiku i ostale stvari kao i kod *DeathMatch-a.* U ovom nivo isto tako su dodani *pathnode-ovi* kako bi bot-ovi mogli navigirati kroz mapu i ostvariti zadani cilj igre, a to je imati više zastavica u zadanom vremenu ili skupiti 3 zastavice u zadanom vremenu . Razlika od *DeathMatch-a* je ta što su se ovdje moral koristiti dodatne stvari kako bi botovi bili malo pametniji. Dodane su naravno zastavice crvene i plave boje te *DefensPoint-i* koji su kroz svoja svojstva povezana sa svojom zastavicom kako bi bot-ovi u određenom timu znali da zastavicu na toj lokaciji treba braniti. Na slici 6.26. prikazana su svojstva objekta *defense point-a.*

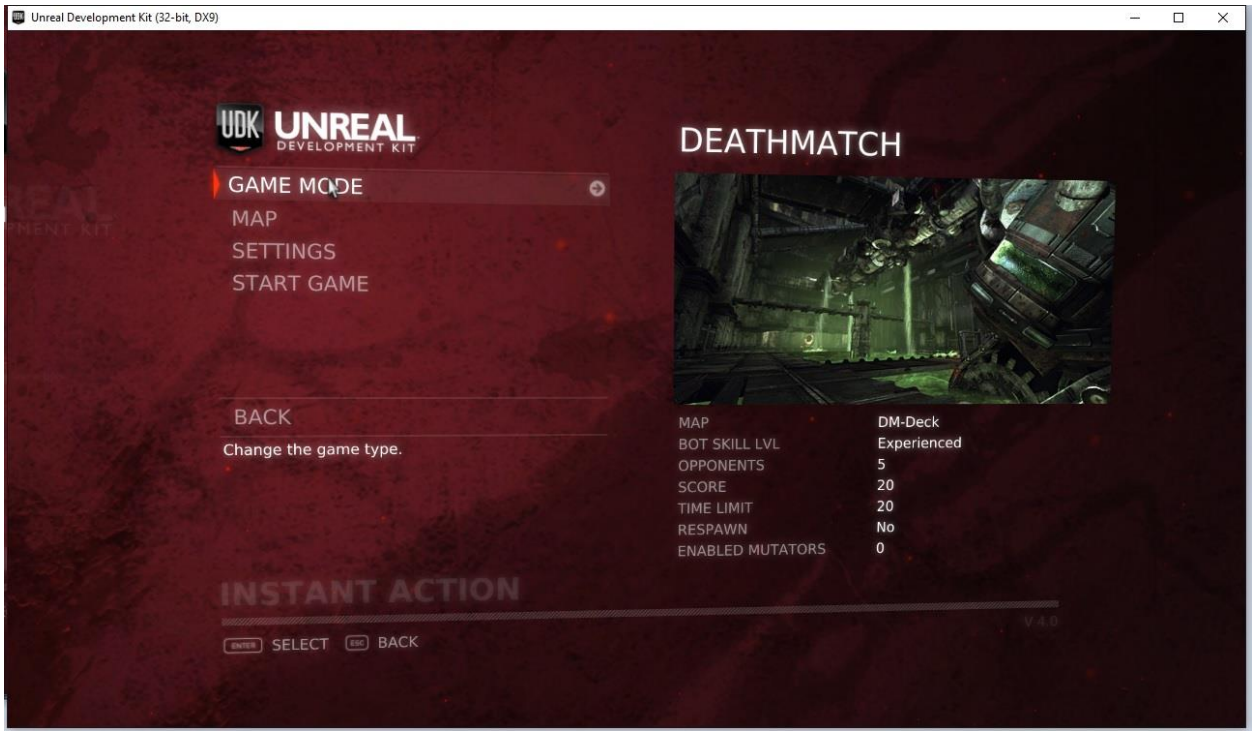



Sl.6.26. Svojstva objekta *Defense point*

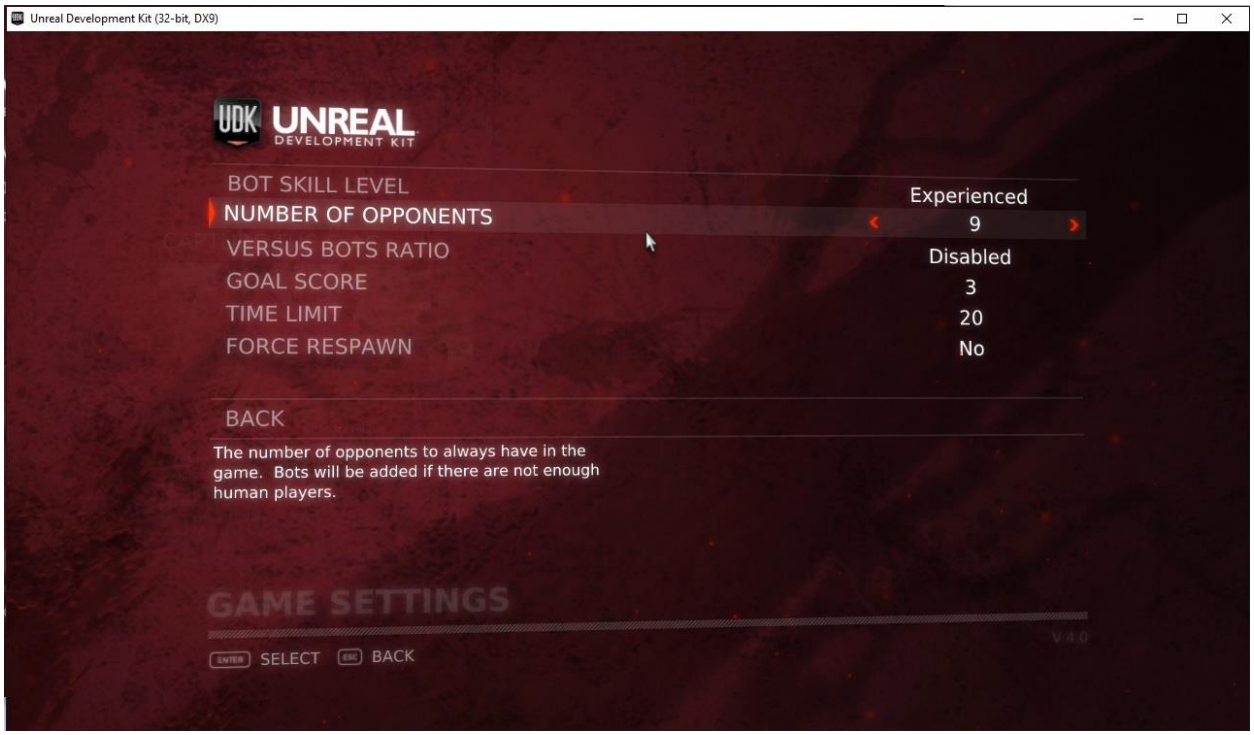
Broj bot-ova u timovima u nivou *Capture The Flag*, težina bot-ova, vrijeme trajanja i broj zastavica izabire se u početnom izborniku . Za početni izbornik kada se ulazi u igru izabran je već postojeći izbornik UDK-a, kroz koji se lagano može utjecati na gore navedene stvari. Bitno je imati konstruiran nivo i potrebne objekte u nivou kako bi igra funkcionirala. Na slikama 6.27., 6.28., 6.29. prikazan je izgled *menu-a* dok je na slikama 6.30., 6.31., 6.32., 6.33. prikazan izgled oba nivoa.



