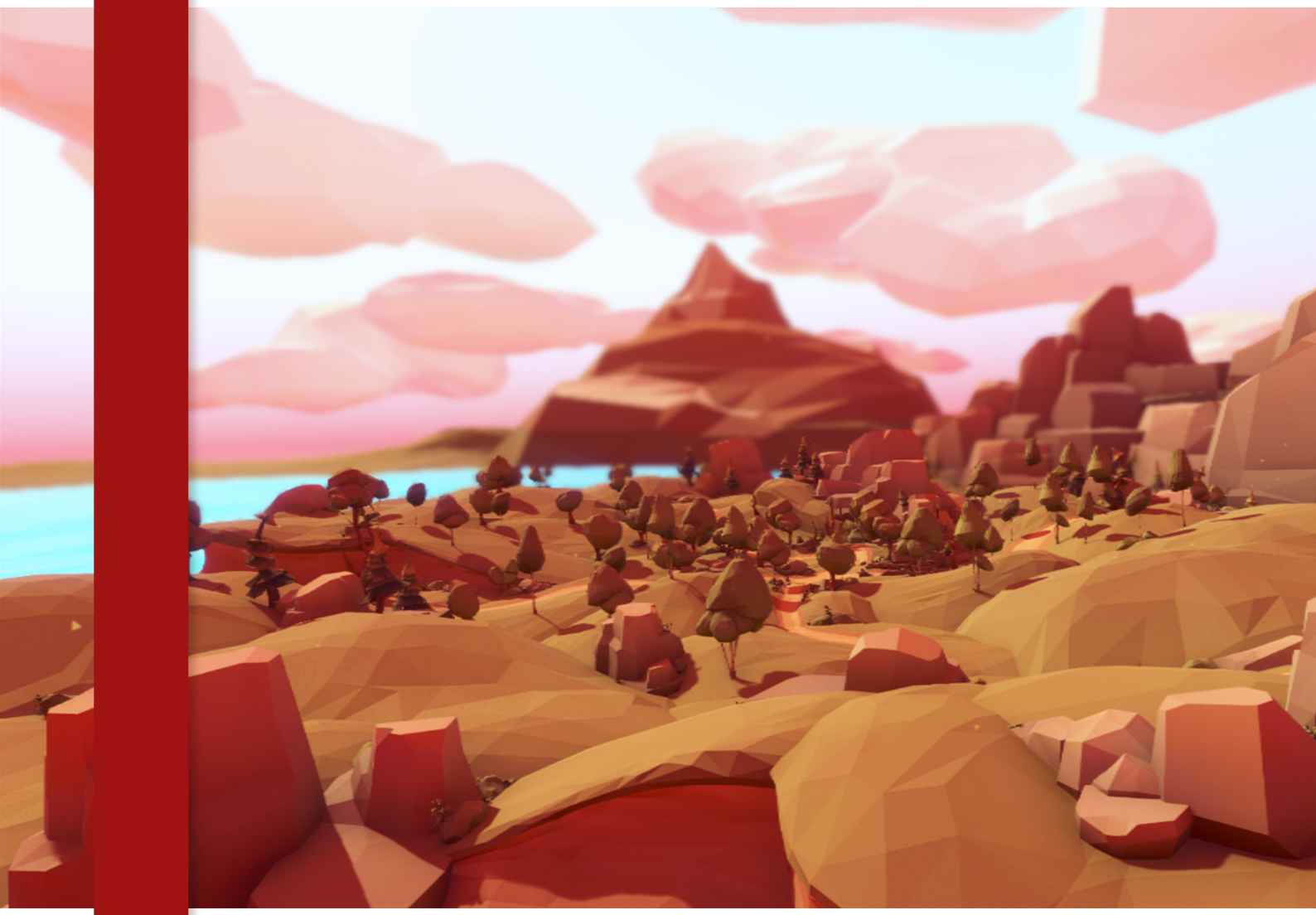
FAKULTET ORGANIZACIJE I INFORMATIKE



ZBORNIK RADOVA

RAČUNALNE IGRE 2020

STRUČNA KONFERENCIJA

UREDNICI

doc. dr. sc. Mario Konecki
doc. dr. sc. Mladen Konecki
Andrej Kovačević

ISSN 2623-7679

IMPRESUM

IZDAVAČ Konferencija Računalne igre 2020

NAKLADNIK Fakultet organizacije i informatike
Pavlinska 2, 42000 Varaždin

KONTAKT racunalne-igre@foi.hr

UREDNICI doc. dr. sc. Mario Konecki
Sveučilište u Zagrebu

doc. dr. sc. Mladen Konecki
Sveučilište u Zagrebu

Andrej Kovačević
Exordium Games

GLAVNI UREDNIK doc. dr. sc. Mario Konecki
Sveučilište u Zagrebu

DATUM ODRŽAVANJA KONFERENCIJE 30. prosinca 2020.

MJESTO ODRŽAVANJA KONFERENCIJE Fakultet organizacije i informatike
Pavlinska 2, 42000 Varaždin

ISSN 2623-7679

STRUČNA KONFERENCIJA

RAČUNALNE IGRE 2020

30. prosinca 2020.

ORGANIZATORI I SUORGANIZATORI

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE VARAŽDIN

CECIIS
Central European Conference on Information and Intelligent Systems

Sveučilište u Zagrebu
Sveučilište u Rijeci
Sveučilište u Splitu
Sveučilište Josipa Jurja Strossmayera u Osijeku
Sveučilište u Zadru
Sveučilište u Dubrovniku
Universitas Studiorum Polensis
Sveučilište Sjever

ALGEBRA VISOKO UČILIŠTE

MACHINA GAME DEV ACADEMY

VERN'

Grad Varaždin

Grad Bjelovar

Grad Daruvar

GENERALNI POKROVITELJ

SISAČKO-MOSLAVAČKA ŽUPANIJA

STRUČNA KONFERENCIJA

RAČUNALNE IGRE 2020

30. prosinca 2020.

ORGANIZATORI I SUORGANIZATORI

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE VARAŽDIN

CECIIS
Central European Conference on Information and Intelligent Systems

ALGEBRA VISOKO UČILIŠTE

MACHINA GAME DEV ACADEMY

VERN'

Grad Varaždin

Grad Bjelovar

Grad Daruvar

GENERALNI POKROVITELJ

SISAČKO-MOSLAVAČKA ŽUPANIJA

SADRŽAJ

Modeliranje „znanstvene metode“ kao igraće mehanike u domaćoj igri Trip the Ark Fantastic

Aleksandar Gavrilović
Gamechuck
gamechuckdev@gmail.com

Sažetak

Rad postavlja tezu da je znanost čest motiv u videoigrama, ali da način na koji se znanstvenike igra u videoigrama ne koristi stvarne metode znanosti već sadrže ludonarativnu disonancu. Nakon toga, rad analizira mehanike u nadolazećoj hrvatskoj igri Trip the Ark Fantastic koja želi simulirati dijelove znanstvene metode te kroz njih potaknuti igrače na razmišljanje o znanosti kako ona funkcionira. Također se okvirno opisuju neke filozofske, znanstvene i političke tema koje igra problematizira.

Ključne riječi: Trip the Ark Fantastic, Gesamtkunstwerk, videoigra, znanstvena metoda, simulacija

Uvod

Anketno istraživanje iz 2014. godine Sveučilišta u Teksasu *Portrayals of Technoscience in Video Games: A Potential Avenue for Informal Science Learning* pokazalo je da znanost igra veliku ulogu u narativima videoigara – po sudionicima ankete (njih 302), znanost igra veliku ulogu u narativu gotovo tri četvrtine igara, a igra *pokazuje* kako se znanost „zapravo događa“ („actually happening“) u 60% igara [1].

Unatoč tome, česta je pojava *ludonarativne disonance* to jest raskoraka između mehanike igre i narativa igre. U igri *Half Life* (1998), igračev lik je doktor teorijske fizike, a igra se prolazi tako da se on uspješno bori protiv cijelih četa treniranih vojnih operativaca.

Želeći se pozabaviti izazovom realnog prikazivanja rada znanosti kroz mehanike igre i izbjegavanja ludonarativne disonance, tim u studiju Gamechuck je odlučio za svoju avanturu Trip the Ark Fantastic pokušati rekreirati znanstvenu metodu.

Ovaj rad ukratko objašnjava kako će se kroz mehaniku igre rekreirati neki osnovni principi znanstvene metode i kako bi takva mehanika mogla biti korisna za demistificiranje znanosti.

Trip the Ark Fantastic

Priča igre Trip the Ark Fantastic je smještena u svijet basne – životinje su svjesne, pričaju i žive u monarhiji zvanoj Životinjsko carstvo, koje tehnološki i znanstveno podsjeća na svijet 19. stoljeća. Igračev lik je jež Charles, eminentni znanstvenik Carstva koji je dobio zadatak pronaći znanstvene dokaze o postojanju mitske Arke Čudesne („Ark Fantastic“). Legenda o arki je u središtu mitologije Životinjskog Carstva i čini osnovu njihovog društvenog sustava i religije te bi dokazivanje njenog postojanja ili nepostojanja imalo velike posljedice na njihovo društvo u cjelosti – hoće li Životinjsko Carstvo ostati monarhija ili se transformirati u demokraciju, ili pak anarhiju – ovo predstavlja osnovno političko pitanje za igrača.

Još više detalja o igri može se pročitati u zborniku radova stručne konferencije Računalne igre 2019 [2].

Charles kroz igru rješava ne samo pitanje postojanja Arke već i mnoga druga pitanja kroz razne misterije i zadatke na koje nailazi u tijeku igre. Osnovni princip na koji Charles rješava zadatke u igri (quests) je trenutno jedinstven u svijetu videoigara – koristeći apstrahiranu formu znanstvene metode.

Viđenja znanstvene metode

Ilustrirat će se različita viđenja znanstvene metode kroz primjer dvije škole misli vezane za nju.

Jedno viđenje znanstvene metode je vezano uz djelo Karla Poppera *The Logic of Scientific* Discovery, gdje se znanstvena teorija razlikuje od neznanstvene zato jer ona koristi empirijske metode – poglavito kroz eksperimente kojima se teorija pokušava osporiti (falsifikabilnost) [3].

Drugo viđenje je ono Thomasa Kuhna koje je definirao u djelu The Structure of Scientific Revolutions, koje kritizira paradigmatičnost znanosti te postavlja tezu da kada znanstvenici naiđu na podatke koji ne pašu u postojeće paradigme, *„normalna znanost“* umjesto preispitivanja paradigme ili nadograđuje paradigmu ili drugačije interpretira podatke, i tako sve dok ne dođe do paradigmatske promjene [4].
Gamechuck, studio iza igre Trip the Ark Fantastic, predstavlja sukob ove dvije teorije kao sukob između dviju vizije – znanost kao racionalistička metoda bez značajnih utjecaja vanjskih sila, ili znanost kao većinski rezultat povijesno-materijalnih sila koje utječu na znanstvenu zajednicu te samim time i na znanost. Kroz mehaniku igre, upravo se pokušava igračima predstaviti i preispitati ove dvije vizije. S tim na umu, prvo je važno napraviti apstrahirani

sustav koji dovoljno dobro aproksimira znanstvenu metodu. Za to su u Gamechucku izabrani neki ključni elementi znanstvene metode:

- Empirizam – dobivanje podataka kroz obzervaciju stvarnog svijeta
- Dedukcija – Interpretacija odabranog niza podataka i stvaranje zaključaka - hipoteze
- Indukcija – Prijašnje recenzirani radovi mogu služiti kao empirijska osnova za buduće radove
- Recenziranje – Objavljivanje hipoteza kako bi ga ostatak zajednice mogao testirati ili kritizirati

Empirizam u Trip the Ark Fantastic

Empirizam se svodi na skupljanje podataka u samom svijetu igre. Igra je predstavljena u 2D prostoru u kojem se lik kreće iz profila kroz razne lokacije – gradove Životinjskog carstva. Interakcija s različitim objektima pokreće istraživački razgovor u kojem igrač može obzervacijom prikupljati podatke.

Slika br. 1: Isječak iz videoigre Trip the Ark Fantastic

Na slici 1 vidi se dio igre u kojem Charles kroz interaktivni dijalog istražuje moguće uzročnike bolesti koja je zadesila zečeve u gradu Burrows, u sklopu znanstvenog rada On the Causes of the Rabbit Epidemic.
Osim izravne opservacije, igrač kroz igru može skupljati razne alate kojima će moći raditi detaljnije opservacije i skupiti više informacija.

Uređaji koje igračev lik može koristiti odražavaju svijet 19. stoljeća u kojem je igra smještena, pa tako Charles može koristiti mikroskop ili kameru, ali je za potrebe igre i zanimljivosti zadataka ovo prošireno i na nestvarne ili fantastične elemente - poput korištenja uzgojenih lišajeva koji mijenjaju boju ovisno o starosti ugljika (takozvani „carbon dating mould“, nepostojeći u stvarnom svijetu).

Neki od ovih uređaja su lako dostupni svima (poput recimo mikroskopa), a neki se tek počinju koristiti i njihova prihvaćenost u znanstvenom svijetu nije potpuna (poput recimo gore spomenutih lišajeva), a neke zbog cijene nije jednostavno dobiti svakom znanstveniku (poput recimo fotoaparata u boji).

Dedukcija u Trip the Ark Fantastic

Do podataka koji su dostupni igraču za stvaranje vlastitih hipoteza se dolazi kroz posebno sučelje igre u kojem se filtriraju svi dostupni podaci te stvaraju novi znanstveni radovi.

Slika br. 2: Sučelje u video igri Trip the Ark Fantastic - kategorije

U desnom dijelu sučelja na slici 2 vidi se kako su podaci sortirani u tri kategorije: opservacije koje je igrač sam pronašao, podaci na koje se igrač poziva iz knjiga i tuđih radova te podaci na koje se igrač poziva iz razgovora s drugim likovima u igri (u ulozi stručnjaka, očevidca i slično). Odabirom određenog izvora (određena knjiga, nečiji iskaz ili opservacija s neke lokacije), dobiva se popis podataka dobivenih iz tog izvora, kao na slici 3.

Slika br. 3: Sučelje u video igri Trip the Ark Fantastic – popis podataka

Konkretno, na slici 3 se u desnom dijelu sučelja vidi popis podataka iz određenog izvora (u ovom slučaju iz knjige koju je Charles pročitao u knjižnici). Igrač bira koje od tih podataka želi dodati u svoj izvještaj.

Klikom na podatak, on se pojavljuje kao ikona na lijevom dijelu sučelja. Ako drugi podaci koji već jesu u izvještaju podržavaju taj podatak, oni se vežu ravnom crtom, a ako su oprečni jedan s drugim onda se vežu valovitom crtom. Ako jedan podatak interpretira drugi, oni su povezani isprekidanom crtom.

Donji dio lijevog sučelja na slici 3 predstavlja sve moguće zaključke koji se s trenutačnim podacima mogu zaključiti. Dodatak podatka iz slike 2 do slike 3 (ikona mača) je povećao broj mogućih zaključaka za +3, što je također prikazano na sučelju. Može se dodati više različitih zaključaka u izvještaj, čak i oprečnih, a različita kombinacija zaključaka može otključati dodatne zaključke.

Završni zaključak je onaj koji predstavlja tezu znanstvenog rada, a svi ostali zaključci mogu biti hipoteze koje se dokazuju ili opovrgavaju. Kada je postavljena teza članka, on se može objaviti pristiskom na gumb SUBMIT („Predaj“) na sučelju, kao što se vidi na slici 4.

Slika br. 4: Sučelje u video igri Trip the Ark Fantastic – objava teze članka

Čitav primjer dostupan je i u video obliku [5].

Indukcija u Trip the Ark Fantastic

Teze Charlesovih radova i njihove recenzije su kasnije dostupne kao podaci s kojima igrač može nastaviti pisati druge radove i razvijati svoju tezu ili je pobijati. Na ovaj način može se doći do zanimljivijih i kompleksnijih stajališta od onih koja su dostupna samo korištenjem podataka „prvog reda“.

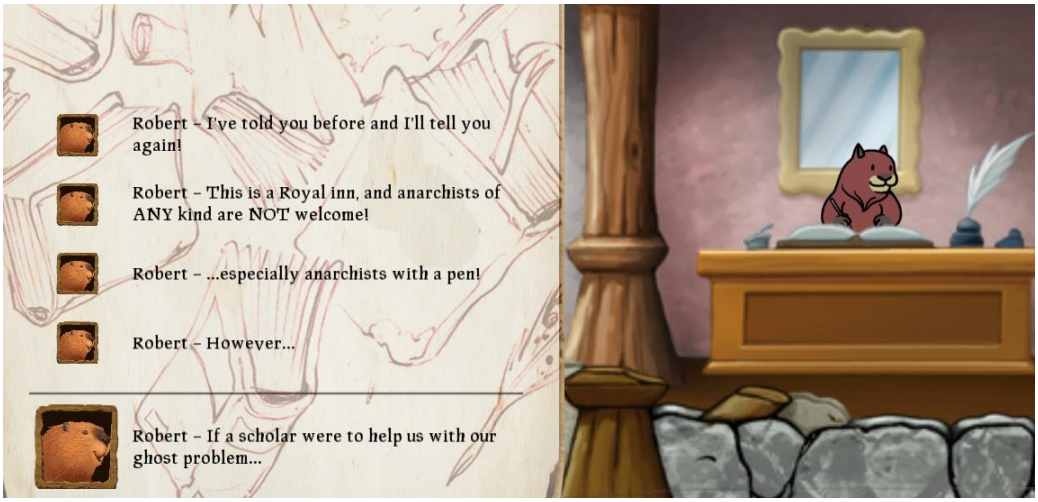
Recenziranje u Trip the Ark Fantastic

Ovisno o reputaciji igrača, sugovornika ili autora knjige, svi podaci koje igrač koristi uz sebe nose i određenu znanstvenu reputaciju. Njihovo korištenje u članku dodaje ili oduzima određenu numeričku vrijednost na vjerodostojnost samog znanstvenog rada.

Također, zaključci (hipoteze i završna teza) vežu uz sebe numeričke vrijednosti koje predstavljaju koliko su oni valjano izvedeni iz dostupnih podataka, koliko su kontroverzni kao zaključak te koliko su politički nabijeni (ako se radi recimo o znanstvenom članku koji preispituje povijest Carstva kao u slikama 2,3,4, ili je na drugi način vezano uz neku društvenu temu – ekonomski članak o razlozima siromaštva u Carstvu i slično). Moguće je u igri napisati članak koji će podići igraču reputaciju u određenim krugovima bliskim kralju, ali mu spustiti reputaciju među antimonarhističkim znanstvenicima u Carstvu.

Također, igračeva prijašnja reputacija može igrati određenu ulogu kao multiplikator koji može ublažiti učinke ovih vrijednosti na reputaciju čitavog članka.

Ovi numerički podaci nisu vidljivi igraču, ali igrač vidi rezultat kroz recenzije svog rada („peer review“) koje može čitati u novinama gdje su njegovi radovi uvijek objavljeni, ali i kroz razgovore s drugim likovima u igri, kao što je vidljivo iz negativno nabijenog dijaloga s krčmarom na slici 5.



Slika br. 5: Sučelje u video igri Trip the Ark Fantastic – dijalog

Ovdje u igru dolazi i političko pitanje koje je središnje u samoj igri – hoće li igrač probati „izigrati“ sustav tako da u Carstvu potvrdi tezu koja je igraču politički bliža (recimo vezano za vladavinu kraljeva Lavova u Carstvu ili za istinitost mita o Arki i slično) ili će se pokušati držati znanstvenih načela i

istine po svaku cijenu. Kroz ovakav sustav, igraču je dopušteno istraživati odgovore na ova pitanja samostalno – bez navođenja.

Reference

[1] „Portrayals of Technoscience in Video Games: A Potential Avenue for Informal Science Learning“, Anthony Dudo, Vincent Cicchirillo, Lucy Atkinson, Samantha Marx; izdavač SAGE Publications, 2014.
[2] „Trip the Ark Fantastic kao Gesamtkunstwerk“, Aleksandar Gavrilović, Matija Vigato, Zbornik radova Računalne igre 2019, izdavač Fakultet organizacije i informatike, Sveučilište u Zagrebu, 2019.
[3] „Logic of Scientific Discovery“, Karl R. Popper, izdavač Julius Springer, 1934
[4] „Structure of Scientific Revolutions“, Thomas S. Kuhn, izdavač University of Chicago, 1962
[5] The Blackbark Discrepancy, dostupno na https://www.youtube.com/watch?v=1jvFR80UoYk, 10.12.2020.

Izrada video igre žanra Metroidvania u programskom alatu Unity

Luka Rožić, Danijel Radošević i Mladen Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
lrozic@foi.hr, darados@foi.hr, mlkoneck@foi.hr

Sažetak

U ovom radu opisat će se glavne karakteristike video igara žanra Metroidvania. Nakon definiranja žanra opisat će se elementi igre „Knivader“, igre koja je izrađena u programskom alatu Unity, a koja ima sve osnovne karakteristike video igara žanra Metroidvania. Opisat će se na koji način su implementirane osnovne mehanike igre. Na kraju će se donijeti zaključak o provedbi izrade video igre ovog tipa u programskom alatu Unity.

Ključne riječi: računalna igra, dvodimenzionalna igra, Unity, C#, Metroidvania, programiranje, algoritmi

Uvod

Razvoj industrije računalnih igara je u neprestanom porastu i predviđanja su da se rast neće zaustaviti [1]. Na tržištu postoji velika ponuda različitih kućnih konzola, igre se igraju na osobnim računalima, a u najvećem porastu je igranje putem mobilnih uređaja. Na tržištu su naravno jako dobro prihvaćene igre velikih proizvođača poput Ubisofta ili Nintenda, no također vidimo da i manji studiji također pronalaze svoje mjesto na tržištu. Upravo zbog velike raznolikosti želja i potreba igrača, vidimo kako na tržištu pronalazimo igre najrazličitijih žanrova. Zbog velike konkurencije, uvijek je potrebno osmišljati nove koncepte igranja, nove mehanike igranja, biti kreativan i inovativan. Jedan on žanrova koji je i danas popularan, koji ima svoju publiku i moderne uspješne naslove je i žanr Metroidvania [2], podžanr akcijsko avanturističkih igara [3].

Elementi Metroidvanie

Sami naziv žanra potječe od serijala igara *Metroid* [4] i *Castlevania* [5]. Metroidvania igre su poznate po pažljivo osmišljenim dinamičnom svijetu koji igrač istražuje, po ugođaju, atmosferi, s naglaskom na narativnom dijelu igre, a jednako tako i na izazovnoj igrivosti (engl. *gameplay*). Na početku igre igrač ima ograničen set mogućnosti i vještina te su mnogi dijelovi svijeta zaključani ili nedostupni. Kako igrač istražuje svijet, tako skuplja iskustvo, opremu, nova

oružja, moći i mogućnosti. Kako stječe nove mogućnosti borbe i kretanja, tako sve više može otkrivati prije nedostupne dijelove svijeta i boriti se s jačim neprijateljima koje prije nije eventualno mogao savladati. Još jedna učestala karakteristika je postojanje tajnih soba i skrivenih putova koji doprinose zanimljivosti svijetu u kojem se igrač nalazi. Dizajn razina je vrlo pomno promišljen, čest je koncept stvaranja određenih regija unutar svijeta i svaka regija ima svoje jedinstvene karakteristike, kako stilske tako i po vrsti neprijatelja i načinu borbe.

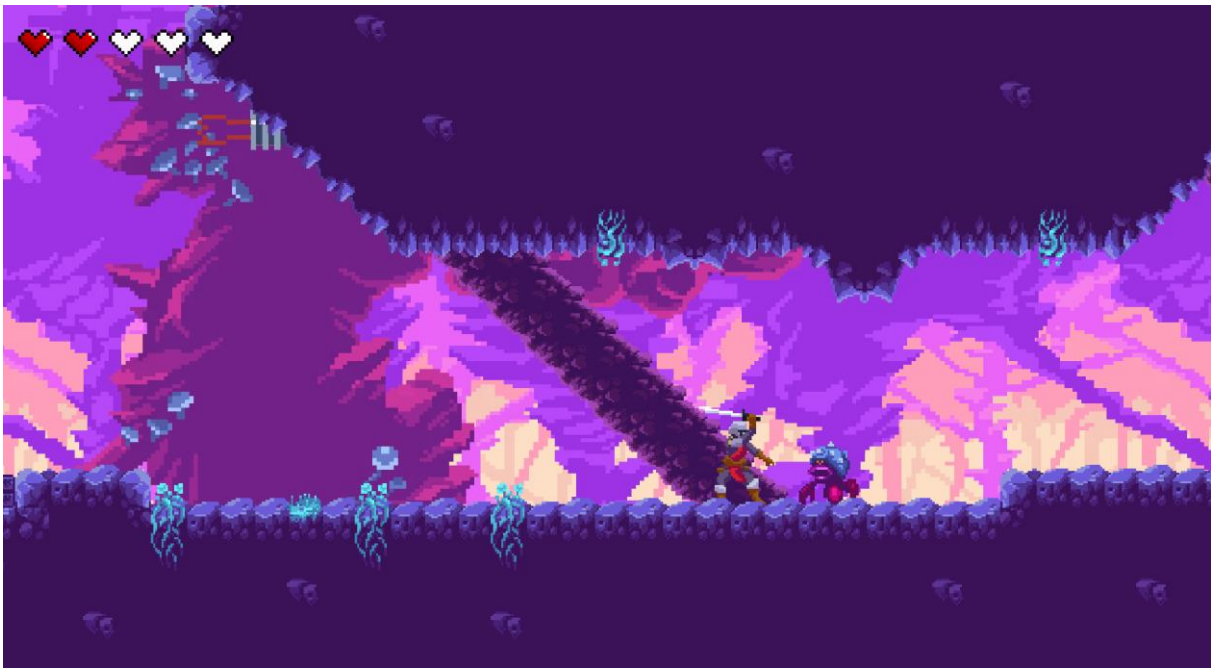
Česta pojava je da igrač mora prolaziti više puta istim dijelovima svijeta. Kako taj segment igre ne bi igraču bio monoton, koristi se rješenje da s vremenom neprijatelji postaju jači ili da se mijenja na određeni način svijet u kojem se igrač nalazi. Lijep primjer implementacije takvog koncepta vidljiv je primjerice u igri *Hollow Knight* [6]. Kako bi se navigacija olakšala igraču, igrač često koristi neki oblik karte svijeta u kojem se nalazi. Na raznim dijelovima igre nalaze se mnogi neprijatelji koje igrač treba savladati kako si napredovao u igri. Također se u igrama pojavljuju i platformerski elementi. Osim toga često igrač mora odgonetnuti razne zagonetke koje se nalaze u svijetu. Jednako tako se i prilikom borbe s neprijateljima javljaju elementi rješavanja problema, najčešće u borbama s glavnim neprijateljima. Igrač mora odgonetnuti vrste napada neprijatelja kako bi ih mogao izbjeći i otkriti u kojem trenutku je neprijatelj ranjiv.

Izrada vlastite igre Knivader

Imajući na umu osnovne karakteristike Metroidvania žanra, kreirana je igra *Knivader*. Samo ime igre dolazi od dvije engleske riječi, *knight* i *invader* što u prijevodu znači vitez i osvajač. Glavnog protagonista igre je zli čarobnjak zajedno s ostalim čudovištima iz svijeta fantazije teleportirao na vanzemaljski planet te je cilj viteza pronaći način kako da se vrati u svoj svijet.

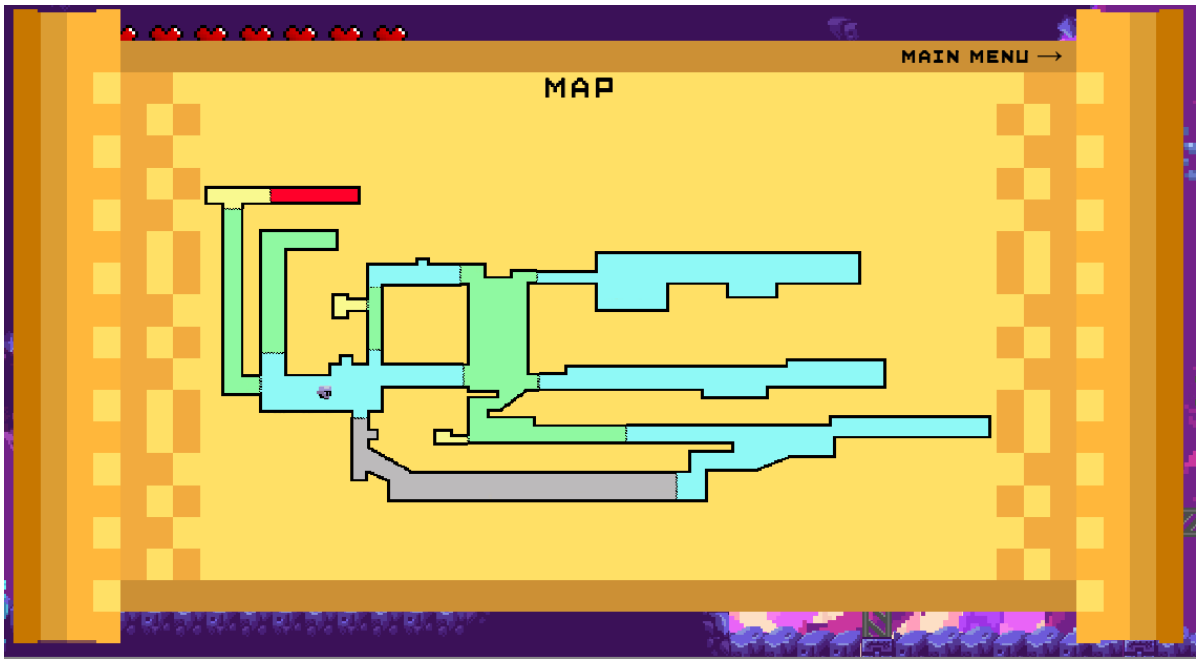
Protagonist igre - vitez

Igrač se nalazi u ulozi viteza koji na početku igre ima 5 životnih bodova koji su vidljivi na sučelju u obliku srca u gornjem lijevom dijelu zaslona. Osim standardne mogućnosti kretanja igrač može također i skočiti. Što se tiče interakcije s neprijateljima, igraču je odmah na raspolaganju mač koji može koristiti dok stoji ili dok se kreće po zemlji. Dok je u zraku tj. dok skače ne može koristiti oružje. Igrač igranjem skuplja iskustvo. Nakon što igrač skupi određenu količinu iskustva (engl. *level up*) igrač postaje sve jači i udarac mača postaje snažniji te nanosi više štete neprijateljima. Na slici br. 1 vidljiva je scena s početka igre gdje igrač napada neprijatelja koji se nalazi ispred njega.



Slika br. 1: Vitez mačem napada neprijatelja

Igraču je dostupna i karta putem koje može lakše navigirati prostorom u kojem se nalazi te kako bi znao koje prostorije je do sada posjetio, a koje su još neistražene. Na mapi su horizontalne prostorije označene plavom bojom, a vertikalne prostorije su naznačene zelenom bojom. Sivom bojom označen je prostor koji je nešto izazovniji i koji je u potpunosti opcionalan. Žute sobe su sobe gdje se nalazi tzv. *checkpoint*, tj. kontrolna točka. To je točka na koju se igrač vraća u slučaju gubitka života tj. u slučaju smrti. Za kraj, crvena soba je soba u kojoj se nalazi glavni neprijatelj igre (engl. *final boss*).

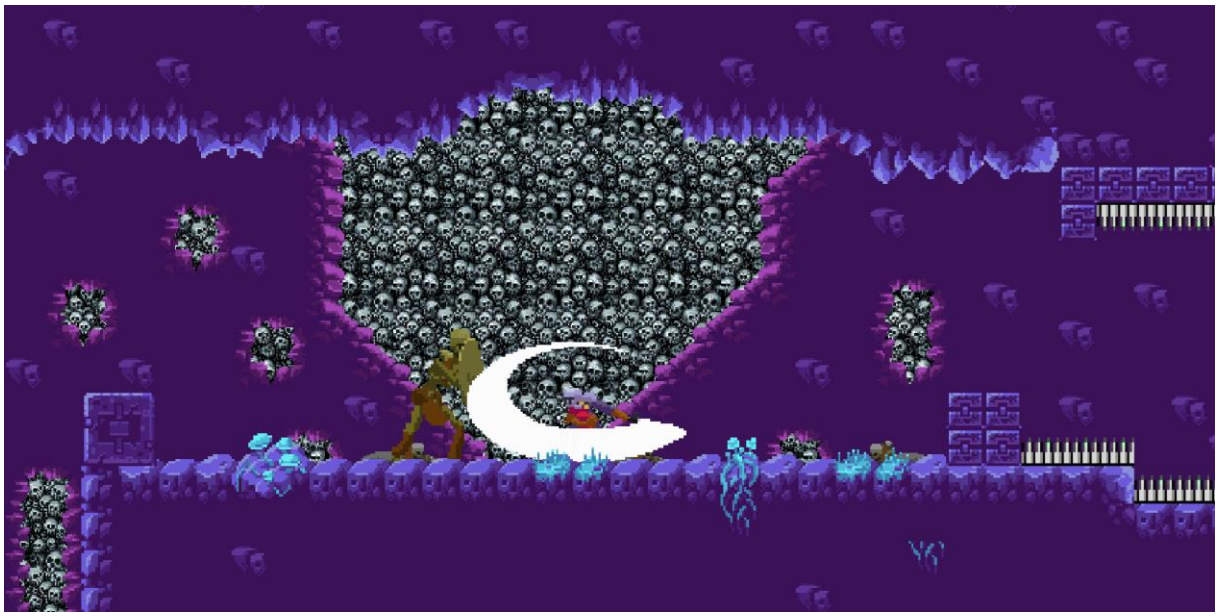


Slika br. 2: Karta svijeta

Posebni predmeti koje igrač može sakupiti u igri su leteće čizme koje igraču omogućavaju dvostruki skok u zraku (engl. *double jump*), vodena medalja koja igraču omogućava kretanje pod vodom istom brzinom kao i izvan vode, kandža za zid koja igraču omogućava penjanje po zidu u vertikalnom smjeru kretanja i polagano klizanje po zidu prema dolje te magična energija pomoću koje igrač može zamahom mača ispucati projektil prema neprijateljima. Posebna vrsta dodatka je i zlatno srce koje povećava kapacitet životnih bodova igrača za jedno srce.

Neprijatelji u igri

U igri postoji 5 vrsta neprijatelja. Prvi neprijatelj je rak koji ima mogućnost kretanja lijevo/ desno po platformi na kojoj se nalazi te u neposrednoj blizini s igračem ga može napasti. U slučaju da rak dođe do kraja platforme ili ako naiđe na nekog drugog neprijatelja, promijenit će svoj smjer kretanja na suprotnu stranu. Druga vrsta neprijatelja je hobotnica koja ima mogućnost ispucavanja otrova na daljinu. U slučaju da se igrač približi ovom neprijatelju na određenu odaljenost, on će ispucati otrov u smjeru igrača. Osim što mogu pucati na daljinu, ovi neprijatelji se kreću brže od rakova i imaju malo više životnih bodova. Treći neprijatelj je neprijatelj koji se nalazi na stropovima prostorija i u slučaju da se igrač nađe ispod njih oni „padaju“ prema dolje i pokušavaju na taj način ozlijediti igrača. Ako igrač dođe u neposredan dodir s ovim neprijateljem, gubi životne bodove.



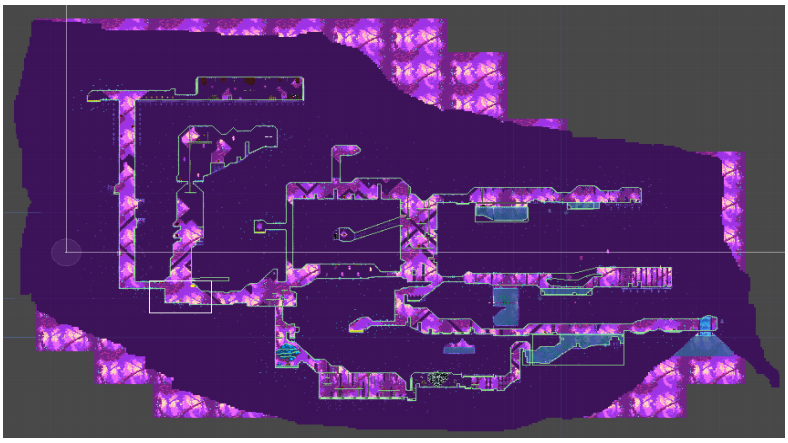
Slika br. 3: Mini glavni neprijatelj kostur se štiti

Četvrta vrsta neprijatelja je kostur, mini glavni neprijatelj koji predstavlja nešto veći izazov za savladati od prva tri neprijatelja. Kostur ima mogućnost praćenja pozicije igrača i neprestano se kreće prema igraču. Za detekciju

igrača koristi se funkcija *Vector2.Distance()* [7] koja vraća vrijednost udaljenosti između pozicija dvaju objekata, u ovom slučaju pozicija viteza i kostura. Ako se igrač dovoljno približi kosturu, kostur će napasti igrača sa svojim mačem. Ako igrač zamahne sa svojim mačem, neće moću udariti kostura jer će kostur iskoristiti obranu štitom koji drži u drugoj ruci. Jednako tako, ako igrač preskoči kostura i pokuša ga napasti, neće mu nauditi jer kostur drži štit na suprotnoj strani od strane prema kojoj gleda. Igrač mora prepoznati da je jedini trenutak kada je kostur ranjiv jest trenutak nakon što kostur zamahne svojim mačem. Tada se igrač mora približiti kosturu i napasti ga. Obrana kostura štitom vidljiva je na slici br. 3.

Slika br. 4: Glavni neprijatelj napada mačem

Zadnji neprijatelj u igri je glavni neprijatelj u igri, a to je zli čarobnjak. Čarobnjak koristi kombinaciju 4 različita napada (mačem i čarolijama). Prva vrsta napada je neposredni napad mačem koji će čarobnjak izvesti ako se igrač nalazi u neposrednoj blizini. Drugi napad je napad vatrom – čarobnjak će se zatrčati prema igraču, zakotrljati i ostaviti trag vatre. Kako bi izbjegao ovu vrstu napada igrač mora preskočiti čarobnjaka. Treća vrsta napada je grmljavina koju igrač također mora preskočiti. Za vrijeme tog napada čarobnjak ima obranu s one strane na kojoj mu se nalazi štit. Posljednji napad je napad ledom. Kako bi izbjegao ovu vrstu napada igrač se mora pozicionirati između padajućih komada leda. Slika br. 4. prikazuje napad mačem zlog čarobnjaka.



Slika br. 5: Vanzemaljski svijet igre „Knivader“ vidljiv unutar Unityja

Dizajniranje svijeta

Za izradu dvodimenzionalnog svijeta u kojem se igrač nalazi korišten je *Tile Palette* [8] prozor u programskom alatu Unity u kojem se definiraju sličice koje se kombiniranjem koriste za stvaranje svijeta. Sličice koje su korištene za izradu igre su preuzete s Unity Asset Storea [9]. Na slici br. 5 moguće je vidjeti konačan dizajn razine igre.

Zaključak

Metroidvania je poseban žanr računalnih igara koji lijepo balansira akcijske i avanturističke aspekte igara u jednu cjelinu. Na neki način bi se moglo reći da su ovo 2D igre otvorenog svijeta. Igrač istražuje svijet u kojem se nalazi kroz eksperimentiranje, ne postoji zacrtani put već postoji više putova kako je moguće doći do krajnjeg cilja. Programski alat Unity je jako dobar razvojni alat koji nudi dobru podršku za razvoj video igara ovog tipa. Za implementaciju mnogih mehanika i aspekata igre, Unity ima dobru podršku u obliku raznih prozora, pomagala i funkcija. Također postoji pregršt dobrih poduka u slučaju da postoji problem s implementacijom određenog željenog segmenta igre ili mehanike.

Reference

[1] Market Analysis Report: Video Game Market Size, Share & Trends Analysis Report By Device (Console, Mobile, Computer), By Type (Online, Offline), By Region, And Segment Forecasts, 2020 – 2027. (21. rujna 2020.). Dostupno na: https://www.grandviewresearch.com/industry-analysis/video-game-market

[2] Wikipedia: Metroidvania. (22. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Metroidvania

[3] Wikipedia: Action-adventure game. (22. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Action-adventure_game

[4] Wikipedia: Metroid. (22. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Metroid

[5] Wikipedia: Castlevania. (22. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Castlevania

[6] Hollow Knight. (22. rujna 2020.). Dostupno na: https://www.hollowknight.com/

[7] Unity Documentation: Vector2. Distance. (23. rujna 2020.). Dostupno na: https://docs.unity3d.com/ScriptReference/Vector2.Distance.html

[8] Unity Documentation: Tile Palette. (23. rujna 2020.). Dostupno na: https://docs.unity3d.com/Manual/Tilemap-Palette.html

[9] Unity Asset Store. (23. rujna 2020.). Dostupno na: https://assetstore.unity.com/

Izrada horor videoigre preživljavanja u programskom alatu Unity

Dario Alilović i Mario Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
dalilovic@foi.hr, mario.konecki@foi.hr

Sažetak

U ovom radu prikazat će se rad u programskim alatu Unity kroz izradu vlastite horor videoigre preživljavanja. Bit će prikazan detaljan opis i kratka povijest programskog alata Unity, a bit će opisan i žanr horor videoigara preživljavanja. Prikazat će se nekoliko tehnika, metoda i algoritama koji se koriste prilikom izrade videoigara te će sve navedeno biti ilustrirano kroz primjenu prilikom izrade vlastite videoigre.

Ključne riječi: Unity, videoigra, horor, programiranje

Uvod

Horor videoigra preživljavanja (engl. Survival Horror videogame) je žanr videoigara koji koristi elemente horor igara, ali s naglaskom na oskudnost resursa [26]. Prva videoigra koja je predstavljena 1996. kao horor igra preživljavanja je Resident Evil. Ova videoigra je danas među najpoznatijim horor videoigrama. Ono što definira horor videoigru preživljavanja su zagonetke, korištenje manjeg inventara i borba s neprijateljima. Borba s neprijateljima nije u svakoj videoigri jednaka. U nekima videoigrama borba jednostavno ne postoji, nego se više fokusira na izbjegavanje neprijatelja [27]. Horor igre najčešće kreću polako te s vremenom stvaraju sve veću napetost dok se ne dođe do točke kada priča dolazi do vrhunca. U tom trenutku često dolazi do nekog velikog obračuna ili nekog velikog preokreta u priči [28]. Za potrebe izradu horror videoigre preživljavanja prikazane u ovom radu korišten je programski alat Unity.

Unity je programski alat za izradu videoigara koji podržava razvoj videoigara na većini platformi, uključujući desktop računala, računalne konzole, virtualnu stvarnost, proširenu stvarnost, mobilne uređaje (Android, iOS), TV platforme, web platforme i ostale platforme [1]. Unity je danas među vodećim razvojnim okolinama za izradu videoigara, ako ne i najveća [2].

U nastavku rada bit će opisana izrada First Person kontrolera, neprijatelja te interakcija među njima.

First Person Controller

U alatu Unity moguće je na dva načina stvoriti „igrača“. Igrač može biti stvoren kao „rigid body“ koji ima već ugrađenu gravitaciju, trenje i interakciju s objektima, ili kao „character controller“ koji ima bolju interakciju sa stepenicama i kosinama te dobru interakciju sa zidovima, u smislu da se igrač ne zaglavljuje u njima [33]. U prikazanoj implementaciji koristi se „character controller“.

Character controller sam po sebi nema ugrađenu gravitaciju niti funkcije za kretanje, tako da se one moraju dodati kroz skripte.

Skripta za CameraMove sadrži funkcije za kontrolu rotacije kamere na kontroleru pomoću miša. Prilikom učitavanja ova skripta sakrije kursor i zaključa ga u sredini ekrana, a nakon toga prilikom svakog ažuriranja provjerava poziciju miša i rotira kontroler, ako je rotacija horizontalna, ili kameru, ako je rotacija vertikalna.

```
public class CameraMove : MonoBehaviour
{
    public float mouseSensitivity = 2f;
    public Transform playerBody;
    public GameObject menu;
    public GameObject menuOptions;
    public bool menuOn = false;
    float xRotation = 0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    void LateUpdate()
    {
      float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity;
      float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity;

      xRotation -= mouseY;
      xRotation = Mathf.Clamp(xRotation, -90f, 90f);
      transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);

      playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

Kontroleru je potrebno dodati gravitaciju i mogućnost kretanja. To je ostvareno kroz skriptu PlayerMove. Ova skripta sadrži funkcije za kontrolu kretanja kontrolera, njegovu gravitaciju te funkcije za provjeru podloge na kojoj se kontroler kreće i funkcije za paljenje zvuka definiranog za tu podlogu.

```
public class PlayerMove : MonoBehaviour
{
    [SerializeField] private AudioClip[] dirtFootsteps;
    [SerializeField] private AudioClip dirtJump;
    [SerializeField] private AudioClip rockJump;
    [SerializeField] private AudioClip grassJump;
    [SerializeField] private AudioClip sandJump;
    [SerializeField] private AudioClip gravelJump;
    [SerializeField] private AudioClip waterJump;
    [SerializeField] private AudioClip woodJump;
    [SerializeField] private AudioClip JumpUp;
    [SerializeField] private AudioClip DivingSound;
    [SerializeField] private AudioClip SwimmingSound;

    private int lastIndex = 0;
    private int newIndex = 0;
    public int surfaceIndex = 0;
    public Terrain terrain;
    private TerrainData terrainData;
    private Vector3 terrainPos;
    private TerrainLayer[] terrainLayers;
    private AudioSource audioSource;
    private int currentDelay;
    private Vector3 lastPosition;
    private Terrain[] _terrains;
    public CharacterController controller;
    public float walkingSpeed = 12f;
    public float runningSpeed = 24f;
    public float lastGravity;
    public float gravity = -9.81f;
    public float waterGravity = -300f;
    public float jumpHeight = 3f;
    public int stepDelay = 20;
    public int runDelay = 15;
    public Transform groundCheck;
    public float groundDistance = 0.4f;
    public LayerMask groundMask;

    Vector3 velocity;
    Vector2 input;
    bool inWater;
    bool inBuilding;
    bool inRuinedBuilding;
    public bool isSwimming;
    bool isDiving;
    bool isJumping;
    bool isWalking;
    [SerializeField] bool isGrounded;
    [SerializeField] bool wasGrounded;
    bool isMoving;

    void Start(){
        _terrains = Terrain.activeTerrains;

        lastGravity = gravity;
        inWater = false;
        lastPosition = new Vector3(0, 0, 0);
        currentDelay = 0;
        audioSource = GetComponent<AudioSource>();
```

```
    isJumping = false;
    wasGrounded = true;

    QualitySettings.vSyncCount = 0;

    PlayerHealth.currentHealth = 20;
}

void Update(){
    terrain = GetClosestCurrentTerrain(groundCheck.transform.position);
    terrainData = terrain.terrainData;
    terrainPos = terrain.transform.position;
    surfaceIndex = GetMainTexture(transform.position);
    terrainLayers = terrain.terrainData.terrainLayers;

    wasGrounded = isGrounded;
    isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

    if (!wasGrounded && isGrounded && !isDiving){
        PlayLandingSound();
        velocity.y = -2f;
        isJumping = false;
        wasGrounded = true;
    }

    if (wasGrounded && !isGrounded && !isDiving){
        audioSource.clip = JumpUp;
        audioSource.Play();
    }

    if (!isGrounded && !isJumping && wasGrounded){
        velocity.y = -2f;
    }
}

private void PlayLandingSound(){
    string textureName = GetTextureName();

    if (inWater){
        audioSource.clip = waterJump;
        audioSource.Play();
        return;
    }

    if (inBuilding){
        audioSource.clip = woodJump;
        audioSource.Play();
        return;
    }

    switch (textureName){
        case "moss":
            audioSource.clip = grassJump;
            audioSource.Play();
            break;
        case "dirt":
            audioSource.clip = dirtJump;
            audioSource.Play();
            break;
        case "rock":
```

```
            audioSource.clip = rockJump;
            audioSource.Play();
            break;
        case "scree":
            audioSource.clip = rockJump;
            audioSource.Play();
            break;
        case "sand":
            audioSource.clip = sandJump;
            audioSource.Play();
            break;
        case "gravel":
            audioSource.clip = gravelJump;
            audioSource.Play();
            break;
        default:
            break;
    }
}

private void FixedUpdate(){
    float speed;
    Calculate(out speed);

    Vector3 move = transform.right * input.x + transform.forward * input.y;

    RaycastHit hitInfo;
    Physics.SphereCast(transform.position, controller.radius, Vector3.down, out hitInfo, controller.height / 2f,
    Physics.AllLayers, QueryTriggerInteraction.Ignore);
    move = Vector3.ProjectOnPlane(move, hitInfo.normal).normalized;

    controller.Move(move * speed * Time.deltaTime);
    isSwimming = CheckIfSwimming();

    if (isSwimming){
        gravity = 0;

        if (Input.GetKey(KeyCode.LeftShift) && !isDiving){
            velocity.y = -50;
            controller.Move(velocity * Time.deltaTime);
            isDiving = true;
        }

        if (isDiving){
            if (Input.GetKey(KeyCode.Space)){
                velocity.y = -waterGravity * Time.deltaTime;
            }
            else{
                velocity.y = 0;
            }

            if (Input.GetKey(KeyCode.LeftShift)){
                velocity.y = waterGravity * Time.deltaTime;
            }

            if (controller.transform.position.y > 38.35){
                velocity.y = 0;
                isDiving = false;
            }
        }
```

```
        else{
            if (controller.transform.position.y > 38.3){
                velocity.y = 0;
            }
            else{
                velocity.y = 5;
            }
        }
    }
    else{
        gravity = lastGravity;
        velocity.y += gravity * Time.deltaTime;
    }

    controller.Move(velocity * Time.deltaTime);

    isMoving = CheckIfMoving();

    if ((isGrounded || isSwimming) && isWalking && isMoving){
        PlayFootstepSound(stepDelay);
    }

    if ((isGrounded || isSwimming) && !isWalking && isMoving){
        PlayFootstepSound(stepDelay);
    }

    if (isDiving){
        PlayFootstepSound(stepDelay);

    }

    if (Input.GetButtonDown("Jump") && isGrounded){
        velocity.y = Mathf.Sqrt(jumpHeight * -2 * gravity);
        isJumping = true;
    }
}

private bool CheckIfSwimming(){
    if (controller.transform.position.y < 38.4){
        return true;
    }
    return false;
}

private void OnTriggerEnter(Collider other){
    if (other.gameObject.name == "WaterProDaytime"){
        inWater = true;

        runningSpeed -= 5;
        walkingSpeed -= 5;

        stepDelay += 3;
        runDelay += 3;
    }
}

private void OnTriggerStay(Collider other)
{
    if (other.gameObject.name == "OldShed" || other.gameObject.name == "aband_house"){
        inBuilding = true;
```

```
    }

    if (other.gameObject.name == "RuinedBuilding"){
        inRuinedBuilding = true;
    }
}

private void OnTriggerExit(Collider other){
    if (other.gameObject.name == "WaterProDaytime"){
        inWater = false;

        runningSpeed += 5;
        walkingSpeed += 5;

        stepDelay -= 3;
        runDelay -= 3;
    }

    if (other.gameObject.name == "OldShed" || other.gameObject.name == "aband_house"){
        inBuilding = false;
    }

    if (other.gameObject.name == "RuinedBuilding"){
        inRuinedBuilding = false;
    }
}

private bool CheckIfMoving(){
    if (transform.position != lastPosition){
        lastPosition = transform.position;
        return true;
    }
    return false;
}

private void Calculate(out float speed){
    float horizontal = Input.GetAxis("Horizontal");
    float vertical = Input.GetAxis("Vertical");

    isWalking = !Input.GetKey(KeyCode.LeftControl);

    if (isWalking){
        speed = walkingSpeed;
    }
    else{
        speed = runningSpeed;
    }

    input = new Vector2(horizontal, vertical);

    if (input.sqrMagnitude > 1){
        input.Normalize();
    }
}

public void PlayFootstepSound(int delay){
    currentDelay++;
    if (currentDelay < delay){
        return;
    }
```

```
currentDelay = 0;
string textureName = GetTextureName();

System.Random rand = new System.Random();

CheckIfMoving();

if (inWater && isMoving && !isGrounded && !isDiving){
    audioSource.clip = SwimmingSound;
    if (!audioSource.isPlaying){
        audioSource.Play();
    }
    return;
}

if (isDiving){
    audioSource.clip = DivingSound;
    if (!audioSource.isPlaying){
        audioSource.Play();
    }
    return;
}

if (inWater && isGrounded){
    while (newIndex == lastIndex){
        newIndex = rand.Next(0, waterFootsteps.Length);
    }
    lastIndex = newIndex;
    audioSource.clip = waterFootsteps[newIndex];
    audioSource.Play();
    return;
}

if (inBuilding){
    while (newIndex == lastIndex){
        newIndex = rand.Next(0, woodFootsteps.Length);
    }
    lastIndex = newIndex;
    audioSource.clip = woodFootsteps[newIndex];
    audioSource.Play();
    return;
}

if (inRuinedBuilding){
    while (newIndex == lastIndex){
        newIndex = rand.Next(0, sandFootsteps.Length);
    }
    lastIndex = newIndex;
    audioSource.clip = sandFootsteps[newIndex];
    audioSource.Play();
    return;
}

if (!inBuilding){
    switch (textureName){
        case "moss":
            while (newIndex == lastIndex){
                newIndex = rand.Next(0, grassFootsteps.Length);
            }
            lastIndex = newIndex;
```

```
            audioSource.clip = grassFootsteps[newIndex];
            audioSource.Play();
            break;
        case "dirt":
            while (newIndex == lastIndex){
                newIndex = rand.Next(0, dirtFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = dirtFootsteps[newIndex];
            audioSource.Play();
            break;
        case "rock":
            while (newIndex == lastIndex){
                newIndex = rand.Next(0, rockFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = rockFootsteps[newIndex];
            audioSource.Play();
            break;
        case "scree":
            while (newIndex == lastIndex){
                newIndex = rand.Next(0, rockFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = rockFootsteps[newIndex];
            audioSource.Play();
            break;
        case "sand":
            while (newIndex == lastIndex){
                newIndex = rand.Next(0, sandFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = sandFootsteps[newIndex];
            audioSource.Play();
            break;
        case "gravel":
            while (newIndex == lastIndex){
                newIndex = rand.Next(0, gravelFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = gravelFootsteps[newIndex];
            audioSource.Play();
            break;
        default:
            break;
    }
  }
}

private string GetTextureName(){
  return terrainLayers[surfaceIndex].name;
}

Terrain GetClosestCurrentTerrain(Vector3 playerPos){
  var center = new Vector3(_terrains[0].transform.position.x + _terrains[0].terrainData.size.x / 2, playerPos.y,
                           _terrains[0].transform.position.z + _terrains[0].terrainData.size.z / 2);
  float lowDist = (center - playerPos).sqrMagnitude;
  var terrainIndex = 0;

  for (int i = 0; i < _terrains.Length; i++){
```

```
            center = new Vector3(_terrains[i].transform.position.x + _terrains[i].terrainData.size.x / 2, playerPos.y,
                                 _terrains[i].transform.position.z + _terrains[i].terrainData.size.z / 2);

            var dist = (center - playerPos).sqrMagnitude;
            if (dist < lowDist){
                lowDist = dist;
                terrainIndex = i;
            }
        }
        return _terrains[terrainIndex];
    }

    private float[] GetTextureMix(Vector3 WorldPos){
        int mapX = (int)(((WorldPos.x - terrainPos.x) / terrainData.size.x) * terrainData.alphamapWidth);
        int mapZ = (int)(((WorldPos.z - terrainPos.z) / terrainData.size.z) * terrainData.alphamapHeight);

        if (mapX > 511 || mapX < 0 || mapZ > 511 || mapZ < 0){
            mapX = 0;
            mapZ = 0;
        }
        float[,,] splatmapData = terrainData.GetAlphamaps(mapX, mapZ, 1, 1);

        float[] cellMix = new float[splatmapData.GetUpperBound(2) + 1];

        for (int n = 0; n < cellMix.Length; n++){
            cellMix[n] = splatmapData[0, 0, n];
        }
        return cellMix;
    }
    private int GetMainTexture(Vector3 WorldPos){
        float[] mix = GetTextureMix(WorldPos);
        float maxMix = 0;
        int maxIndex = 0;

        for (int n = 0; n < mix.Length; n++){
            if (mix[n] > maxMix){
                maxIndex = n;
                maxMix = mix[n];
            }
        }
        return maxIndex;
    }
}
```

U svakoj igri je potrebno ostvariti nekakvu interakciju s okruženjem u kojem se igrač nalazi. Primjer skripte u kojoj se navedeno ostvaruje je PickUpPistol, koja sadrži logiku kupljenja stvari s poda, a koja se može modificirati za bilo kakvu drugu interakciju (npr. otvaranje vrata). Stvari se kupeo tako da kada se kontroler nalazi na zadanoj udaljenosti, koja je ostvarena kroz skriptu koja dolazi u nastavku, i gleda u stvar koju želi pokupiti, postavi tekst i crosshair u njegov GUI kako bi se naglasila mogućnost interakcije. Nakon toga, kada je interakcija omogućena, pritiskom na određenu tipku (u ovom slučaju „E“), omogućuje se određena stvar koja se nalazi u svijetu i omogućava se kopiju te stvari, koja je prikvačena na sam kontroler (u slučaju da je želimo prikazati).