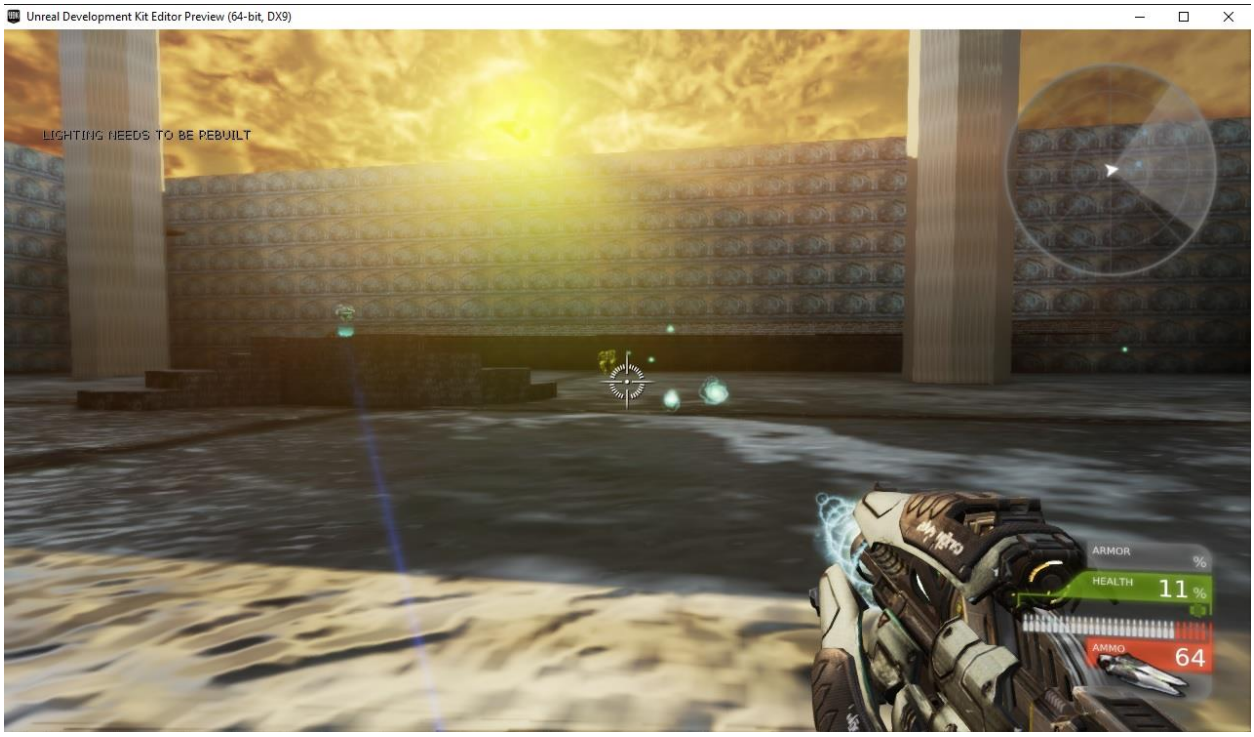

Sl.6.27. Izgled izbornika 1



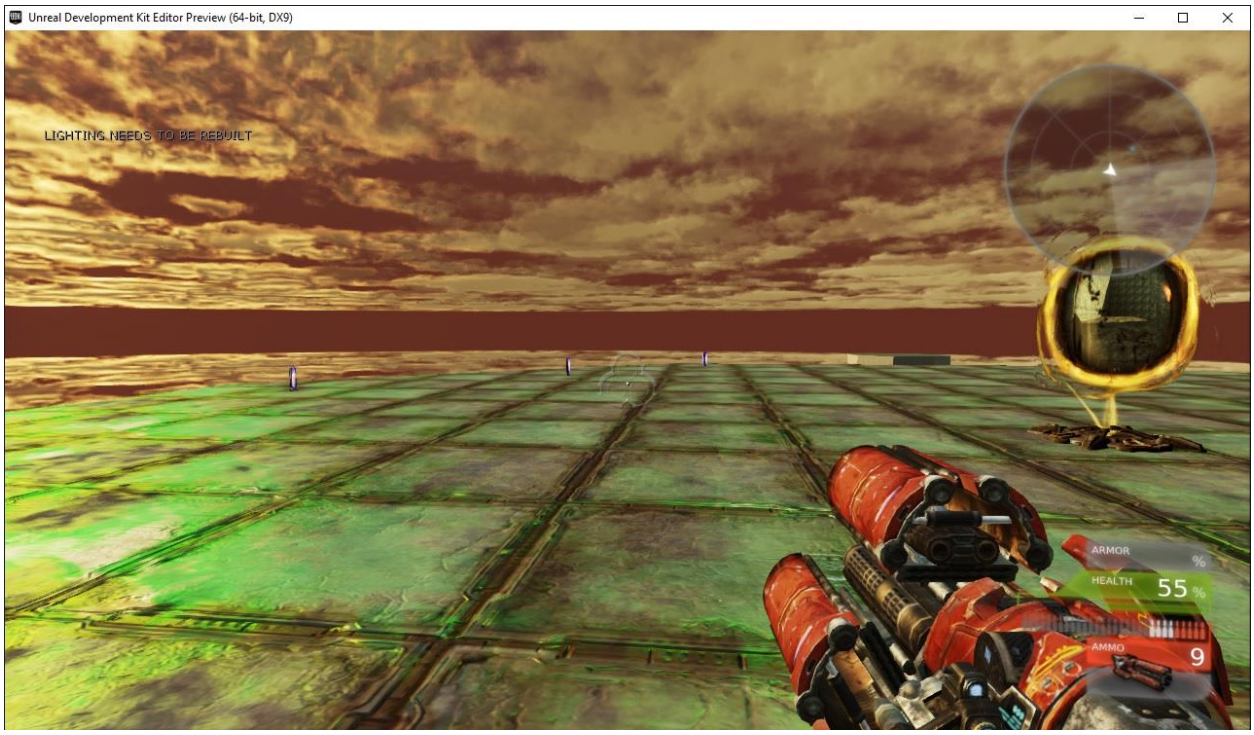

Sl.6.28. Izgled izbornika 2




Sl.6.29. Izgled izbornika 3

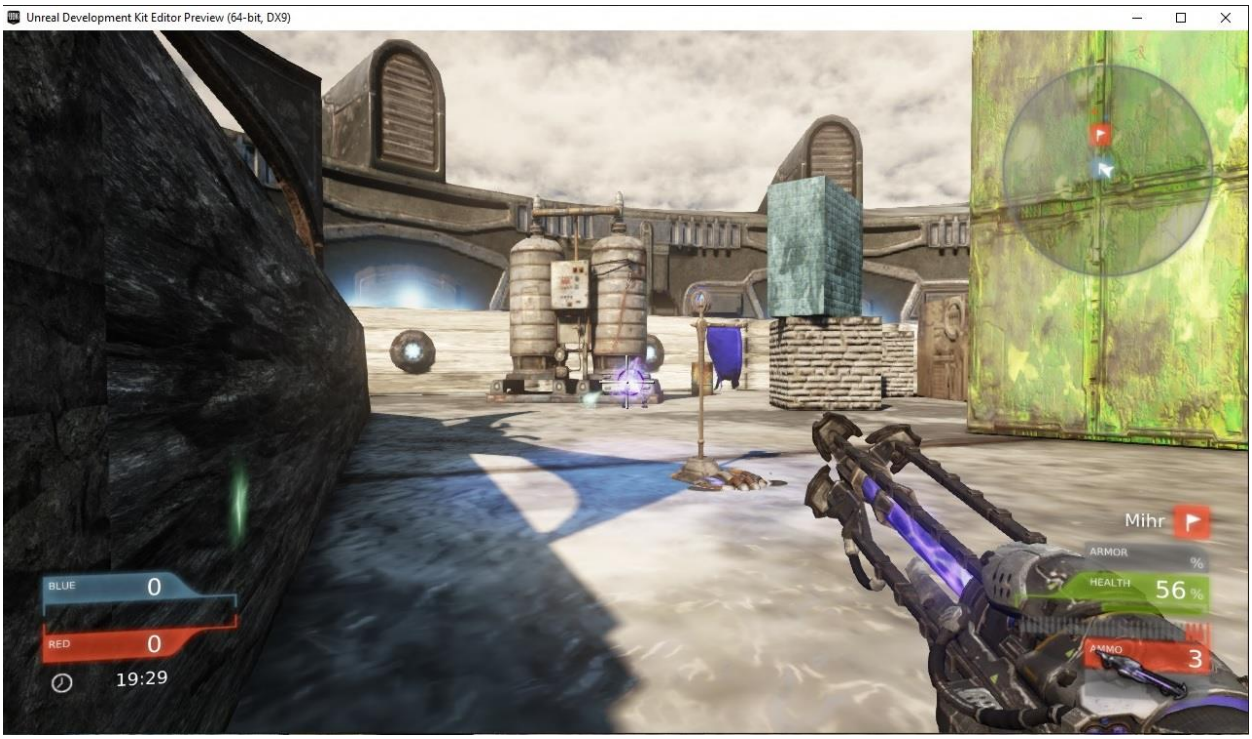



Sl.6.30. *Ingame DeathMatch* 1




Sl.6.31. *Ingame DeathMatch* 2


Sl.6.32. *Ingame Capture the flag* 1




Sl.6.33. *Ingame Capture the flag* 2

# 7. ZAKLJUČAK

Cilj ovog rada je izraditi 3D igru u Unreal engine-u. U početku rada objašnjena je općenito 3D grafika i alati koji su doveli da ona izgleda kao danas. Naglasak je stavljen na sam program koji se koristi za izradu igre, Unreal engine. Opisane su metode koje se pojavljuju unutar programa te alati koji su se koristili kako bi se ovaj radi proveo do kraja. Opisana je sama struktura igre i njezin tijek. Najveći problem nekome tko počinje sa programiranjem i izradom igara je ustvari opseg mogućnosti programskog okruženja u kojem se igra izvodi. Kako je ova igra rađena u UDK-u, ovo je tek početak onoga za što je ovaj alat sve moguće koristiti. Ovaj rad je dobar uvod u područje razvoja video igara, a je UDK kao takav jako moćan alat u izradi iste.