```
public class PickUpPistol : MonoBehaviour
```

```
{
    public float Distance;
    public GameObject DisplayActionKey;
    public GameObject DisplayText;
    public GameObject ActionCrosshair;
    public GameObject PistolPickup;
    public GameObject Pistol;
    private Text text;

    void Update(){
        Distance = DetectDistance.TargetDistance;
    }

    private void OnMouseOver(){
        if (Distance < 5){
            DisplayActionKey.SetActive(true);
            DisplayText.SetActive(true);
            ActionCrosshair.SetActive(true);
            text = DisplayText.GetComponent<Text>();
            text.text = "Pick up pistol";
        }
        if (Distance >= 5){
            DisplayActionKey.SetActive(false);
            DisplayText.SetActive(false);
            ActionCrosshair.SetActive(false);
        }
        if (Input.GetKey(KeyCode.E) && Distance < 5){
            this.GetComponent<BoxCollider>().enabled = false;
            PistolPickup.SetActive(false);
            Pistol.SetActive(true);
            DisplayActionKey.SetActive(false);
            DisplayText.SetActive(false);
            ActionCrosshair.SetActive(false);
            text = DisplayText.GetComponent<Text>();
            text.text = "";
            Pistol.GetComponent<FireGun>().ammoInGun = 10;
        }
    }

    private void OnMouseExit(){
        DisplayActionKey.SetActive(false);
        DisplayText.SetActive(false);
        ActionCrosshair.SetActive(false);
    }
}
```

Kao što je već spomenuto skripta DetectDistance mjeri udaljenost kontrolera od prve prepreke uz pomoć „zraka“ (engl. raycast).

```
public class DetectDistance : MonoBehaviour
{
    public static float TargetDistance;

    void Update(){
        RaycastHit hit;
        if(Physics.Raycast (transform.position, transform.TransformDirection (Vector3.forward), out hit)){
            TargetDistance = hit.distance;
        }
    }
```

```
}
```

Neprijatelji

Neprijatelji su važan dio horor videoigara. Principi oblikovanja neprijatelja mogu varirati, ali neke osnovne stvari su u svakom slučaju iste. Potreban je model neprijatelja, animacije i umjetna inteligencija. U alatu Unity već gotove modele s animacijama moguće je preuzeti u trgovini Asset Store, tako da još samo preostaje dodavanje umjetne inteligencije. Skripta EnemyAI sadrži vrlo jednostavnu umjetnu inteligenciju koja neprijatelju daje gravitaciju i mogućnost praćenja igrača, te kretanje prema igraču. Uz to, u skripti su definirani zvukovi i animacije koje se uključuju ili isključuju u određenoj situaciji.

```
public class EnemyAI : MonoBehaviour
{
    public GameObject Player;
    public GameObject PlayerController;
    public GameObject Enemy;
    public float enemySpeed = 0.01f;
    public int enemyDamage = 5;
    public bool attackTrigger = false;
    public bool isAttacking = false;
    public bool isDead = false;
    public AudioClip ZombieSound;
    public AudioClip ZombieAttack;
    public CharacterController controller;
    Vector3 velocity;
    private float speed;
    private AudioSource audioSource;

    private void Start(){
        speed = enemySpeed;
        audioSource = GetComponent<AudioSource>();
    }

    void Update(){
        if (isDead){
            audioSource.Stop();
        }
        if (!isDead && Vector3.Distance(Enemy.transform.position, Player.transform.position) <= 130){
            Enemy.GetComponent<Animation>().Stop("Z_Idle");
            controller.Move(velocity * Time.deltaTime);
            if (!Enemy.GetComponent<Animation>().IsPlaying("Z_Attack") && isAttacking){
                isAttacking = false;
            }
            transform.LookAt(Player.transform);
            if (!attackTrigger && !isAttacking){
                if (PlayerController.GetComponent<PlayerMove>().isSwimming){
                    enemySpeed = 0f;
                    Enemy.GetComponent<Animation>().Play("Z_Idle");
                }
                else{
                    enemySpeed = speed;
                    Enemy.GetComponent<Animation>().Play("Z_Run_InPlace");
```

```
                if (!Enemy.GetComponent<AudioSource>().isPlaying){
                    PlayZombieSound();
                }
                else{
                    if (Enemy.GetComponent<AudioSource>().clip != ZombieSound &&
Player.GetComponent<PlayerHealth>().myHealth > 0){
                        PlayZombieSound();
                    }
                }
            }
            velocity.y = -4f;
            controller.Move(velocity * Time.deltaTime);
            transform.position = Vector3.MoveTowards(transform.position, Player.transform.position,
enemySpeed * Time.deltaTime);
        }
        if (attackTrigger && !isAttacking){
            enemySpeed = 0f;
            isAttacking = true;
            Enemy.GetComponent<Animation>().Play("Z_Attack");
            if (PlayerController.GetComponent<PlayerHealth>().myHealth > 0){
                PlayAttackSound();
            }
            PlayerHealth.currentHealth -= enemyDamage;
        }
    }
    else{
        if (!isDead){
            enemySpeed = 0f;
            Enemy.GetComponent<Animation>().Play("Z_Idle");
            velocity.y = -4f;
            controller.Move(velocity * Time.deltaTime);
        }
    }
}

private void PlayZombieSound(){
    audioSource.clip = ZombieSound;
    audioSource.Play();
}

private void PlayAttackSound(){
    audioSource.clip = ZombieAttack;
    audioSource.Play();
}

private void OnTriggerEnter(Collider other){
    if (other.gameObject.name == "AttackTrigger" && !isDead){
        attackTrigger = true;
    }
}

private void OnTriggerExit(Collider other){
    if (other.gameObject.name == "AttackTrigger" && !isDead){
        attackTrigger = false;
    }
}
}
```

Interakcija

Uz sve navedeno, potrebna je još i interakcija igrača i neprijatelja. Za to je zaduženo nekoliko skripti. Skripta PlayerHealth sadrži statičku varijablu currentHealth koja sadrži trenutno stanje života igrača te funkcije za pokretanje zvukova i funkcije za pokretanje game over scene.

```
public class PlayerHealth : MonoBehaviour
{
    public static int currentHealth = 20;
    public int myHealth;
    public GameObject fadeOut;
    public AudioClip PlayerHurtSound1;
    public AudioClip PlayerHurtSound2;
    public AudioClip PlayerDeadSound;
    public int randomNum;
    public AudioSource audioSource;
    private bool died;
    private System.Random rand;

    private void Start(){
        died = false;
        rand = new System.Random();
    }

    void Update(){
        if(currentHealth < myHealth){
            PlayHurtSound();
        }
        myHealth = currentHealth;
        if(currentHealth <= 0){
            StartCoroutine(Wait());
        }
        IEnumerator Wait(){
            PlayDyingSound();
            fadeOut.GetComponent<Animation>().Play("FadeOutAnimation");
            yield return new WaitForSeconds(1);
            SceneManager.LoadScene(2);
        }
    }

    private void PlayDyingSound(){
        if (!died){
            audioSource.clip = PlayerDeadSound;
            audioSource.Play();
            died = true;
        }
    }

    private void PlayHurtSound(){
        randomNum = rand.Next(0, 2);
        if (randomNum == 1){
            audioSource.clip = PlayerHurtSound1;
        }
        else{
            audioSource.clip = PlayerHurtSound2;
        }
        audioSource.Play();
```

```
    }
}
```

Uz to potrebna je još i interakcija igrača s neprijateljem. Skripta EnemyDeath sadrži funkciju za ozljeđivanje neprijatelja koju je moguće pozvati iz druge skripte te logiku za gašenje inteligencije i animacija nakon umiranja.

```
public class EnemyDeath : MonoBehaviour
{
    public int EnemyHealth = 20;
    public GameObject Enemy;
    public GameObject Leg1;
    public GameObject Leg2;
    public int StatusCheck;
    public int randomNumber;
    private System.Random rand;

    void DamageEnemy(int damage){
        rand = new System.Random();
        EnemyHealth -= damage;
    }

    void Update(){
        if(EnemyHealth <= 0 && StatusCheck == 0){
            Enemy.GetComponent<EnemyAI>().isDead = true;
            StatusCheck = 2;
            Enemy.GetComponent<Animation>().Stop();
            randomNumber = rand.Next(0, 2);
            Enemy.GetComponent<CharacterController>().enabled = false;

            Leg1.GetComponent<MeshCollider>().enabled = false;
            Leg2.GetComponent<MeshCollider>().enabled = false;

            Enemy.GetComponent<AudioSource>().Stop();
            if (randomNumber == 1){
                Enemy.GetComponent<Animation>().Play("Z_FallingBack");
            }
            else{
                Enemy.GetComponent<Animation>().Play("Z_FallingForward");
            }
        }
    }
}
```

Zaključak

Nakon implementacije gore navedenih elemenata, još je potrebno samo izraditi svijet u kojem će igrač i neprijatelji biti pozicionirani i tako se dobiva vrlo jednostavna horor videoigra, na koju je onda moguće dodati još neke detalje kao što je glavni izbornik, više nivoa igre, više interaktivnih mjesta i slično. Važno je postaviti ambijentalne zvukove i smanjiti svjetlinu u svijetu, kako bi se postigla veća napetost u igri.

Unity je svestran alat koji omogućuje brzu i laganu izradu svega gore navedenog te uvelike olakšava čitav proces razvoja. Iako se na prvu proces izrade videoigre čini kompliciranim, veoma je lagano naučiti osnovne funkcije potrebne za razvoj videoigara, pogotovo uz korištenje Unity priručnika koji vrlo detaljno opisuje sve njegove funkcije. Zbog navedenog Unity predstavlja dobar izbor za početnike koji se žele iskušati u razvoju videoigara.

Reference

[1] LoveToKnow, Corp. (bez dat.) survival-horror. Dostupno: https://www.yourdictionary.com/survival-horror [pristupano 27.06.2020.].

[2] Jason Ashman (25.10.2015.) What Makes a Survival Horror Game? Dostupno: https://techraptor.net/gaming/opinion/what-makes-survival-horror-game [pristupano 27.06.2020.].

[3] World of Level Design LLC by Alex Galuzin (29.06.2009.) Horror/Survival Level Design: Part 2 – Anticipation. Dostupno: https://www.worldofleveldesign.com/categories/level_design_tutorials/horror-fear-level-design/part2-survival-horror-level-design-anticipation-pacing.php [pristupano 27.06.2020.].

[4] Unity Technologies (bez dat.) Multiplatform. Dostupno: https://unity.com/features/multiplatform [pristupano 22.06.2020.].

[5] Jon Brodkin, „How Unity3D Became a Game-Development Beast“, 2013. [Na internetu]. Dostupno: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ [pristupano 22.06.2020.].

[6] Brackeys (27.10.2019.) FIRST PERSON MOVEMENT in Unity - FPS Controller, Youtube. Dostupno: https://www.youtube.com/watch?v=_QajrabyTJc [pristupano 27.06.2020.].

Izrada 2D platformera u programskom alatu Unreal Engine 4

Domagoj Mahnet i Mario Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
dmahnet@foi.hr, mario.konecki@foi.hr

Sažetak

U ovom radu opisan je žanr videoigara platformer, razvoj videoigara ovakvog tipa od samih početaka pa sve do danas te su dani primjeri videoigara koji najbolje oslikavaju navedeni žanr. U sljedećem dijelu rada opisan je proces izrade videoigre ovakvog tipa pomoću Unreal Enginea 4. Dio je objašnjen kroz C++ kod, a dio kroz vizualno skriptiranje Unreal Enginea. Igrač upravlja glavnim likom koji ima izgled jednog srednjovjekovnog viteza. Cilj igre je doći do kraja nivoa preskačući otrovne gljive, pri čemu se na putu nalaze agresivni zombiji.

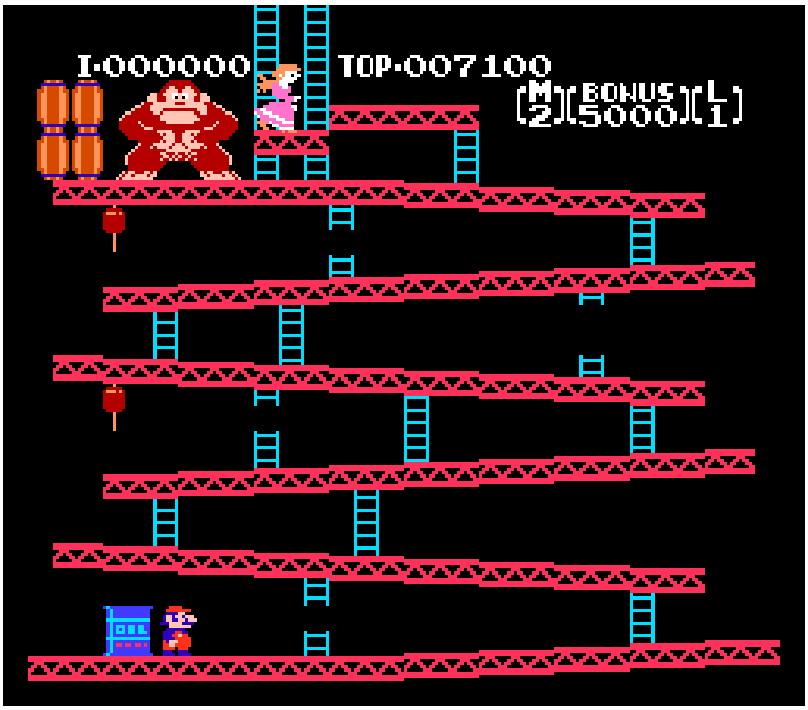
Ključne riječi: Unreal Engine, platformer, C++, 2D videoigra, zombi

Uvod

Razvoj videoigara postao je jedna velika i unosna grana računalne industrije. Danas je gotovo nemoguće pronaći osobu koja nikada nije odigrala barem jednu videoigru. Tome je uvelike doprinijela njihova široka dostupnost, bilo to na računalu ili na mobilnom uređaju. Videoigre su također postale veoma važan dio današnje kulture.

2D platformer

Platformer je žanr videoigara i podžanr akcijskih videoigara. U tipičnim igrama ovog žanra igrač kontrolira svog lika u videoigri te mu daje naredbe za kretanje. Igrač mora skakati između platformi te izbjegavati prepreke na putu. Platformere dijelimo na one sa statičkim prikazom i one s pomičnim prikazom. Platformeri sa statičkim prikazom svaki nivo prikažu u cjelini, dok se kod platformera s pomičnim prikazom okolina mijenja ovisno o kretanju lika kojim igrač upravlja. Prve igre ovog žanra koristile su statičke prikaze, zbog tehničkih ograničenja u to vrijeme [1].



Slika br. 1: Donkey Kong [2]

Donkey Kong se smatra prvom videoigrom ovog žanra. Objavljena je 1981. godine, a cilj igre je stići do najviše platforme izbjegavajući kolutajuće bačve i preskačući preko njih. Rast popularnosti Donkey Konga doveo je do nastanka brojnih platformera fokusiranih na mehanike skoka. Donkey Kong se također smatra i prvom „hop and bop" igrom, najčešćim podžanrom žanra platformer. Hop and bop svoje ime dobiva po mehanici skoka na neprijatelja, kojim se neprijatelj poražava. Najpoznatija videoigra ovog podžanra je Super Mario [3].

U 1990-im igre žanra platformer prelaze s pseudo 3D-a na pravi 3D. Rendering ovih igara postao je veoma zahtjevan, stoga su za njihovo igranje bila potrebna računala jaka za to doba. 1994. Exact je izdao videoigru naziva Geograph Seal. To je bila potpuno 3D poligonalna shooter igra u prvom licu. Smatra se prvom iskonski 3D action-shooter igrom sa „free roaming" okolinom, ali je ostala nezamijećena na zapadu jer je izdana samo u Japanu. Sljedeće godine Exact izdaje Jumping Flash! za Sonyevu novu konzolu Playstation. Prema Guiness World Recordsu Jumping Flash! smatra se prvom pravom igrom žanra platformer u 3D-u.

Početkom trećeg tisućljeća igre žanra platformer uvelike gube na popularnosti. 1998. njihov udio na tržištu iznosio je 15%, dok je 2002. taj udio spao na samo 2% [4]. 2002. Nintendo izdao svoju drugu videoigru iz grupacije 3D Super Mario, imena Super Mario Sunshine. Ova videoigra je bila kritizirana zbog kratke duljine, nemaštovito dizajniranih razina te sporosti samog gameplaya [5]. Ovaj žanr više nikada nije dosegao popularnost koju je nekad imao [4].

Glavni lik

Kako bi glavni lik dobio svoj izgled potrebno je uvesti slike u projekt, što se postiže jednostavnom drag-and-drop akcijom. Iz ovih slika je tada potrebno iz tih slika izvući Sprite. Za potrebe animiranja lika poželjno je izvući barem deset Spriteova kako bi animacija izgledala što uvjerljivije. Nakon toga potrebno je kreirati Flipbook. Desni klik u Content Browseru > Animation > Paper Flipbook.

Pod detaljima potrebno je dodati onoliko Key Frameova koliko jedna animacija sadrži slika. Zatim je svakom Key Frameu potrebno dodati pripadajući Sprite. Ovisno o željenoj brzini izvođenja animacije, moguće je modificirati Frames Per Sec kako bi se ona ubrzala, odnosno usporila. Klikom na Save, Flipbook se sprema te ga je moguće implementirati. U Editoru glavnog lika pod sekcijom Details nalazi se kategorija Animations. Ovdje se animaciji dodjeljuje njezin Flipbook source.

Važno je napomenuti kako se ovdje po defaultu nalaze samo Running Animation te Idle Animation. Kako bi bilo omogućeno dodjeljivanje drugih animacija, poput Jump ili Attack animacija, potrebno je u header klasi glavnog lika dodati nove animacije. To se omogućuje unosom sljedećeg koda:

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Animations)
	class UPaperFlipbook* JumpAnimation;
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Animations)
	class UPaperFlipbook* AttackAnimation;
```

Sada se u editoru pod sekcijom Details u kategoriji Animations ne nalaze samo Idle i Running Animations, nego i Jump Animation te Attack Animation.
Zadnja važna stvar kod animacija je određivanje u kojem trenutku će se koja animacija prikazivati. U klasi glavnog lika potrebno je kreirati funkciju koja će neprestano provjeravati stanja u kojem se glavni lik nalazi te određivati prikladnu animaciju.

```
void APlatformerCharacter::UpdateAnimation()
{
	const FVector PlayerVelocity = GetVelocity();
	const float PlayerSpeedSqr = PlayerVelocity.SizeSquared();
	UPaperFlipbook* DesiredAnimation;
	DesiredAnimation = IdleAnimation;
	if (PlayerSpeedSqr > 0.0f) {
		DesiredAnimation = RunningAnimation;
	}
	if (GetCharacterMovement()->IsFalling())
```

```
	{
		DesiredAnimation = JumpAnimation;
	}
	if (isAttacking == true) {
		DesiredAnimation = AttackAnimation;
	}
	GetSprite()->SetFlipbook(DesiredAnimation);
}
```

Ova funkcija prvo dohvaća brzinu kojom se glavni lik kreće. Ako se glavni lik kreće brzinom većom od 0.0f, odnosno ne stoji na mjestu, željena animacija postavlja se na Running Animation. Sljedeće što ta funkcija provjerava je nalazi li se lik u skoku. Ovdje funkcija željenu animaciju postavlja na Jump Animation neovisno o tome kreće li se glavni lik horizontalno ili ne, zbog toga što lik ne može samo skakati na mjestu, nego može i skakati i kretati se horizontalno istovremeno. Zadnje što funkcija provjerava je napada li glavni lik trenutačno protivnika. Ako napada, željena animacija postavlja se na Attack Animation, neovisno o rezultatima prethodnih provjera, zbog toga što glavni lik može napasti dok se kreće horizontalno, dok skače, ili dok izvršava obje akcije istovremeno. Ako sve ove provjere vrate negativan rezultat, željena animacija postavlja se na Idle Animation.

Zombi

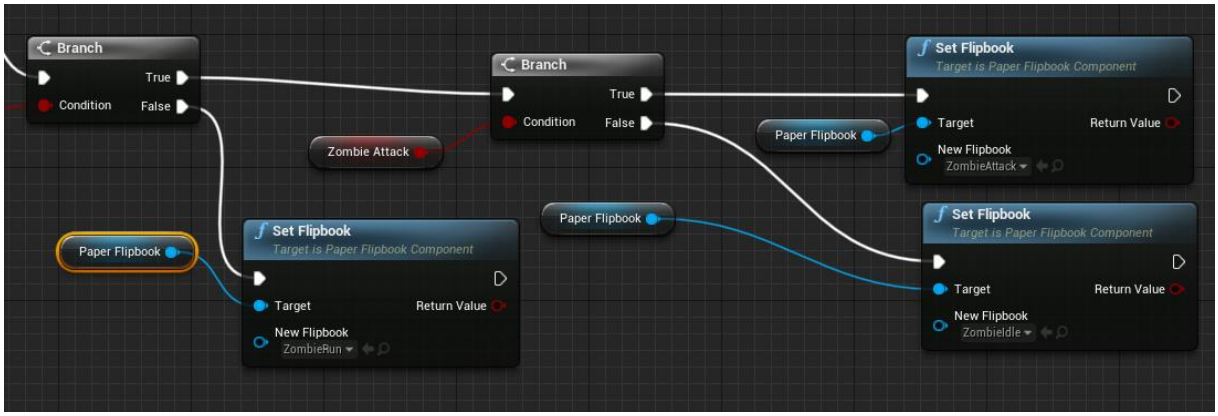
Za razvoj funkcionalnosti glavnog lika korišten je odgovarajući C++ kod. S obzirom na to da Unreal Engine 4 ima mogućnost i vizualnog skriptiranja preko Blueprint sustava, upravo taj sustav je korišten za razvoj protivnika kako bi se pokazala i ta strana Unreal Enginea. Zombiji se kreiraju jednako kao i glavni lik, a na isti način se rade i animacije.

Slika br. 2: Zombi

Kao i kod glavnog lika, implementiran je algoritam koji odabire prihvatljivu animaciju za trenutačno stanje zombija. Na početku se dohvaća Health te brzina kretanja glavnog lika.

Ako je zombijev Health manji ili jednak nuli, animacija se postavlja na ZombieDeath, koja se izvodi jednu sekundu te se nakon toga zombi uništi.

Ako je zombijev Health veći od nule, provjerava se kretnje zombija. Ako se on kreće animacija se postavlja na ZombieRun. U suprotnom, provjerava se napada li zombi glavnog lika. Ako da, animacija se postavlja na ZombieAttack, a ako ne, postavlja se na ZombieIdle.

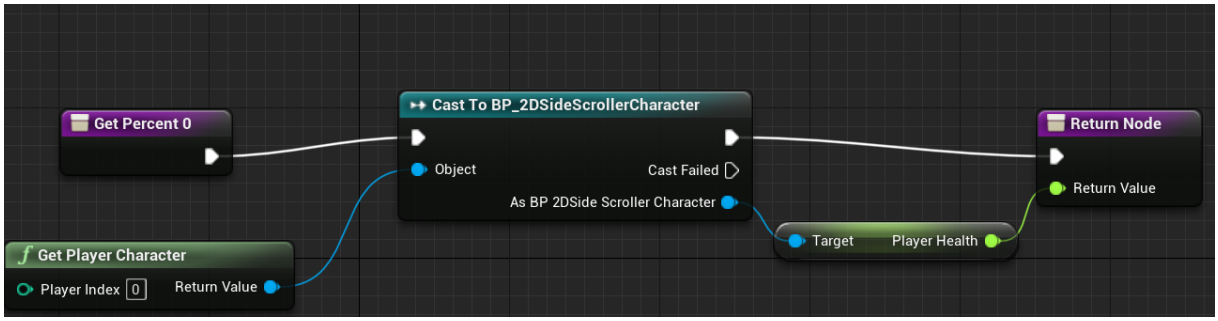


Slika br. 3: Logika animiranja zombija

Grafičko sučelje

Grafičko sučelje koje prikazuje Health glavnog lika kreira se slično kao i ostali elementi. Desni klik u Content Browseru > User Interface > Widget Blueprint. Kreirano je prozirno grafičko sučelje koje prikazuje trenutačni Health glavnog lika.

Kako bi se ažurirao te ispravno prikazivao Health potrebno je dodati sljedeći kod u Blueprint editor:



Slika br. 4: Povezivanje healtha sa sučeljem

Slika br. 5: Prvi nivo

Zaključak

U ovom radu prikazan je proces izradu jedne 2D videoigre žanra platformer. Opisani su ključni dijelovi razvoja jedne takve videoigre koristeći Unreal Engine 4. Unreal Engine 4 i Visual Studio pokazali su se moćnim, ali relativno jednostavnim alatima za razvoj jedne ovakve videoigre.

Reference

[1] M. Klappenbach, What is a platform game? Dostupno: https://www.lifewire.com/what-is-a-platform-game-812371 [Pristupano 30.6.2020.]

[2] https://en.wikipedia.org/wiki/Donkey_Kong#/media/File:Donkey_Kong_NES_Screenshot.png

[3] racketBOY, Platforming Games. Dostupno: http://www.racketboy.com/retro/platforming-games-101-all-you-need-to-know [Pristupano 30. 6. 2020.]

[4] T. Fahs, the-nextlevel. Dostupno: https://web.archive.org/web/20160129224512/http://www.the-nextlevel.com/review/retro/geograph-seal-x68000/ [Pristupano 30.8.2020]

[5] D. Boutros, GamaSutra. Dostupno: https://www.gamasutra.com/view/feature/1851/a_detailed_crossexamination_of_.php [Pristupano 30. 8. 2020.]

[6] S. Maiorana, The Jaded Gamer: Super Mario Sunshine. Dostupno: https://web.archive.org/web/20051125062547/http://www.thejadedgamer.net/review_gcn_supermariosunshine.shtml [Pristupano 30.8.2020.]

Usporedba žanrova računalnih igara

Domagoj Curić i Mario Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
dcuric@foi.hr, mario.konecki@foi.hr

Sažetak

U radu će biti dan pregled žanrova računalnih igara, iako je doista teško navesti jednu podjelu s obzirom na to da se industrija računalnih igara razvija takvom brzinom da novi žarnovi nastaju kao rezultat inovativnih projekata i tehnologija, ili se pak više žanrova udružuje u novi žanr. Bit će dan pregled povijesti i specifičnosti pojedinih žanrova računalnih igara.

Ključne riječi: računalne igre, žanrovi, usporedba žanrova

Uvod

Računalne igre su se kroz svoju kratku povijest razvijale velikom brzinom, što zbog slobodne maštovitosti u stvaranju različitih igara, što zbog razvoja same tehnologije na kojoj se računalne pokreću. Činjenica je da se svake godine pokušavaju napraviti računalne igre kakvih na tržištu još nema ili se postojeće računalne igre pokušava dovesti na razinu kakva do još nije viđena. Probijaju se granice maštovitosti i vizualizacije same računalne igre, a to dovodi do sve veće zainteresiranosti za računalnim igrama. Novi igrači ulaze u svijet industrije računalnih igara svakodnevno i time ova industrija postaje sve popularnija.

Igrači računalnih igara imaju svoje individualne interese za igranjem igara koje njima odgovaraju, pa prema svom kriteriju biraju koji žanr igre žele igrati. A kako ima i više nego dovoljno igara u svim različitim žanrovima, mogu se lagano i odlučiti koju će igru igrati.

Prema tome, koju računalnu igru uopće odabrati i kako prepoznati i odlučiti se za pojedinu igru u moru žanrova svih vrsta igara? U ovom radu bit navedeni žanrovi računalnih igara i bit će dana usporedba žanrova i razlika među njima.

Prikazat će se i povijest računalnih igara, gdje će se vidjeti kako su izgledale i nastale prve računalne igre, a bit će komentirana i industrija računalnih igara u Hrvatskoj.

Povijest računalnih igara

Prva računalna igra u povijesti bila je OXO, tzv. križić kružić. Igru je osmislio i izradio Alexander Sandy Douglas i to na automatskom kalkulatoru elektroničkog odlaganja (EDSAC) 1952. godine [1].

Još jedna značajna računalna igra je „Tennis for Two“. Tu je igru 1958. godine osmislio William Higinbotham u Brookhaven nacionalnom laboratoriju u New Yorku. Igra se pokretala na osciloskopu i analognom računalu s vakuumskom cijevi [2] [3]. Prvom značajnijom računalnom igrom smatra se Spacewar!, osmislio ju je Steve Russell sa svojim timom 1962. godine. Tom timu na čelu sa Russell-om je trebalo oko 200 sati da napišu prvu verziju Spacewar!-a. Napisana je na PDP-1, ranom interaktivnom mini računalu DEC, a on je koristio ekran s katodnom cijevi i unos putem tipkovnice. Ovu su igru mogla igrati dva igrača, tako što je svaki od igrača imao pod kontrolom svoj svemirski brod koji ispaljuje rakete. Bodovi su se skupljali na način da se rakete ispaljuju na protivnika te se moralo odupirati gravitaciji sunca [4].

Prva igra koja je komercijalno bila dostupna je Computer Space koja je razvijena 1971.godine, a stvorili su je Nolan Bushnell i Ted Dabney i ona je bila prva arkadna igra, a predstavljala je nadograđenu verziju igre Spacewar! [5].

Devedesetih godina 20. stoljeća ubrzano počinje rasti broj igara. Toru Iwatani je 1980. godine izdao igru Pac-Man. U igri je bilo potrebno pojesti sve točkice i izbjegavati duhove koji love igrača. Ova igra je jedna od najpopularnijih igara u povijesti [6].

Jošš jedna od najpopularnijih igara koja je u ovome slučaju na igraćoj konzoli NES (Nintendo Entertainment System) je Super Mario Bros u izdanju Nintenda 1983. godine.

Samo deset godina kasnije izlazi prva avanturistička igra Myst gdje igrač istražuje i rješava svakojake zagonetke.

Žanrovi računalnih igara

Trenutačno najaktualniji žanrovi računalnih igara uključuju:

- Akcijske igre,
- Avanturističke igre,
- RPG - Igra uloge,
- Simulacijske igre,
- Strateške igre,
- Sportske igre,

- Igra utrka,
- Pucačine (FPS, TPS), te
- MMO (pretežito multiplayer online igre)
- Kraljevske bitke (Battle – Royale games)

Kao najzanimljiviji i najigraniji žanr nameće se Battle – Royale games. Najpopularnija i najigranija Battle – Royale game svakako je Fortnite, koji je zaludio populaciju od mlađih do starijih igrača. Upravo je u ovoj igri igrač pod imenom Ninja na online platformi za gledanje videa u kojoj igrači igraju video igre, Twitch, oborio rekord gledanosti od oko 635.000 gledatelja [8].

Akcijske igre

Akcijske igre su jedan od najjednostavnijih i najstarijih žanrova računalnih igara. U početku su se ove igre odvijale u 2D okruženju. Razvojem računala igre su postajale sve naprednije i radnja akcijskih igara se počela odvijati i u 3D okruženju [9]. Za igranje akcijskih igara igrač treba biti dosta spretan i imati dobru usklađenost očiju i ruku. Popularnost ovog žanra dovela je do niza programerskih inicijativa u razvoju ovog žanra [10].

Avanturističke igre

Ovaj žanr računalnih igara karakterizira istraživanje i rješavanje zagonetki. Ovaj žanr može biti fokusiran na mnoge književne žanrove kao što su horori, znanstvena fantastika, misterij itd. I ovaj žanr je jedan od najstarijih žanrova, njegovo podrijetlo seže u 1970-te kada je izdana igra Colossal Cave, jedna od prvih avanturističkih igara. Ova igra se temeljila na tome da igrač unosi jednostavne tekstualne naredbe za istraživanje špilje.

Većina igrača i kritičara smatra kako ovaj žanr izumire, no međutim on se i dalje proizvodi i kupuje. Mana je jedino što se tržišni udio ovih avanturističkih igara smanjuje, ali elementi ovoga žanra prelaze u druge najpopularnije žanrove kao što je akcijsko – avanturistički žanr [10] [11].

RPG – Igra uloge

RPG ili „Role Playing Game“ su vizualno bogate igre u kojima se igraču stvara dojam realnog svijeta. Igrač RPG igre kontrolira izmišljenog lika kojeg odabere i koji rješava različite zadatke u izmišljenom svijetu. Likovi u igrama mogu imati različite osobine ili karakteristike, koje im služe za rješavanje raznih izazova. [10] [12].

Simulacijske igre

Simulacijske igre pokušavaju kopirati razne aktivnosti iz stvarnog života, poput treninga, vožnje ili poslovanja. Obično u igri ne postoje strogo definirani ciljevi, a igraču je umjesto toga dozvoljeno slobodno kontrolirati lika ili okolinu. Postoje razne simulacijske igre u kojima igrač može voziti traktor, avion, kamion, auto, graditi zgrade, prodavati robu itd. Najzastupljenije simulacijske igre su one ekonomskog tipa, u kojima se pokušava prikazati kako funkcionira realan svijet [10].

Strateške igre

skillful thinking and planning to achieve victory.[1] It emphasizes strategic, tactical, and sometimes logistical challenge

Strateške računalne igre uključuje promišljanje, planiranje i vješt odabir poteza u svrhu postizanja cilja tj. pobjede. Naglasak je dan na strateške, taktičke i logičke kompetencije igrača. Strateške igre mogu biti jednog od niza podžanrova [13].

Neki od ovih žanrova su Real-Time Strategy (RTS), Real-Time Tactics (RTT) i Multiplayer Online Battle Arena (MOBA). Real-time strategy ili strategija u stvarnom vremenu je primjer igre u kojoj se igrača potiče da sakuplja i regulira svoje resurse, istovremeno povećavajući i svoje borbene snage i izvore za uzdržavanje. Real-time tactics ili taktika u stvarnom vremenu podrazumijeva da se igra usredotočuje na taktike koje igrač koristi na virtualnom bojnom polju ili tijekom igranja rata, umjesto na održavanje resursa. MOBA ili multiplayer online borilište je možda i jedan od najpopularnijih žanrova jer se pod ovim žanrom smatra jedna od najpopularnijih računalnih igara na svijetu, a to je League of Legends. Igrači uglavnom kontroliraju jednog lika u jednoj od dvije momčadi koje surađuju kako bi pokušale preuzeti ili uništiti domaću bazu neprijateljskog tima [14].

Sportske igre

Ovaj žanr je vrlo jednostavan za opisati jer njegov opis proizlazi iz samog naziva žanra. Računalne igre koje potpadaju pod ovaj žanr su igre poput košarke, nogometa, tenisa itd.

Sportske igre pojavile su se rano u povijesti igara i danas su i dalje jedan od najpopularnijih žanrova igara na svijetu. Baš kao i stvarni sportovi, sportske igre simuliraju tradicionalne fizičke sportove u kojima protivnički tim kontrolira umjetna inteligencija (AI) ili druge osobe iz stvarnog života [10].

Igra utrka

Igra utrka smatra se popularnim žanrom koji se dosta rano pojavio. Ove igre dosta su popularne na konzolama, no računalni svijet trkaćih igara svojim igračima nudi dosta rekvizita poput volana, papučica, mjenjača. Ovaj žanr se smatra jednim od najpopularnijih žanrovi svih vremena. Razvijanjem igračke industrije i poticanjem na sve više online igranje, došlo je do sve većeg broja igara ovog žanra koje nude opciju igranja preko Interneta s drugim stvarnim igračima [10].

Pucačine – FPS

Pucačine su podžanr akcijskih igara, no ovaj žanr postaje sve popularniji u svijetu igara te se može svrstati i pod samostalan žanr. Igre u ovome žanru često testiraju prostornu svijest, reflekse i brzinu igrača. Mogu se igrati u prvom licu (FPS) ili trećem licu (TPS).

Nakon izlaska devedesetih godina vrlo popularne igre Doom ovaj žanr zapravo biva rođen, te mnogi smatraju upravo igru Doom prekretnicom za razvijanje računalnih igara. U prvome licu igrač vidi samo ruku koja drži pušku i njome upravlja kroz cijelu igru, dok se u trećem licu vidi cijeli avatar kojim igrač upravlja. Sve više igara ovoga žanra ima naglasak na online igranje. [15]

MMO – Pretežito multyplayer online igre

MMO ili Massively Multiplayer Online zapravo znači pretežito multiplayer online. Jednostavno rečeno, MMO je mrežna igra za više igrača koju veliki broj igrača može igrati istovremeno. Ne igra se s ili protiv samo nekolicine igrača, nego s tisućama, ponekad čak i milijunima igrača odjednom [16].

MMO počinje uzimati sve veći zamah u industriji računalnih igara. Razlog tome je prije svega mogućnost socijalne interakcije igrača. Naravno, ono što je tu zanimljivo i što privlači veliki broj publike je igranje i natjecanje u timovima. Podžanrovi ovoga žanra su [16]:

- MMORPG (Role-Play Game)
- MMOFPS (First-Person Shooter)
- MMORTS (Real-Time Strategy)
- MMO Sports, itd. (Fritts, n.d.)

Kraljevske bitke (Battle – Royale games)

Battle-Royal games ili kraljevske bitke su online multiplayer igre. Ovaj žanr je igračima doista poseban, zanimljivi i uzbudljiv iz jednostavnog razloga, a to je što moraju biti domišljati kako bi pobijedili. U većini slučajeva kod računalnih

igara ovoga žanra sve kreće tako da igrač ili cijeli tim tj. lik kojim se upravlja iskače iz aviona te slijeće padobranom na željeno mjesto određene mape u kojoj se nalazi (najčešće je to otok).

Nakon slijetanja igrači, da bi preživjeli i opstali na mapi, moraju tražiti oružja, opremu i mnoge druge stvari. Na mapi uglavnom sleti oko 100 igrača. Tu je i tzv. „Oluja“ koja se sužava i stvara opasnost za igrača ako uđe u nju jer će izgubiti život ako ne izađe na vrijeme iz nje. Na posljetku je pobjednik onaj koji ostane zadnji na mapi [17].

Usporedba žanrova računalnih igara

Usporedimo li žanr akcijskih igara i pucačina, može se uvidjeti da su to slični žanrovi tj. da je svaka računalna igra koja je pucačina ujedno i akcijska igra, ali da svaka igra koja je akcijska ne mora nužno biti pucačina jer postoje akcijske igre koje nisu pucačine nego imaju neku drugu priču i akcije koje se u njima događaju.

Kada bi se pokušalo objasniti i usporediti danas vrlo vjerojatno dva najpopularnija žanra, MMO i Battle Royale, može se zaključiti da su to žanrovi kojima je zajedničko online igranje, a to znači da se bez pristupa Internetu ne mogu igrati. Naravno, unutar igre postoji i poseban dio koji omogućuje da se igra i bez pristupa Internetu, no ove igre za to ne služe i nikome nisu zbog toga zanimljive. Razlike ova dva žanra leže u tome da u žanru MMO nije cilj ostati zadnji na mapi i tako pobijediti, a što je slučaj kod žanra Battle Royale.

Zaključak

Kroz svoju dosta kratku povijest žanrovi računalnih igara su se uvelike promijenili i može se primijetiti kako je razvoj računalnih igara doista brzo napredovao, a i još uvijek eksponencijalno raste. Igrama se u pravilu ne određuje samo jedan žanr nego se gledaju sve karakteristike pojedinih žanrova u određenoj računalnoj igri te se onda toj igri dodijele žanrovi kojima pripada.

Može se zaključiti kako su neki žanrovi popularniji od drugih te da oni opći žanrovi poput recimo akcijskih igara imaju svoje podžanrove koji imaju osnovne karakteristike akcijskih igara, ali i svoje dodatne specifičnosti.

O budućnosti računalnih igara i njegovih žanrova teško je govoriti jer se pokazalo kako se glavni i najpopularniji žanrovi stalno mijenjaju dolaskom novih igara, a za očekivati je da će se ovaj trend i nastaviti.

U svakom slučaju, igrači uživaju u igranju svojih najomiljenijih igara i to im predstavlja veliko zadovoljstvo i igrači će uvijek tražiti igre po svojoj mjeri. Upravo zbog ovog razloga razvoj novih igara i novih žanrova će se nastaviti i dalje.

Reference

[1] D. Cohen, »OXO aka Noughts and Crosses - The First Video Game,« 2019. [Mrežno]. Dostupno na: https://www.lifewire.com/oxo-aka-noughts-and-crosses-729624. [Pristupano 1.9.2020.].

[2] J. Norman, »The First Video Game: "Tennis for Two",« [Mrežno]. Dostupno na: https://www.historyofinformation.com/detail.php?id=760. [Pristupano 1.9.2020.].

[3] C. Kohler, »Engineers Restore 1958 'Tennis For Two' Game, Now With Vacuum Tubes,« 2010. [Mrežno]. Dostupno na: https://www.wired.com/2010/12/tennis-for-two/. [Pristupano 1 9 2020].

[4] M. Bellis, »The History of Spacewar: The First Computer Game,« 2019. [Mrežno]. Dostupno na: https://www.thoughtco.com/history-of-spacewar-1992412. [Pristupano 1.9.2020.].

[5] K. Willaert, »Pixels In Print (Part 1): Advertising Computer Space – The First Arcade Video Game,« 2018. [Mrežno]. Dostupno na: https://gamehistory.org/first-arcade-game-advertisement-computer-space/. [Pristupano 1.9.2020.].

[6] J. Rosenberg, »Pac Man,« [Mrežno]. Dostupno na: https://hr.eferrit.com/pac-man/. [Pristupano 1.9.2020.].

[7] J. Juul, »A history of the computer game,« [Mrežno]. Dostupno na: https://www.jesperjuul.net/thesis/2-historyofthecomputergame.html. [Pristupano 1.9.2020.].

[8] C. Leslie i P. Byers, »How much money does Ninja make?,« 9 3 2020. [Mrežno]. Dostupno na: https://dotesports.com/culture/news/how-much-money-does-ninja-make-21954. [Pristupano 1.9.2020.].

[9] H. Bedeković, »Žanrovi igara,« [Mrežno]. Dostupno na: http://web.studenti.math.pmf.unizg.hr/~hrc313/zadaca.html. [Pristupano 1.9.2020.].

[10] technavio Blog, »Top 10 Most Popular Game Genres in the World 2018,« 2018. [Mrežno]. Dostupno na: https://blog.technavio.com/blog/top-10-most-popular-game-genres. [Pristupano 1.9.2020.].

[11] Wikiwand, »Colossal Cave Adventure,« [Mrežno]. Dostupno na: https://www.wikiwand.com/en/Colossal_Cave_Adventure. [Pristupano 1.9.2020.].

[12] PCChip, »Najbolji RPG naslovi za Windows računala svih vremena!,« 2018. [Mrežno]. Dostupno na: https://pcchip.hr/gaming/najbolji-rpg-naslovi-za-windows-racunala-svih-vremena/. [Pristupano 1.9.2020.].

[13] »WHAT IS A STRATEGY GAME?,« [Mrežno]. Dostupno na: https://www.kathleenmercury.com/what-is-a-strategy-game.html#. [Pristupano 1.9.2020.].
[14] M. Wilson, »Making Moves: Best Turn-Based Strategy Games on PC,« 2020. [Mrežno]. Dostupno na: https://store.hp.com/us/en/tech-takes/best-turn-based-strategy-games-pc. [Pristupano 1.9.2020.].
[15] E. Wilson, »A Matter Of Perspective: Comparing First And Third-person Shooters,« [Mrežno]. Dostupno na: https://thegamesedge.com/320/a-matter-of-perspective-comparing-first-and-third-person-shooters/#.XhcbNvGGuc. [Pristupano 1.9.2020.].
[16] Plarium, »What Is an MMO, What Is an MMORPG and the Difference Between Them,« 2018. [Mrežno]. Dostupno na: https://plarium.com/en/blog/difference-between-mmo-and-mmorpgs/. [Pristupano 1.9.2020.].
[17] Gaming street, »How battle royale games exploded in popularity,« 2019. [Mrežno]. Dostupno na: https://gamingstreet.com/how-battle-royale-became-popular/. [Pristupano 1.9.2020.].
[18] Computer History Museum, »Timeline of Computer History,« [Mrežno]. Dostupno na: https://www.computerhistory.org/timeline/graphics-games/. [Pristupano 1.9.2020.].
[19] J. Fritts, »Computer & Video Game Geners,« [Mrežno]. Dostupno na: https://cs.slu.edu/~fritts/csci1030/schedule/csci1030_game_genres.pdf. [Pristupano 1.9.2020.].

Virtualni interaktivni prikaz nastavnih materijala uz pomoć videoigara

Mislav Matijević i Mario Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
mmatijevi@foi.hr, mario.konecki@foi.hr

Sažetak

Nastavni proces je puno puta izazovan za mnoge studente. Puno puta je vrlo teško zadržati koncentraciju i shvatiti dovoljno detaljno sve aspekte izloženog gradiva. U ovom radu dan je prikaz prototipa sustava za generiranje prezentacija u 3D prostoru, čime se stvara podloga za interaktivno učenje putem virtualnih svjetova i prostora. Ovaj način poučavanja i učenja je lakši, zabavniji i zornije prikazuje nastavno gradivo na način koji studente drži zainteresiranima tijekom dužeg vremena. U radu je dan i kratki osvrt na mogućnosti korištenja 3D prikaza u obrazovanju.

Ključne riječi: Virtualni prikaz, interaktivni prikaz, 3D prostor, nastavni materijali, prezentacije

Uvod

Kad sam imao 9 godina, sasvim slučajno sam otkrio mogućnost stvaranja mapa za igricu Far Cry 2. Stvorio sam mapu i divio se svojoj tvorevini. Nije bitno što su to bili neki neuvjerljivo visoki brjegovi, nekakve razbacane eksplozivne bačve i par vozila, bitno je što je tada počela moja ljubav prema stvaranju 3D prostora, prostora koje možemo stvoriti po svojoj volji i po njima se potom kretati.

Nekoliko godina učenja kasnije, kada sam u 2. razredu gimnazije dobio zadatak iz likovnog napraviti prezentaciju o nekoj rimskoj građevini, odlučio sam ne koristiti Powerpoint, već eksperimentirati rekreirajući građevinu u Source Engineu i prezentirajući tako da letim oko nje, prikazujem bitne dijelove tako da im se približim itd. Uradak je prošao više-manje neshvaćen, no to me nije obeshrabrilo da nastavim izrađivati prezentacije za likovni na takav način.

Kako je vrijeme prolazilo, postavio sam si pitanje: zašto koristimo 2D prezentacije? One su po meni anakroni ostatak prošlosti kada su ljudi postavljali papire s tekstom na svjetlo ispred leće. Danas je taj plosnati oblik potpuno nepotrebno ograničenje. Zato sam odlučio napraviti program za izradu 3D prezentacija.

Zašto igre?

Proizvođači videoigara vrlo su rano krenuli u izradu igara u 3D prostorima [1]. Za razliku od 2D igara, 3 dimenzije omogućuju realniji, uvjerljiviji i dosljedniji prikaz okoline koju igrač istražuje. Doduše, cijena toga jest zahtjevna izrada tzv. „enginea“, odnosno kompleksnog skupa programa koji olakšavaju i uopće omogućuju prijevod 3D prostora u oblik razumljiv glavnom procesoru koji potom pod uputama (na Windowsima) DirectX-a šalje napredne naredbe grafičkom procesoru [2].

Na primjer, ako je igrica automobil, onda je engine, logično, motor koji sve pokreće i osigurava da auto ne eksplodira na uzbrdici, a boja sjedala, kvaliteta prozora, MP3 player, GPS sustav itd. su dijelovi koji čine ono što igrač vidi, tu igricu [3].

Kao što postoji više vrsta motora različitih proizvođača, tako postoji i mnogo game enginea, poput Unreala, Unityja, Sourcea itd. Sebe bih nazvao pomoćnim automehaničarem za Source Engine koji je osmislio kako ga pogoniti u obrazovanju.

Zašto Source Engine?

Ovaj engine pokreće mnoge popularne igrice (zajedno Source i Source 2): Counter Strike Global Offensive, Dota, Apex Legends, Insurgency (ne Sandstorm), Portal 2, Half Life Alyx i mnoge druge [4]. Stvorilo ga je poduzeće Valve Software na osnovi njihovog starijeg enginea „GoldSrc“.

Source je popularnost stekao kada je Valve Software izdao Half Life 2 davne 2004. Taj klasik u svijetu videoigara dobio je 39 nagrada za igru godine [5]. Sjećam se još iz razdoblja kada sam imao 5 ili 6 godina i igrao ga na najnižim postavkama na starom CRT monitoru rezolucije 640x480 px. Malome meni, čak i na takvim postavkama, bilo je teško razlučiti kakva grafika može biti bolja od toga. Kutije koje se mogu uzeti i nositi, razbiti, eksplozivne bačve, detaljni modeli lica (jedni od prvih takvih uopće!), revolucionarne refleksije, a kasnije je dodan i HDR (High-Dynamic Range) koji daje efekt privikavanja očiju na svjetlost. Apsurd je da čak i dan danas, kada ponovno po n-ti put krenem prelaziti taj klasik, grafika je i dalje vrhunska (a o gameplayju i priči da ni ne govorim).

Uglavnom, Source je postao vrlo popularan engine, pogotovo zato što je dozvoljavao jednostavno stvaranje novih mapa, modova i drugih elemnata 3D svijeta. Garry's mod je primjer kako je mod za Half Life 2 svome stvaratelju Garryju Newmanu donio milijune dolara [6].

Mogućnosti ovog enginea također su dobro prikazane u još jednoj fantastičnoj videoigri, Valveovom Portalu. U toj se pak videoigri, osim priče pune genijalnog crnog humora, ističe mogućnost ovoga enginea da stvara prolaze u prostoru koji vode do drugih dijelova istog prostora.