# LITERATURA


[1] Cornell University Program of Computer Graphics, "What is Computer Graphics?", Accessed November 17, 2009.

[2] A. Kapoor, "Shading", https://www.slideshare.net/akbrightfuture/shading-10305107, Nov 24, 2011

[3] H. Gouraud, "Continuous shading of curved surfaces," IEEE Transactions on Computers, 20(6):623–628, 1971

[4] Watt, Alan H.; Watt, Mark (1992), "Advanced Animation and Rendering Techniques: Theory and Practice.", Addison-Wesley Professional pp.21–26. ISBN 978-0-201-54412-1

[5] B. T. Phong, "Illumination for computer generated pictures", Communications of ACM 18 (1975), no. 6, 311–317.

[6] Wikipedia, Hightmap, https://en.wikipedia.org/wiki/Heightmap

[7] J. Kautz1, W. Heidrichy and Hans-Peter Seidel, "Real-Time Bump Map Synthesis", Max-Planck-Institut für Informatik, University of British Columbia

[8] Heidrich and Seidel, "Realistic, Hardware-accelerated Shading and Lighting", SIGGRAPH 1999

[9] http://web.archive.org/web/20060325144730/http://www.neoseeker.com/Hardware/faqs/kb/10,72.html, Quoted From NVIDIA Corporation

[10] Wikipedia, Multisample anti-aliasing, https://en.wikipedia.org/wiki/Multisample_anti-aliasing

[11] Lottes, Timothy (February 2009). "FXAA", NVIDIA. Retrieved 29 September 2012.

[12] Atwood, Jeff (December 7, 2011). "Fast Approximate Anti-Aliasing (FXAA)". Coding Horror. Retrieved September 30, 2012.

[13] NVIDIA,TXAA, http://www.geforce.com/hardware/technology/txaa/technology

[14] Parent 2012, pp. 193–196.

[15] Parent 2012, pp. 324–326.

[16] Parent 2012, pp. 111–118.

[17] Masson 1999, p. 118.

[18] Masson 1999, p. 204.

[19] Reilly, Luke (30 September 2013). "The Most Impressive Physics Engine You've Never Seen". IGN. Retrieved 1 December 2013.

[20] Wikipedia, Physics engine , https://en.wikipedia.org/wiki/Physics_engine

[21] "What is a Game Engine?". GameCareerGuide.com. Retrieved 2013-11-24.

[22] Cowan, Danny. "Joystiq". Gamedaily.com. Retrieved 2013-11-24.

[23] Wikipedia, Unreal Engine , https://en.wikipedia.org/wiki/Unreal_Engine

[24] UDK home website, UDK, https://docs.unrealengine.com/udk/Three/WebHome.html

# SAŽETAK

Pri izradi određene igre potrebno je unaprijed odrediti u kojem će se programskom okruženju igra izvesti. Zatim treba dobro razraditi plan i dijagram tijeka izvršavanja same igre, od kreiranja samih nivo do umjetne inteligencije koju koriste akteri u samoj igri. U ovom radu igra je rađena u UDK-u koji ima sve dostupne alate kako bi netko uspio izraditi željenu igru. Sam UDK kao program ima jako puno mogućnosti od kojih su u ovom radu prezentirane samo neke, potreban da se igra izvede do kraja.

**Ključne riječi**: igra, programsko okruženje, nivo, umjetna inteligencija, UDK

## ABSTRACT, THE USE OF UNREAL ENGINE IN THE DEVELOPMENT OF 3D COMPUTER GAMES

When designing a particular game, it is necessary to determine in advance which program environment will be used to make the game. Then, it is necessery to develop a plan and diagram of the course of the game itself, from creating the level to the artificial intelligence used by the players in the game itself. In this paper, game is made in UDK, which has all the tools available to help someone make the desired game. The UDK itself as a program has a lot of features, of which only some are presented in this paper, crucial for making the game.



**Keywords :** game, program environment, level, artificial intelligence, UDK

# ŽIVOTOPIS

Dino Ćurić rođen je 28.01.1989. godine u Osijeku. Osnovnu školu „Retfala“ završava 2003. godine u Osijeku nakon čega upisuje III. Gimnaziju u Osijeku. Preddiplomski studij računarstva na Elektro-tehničkom fakultetu u Osijeku upisuje 2007. Diplomski studij računarstva, smjer procesno računarstvo upisuje 2014.