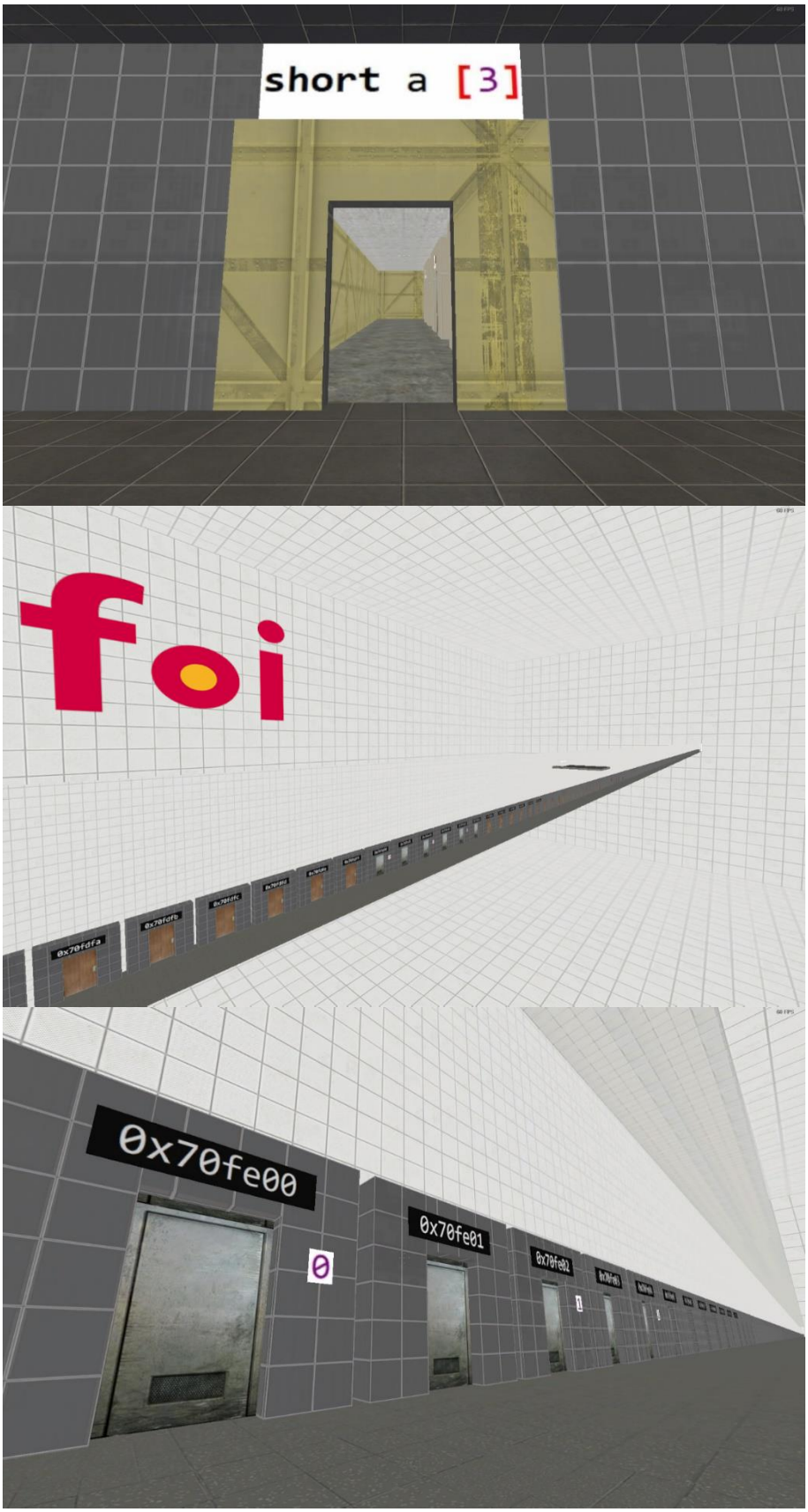
Je li Source Engine najbolji engine današnjice? Nije. To bi vjerojatno bio Unreal Engine. Valve se nije toliko bavio razvijanjem svog enginea koliko se bavio drugim proizvodima poput Steama. Doduše, nedavno izašli Half Life Alyx u potpunosti je izrađen u Source 2 engineu i malo koja videoigra današnjice se može po grafičkim mogućnostima usporediti s ovom videoigrom. Međutim, Source 2 još uvijek nije ni blizu rasprostranjen kao što će (barem po mom mišljenju) biti u budućnosti.

Zašto 3D prezentacija?

Mogućnosti koje nam videoigre nude su beskonačne. Moguće je potrošiti mjesece stvarajući već postojeću okolinu, pazeći na najmanje detalje, a moguće je stvoriti i neku novu okolinu s novim zakonima fizike.

Interakcija s takvim okolinama moguća je na mnogo načina: pritisak na gumb, povezivanje kabela od jedne stavke kroz prostor prema drugoj, običan prolazak kroz vrata, a čak i najjednostavniji pogled prema nekom elementu može biti okidač neke akcije nad njime. Tranzicije sa slajda na slajd više ne moraju biti pretapanje ili nekakav kič otkrivanja zastora. To sada može biti lift ili automobil kojim se korisnik doveze do sljedećeg slajda: savršena podloga za razvoj nastavnih materijala! Nastavni materijali mogli bi biti realizirani kao hodnik s vratima u kojemu svaka vrata vode u sobu s određenim slajdom i s raspoređenim modelima na koje se slajd nadovezuje. To bi mogla biti i samo jedna soba heksagonskog oblika u kojoj na svakom zidu postoji slajd, a ispred svakog slajda nalaze se određeni modeli. Zamislimo u bližoj budućnosti predavanje na fakultetu kao online server gdje je profesor administrator, a studenti su igrači, pa svi sa svojim naočalama za virtualnu stvarnost hodaju kroz prezentaciju i ostvaruju interakciju s nastavnim materijalima koji su modeli s fizikom.

Kao demonstrator na predmetu Programiranje 1, odlučio sam ozbiljnije testirati ovaj koncept te sam video lekciji koja se bavi pokazivačima dodao i 3D dio. Prvo je prikazano polje 'a' s tri elementa – malen hodnik s troja vrata koja se mogu otvoriti. Potom je pokazana radna memorija koja je u biti jedan divovski hodnik s hrpom vrata, a iznad svakih vrata nalazi se njihova heksadecimalna adresa. Određenim vratima se može pristupiti, otvoriti ih i pogledati/pročitati sadržaj iza njih. Na kontrolnoj ploči može se alocirati prostor i potom se neki podatak može pohraniti u novo dobivena vrata. Navedeno je prikazano na slici br. 1.



Slika br. 1: 3D prezentacija pokazivača

Studenti su mi se osobno javljali s pozitivnim reakcijama, rekavši kako im je takav 3D prikaz uvelike pomogao pri savladavanju tog najtežeg dijela gradiva. Pretpostavljam da im je takav prikaz pomogao ne samo zato što je drugačiji od obične prezentacije natrpane tekstom, već i zato što im je nesvjesno zanimljiviji prikaz kroz prizmu videoigre i pokreta kroz virtualni prostor.

Virtualni interaktivni prikaz nastavnih materijala

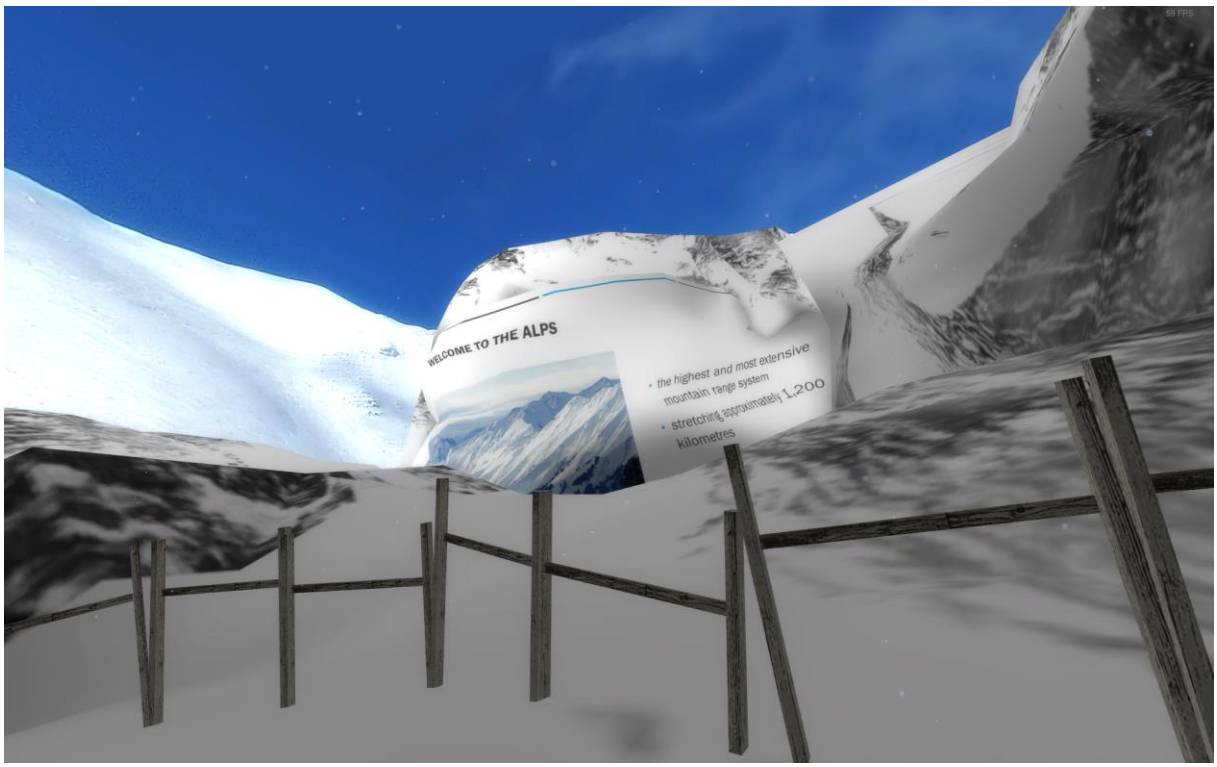
U nastavku je dodatno pokazano nekoliko isječaka onoga što je moguće postići ovakvom tehnologijom (slike br. 2 - 6).

Slika br. 2: 3D prikaz slika

Na slici br. 2 prikazane su slike baroknog slikara Caravaggia u odgovarajućem ambijentu: u špilju se ulazi iz mračne šume, unutra šumi vjetar i pucketaju kamenčići, a prilikom približavanja ovoj glavnoj sobi, u pozadini započinje svirati Mozartova Lacrimosa. To sve budi jedan set emocija i osjećaja koji se nikada nije mogao niti će se ikada moći postići klasičnim prezentacijama. Ovaj prikaz njegovih slika nema teksta, očekuje se da predavač kaže sve ono što je bitno. Ovakva prezentacija ima zadatak osvojiti gledatelja dojmovima uz koje će mu sadržaj prezentacije još dugo ostati u sjećanju.

Slika br. 3: Prikaz slike u 3D svijetu

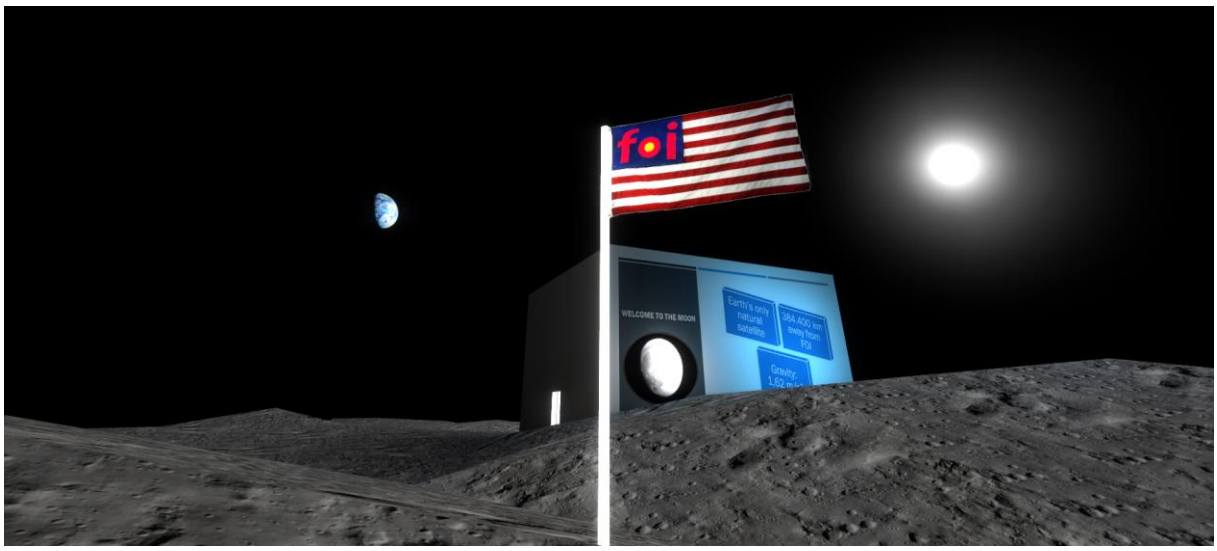
Galerija slikara Tiziana (slika br. 3) u kojoj su pojedinačne sobe visoke 50-ak metara. Svaka soba sadržava vlastitu sliku veličine stotina metara kvadratnih. Neizvedivo u stvarnosti postaje izvedivo u videoigri.



Slika br. 4: Učenje o Alpama u 3D svijetu

Slajd o Alpama prikazan na slici br. 4 dio je Alpa. Slajdovi više nikada ne trebaju biti plohe, sada mogu biti savijeni, zgužvani, kakvi god. Slajd može biti i

pokretan, može biti objekt na koji djeluje fizika, može ispuštati zvuk… Mogućnosti su neograničene.



Slika br. 5: Učenje o mjesecu putem šetnje mjesecom

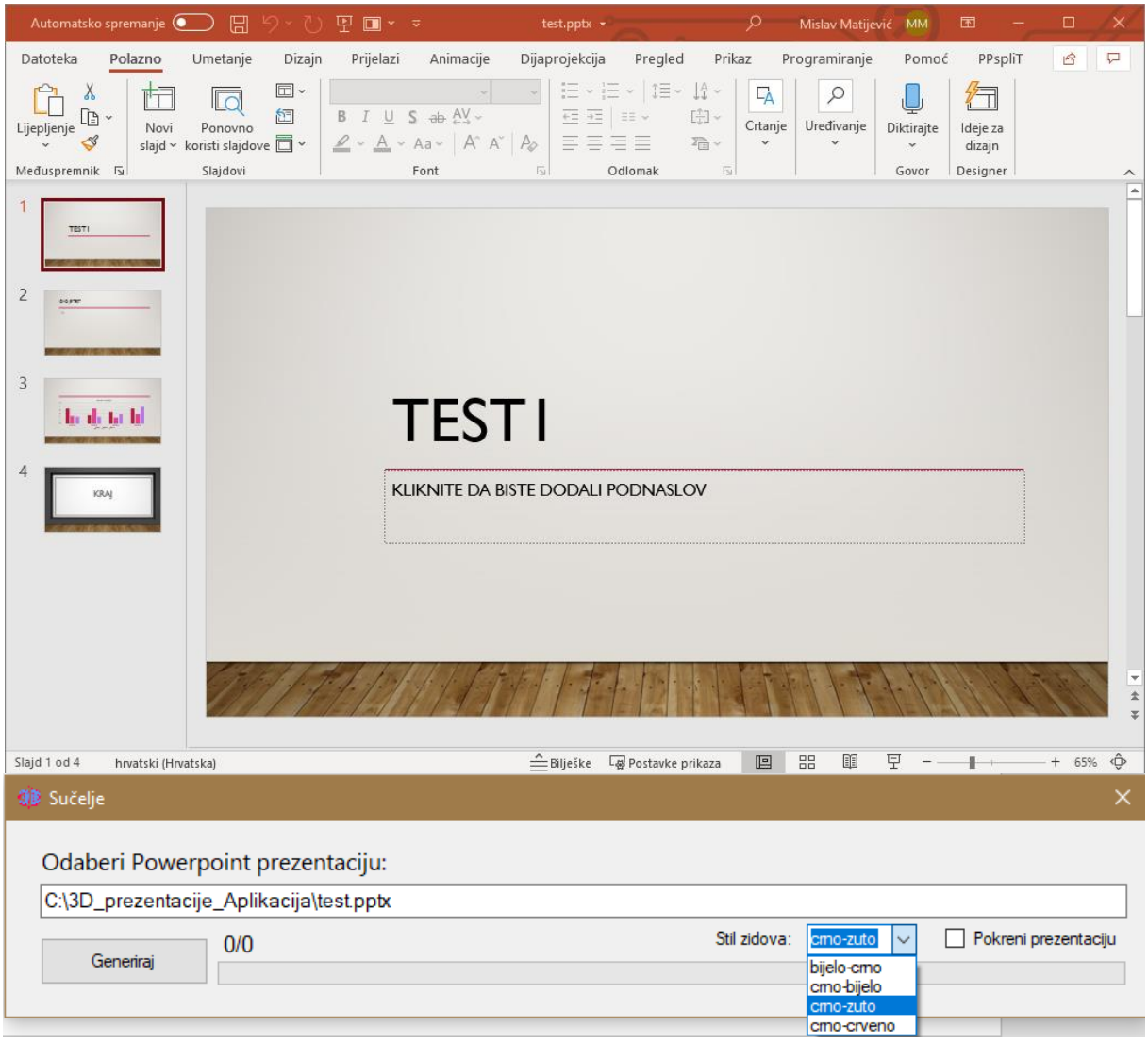
Slajd o Mjesecu prikazan na slici br. 5 se nalazi na Mjesecu. Gravitacija je smanjena da bude točno onakva kakva je na Mjesecu, a prikazano je i kako se Zemlja vidi s površine Mjeseca. Isto bi se moglo napraviti za svaki drugi planet. Tu je naravno i zastava Fakulteta organizacije i informatike.

Slika br. 6: Učenje o Internetu stvari (engl. IoT Internet of Things)

Slajd vezan za IoT prikazan na slici br. 6 objašnjava kako ispravno spojiti neki rotor u stvarnom svijetu. U 3D prezentaciji, korisnik isprobava ovo spajanje i ako uspješno izvrši spajanje, model rotora se počinje vrtjeti.

Realizirani program za virtualni interaktivni prikaz nastavnih materijala

Program je realiziran korištenjem razvojnog okruženja Visual Studio i programskog jezika C#. Na slici br. 7 prikazan je procesa generiranja 3D prostora temeljem odabranog seta slajdova .



Slika br. 7: Pretvaranje prezentacije u 3D prostor

Slika br. 8: Slajdovi u 3D prostoru

Na slici br. 9 prikazano je kako bi izgledali slajdovi kada bi ih se gledalo iz druge perspektive, izvan generiranog prostora. Program je generirao 4 sobe za 4 slajda te je u svaku sobu u 3D prostoru smjestio drugi slajd, a korisnik se stvorio u malenoj početnoj sobici prije prvog slajda.

Kako postići automatski generirani prostor?

Videoigra (u ovom slučaju Half Life 2) razumije prostor zapisan u datoteci s *.bsp ekstenzijom.

Ekstenzije *.bsp (kratica za Binary Space Partitioning) svoj korijen vuku daleko u prošlost, iz starog Quake game enginea. Istu ekstenziju za binarni zapis kompajliranih mapa koriste i stari Medal of Honor (npr. Allied Assault), ranije inačice videoigre Call of Duty. Kao što je već spomenuto, stari MoH-ovi i rani CoD-iji imaju jednu zajedničku karakteristiku, oba ova serijala koriste isti program (Radiant) za izradu njihovih mapa [7].

Zašto je važan program u kojemu se rade mape? Zato što su one, prilikom svoje izrade, zapisane u tekstualnom obliku, baš poput programskog koda. Dok osoba zadužena za kreiranje mapa izrađuje mapu u programu za Source Engine koji se zove Hammer, taj program u biti tekstualno zapisuje tu mapu. Upravo je odgovarajuće rješenje u C#-u, koje je dio prikazanog prototipa za generiranje prezentacija u 3D prostoru, izrađeno kako bi određene blokove

koda kojim se opisuje mapa bilo moguće multiplicirati onoliko puta koliko prezentacija ima slajdova te time stvoriti sobu za svaki pojedinačni slajd.

Zaključak

2020. je na izmaku, a za potrebe prezentiranja se svejedno manje-više standardno koriste 2D prezentacije kao i prije 80 godina. Mogućnosti koje donosi razvoj videoigara i njihovih enginea su beskonačne, a dodatnu svrhu bi im svakako mogli dati i nastavni materijali iz najrazličitijih područja. To bi približilo nastavne materijale studentima jer bi preuzeli zanimljiviji i uzbudljiviji model videoigara. Uvedene bi bile i razni do sada nemogući aspekti i elementi: promjenjivi ambijent s obzirom na lokaciju slajda, ozračje i tematiku prezentacije, interaktivni modeli putem kojih student može provjeriti i potvrditi naučeno gradivo, gumbi, vozila itd.

Konačno, u budućnosti će zasigurno biti moguće predavanja organizirati putem online servera, pri čemu svi sudionici imaju naočale za virtualnu stvarnost i hodaju kroz nastavne materijale s predavačima, djelujući nad 3D modelima izrađenima po uzoru na dijelove stvarnog svijeta na koje se nastavni materijali odnose.

Reference

[1] Which is the first 3D game?, https://www.quora.com/Which-is-the-first-3D-game, pristupljeno 7.4.2020.
[2] DirectX 12: what is it, and why it matters to PC gamers, https://www.techradar.com/news/gaming/directx-12-what-is-it-and-why-it-matters-to-pc-gamers-1318636, pristupljeno 7.4.2020.
[3] What is a Game Engine?, https://www.gamecareerguide.com/features/529/what_is_a_game_.php, pristupljeno 7.4.2020.
[4] Source 2, https://developer.valvesoftware.com/wiki/Source_2, pristupljeno 7.4.2020.
[5] Awards and Honors, https://web.archive.org/web/20140108083038/http://www.valvesoftware.com/awards.html, pristupljeno 7.4.2020.
[6] How a 'total accident' led to Garry's Mod's funniest feature and 15 years of twisted success, https://www.pcgamer.com/garrys-mod-interview/, pristupljeno 7.4.2020.
[7] .BSPFile Extension, https://fileinfo.com/extension/bsp, pristupljeno 7.4.2020.

Izrada akcijske 3D igre u programskom alatu Unity

Antonio Oletić i Mario Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
aoletic@foi.hr, mario.konecki@foi.hr

Sažetak

Na početku ovog rada opisan je žanr 3D akcijskih videoigara. Nakon toga je dan prikaz razvojnog sučelja Unity koje je korišteno za razvoj videoigre prikazane u ovom radu. Uz opis korisničkog sučelja i alata Unity-ja opisani su i svi koraci potrebni za izradu jedne jednostavne 3D akcijske videoigre u kojoj se prelazak razine postiže skupljanjem određenog broja objekata što je otežano pomoću AI neprijatelja. Na kraju će prikazan izrađen programski primjer jedne 3D akcijske videoigre.

Ključne riječi: Unity, videoigra, akcijska igra, 3D

Uvod

Ovaj rad prikazuje kompletan proces izrade 3D videoigre. Industrija videoigara konstantno raste pa se tako pojavljuje sve više razvojnih okolina za izradu videoigara. Neka velika poduzeća koje se bave izradom videoigara su se čak odlučila za kreiranje svojih razvojnih okolina, no unatoč tome tržištem još uvijek dominiraju dva „diva“ za izradu videoigara, a to su: Unity i Unreal Engine. U ovom radu bit će prikazana proces izrade videoigre u razvojnom okruženju Unity. Prvo će biti dan prikaz 3D akcijske videoigre i opis ovog žanra, a nakon toga će korištena razvojna okolina biti uspoređena s drugom najpopularnijom razvojnom okolinom.

Unity je vrlo sadržajna i intuitivna razvojna okolina. Sadrži puno gotovih funkcija koje početnicima u toj industriji pomažu kod razvoja videoigara. Unity Asset Store je Unity-jeva online trgovina resursima koja je jedan od razloga zašto je Unity toliko popularan među početnicima. Tamo se jednostavno mogu pronaći proširenja za Unity, kao i gotovi 3D modeli, zvukovi te animacije. Odabir pristupa razvoju videoigara je vrlo važan i bit će prokomentiran u okviru ovog rada.

Koncept

Tema projekta je prototip 3D akcijske igre. Na početku razvoja videoigre, bilo je potrebno osmisliti generalni koncept. Odabran je sustav prikupljanja

objekata nalik na kovanice u videoigri Super Mario. Cilj videoigre je prikupljanje određenog broja objekata. Kao teren odabrana je dolina s pokojim brdom, omeđena planinama sa svih strana. Planine predstavljaju kraj svijeta. U dolini se nalaze i dva jezera spojena malom rječicom. Nakon osmišljavanja koncepta terena, cilja i načina igranja videoigre, bilo je potrebno unijeti neke funkcionalnosti koje bi otežale igru te je samim time učinile zanimljivijom. Definirani su neprijatelji koji će slijediti glavnog lika te mu oduzimati životne bodove. Životni bodovi se prikazuju na ekranu i ako glavni lik ostane bez životnih bodova, umire. Kada glavni lik umre, igra počinje ispočetka. Nakon definiranja navedenih aspekata, moguće je krenuti s izvedbom u razvojnoj okolini Unity. Kreiran je novi projekt i klikom na gumb New Scene stvorena je nova prazna scena. Scene se može interpretirati kao jedan nivo u Unity-ju.

Kretanje glavnog lika

Kretanje glavnog lika je vrlo važno u svakoj videoigri. U prikazanoj videoigri, glavni lik se kreće u tri osi zbog toga što je igra rađena u 3D-u. Glavni lik se može kretati tipkama W, A, S, D ili strelicama. Glavni lik može i skakati pritiskom na tipku Space te može brzo trčati uz držanje tipke Left Shift.

Unity sadrži besplatni paket koji se zove Standard Assets. U tom paketu postoji verzija gotovog upravljača iz prvog lica koju je moguće koristiti s funkcionalnostima hodanja, trčanja, okretanja i skakanja. Kod koji implementira trčanje i hodanje glavnog lika prikazan je u vidjeti u nastavku.

Na početku koda provjerava se unos kako bi se utvrdilo kreće li se igrač horizontalno ili vertikalno. Ako se igrač kreće i drži tipku Left Shift, ta kretnja se registrira kao trčanje. Na kraju koda se postavlja brzina kretanja ovisno o tome radi li se o trčanju ili hodanju.

```
private void GetInput(out float speed)
{
        // je li unos horizontalno kretanje ili vertikalno
float horizontal= CrossPlatformInputManager.GetAxis("Horizontal");
        float vertical = CrossPlatformInputManager.GetAxis("Vertical");
        bool waswalking = m_IsWalking;

#if !MOBILE_INPUT
        // ukoliko je pritisnuta tipka left shift onda trči
        m_IsWalking = !Input.GetKey(KeyCode.LeftShift);
#endif
        // postavljamo željenu brzinu na hodanje ili trčanje
        speed = m_IsWalking ? m_WalkSpeed : m_RunSpeed;
        m_Input = new Vector2(horizontal, vertical);

}
```

Funkcionalnost neprijatelja koji prati igrača

Ranije je dodan objekt vojnika unutar prve scene koji će u videoigri predstavljati neprijatelja. Nakon toga je dodan Rigidbody i Capsule Collider te sada vojnik ima fizički oblik. Trenutačno vojnik ne radi ništa te je potrebno napraviti funkcionalnost da vojnik slijedi i napada igrača. Implementirat će se funkcionalnost da se, ako glavni igrač svojim kretnjama dođe na određenu daljinu, vojnik okreće za njim.

Klasa ChasePlayer služi u tu svrhu. Na početku se deklarira igrača prema kojemu će se transformirati kretnje vojnika i deklarira se animator pomoću kojega će vojnik biti animiran.

Detaljnije o animatoru će biti riječi u sljedećem poglavlju. Funkcija Update(), koja se izvršava više puta u sekundi, definira da se, ako se glavni igrač nalazi na lokaciji bližoj od 15 mjernih jedinica, vojnik okreće prema igraču. Nakon što se vojnik okrene prema igraču, animator se postavlja na hodanje te vojnik kreće prema igraču brzinom od 0.05f. Sada to izgleda kao da vojnik lebdi jer još uvijek ne postoje nikakve animacije vezane za kretnje vojnika. Nakon toga provjerava se je li vojnik bliži od 2 mjerne jedinice. Tada se animator postavlja na napadanje i vojnik će početi napadati glavnog igrača. Taj napad se ne vidi jer još ne postoji nikakva animacija. U slučaju da je vojnik dalje od glavnog igrača za 15 mjernih jedinica, animator se postavlja na njegovu Idle animaciju, u kojoj se on lagano pomiče na mjestu.

```
public class ChasePlayer : MonoBehaviour
{
   public Transform player;
   private Animator anim;

   void Start() {
       anim = GetComponent<Animator>();
   }


    // Update is called once per frame
    void Update()
    {
        if(Vector3.Distance(player.position, this.transform.position) < 15)
        {
            Vector3 direction = player.position - this.transform.position;
            direction.y=0;

          this.transform.rotation=Quaternion.Slerp
            (this.transform.rotation,
            Quaternion.LookRotation(direction), 0.1f);

            anim.SetBool("isIdle", false);
            if(direction.magnitude > 2)
            {
                this.transform.Translate(0, 0 ,0.05f);
```

```
                anim.SetBool("isWalking", true);
                anim.SetBool("isAtacking", false);
            }
            else{
                anim.SetBool("isAtacking", true);
                anim.SetBool("isWalking", false);
                player.GetComponent<PlayerHealth>().TakeDamage(1);


            }
        }
        else{
            anim.SetBool("isIdle", true);
            anim.SetBool("isWalking", false);
            anim.SetBool("isAtacking", false);
        }

    }
}
```

Funkcionalnost smanjivanja životnih bodova

Napadi protivnika u videoigri bili bi beskorisni da ne predstavljaju prijetnju glavnom igraču. U tu svrhu potrebno je kreirati životne bodove igrača. Kreirani su u obliku trake koja se nalazi na sredini ekrana. Traka je zapravo takozvani Slider te se popunjenost trake bojom može kontrolirati povećavanjem i smanjivanjem vrijednosti. Za kreiranje te trake zadužena je klasa HealthBar.

```
public class HealthBar : MonoBehaviour
{
    public Slider slider;

    public void SetMaxHealth(int health){
        slider.maxValue = health;
        slider.value = health;
    }
    public void SetHealth(int health){
        slider.value =health;
    }
}
```

Na početku je deklariran Slider koji je predstavlja traku životnih bodova. U funkciji SetMaxHealth postavlja se vrijednost maksimalne vrijednost slidera na varijablu health. Također se postavlja trenutačna vrijednost slidera na varijablu health. U funkciji SetHealth trenutačna vrijednost slidera se postavlja na varijablu health. Te funkcije ćemo pozivati u sljedećoj klasi koja je zadužena za određivanje igračevih životnih bodova. To je klasa PlayerHealth.

```
public class PlayerHealth : MonoBehaviour
{
    public int maxHealth = 3000;
    public int currentHealth;

    public HealthBar healthBar;
```

```
    void Start()
    {
        currentHealth = maxHealth;
        healthBar.SetMaxHealth(maxHealth);
    }

    public void TakeDamage(int damage){
        if(PauseMenuScript.GameisPaused==false){
       currentHealth -= damage;

       healthBar.SetHealth(currentHealth);
       if(currentHealth==0){
           FindObjectOfType<GameManager>().EndGame();
       }
        }
   }
}
```

Na početku je deklarirana varijabla maxHealth te je postavljena na vrijednost 3000. Također je deklarirana i varijablu currentHealth koja predstavlja trenutačne životne bodove. Deklariran je i objekt healthBar koji predstavlja traku životnih bodova. Prije prikaza prve sličice u igri, izvršava se funkcija Start(). U toj funkciji deklarirani su trenutačni bodovi, na početku ujedno i maksimalni bodovi te je u klasi HealthBar definirano da su maksimalni bodovi jednaki varijabli koja je deklarirana na početku ove klase. U funkciji TakeDamage koja se poziva kada vojnik napada provjerava se je li igra pauzirana ili ne. Ukoliko igra nije pauzirana, od trenutačnih životnih bodova se oduzima šteta koja se zadaje u postavkama HealthBara. Kada životni bodovi igrača dođu do nule, igra se završava. Kako su životni bodovi prikazani izrašenoj videoigri može se vidjeti na slici 1.

Slika br. 1: Životni bodovi igrača

Zaključak

Akcijske igre su samo jedan od mnogih žanrova videoigara. Njihova dinamična priroda i borba s preprekama i neprijateljima, korištenje oružja i alata te predivni svjetovi u kojima se odvijaju, doprinose tome da su akcijske videoigre jedne od najigranijih, ako ne i naigraniji žanr videoigara. Odabir razvojne okoline za izradu akcijske igre je također vrlo važan. Unity je u prošlosti bio dominantan u usporedbi s Unreal Engine-om. Radikalnim promjenama Unreal Engine-a smanjuje se razlika u dominantnosti Unity-jeve razvojne okoline te se može reći da su danas obje razvojne okoline u puno segmenata podjednake. Za izradu prikazanog prototipa videoigre korišten je Unity s programskim

jezikom C#. Unity je vrlo jednostavan za korištenje te sadrži pregršt alata za izradu jednostavne 3D akcijske igre. Lako se snalaziti sučeljem Unity-a što doprinosi tome da je Unity vrlo popularan među početnicima. Unity također uvelike olakšava uvođenje 3D modela i primjenjivanje animacija na te iste modele.

Reference

[1] History.com Editors, 2017, Video Game History. Dostupno: [20.6.2020.]

[2] Fandom, Action adventure game. Dostupno: https://videogamehistory.fandom.com/wiki/Action-adventure_game [20.6.2020.]

[3] Imagine Media, The Next Generation 1996 Lexicon A to Z, Časopis broj 15, US:GP Publications (1996)

[4] Dice, How Unity3D Became a Game-Development Beast. Dostupno: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast [21.6.2020.]

[5] ArsTechnica, Unity at 10: For better—or worse—game development has never been easier. Dostupno: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/ [21.6.2020.]

[6] Unity Technologies, 2020, Unity for all. Dostupno: https://unity.com [21.6.2020.]

[7] Epic games, 2020, Unreal engine. Dostupno: https://www.unrealengine.com/en-US/features [21.6.2020.]

[8] Program-Ace, Unity vs. Unreal: What to Choose for Your Project? Dostupno: https://program-ace.com/blog/5-years-of-unity-vs-unreal/ [20.8.2020.]

[9] M. Gligora Marković, M. Antić, M. Rauker Koch: Proces razvoja multimedijske računalne igre, Zbornik Veleučilišta u Rijeci, Vol. 1 (2013)

Razvoj alata za izradu mozaika pomoću GameMaker studija 1.4

Martin Starešinčić
Fakultet organizacije i informatike, Sveučilište u Zagrebu
mstaresin@foi.hr

Sažetak

U radu je dan prikaz razvoja alata za izradu mozaika pomoću GameMaker studija (GMS) 1.4. Opisana su rješenja niza problema koji se javljaju prilikom izrade alata ovakvog tipa unutar GMS-a, poput procesa unutar crne kutije, sandbox, ispis i upis u datoteku i slično. Alat za izradu mozaika bi trebao sadržavati sve potrebne funkcije kako bi se u njemu mogla realizirati kreacija mozaika, izvoz slike tog mozaika te inventar potrebnih dijelova za izradu istog.

Ključne riječi: razvoj, alat, GameMaker studio, GMS 1.4, mozaik

Uvod

U radu je dan prikaz postupka razvoja alata za izradu mozaika, kao i izazova s kojim će se početnici susresti na ovakvom projektu. Sam alat je izrađen u okviru stvarnog komercijalnog projekta koji je naručio jedan singapurski umjetnik.

Iako bi mnogi mogli pomisliti da bi za izradu ovakvog alata bio potreban C++, kao i u slučaju mnogih drugih projektnih zadataka, rješenje je moguće postići na više načina pa se tako u ovom slučaju koristio GameMaker studio 1.4.

GameMaker studio 1.4

GameMaker studio je 2D programski alat za izradu videoigara napravljen od strane poduzeća YoYo Games. Dizajniran je na način koji omogućava jednostavan razvoj videoigara bez potrebe za učenjem složenog programskog jezika poput C++-a, korištenjem drag-and-drop sustava.

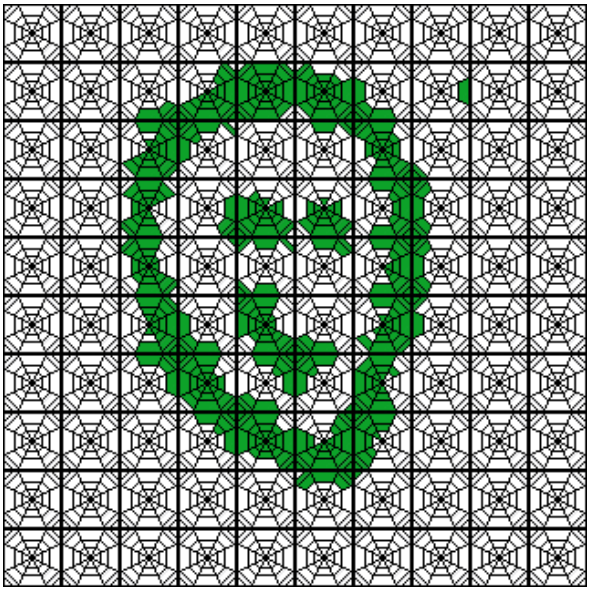
GameMaker također ima ugrađeni programski jezik GameMaker Language ili skraćeno GML. GML omogućuje korisnicima upisivanje koda koji će se pokretati tijekom njihove videoigre i pruža veću razinu fleksibilnosti i finog podešavanja u odnosu na drag-and-drop sustav.

GameMaker prvenstveno je dizajniran za 2D videoigre. GameMaker: Studio omogućava stvaranje programa i izvoz za upotrebu na više uređaja i

operativnih sustava, uključujući PC, Mac, Linux, i Android, i na mobilnoj i na stolnoj verziji.

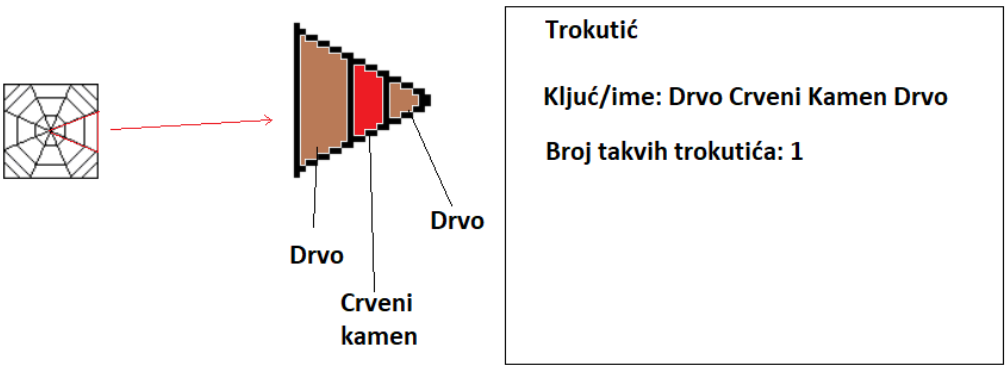
Potrebne funkcionalnosti alata za izradu mozaika

Funkcionalnost alata je izrada mreže mozaika, izrada materijala tj. boja kojima se mozaik ispunjava, prikaz mozaika do najmanjeg djelića. S obzirom na to da se mozaik sastoji od kocki, te se kocke sastoje od 8 trokutića plus 4 jednodijelna trokutića, dok se ostali trokutići sastoje od 3 dijela. Bojanje i uređivanje samog mozaika je isto tako sastavni dio funkcionalnosti alata.



Slika br. 1: Prikaz mozaika 10x10

Jedna od najvažnijih funkcija je funkcija inventar trokutića. Naime, kada bi se svi dijelovi pojedinog trokutića popunili s nekim materijalima program bi trebao kreirati ključ za takav tip trokutića te ga dodati u inventar i ako takav trokutić postoji inkrementirati broj tipa takvog trokutića.

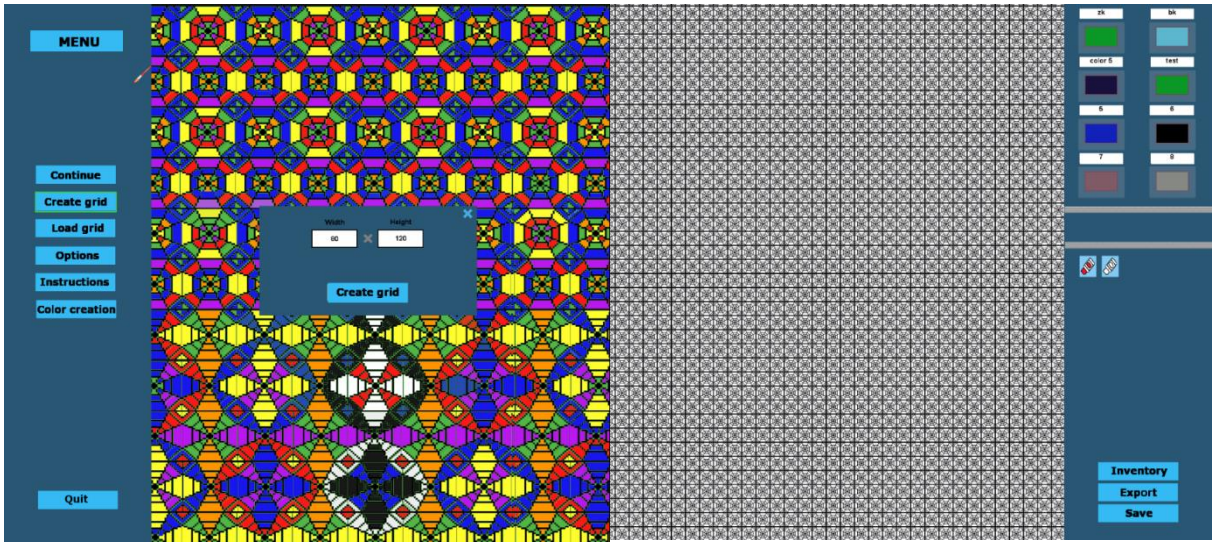


Slika br. 2: Ideja inventara trokutića

Trebala je isto tako postojati mogućnost spremanja mozaika i ponovnog otvaranja, a uz to da sadrži sve informacije upisane u inventar tog mozaika. Isto tako, bila je potrebna mogućnost izvoza mozaika u obliku slike. Kao što je potrebno spremanje mozaika isto tako je potrebno i spremanje paleta tj. materija koje je korisnik napravio.

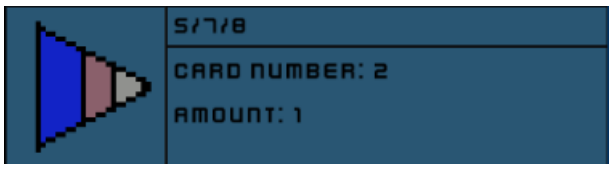
Izrada alata za izradu mozaika

U izradi ovog alata bilo je potrebno izraditi jezgru alata, ali i odgovarajući GUI Prvi problem koji je uočen je velika količina objekata koji će se nalaziti na ekranu jer alati poput GMS-a nisu previše optimizirani te vrlo često tijekom rada koriste veliku količinu memorije i procesorske snage. Korištenje ugrađenih funkcija za detekciju dodira u GMS-u je odmah otpalo kao rješenje te je izrađena vlastita funkcija detekcije koja radi tek kada se korisnik dovoljno približi mišem kako bi ostvario interakciju s objektom. Ovo je napravljeno na opisani način jer kalkulacija udaljenosti zahtijeva manje resursa od provjere područja. Isto tako, program je sporiji čim je aktivno više instanci objekata, koji često sadrže puno procesa. U izrađenoj implementaciji nepotrebne instance objekta se isključuju, npr. ako je nešto izvan ekrana. Jedan od trikova za optimizaciju je i način na koji se detektira dodir mišem. Naime, ako je korisnik povećao svoj kist, sustav za detekciju dobiva informaciju o veličini kista te preskače provjeru nepotrebnih područja, a to su sva područja udaljena za polovicu veličine kista od područja provjere. Kako bi se dodatno naglasila važnost optimizacije ovog programa, važno je naglasiti da u mozaiku veličine 120x120 postoji 547200 dijelova te svaki pojedini dio može imati utjecaj na inventar trokutića i izgled mozaika.



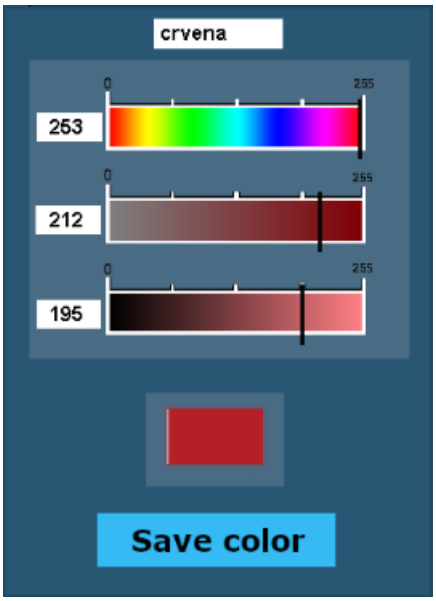
Slika br. 3: Meni za kreaciju mozaika i editor mozaika

Upis u inventar funkcionira tako da se trokut upisuje na način da se u trenutku popunjavanja manjih dijelova trokuta vrši provjera prethodnog stanja trokuta te se utvrđuje je li bio prazan ili popunjen i ako je bio popunjen to znači da postoji ključ u inventaru te da treba oduzeti jedan od broja trokutića toga tipa te onda pogledati trenutačno stanje i opet provjeriti postoji li ključ za novi tip trokutića te inkrementirati broj takvih trokutića. Ako nije postojao, kreirat će se novi ključ, a ključ je u biti niz imena materija od kojih se sastoji (npr. „Drvo Kamen Crveni kamen“).



Slika br. 4: Primjer kartice u inventaru trokutića koji se sastoji od materijala nazvanih 5,7,8

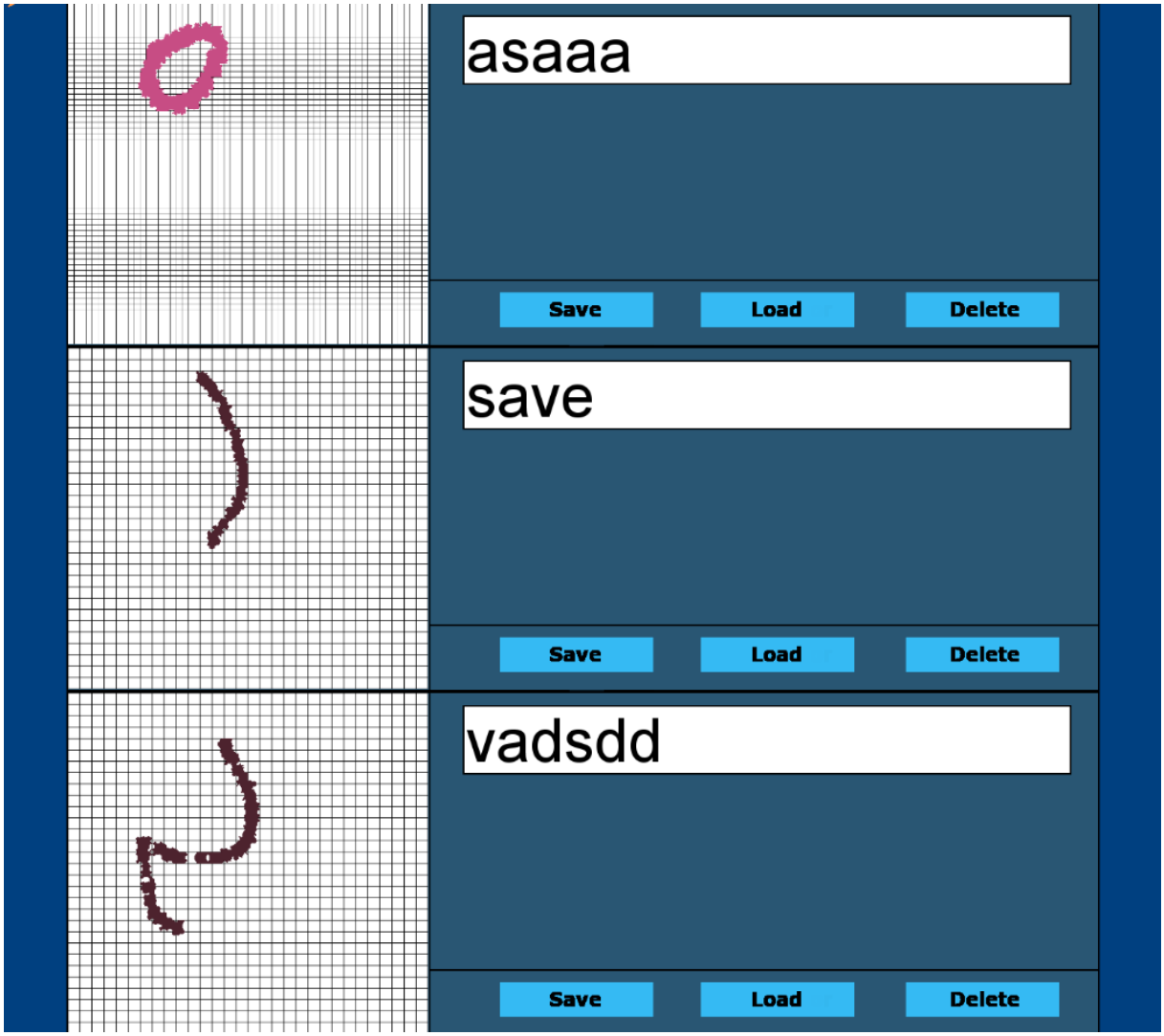
Kreator boja funkcionira po HSV sistemu što je prikazano na slici 5. Nakon kreacije boje, korisnik je može spremiti te joj pristupiti preko palete u mozaik editoru.



Slika br. 5: Meni za kreaciju boja

Od velike važnosti je mogućnost otvaranja i spremanja mozaika. Samo spremanje informacija mozaika je bilo zahtjevno s obzirom na to da se pri svakom upisu u datoteku pokazivač vraća na početak. Problem upisa u nekom racionalnom vremenu riješen je na način da je realiziran sustav koji kreira novu datoteku nakon optimalnog broja unosa u nju te se tako izbjegava veliko pomicanje pokazivača. Treba imati na umu da se u većini slučajeva upisuje stotinjak tisuća podataka te sustav pomicanja pokazivača svaki put do kraja datoteke koja ima desetke tisuća informacija predstavlja vremenski zahtjevan

posao. Tijekom upisa vrši se aktivacija i deaktivacija objekata ovisno o poziciji na kojoj se pokazivač nalazi, a sličan princip je primijenjen i kod ispisa, jedino što se kod ispisa nije trebala provoditi aktivacija nego kreacija objekata. Postoji još jedan problem kod spremanja datoteka, to da je GMS 1.4 u sandbox-u, što znači da ne može pristupiti datotekama van svog korijenskog foldera. Taj problem je riješen korištenjem DLL-a napisanog u C++-u „Non-Sandboxed FileSystem“ od strane kreatora YellowAfterlife. Što se tiče izvoza slika GMS ima sustav površina te je bilo vrlo jednostavno kreirati dodatnu površinu nad trenutačnim prikazom i preslikati trenutačni prikaz na novu površinu i tu površinu konvertirati u sliku.



Slika br. 6: Meni za učitavanje spremljenih mozaika

Zaključak

Razvijeni alat za izradu mozaika efektivno provodi svoje funkcije. Alat je opisan, a opisani su i problemi do kojih je došlo prilikom razvoja alata. Dan je prikaz funkcionalnosti GMS-a i načina na koje su spomenuti problemi riješeni.

Isti principi bi se mogli primijeniti prilikom razvoja drugih alata za razvoj videoigara. Ideja razvoja alata unutar alata za izradu videoigra je dosta intrigantna te je dosta toga izvedivo uz dobru dozu kreativnosti.

Reference

[1] GameMaker: Studio user manual, YoYo Games, (29.12.2020.), dostupno na: https://docs.yoyogames.com/
[2] YellowAfterlife, Non-Sandboxed FileSystem extension (29.12.2020.), dostupno na: https://yal.cc/r/17/nsfs/
[3] N. Aucket (2015.), GameMaker Essentials, Packt Publishing, 29.12.2020.

Izrada klona strateške igre “Plants vs. Zombies” u programskom alatu Unity

Ilija Vuk i Mladen Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
ivuk@foi.hr, mlkoneck@foi.hr

Sažetak

U ovom radu obrađena je tematika izrade klona računalne igre “Plants vs. Zombies” u programskom alatu Unity. U radu je opisana implementacija osnovnih mehanika igranja navedene igre.

Ključne riječi: 2D računalna igra, Unity, Plants vs. Zombies, strateška igra, mehanike igranja, programiranje, algoritmi

Uvod

Izraditi u potpunosti samostalno računalnu igru je vrlo zahtjevan proces koji zahtjeva znanja širokog spektra. Što se tiče implementacije programske logike računalne igre, u današnje vrijeme moguće je koristiti alate koji su već gotovi pogoni (eng. *engine*) za razvoj računalnih igara i time je taj proces značajno olakšan [4]. Nije potrebno implementirati mnoge funkcije najniže razine već imamo gotove funkcije koje nam omogućavaju da programiramo na višoj razini apstrakcije. Također unutar takvih alata postoji podrška za realizaciju tipičnih stvari prilikom izrade jedne računalne igre: postavljanje objekata u prostor, njihova transformacija, skriptiranje, animiranje i sl. Jedan od tih pogona je i programski alat Unity koji je korišten u svrhu izrade ovog rada [6].

Korišteni programski alati

Unity je jedan od najpopularnijih besplatnih pogona za razvoj računalnih igara. Unity omgućava izvoz kreirane igra na preko 25 platformi. Također, nudi i podršku za Wolfram matematičke funkcije što može biti korisno kod nekih složenijih izračuna [8]. Unity ima i dobru podršku za uvod objekata, tekstua i animacija iz programa kao što su Blender, 3DS Max, Maya, 4D Cinema i sl. [1]. Osim alata Unity, korišten je i alat GIMP za uređivanje grafičkih elemenata igre. Također, u svrhu izrađivanja skripti korišten je programski alata Visual Studio 2017.

Svrha ovog rada je kreiranje klona strateške računalne igre „Plants vs. Zombies“ u edukacijske svrhe: radi upoznavanja s procesom izrade računalnih igara.

„Plants vs. Zombies“ franšiza

Razvoj franšize započeo je prije više od 11 godina prvom igrom imena „Plants vs. Zombies“ [7] a franšizu je razvilo poduzeće PopCap Games [5]. Primarno, ovo poduzeće je razvijalo igre strateškog žanra, podžanra obrane tornjeva no s vremenom i u ovoj frašizi došle su igre i drugih žanrova: edukacijskog karaktera, kartaške i akcijske igre. Čak je razvijena i društvena igra *Plants vs. Zombies* [2].

Izvorna verzije igre je postala izuzetno popularna zbog svoje maštovite tematike. Čak su se određena iznenađenja (eng. *Easter Eggs*) u drugim igrama referencirala na ovu popularnu igru. Takva iznenađenja postoje u igrama *World of Warcraft*, *League of Legends*, *Hearthstone*, *The Sims* i dr. Osim toga, postoje i pojavljivanja u knjigama *The Simsons* i *Diary of a Minecraft Zombie*.

Osnovne mehanike igranja igre Plants vs. Zombies

Kao što je već rečeno, igra *Plants vs. Zombies* je igra strateškog tipa podžanra obrane tornjevima. Zadatak igrača je graditi biljke na travnjaku. Neprijatelji dolaze po travnjaku s desne strane prema lijevoj i pokušavaju doći do kuće iza travnjaka. Ako u tome uspiju, igrač gubi igru. Cilj igre je izgradnjom biljaka savladati nadolazeće valove zombija.

Postoje tri osnovne vrste biljaka koje su igraču na raspolaganju: biljke za stvaranje prihoda, biljke koje napadaju neprijatelje i obrambene biljke. Biljke koje stvaraju prihod u fiksnim intervalima stvaraju prihod (zrake sunca) koji se koristi kako bi se mogle saditi nove biljke. Kao i kod velike većine strateških igara, i u ovoj igri je bitno balansirati potrošnju prihoda na ekonomiju, napad i obranu. Biljke koje napadaju neprijatelje mogu se dalje podijeliti u tri osnovne kategorije: biljke koje zadaju štetu jednom neprijatelju, biljke koje imaju područje napadanja (napadaju sve neprijatelje u određenoj regiji) a postoje i jednokratne biljke koje nakon jednog napada nestaju. Efikasnost ofenzivne biljke mjeri se količinom štete koju može napraviti u jednoj sekundi (eng. *damage per second*) [3]. Defanzivne biljke služe kao prepreka koja zaustavlja kretanje neprijatelja po travnjaku. S obzirom da su defanzivne, one imaju veću količinu zdravlja od ostalih biljaka.

U kreiranom prototipu implementirana je jedna biljka koja stvara prihod (suncokret), 4 ofenzivne biljke (pucač graška, biljka žderačica, krumpir mina i trešnja bomba) i jedna defanzivna biljka (zid od oraha).

Što se tiče zombija, svi zombiji su ofenzivnog tipa. Razlikuju se prema količini životnih bodova, brzini kretanja i određeni zombiji imaju posebne sposobnosti. Kako bi igrač mogao prepoznati o kojoj vrsti neprijatelja se radi, zombiji se razlikuju po svom izgledu. U kreiranom prototipu implementirane su 4 vrste zombija: obični zombi, zombi s dodatnim štitom i zombi sa zastavom i zombi s motkom (posebna vještima: može preskočiti prvu biljku na koju naiđe na travnjaku).

Posljednje, u slučaju da je igrač postavio biljku na neku neželjenu poziciju ili se predomislio oko rasporeda biljaka, ima mogućnost „iskopati“ pojedinu biljku kako bi se na tom mjestu na travnjaku oslobodilo mjesto za izgradnju neke druge biljke.

Izrada prototipa igre

Prva scena koja se igraču pokazuje je izbornik. U izborniku igrač ima mogućnost igrati igru. Ako odabere tu mogućnost pojavljuje se dodatni izbornik gdje može odabrati razinu koju želi igrati. U glavnom izborniku također ima pristup određenim postavkama igre te mogućnost izlaska iz igre pritiskom na znak „Back“.

Slika br. 1: Izgled početnog izbornika igre

Početne razine su kreirane po uzoru na mnoge druge igre gdje se na pojedinoj razini uče osnovne mehanike igranja. Na prvoj razini igre dostupna je samo jedna „traka“ travnjaka te je igraču dostupno zasaditi samo jednu biljku a to je pucač graška. To je najosnovnija ofenzivna biljka koja ima mogućnost napadanja jednog zombija: onog koji se nalazi u istoj traci sa zasađenom biljkom i koji je najbliži biljci.

Također, u ovom nivou se igrač upoznaje s mehanikom skupljanja osnovnog resursa za gradnju biljaka a to su sunčeve zrake. U pravilnim odmacima, igraču odozgo prema dolje padaju sunčeve zrake u obliku malih sunaca. Klikom na sunce koje pada igrač skuplja zrake sunce. Nakon što skupi odgovarajuću količinu, može izgraditi biljku koja mu je na raspolaganju. Igrač se također poznaje s osnovnom mehanikom obrane. Na prvoj razini dolazi obični zombi kojeg igrač mora savladati.

Slika br. 2: Izgled prve razine igre

Na svakoj novoj razini igraču se otključavaju nove vrste biljaka, dolaze nove vrste zombija u drugačijim kompozicijama valova i igrač mora balansirati izgradnju biljaka kako bi imao dovoljno resursa za izgradnju dovoljne količine biljaka kako bi savladao nadolazeće neprijatelje. Na svakoj razini igraču je također prikazana informacija koliko zombija dolazi na toj razini.

Slika br. 3: Izgled kasnije razine

Izrada projekta u programskom alatu Unity

Razvoj računalne igre započinje kreiranjem novog projekta. Osnovne mehanike igre se implementiraju tako da kreiramo naše objekte igre i onda im pridružujemo odgovarajuće skripte kako bi se postigla željena mehanika koju želim ostvariti. Jedna od osnovnih mehanika u ovoj igri je mehanika pucanja biljaka na dolazeće zombije. Kako bismo kreirali biljku, potrebno je imati spremne grafičke elemente u obliku slika ili animacija koje će biti vidljivi dio biljke. U nastavku se nalazi skripta za pucanje biljaka na neprijatelje.

```
public void Shoot(){
   AudioSources[Random.Range(0, AudioSources.Lenght)].Play();
   Instantiate(Prefab, pos, Quaternion.identity);
}

void OnTriggerStay2D (Collider2D other){
   if(other.gameObject.tag == "Zombie"){
       anim.SetBool("shooting", true);
   }
}

void OnTriggerExit2D (Collider2D other){
   if (other.gameObject.tag == "Zombie"){
       anim.SetBool("shooting", false);
   }
}
```

Biljka na sebi ima okidač koji detektira koliziju s bilo kojim drugim objektom. Prilikom kolizije, funkcija *OnTriggerStay2D* provjerava oznaku objekta koji se nalazi u koliziji. Ako taj objekt ima oznaku *Zombie*, tada se aktivira animacija pucanja biljke. Kod kreiranja animacija u programskom alatu Unity na određenom okviru (eng. *frame*) moguće je kreirati određeni događaj. U odgovarajućem trenutku potrebno je ispucati zrno graška, tj. projektila koji će se instancirati i ići prema zombiju kojeg napada.

Grašak također na sebi ima okidač koji se aktivira prilikom sudara sa zombijem. Ako grašak udari zombija, tom zombiju s kojim je detektirana kolizija se oduzimaju njegovi životni bodovi. Nakon toga instanca projektila sama sebe uništava.

Što se tiče samih zombija, oni imaju dva osnovna stanja u kojima se mogu nalaziti. Prvo stanje je kretanje po travnjaku s desne strane prema lijevoj strani travnjaka. Prilikom te radnje se aktivira animacija hodanja. Kada zombi dođe u koliziju sa biljkom, tada prelazi u stanje napadanja biljke. Zombi biljku napada sve dok ju ne uništi. Ako je bilja uništena, zombi prelazi u stanje hodanja dok ne dođe no iduće biljke.

```
void DoDamage(){
    timer -= Time.deltaTime;
    if (timer <= 0){
        BiteSounds[Random.Range(@, BiteSounds.Length)].Play();
        go.GetComponent<PlantsHP>().SmanjiHP(damage);
        timer = hittingCooldown;
    }
}

void OnCollisionStay2D(Collision2D other){
    if (other.collider is CircleCollider2D &&
        other.gameQbject.tag == "Plant" && !hitting){
        animator.SetBool("biting", true);
        hitting = true;
        go = other.gameObject;
        InvokeRepeating("DoDamage", 0, .01667f);
    }
}

void OnCollisionExit2D(Collision2D other){
    if (other.gameObject.tag == "Plant"){
        hitting = false;
        animator.SetBool("biting", false);
        CancelInvoke();
    }
}
```

Kao što je vidljivo iz programskog koda, prilikom detekcije kolizije s objektom koji ima oznaku „Plant“ i ako zombi već nije u stanju napadanja, stanje se iz hodanja mijenja u „biting“, mijenja se animacija zombija te se izvršava funkcija zadavanja štete. Šteta se rad u odgovarajućim vremenskim razmacima te se ponavlja sve dok zombi ne izađe iz stanja napadanja.

Zaključak

Kroz ovaj rad napravljen je prototip igre Plants vs. Zombies, implementirano je prvih 10 razina izvorne verzije ove igre. Iako se radilo o razvoju klona postojeće igre, proces izrade računalnih igara je vrlo kreativan proces u kojem se programer susreće s brojnim izazovima. Mnoge funkcionalnosti se mogu implementirati na različite načine. Ovaj rad se lako može nadograditi s dodatnim razinama, dodatnim vrstama biljaka i dodatnim vrstama zombija. Programski alat Unity nudi odličnu podršku za razvoj igara ovog tipa i generalno gledano, putem ovog alata moguće je na vrlo intuitivan način implementirati osnovnu logiku računalne igre širokog spektra žanrova.

Reference

[1] A. McAloon: Unity 2017.2 brings Autodesk integration into the fold. (6. listopada 2020.). Dostupno na: https://www.gamasutra.com/view/news/307050/Unity_20172_brings_Autodesk_integration_into_the_fold.php

[2] E. Arneson: Is There a Plants vs. Zombies Bord Game? (6. listopada 2020.). Dostupno na: https://www.thesprucecrafts.com/plants-vs-zombies-the-board-game-412552
[3] Fandom: Tower defense. (6. listopada 2020.). Dostupno na: https://plantsvszombies.fandom.com/wiki/Tower_defense
[4] GameDesigning: The Top 10 Video Game Engines. (5. listopada 2020.). Dostupno na: https://www.gamedesigning.org/career/video-game-engines/
[5] PopCap Studios. (6. listopada 2020.). Dostupno na: https://www.ea.com/ea-studios/popcap
[6] Unity. (5. listopada 2020.). Dostupno na: https://unity.com/
[7] Wikipedia: Plants vs. Zombies. (6. listopada 2020.). Dostupno na: https://en.wikipedia.org/wiki/Plants_vs._Zombies
[8] Wolfram. (5. listopada 2020.). Dostupno na: https://www.wolfram.com/language/12/built-in-interface-to-unity-game-engine/

Implementacija osnovnih mehanika top down shooter igre u programskom alatu Unity

Josip Bezi i Mladen Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
Josip.bezi@gmail.com, mlkoneck@foi.hr

Sažetak

U ovom radu su opisane osnovne mehanike implementacije top down shooter igre u programskom alatu Unity. U radu će biti objašnjene mehanike kretanja, pucanja, upravljanje životnim bodovima, detekcija kolizije i logika dolaska neprijatelja u valovima.

Ključne riječi: računalna igra, Unity, top down shooter, shoot 'em up, C#, mehanike igranja, programiranje, algoritmi

Uvod

Top down shooter igre su igre specifičnog tipa koje bi se po svojoj vrsti mogle prikloniti žanru shoot 'em up, podžanru igara pucanja unutar žanra akcijskih igara [5]. U igrama ovog tipa igrač se nalazi u ulozi protagonista koji se bori protiv velike količine različitih neprijatelja. Igrač mora savladati neprijatelje koji ga napadaju dok u isto vrijeme mora izbjegavati kontakt s njima i njihovu paljbu. U klasičnoj izvedbi ovaj žanr je u 2D ili 2.5D izvedbi gdje neprijatelji dolaze odozgo prema dolje (ili ponekad s desne prema lijevoj strani). U ovom radu kreirana je 3D izvedba igre u kojoj neprijatelji dolaze sa svih strana i napadaju igrača.

Povijesni razvoj ovog žanra seže do samih početaka razvoja računalnih igara. Prva igra ovog tipa razvijena je davne 1961. godine gdje su studenti M.I.T.-a na prvom korisnički-orijentiranom računalu, PDP-1, kreirali igru Spacewar! [1]. Dvadesetak godina kasnije razvijena je legendarna igra Space Invaders koja je bila nevjerojatno popularna. Popularnost žanra se održala i do danas te se i dalje razvijaju i igraju igre ovog tipa. Neki najpopularniji naslov današnjice na osobnim računalima su *Enter the Gungeron*, *Jamestown*, *Nex Machina* i druge [2].

Razvoj igre Sleepy Hero

U nastavku slijedi opis razvoja prototipa igre pucanja odozgo (eng. *top down shooter*) pod imenom *Sleepy Hero*.

U igri igrač se nalazi u ulozi glavnog protagonista koji se nalazi u šumi u kojoj ga napadaju plišani zombiji životinja. Cilj igrača je preživjeti napade neprijatelja koji dolaze u valovima. Svaki val je naravno sve teži: u svakom valu dolazi sve više neprijatelja koji su ujedno i sve jači i imaju sve više života. Što se tiče osnovnih mehanika igranja, igrač se može kretati slobodno kroz prostor u svim smjerovima a što se tiče napada na raspolaganju su mu 3 različite vrste oružja i bomba koja ima razorni efekt u određenom radijusu.

Slika br. 1: Prikaz igrača i okoline igre Sleepy Hero

Što se tiče sučelja, u donjem lijevom kutu ekrana je moguće vidjeti stanje životnih bodova igrača, u gornjem lijevom kutu igra može vidjeti koliko mu je bombi preostalo a s desne strane vidi se trenutno odabrano oružje. U sredini s gornje strane vidi se informacija koji val neprijatelja je trenutno aktivan. Što se tiče okoline u kojoj se igrač nalazi, igrač se može slobodno kretati kroz prostor. U šumi postoje određene prepreke (npr. stablo, kamen) a također postoje i prepreke koje igrač mora izbjegavati (npr. bodlje). U slučaju da igrač stane na zamku izgubit će određenu količinu životnih bodova. U okolini se nalaze i mjesta gdje igrač može regenerirati životne bodove prelaskom preko odgovarajućih polja.

Igrača napadaju 3 vrste neprijatelja a to su: plišani medvjed (plave boje), plišani zec (ljubičaste bolje) i plišani slon (žute boje). Svaki neprijatelj ima svoju razinu životnih bodova i nanosi odgovarajuću razinu štete. Svaki od ovih neprijatelja dolazi u dva oblika: mali i veliki neprijatelj. Velika verzija neprijatelja ima više životnih bodova i radi veću štetu igraču od male verzije neprijatelja.

Osnovna struktura projekta u programskom alatu Unity

Osnovna struktura svake igre koja se radi u programskom alatu Unity je da se igra dijeli u odgovarajuće scene. To su logički odvojeni dijelovi igre. U projektu su korištene svije scene: jedna za početni izbornik a druga za igranje same igre. Svaka scena ima svoje osnovne objekte igre (eng. *GameObject*) [3]. *GameObject* je temeljni objekt i putem njega se stvaraju likovi, okolina i ostali elementi igre. Primjerice, gumb na početnom ekranu je *GameObject* koji ima pridruženu skriptu *UIButton* i upravitelj događajima. Pritiskom na gumb, upravitelj događaja interpretira sami događaj i poziva odgovarajuću metodu iz skripte.

Kretanje igrača

Kretanje igrača je implementirano pomoću komponente *RigidBody* koja objektu kojem je ta komponenta pridružena daje osnovna fizikalna svojstva [4]. Ta komponenta se koristi i kod igrača i kod neprijatelja kako bi se implementirala mogućnost kretanja tih entiteta. Što se tiče samog igrača, u metodi *Update()* koja se izvodi prilikom učitavanja svake sličice (eng. *frame*) igre ažurira se pozicija igrača na temelju pritisnutih tipki za kretanje (zadane tipke su WASD i strelice na tipkovnici).

U metodi *Update()* dohvaćaju se osi kretanja te se vrijednosti spremaju u varijable.

```
float moveHorizontal = Input.GetAxisRaw("Horizontal");
float moveVertical = Input.GetAxisRaw("Vertical");
```

U metodi *Move()* se kreira vektor smjera kretanja koji tada pomiče objekt igrača u zadanom smjeru putem *RigidBody* komponente.

```
void Move(float h, float v) {
   movement.Set(h, 0f, v);
   movement = movement.normalized * MovementSpeed * Time.deltaTime;
   rb.MovePosition(transform.position + movement);
}
```

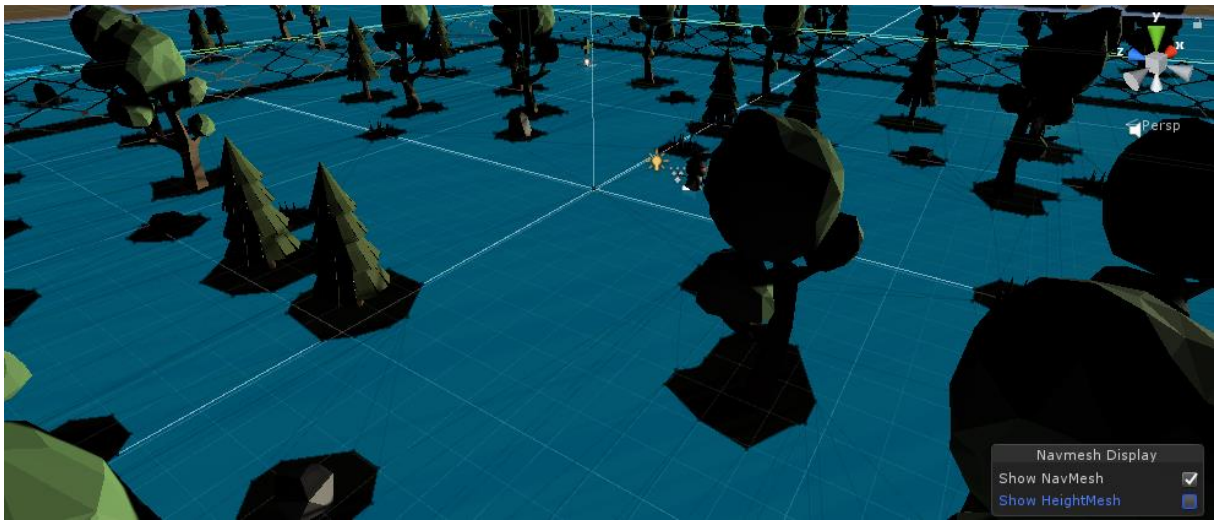
Osim pomicanja igrača je potrebno i rotirati igrača prema poziciji miša jer je to smjer u kojem želimo da igrač bude okrenut kako bi igrač mogao u smjeru miša pucati.

```
void Turning() {
   Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
   RaycastHit floorHit;
   if (Physics.Raycast(camRay, out floorHit, camRayLength, floorMask)) {
      Vector3 playerToMouse = floorHit.point - transform.position;
```

```
        playerToMouse.y = 0f;
        Quaternion newRotatation = Quaternion.LookRotation(playerToMouse);
        rb.MoveRotation(newRotatation);
    }
}
```

Kretanje neprijatelja

Za implementaciju kretanja neprijatelja prema igraču koriste se odgovarajuće ugrađene komponente programskog alata Unity za definiranje podloge kretanja i prohodnosti na toj podlozi. Na temelju definirane podloge hodanja radi se izračun traženja najkraćeg puta od pozicije na kojoj se pojedini neprijatelj nalazi i pozicije igrača. Za ovu funkcionalnost potrebno je koristiti *NavMesh* komponentu na okolini igre. Nakon što se ona odredi (eng. *bake*) moguće je dobiti željenu funkcionalnost. Na slici br. 2 vidljiva je ta definirana podloga kretanja plavom bojom.



Slika br. 2: NavMesh mape

Kako bi se omogućilo kretanje neprijatelja, uz *RigidBody* komponentu neprijatelji imaju i *NavMeshAgent* komponentu. Ta komponenta je zadužena za pomicanje objekta od trenutne pozicije do ciljane točke, tj. traženje najkraćeg validnog puta (izbjegavanje prepreka).

U metodi Update() koja se izvodi svaku sličicu iscrtavanja igre ažurira se željeno odredište kretanja pojedinog neprijatelja a to je pozicija igrača.

```
void Update() {
    if (!(enemyHealth.CurrentHealth <= 0)) {
        nav.SetDestination(player.position);
    }
}
```

Implementacija mehanika napadanja

Kako bi igrač savladao neprijatelje koji mu se približavaju, mora im oduzeti životne bodove pucanjem u njih. Kako bi se odredilo koji neprijatelj je pogođen, koristi se *RayCast* metoda. Metoda *Shoot()* provjerava je li zraka pogodila neku odgovarajuću metu. Svi neprijatelji se nalaze u odgovarajućem sloju i na taj način su podložni tome da budu pogođeni. Metodom *TakeDamage()* se smanjuju životni bodovi pogođenom neprijatelju.

```
void Shoot() {
   timer = 0f;
   gunAudio.Play();
   gunLight.enabled = true;
   faceLight.enabled = true;
   gunLine.enabled = true;
   gunLine.SetPosition(0, transform.position);
   shootRay.origin = transform.position;
   shootRay.direction = transform.forward;
   if (Physics.Raycast(shootRay, out shootHit, range, shootableMask)) {
      EnemyMovement enemyMovement =
      shootHit.collider.GetComponent<EnemyMovement>();

      if (enemyMovement != null) {
         enemyMovement.TakeDemage(damagePerShot);
      }
   gunLine.SetPosition(1, shootHit.point);
   }
   else {
      gunLine.SetPosition(1, shootRay.origin + shootRay.direction * range);
   }
}
```

Neprijatelj može napasti igrača ako se nađe u njegovoj neposrednoj blizini. Ako se igrač nađe na odgovarajućoj udaljenosti i ako je prošlo odgovarajuće vrijeme od eventualnog prijašnjeg napada, neprijatelj će napasti igrača.

```
void Update() {
   timer += Time.deltaTime;
   if (timer >= timeBetweenAttacks && playerInRange) {
      Attack();
   }
}

void OnTriggerEnter(Collider other) {
   if (other.gameObject == player) {
      playerInRange = true;
   }
}
```

```
void OnTriggerExit(Collider other) {
   if (other.gameObject == player) {
      playerInRange = false;
   }
}

void Attack() {
   timer = 0f;
   player.GetComponent<PlayerMovement>().TakeDeamge(attackDamage);
}
```

Igrač i svi neprijatelji imaju komponentu *Health* koja ima dvije metode za oduzimanje i povećavanje životnih bodova te dohvaćanje trenutne vrijednosti životnih bodova.

Razine i stvaranje neprijatelja

Skripta EnemyManager je skripta koja se odvija u pozadini i zadužena je za upravljanje valovima neprijatelja. U slučaju da je igrač eliminirao sve neprijatelje na pojedinoj razini učitava se iduća razina (ili val neprijatelja). Svaka razina ima definiranu vrstu i broj neprijatelja koji se na toj razini stvaraju.

```
void Update()  {
   if (spawnNextWave) {
      switch (wave) {
         case 1:
            spawnNextWave = false;
            SpawnSmallerSmallEnemyManager(15);
            break;
         case 2:
            spawnNextWave = false;
            SpawnSmallerSmallEnemyManager(15);
            SpawnSmallerMediumEnemyManager(5);
            break;
            . . .
         case 10:
            SpawnSmallerSmallEnemyManager(75);
            SpawnSmallerMediumEnemyManager(20);
            SpawnSmallerBigEnemyManager(10);
            SpawnSmallEnemyManager(25);
            SpawnMediumEnemyManager(25);
            SpawnBigEnemyManager(20);
            break;
         default:
            break;

      }
   }
   if (wave == 11) {
      SceneManager.LoadScene("WinGameScene");
```

```
    }
    CheckEnemies();
}
```

Zaključak

Programski alat Unity je pokretač za razvoj računalnih igara koji ima mnogo korisnih komponenti koje mogu uvelike olakšati razvoj prototipova igara a naravno onda i finalnih proizvoda igara. Mnoge funkcionalnosti koje su u ovom prototipu implementirane se oslanjaju na gotove komponente koje omogućavaju da se na jednostavan način dođe do željenih funkcionalnosti: kretanje na temelju fizikalnih svojstava, traženje najkraćeg puta, detekcija kolizije i sl. Također od velike pomoći mogu biti i brojni resursi koji su dostupni svakome tko želi razvijati svoju igru u programskom alatu Unity.

Reference

[1] C. J. Barrish: A Detailed History of Shoot ‘Em Up Arcade Games (Liberty Games Blog). (10. veljače 2020.) Dostupno na: https://www.libertygames.co.uk/blog/a-detailed-history-of-shoot-em-up-arcade-games/

[2] Slant: What are the best Shoot 'Em Up games on Steam? (11. veljače 2020.). Dostupno na: https://www.slant.co/topics/6383/~shoot-em-up-games-on-steam

[3] Unity Documentation: GameObject (17. veljače 2020.). Dostupno na: https://docs.unity3d.com/ScriptReference/GameObject.html

[4] Unity Documentation: RigidBody (17. veljače 2020.). Dostupno na: https://docs.unity3d.com/ScriptReference/Rigidbody.html

[5] Wikipedia: Shoot 'em up. (18. rujna 2020). Dostupno na: https://en.wikipedia.org/wiki/Shoot_%27em_up

Izrada računalne igre Biljar u programskom alatu Unity

Karlo Nenadić, Danijel Radošević i Mladen Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
knenadic@foi.hr, darados@foi.hr, mlkoneck@foi.hr

Sažetak

U ovom radu je opisan proces izrade računalne igre biljar za dva igrača u programskom alatu Unity. Modeli korišteni u kreiranoj igri napravljeni su u programskom alatu Blender. U radu su opisane osnovne mehanike igre i implementacija tih mehanika.

Ključne riječi: biljar, računalna igra, Unity, 3D, C#

Uvod

Industrija računalnih igara je jedna od najvećih industrija na svijetu. Predviđa se da će do 2025. godine tržište doseći vrijednost od 256 milijardi dolara. Danas više od 2,5 milijarde ljudi igra računalne igre dok eSport publika broji više od 450 milijuna ljudi [2]. Razvoj računalnih igara nikada nije bio pristupačniji zbog mnogih alata i pogona (eng. *engine*) koji omogućavaju brži razvoj igara. Ne samo da alati omogućavaju brži razvoj već mnogi od tih alata su besplatni što zbilja omogućava svakome da se počne baviti ovim područjem [3].

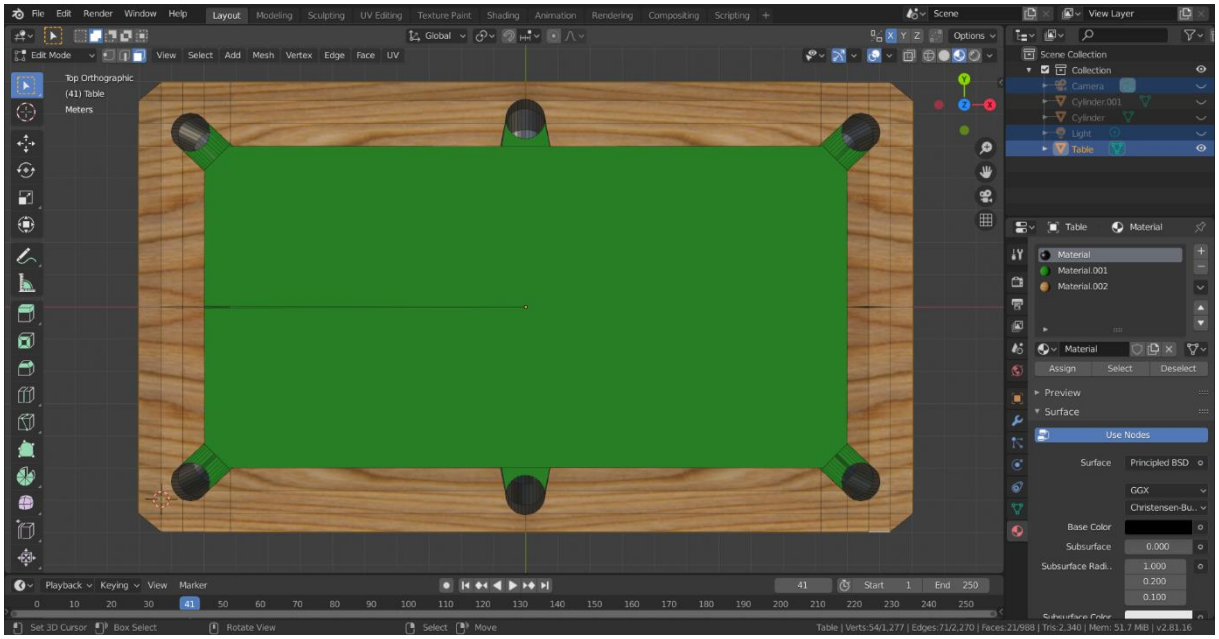
Igra biljara je podžanr igara sportskih simulacija [6]. U ovom radu opisat će se proces izrade igre biljara za dva igrača. Igra će biti napravljena putem programskog alata Unity [4], jedan od najpoznatijih besplatnih pogona igara [5]. Mnogo je lakše izraditi računalnu igru simulacije biljara koristeći već gotov pogon poput Unityja jer su mnoge potrebne stvari implementirane: fizika, upravljanje kamerama, detekcija kolizije, izrada grafičkog sučelja i sl. Igra će biti napravljena za Windows platformu i igrat će se mišem.

Za izradu potrebnih grafičkih elemenata igre korišten je programski alat Blender [1]. U njemu će biti izrađeni 3D modeli potrebnih elemenata za izradu igre: biljarsko stol, štap i kuglice. Osim 3D modela biti će potrebno izraditi grafičke elemente grafičkog sučelja igre. Sam projekt biti će strukturiran u dvije scene u programskom alatu Unity. Prva scena biti će scena izbornika putem kojeg će igrač moći započeti igrati novu igru, nastaviti igrati postojeću

igru, podesiti određene postavke vezano uz igru te izaći iz igre. Druga scena biti će scena same igre gdje će igrač moći igrati simulaciju igre biljara.

Modeliranje

U prvoj fazi izrade igre kreirani su potrebni 3D modeli za igru. Objekti koje je potrebno izraditi su relativno jednostavni za izmodelirati. Stol je napravljen tako da je kreirana ploha koja ima svoj rub te je rub uzdignut u odnosu na plohu. Nakon toga je kreiran cilindar u obliku rupe za kuglice. Korištenjem modifikatora *boolean* su napravljene rupe u stolu na odgovarajućim pozicijama. Dno je zaobljeno kako bi se dobio dojam spajanja sa stolom. Nakon modeliranja na objekt su dodane odgovarajuće teksture: tekstura drveta na rub stola i zelena boja na podlogu stola. Konačan model stola vidljiv je na slici br. 1.



Slika br. 1: Konačan model stola u alatu Blender

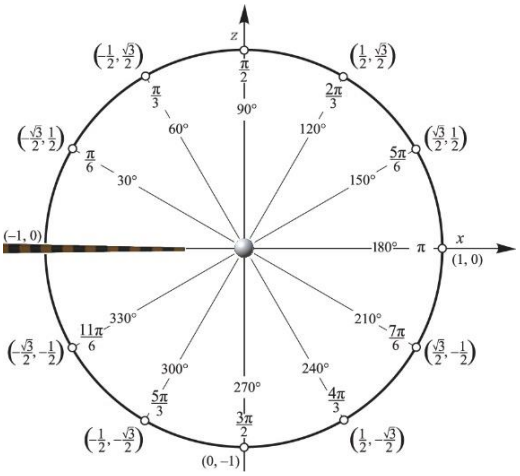
Štap za igranje je izrađen tako što je napravljen jednostavan cilindar odgovarajuće dužine koji je nešto malo manjeg promjera na jednom kraju od drugog. Same biljarske kuglice su jednostavne kugle na koje je potrebno samo staviti odgovarajuće teksture.

Logika igre

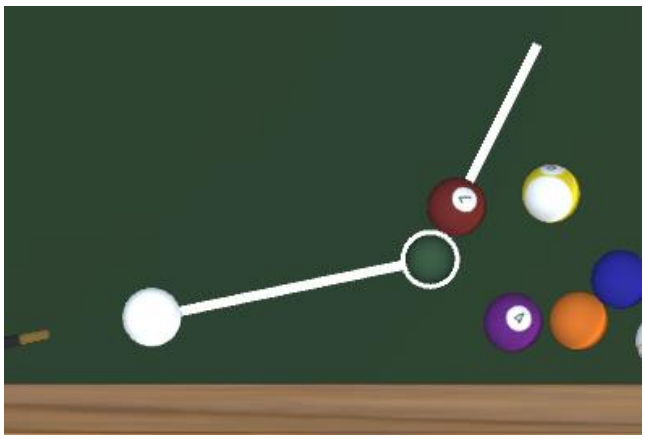
U klasičnom biljaru za dva igrača, igrači igraju naizmjenično. U slučaju da igrač pogodi kuglicu u rupu, igrač igra ponovno. Ako u nekom pokušaju ne ubaci svoju kuglicu u rupu, tada igra drugi igrač. Ako igrač pogodi ili ubaci tuđu (ili bijelu) kuglicu, protivnički igrač ima mogućnost postavljanja bijele kuglice na željenu poziciju. Igrač igra tako što štapom udara direktno u bijelu kuglicu te

pokušava odbijanjem pogoditi svoje kuglice u rupu. U igri postoje dvije vrste kuglica: „pune“ i „prazne“. Odabir kuglica se određuje u prvim potezima igre. Prvi igrač koji pogodi neku od vrsta, to je njegova vrsta kuglica koje treba pogađati u rupe. Posebna kuglica je crna kuglica koja je zajednička i koja se mora spremiti u rupu posljednja. U slučaju da igrač crnu kuglicu pospremi u rupu prije nego što je pospremio sve svoje druge kuglice, automatski gubi igru.

Sama mehanika igranja se oslanja na fizikalna svojstva. Kako bi se osiguralo korektno izvođenje igre, koriste se komponente za detekciju sudara (eng. *Collider*) i komponenta krutih tijela (eng. *Rigidbody*). S obzirom da igrač štapom udara bijelu kuglicu, pozicija štapa se postavlja tako da je centar rotacije (eng. *pivot*) na poziciji bijele kuglice. Na taj način se štap rotira oko bijele kuglice. Smjer u koji štap pokazuje dobiva se kao vektor kojem je x koordinata kosinus kuta rotacije pivota štapa, z koordinata sinus kuta rotacije pivota štapa a y koordinata je nula. Odnos je prikazan na slici br. 2. Jednom kada imamo određen smjer kretanja kuglice, sve što trebamo kako bismo dobili konačni vektor je odrediti jačinu udarca. Jačina se određuje dužinom pritiska lijeve tipke miša.



Slika br. 2: Brojevna kružnica



Slika br. 3: Pomoćne linije i kugla

Standardna mogućnost u simulacijama biljara je i vizualni prikaz smjera u kojem će se bijela kuglica ispucati i koja je to dodirna točka bijele kuglice s prvom kuglicom koja se nalazi na smjeru kretanja bijele kuglice. Također je čest slučaj iscrtavanje smjera kretanja kuglice koju bijela kuglica pogađa, kako bi se igraču olakšalo spremanje te kuglice u rupu. Naravno, iscrtava se samo neposredna udaljenost kretanja inače bi igra bila prejednostavna. Slika br. 3 prikazuje spomenuta iscrtavanja. Kako bismo dobili tu funkcionalnost, koristimo funkciju koja ispucava sferu veličine kuglice u smjeru kretanja bijele kuglice. Putem te sfere se detektira mjesto sudara bijele kuglice s prvom kuglicom na putanji. Time dobivamo mjesto udara kojem je dovoljno dodati normalu udarca pomnoženu s promjerom bijele kuglice kako bismo dobili

središte pomoćne kuglice koju trebamo iscrtati. Tada komponenti za prikaz bijele linije (eng. *Line Renderer*) početak iscrtavanja postavimo na poziciji bijele kuglice a kraj na poziciji pomoćne sfere. Za iscrtavanje druge pomoćne linije koristi se pozicija sfere i središte udarene kuglice. Ovisno o jačini udarca, mijenja se i dužina druge pomoćne linije.

```
Ray ray = new Ray(start, direction);
if (Physics.SphereCast(ray, radius, out RaycastHit raycastHit)) {
      end = raycastHit.point + radius * raycastHit.normal;
      }

Vector3 endLine = raycastHit.point - raycastHit.normal *
Vector3.Angle(raycastHit.normal, end - start) / 150f;
```

Još jedan zanimljiv detalj je početno postavljanje svih kuglica koje se slažu u obliku trokuta. Kako bi se simulirala nesavršenost početnog postavljanja, početna pozicija kuglica se randomizira za mali određeni pomak. Na taj način se dobiva još jedan element slučajnosti koji će utjecati na početno razbijanje kako bi ono bilo dodatno nasumično.

Implementacija većine ostalih pravila vezanih za praćenje stanja igre su relativno jednostavna. Kao što je već istaknuto u pravilima, tip kuglica određenog igrača se određuje na način da prva ubačena kuglica tokom igre postaje „boja“ kuglica igrača koji je na potezu. Postoji nekoliko načina na koje igrač može napraviti prekršaj što će rezultirati time da protivnički igrač u sljedećem potezu može bijelu kuglicu staviti bilo gdje na stolu po želji. To su sljedeće situacije: ako bijela kuglica ne dodirne niti jednu kuglicu, ako bijela kuglica dodirne prvo protivničku kuglicu i ako bijela kuglica dodirne prvo crnu kuglicu. Standardno, ako igrač pogodi barem jednu kuglicu u rupu, igra ponovno, a ako ne uspije ugurati niti jednu kuglicu u rupu, tada je na potezu protivnički igrač.

Kraj igre može doći standardno ako igrač ugura u rupe sve svoje kuglice i na kraju crnu kuglicu. Također se smatra prekršajem ako igrač prilikom ubacivanja crne kuglice također ubaci i bijelu kuglicu. U tom slučaju je igrač izgubio. Jednako tako ako igrač prijevremeno ubaci crnu kuglicu prije ostalih kuglica, izgubio je. Sve ovo su relativno jednostavno provjere koje treba napraviti kako bi dobro funkcionirala logika igre. tj. kako bi se uvažila sva pravila igre. Status igre je moguće pratiti kroz kreirano grafičko sučelje na kojem se nalazi oznaka koji igrač je trenutno na potezu, imena igrača te koliko kuglica je pojedini igrač uspio pospremiti u rupe. Na taj način je jednostavnije pratiti stanje igre.

Dodatne opcije

Osim same igre, igrač ima i još jeke dodatne mogućnosti moje može podesiti vezano za samu igru. Igrač ima mogućnost prikazivanja ili skrivanja pomoćne linije, može mijenjati boju pomoćne linije, modificirati dužinu pomoćne linije, može promijeniti teksturu izgleda stola i podesiti glasnoću zvukova u igri. Na samom kraju igre kreirana je i animacija konfeta prilikom proglašenja pobjednika igre a ona je napravljena putem sustava čestica. Završni ekran je vidljiv na slici br. 4.

Slika br. 4: Animacija za kraj igre

Zaključak

U ovom radu je ukratko opisan proces izrade igre simulacije biljara u programskom alatu Unity. Opisan je način kako su kreirani potrebni resursi, opisana su pravila igre i način implementacije pojedinih segmenata mehanika igranja. Kako bi se implementirala ovakva igra potrebno je znati osnove rada u programskom alatu Unity, potrebno je znati programirati u programskom jeziku C# i potrebno je određeno znanje iz područja matematike i fizike.

Reference

[1] Blender. (13. rujna 2020.). Dostupno na: https://www.blender.org/
[2] Dobrilova, T. How Much Is the Gaming Industry Worth in 2020? (13. rujna 2020.). Dostupno na: https://techjury.net/blog/gaming-industry-worth/

[3] Petty, J. Top 12 Free Game Engines For Beginners & Experts Alike. (13. rujna 2020.). Dostupno na: https://conceptartempire.com/free-game-engines/
[4] Unity. (8. rujna 2020.). Dostupno na: https://unity.com/
[5] Wikipedia: Game engine. (8. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Game_engine
[6] Wikipedia: Sports game. (13. rujna 2020.) Dostupno na: https://en.wikipedia.org/wiki/Sports_game

Alati i platforme za razvoj računalnih igara

Fran Šimović i Mario Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
fsimovic@foi.hr, mario.konecki@foi.hr

Sažetak

U ovom radu obradit će se tema alata i platformi koje se koriste pri izradi računalnih igara. U početku rada će se proći kroz samu povijest računalnih igra, odnosno ukratko će se prikazati njihov napredak do danas, kao i trendovi koji su bili aktualni u raznim razdobljima. Zatim, nakon povijesti računalnih igara, u radu će se prikazati povijesni ciklus alata i platformi te sami sustav izrade računalnih igara kroz nekoliko desetljeća. Glavni dio rada obuhvatit će detaljniji prikaz nekoliko različitih alata i platformi i nekih od najuspješnijih računalnih igara koje su izrađene pomoću ovih alata i platformi.

Ključne riječi: Računalna igara, vrste računalnih igara, povijesni ciklus alata i platformi za razvoj računalnih igara, arhitektura alata, game engine, video igra, Unity, Unreal Engine, Frostbite

Uvod

Početak razvoja industrije videoigara seže još u sredinu 20. stoljeća kada je inženjer Josef Kates izgradio prvo računalo „Bertie the Brain“ na kojem je bilo mogući igrati videoigru „tic-tac-toe“, odnosno križić-kružić [1]. Taj prvi korak, izgradnja računala za vrlo jednostavnu igru, bio je izrazito kompleksan što je rezultiralo jednim početkom potpuno nove industrije, nepoznate tadašnjem svijetu. Sam napredak industrije videoigara uvelike ovisi o napretku tehnološke industrije, stoga je bilo potrebno dosta godina kako bi se počele razvijati kompleksnije igre namijenjene različitim skupinama.

Do 90-ih godina 20. stoljeća poduzeća koja su razvijala računalne igre su ujedno razvijala i računala na kojima su se ove igre igrale, poput poduzeća Atari koje je 1971. godine razvilo svoju prvu igru „Computer Space“. Napretkom tehnologije, aktualna industrija videoigara bavi se isključivo razvojem igara, no svejedno ovisi o razvoju računalne tehnologije.

Većina današnjih alata je namijenjena većim poduzećima koja imaju velike razvojne timove, no postoje i alati koje mogu koristiti i privatne osobe, za zabavu ili komercijalne svrhe uz poseban oblik naplate [2].

Pojam „Game engine“

Generalno koncept „Game engine“, odnosno alati i platforme za razvoj računalnih igara, je prilično jednostavan za razumjeti. To je softver koji izvršava uobičajene zadatke potrebe za izradu videoigre, poput renderiranja, raznih izračuna povezanih s fizikom, a sve kako bi se razvojni programeri mogli fokusirati na razvoj detalja koji videoigru čine jedinstvenom u odnosu na druge igre. Ti alati zajedno čine kolekciju komponenata kojima se može manipulirati kako bi se videoigra realizirala. [3]

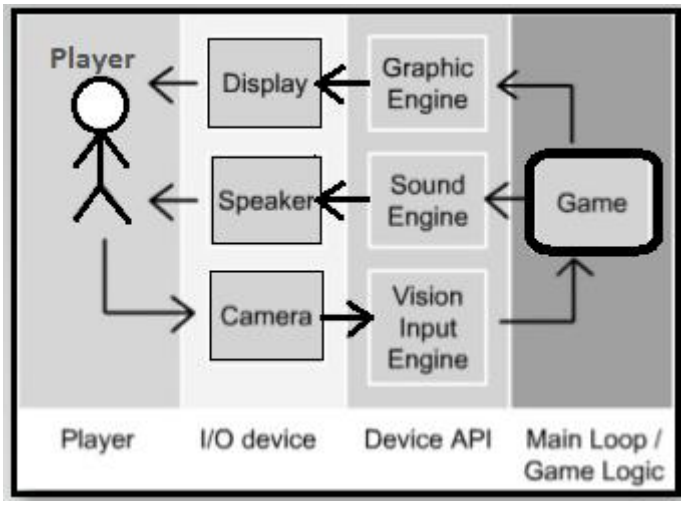
Alati i platforme za razvoj računalnih igra se zapravo ponašaju kao „middleware“. Middleware je softver koji se nalazi između operacijskog sustava i aplikacije koja se pokreće na njemu. U osnovi on djeluje kao skriveni sloj, odnosno srednji sloj koji omogućava komunikaciju i upravljanje podacima za aplikacije. [4]

Iz tog razloga, pomoću takvih alata je moguće provesti apstrakciju, što omogućava prilagodbu videoigara na različitim platformama, poput igraćih konzola uz manju promjenu izvornog koda. Općenito su ti alati izrađeni od nekoliko dodatnih middleware-a, npr. „Havok“ koji je zadužen za izračune vezane uz fiziku, „Miles Sound System“ za zvuk, „Blink“ za video. [3]

Arhitektura alata

Danas se svaka standardna videoigra sastoji se od niza razina koje su ugrađene i usko povezane s pričom videoigre, akcijskim i avanturističkim sekvencama, spektakularnim vizualnim doživljajima, koji naposljetku predstavljaju izazov igračevim mentalnim sposobnostima. Kako bi videoigra bila što zanimljivija, modularnija, pa tako i lakša za razvoj, općeprihvaćena je bazna arhitektura svakog današnjeg alata za njihov razvoj.

Arhitektura alata za razvoj videoigara je vrlo slična arhitekturi svakog klasičnog softvera, uz neke specifičnosti. Svaki od alata za razvoj videoigara ima „main loop“ koji sadržava logiku igre, zatim alate za grafiku, zvuk, renderiranje slike i videa, ulazno izlazne uređaje koje korisnik koristi i DDL datoteke i aplikacijska sučelja. Sve navedene komponente u zajedničkoj interakciji omogućavaju razvoj videoigre u punom opsegu. [5]



Slika br. 1: Interakcija komponenti alata za razvoj videoigara [5]

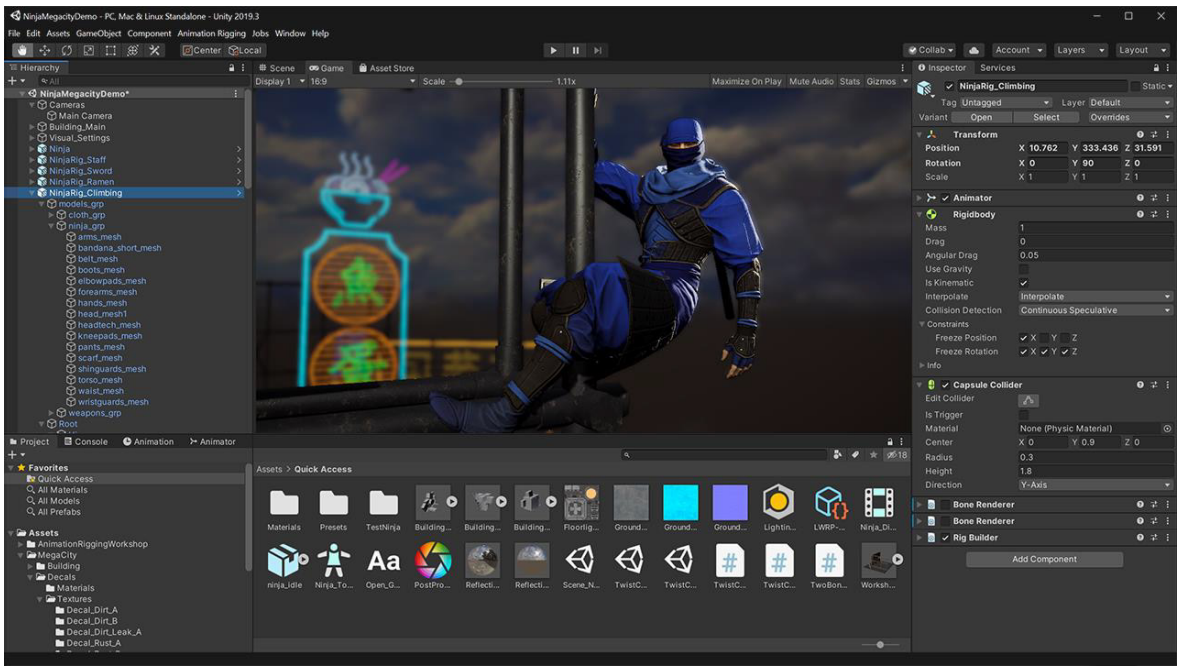
Unity

Unity je višeplatformski „game engine“ za videoigre razvijen od strane poduzeća Unity Technologies, a prvi put je najavljen i objavljen u lipnju 2005. na Apple Inc.-ovoj svjetskoj konferenciji za razvojne programere kao ekskluzivni Mac OS X „engine“ za videoigre. Od 2018. alat je proširen kako bi podržao veliki broj današnjih platformi. Alat se može koristiti za stvaranje trodimenzionalnih, dvodimenzionalnih igara, virtualne stvarnosti i proširene stvarnosti, kao i simulacija i drugih iskustava. Alat je također usvojen i izvan industrije videoigara te ga koriste industrije filma, automobila, arhitekture, inženjerstva i građevine. [6]

Unity omogućava svojim korisnicima razvoj videoigara u 2D i 3D okruženje te je u svrhu toga u svojim počecima podržavao razvijanje videoigara pomoću programskog jezika Boo, a za videoigre koje su se pokretale u web pregledniku podržavao je verziju JavaScript-a nazvanu UnityScript. Dolaskom Unity-ja 5.0, programski jezik Boo prestaje biti primarni jezik razvoja novih videoigara iako je još uvijek bilo mogućnosti za njegovu uporabu za starije videoigre, a izlaskom verzije Unity-ja 2017.1 UnityScript se prestaje koristi za razvoj videoigara u web pregledniku. Funkciju oba programska jezika, Boo i UnityScript, preuzeo je Microsoft-ov C# [7] [8].

Alat Unity podržan je na sustavima Windows i MacOS, s verzijom dostupnom za Linux platformu, iako još u eksperimentalnoj fazi, dok sam alat trenutno podržava izgradnju videoigara za više od 25 različitih platformi. Pored svih popularnih mobilnih i računalnih platformi podržava i razvoj videoigara za platforme virtualne i proširene stvarnosti poput PlayStationVR, Olocus, ARCore, Microsoft HoloLens i Magic Leap , te razvijanje softvera za platforme poput AndroidTV i tvOS. [9]

Više od dvije trećine svih sadržaja stvorenih za proširenu i virtualnu stvarnost izrađeno je alatom Unity. 91% aplikacija na novim platformama za proširenu stvarnost poput Microsofta HoloLens stvoreno je pomoću Unity-ja, dok Samsungov Gear VR ima oko 90% svojih igračkih naslova izrađenih pomoću Unity-ja. Iako su „AR“ i „VR“ tehnologije nevjerojatno mlade, poduzeća koja se bore za dominaciju nad platformama ostvaruju izvrsne rezultate. Zbog široke podrške alata za različite platforme, korisnicima se omogućava istovremeni rad na više njih istovremeno što dodatno potvrđuje raznolikost Unity-ja [10].



Slika br. 2: Izgled sučelja Unity 5.0 [11]

Unreal Engine

Unreal Engine je razvijen 1998. godine od strane poduzeća Epic Games, zajedno s akcijskom igrom u prvom licu pod imenom „Unreal“. Ova inačica Unreal Engine uključivala je brojne sustave, uključujući prikazivanje slike, otkrivanje kolizija, umjetnu inteligenciju, vidljivost, umrežavanje, skriptiranje i upravljanje datotekama. „Unreal Tournament“ pratio je „Unreal“ i napravio je velike korake u poboljšanju prikazivanja slike i mrežnih performansi. Unreal Engine postao je popularan uglavnom zbog modularnog dizajna arhitekture alata i uključivanja vlastitog skriptnog jezika nazvanog UnrealScript. Iako je u velikoj mjeri zasnovan na C++-u, korisnicima je omogućeno lako stvaranje izmjena pomoću UnrealScript-a.

UnrealScript je izvorni skriptni jezik Unreal Engine-a koji se koristio za izradu koda videoigre i igranja događaja prije izlaska Unreal Engine-a 4. Jezik je dizajniran za jednostavno programiranje videoigara na visokoj razini. Jezik je u potpunosti izgrađen na objektno orijentiranim principima i u sintaksi podsjeća na Javu ili C++. Sučelja su bila podržana samo u Unreal Engine generaciji 3 i nekoliko videoigara Unreal Engine-a 2. Na konferenciji programera videoigara 2012. Epic je objavio da se UnrealScript uklanja iz Unreal Engine-a 4 i da se počinje u potpunosti koristiti C++ [12].

Danas Unreal Engine 4 pored računalne, konzolne i mobilne platforme podržava i razvoj videoigara i softvera umjetne, virtualne i mješovite stvarnosti. Neke od danas podržanih platformi su MS Windows, macOS, Linux, HTML5, iOS, Android, PlayStation, Xbox, Magic Leap One, Oculus Roft, Google Daydream, Samsung Gear VR, HoloLens 2. Izlaskom najavljenog Unreal Engine-a 5 stiže i podrška za najavljene konzole PlayStation 5 i Microsoft Xbox Series X [13].

13. svibnja 2020. Epic je najavio izlazak Unreal Engine-a 5 krajem 2021. Ta verzija će uključivati podršku za sve platforme pa tako i za nadolazeći PlayStation 5 i Xbox Series X. Cilj postavljen ovom verzijom je znatno skraćivanje vremena potrebnog razvojnim programerima za kreiranje detalja i okruženja unutar videoigre i omogućavanje pokretanja videoigara u 4k rezoluciji [14].

Ovaj alat sadržava nekoliko glavnih komponenti koje čine svaki alat za razvoj računalnih i drugih igara. Svaka od tih komponenti je zasebna, ali zajedno čine jezgru ovog alata. Kako su sve komponente razvijane modularno, unutar alata je omogućeno prilagođavanje postojećih i dodavanje novih komponenti, bez drastične promjene jezgre alata.

Unreal Engine je pronašao mogućnost uporabe i u filmskom stvaranju, poput produkcije televizijskih serija poput The Mandalorian i Westworld. U tim serijama virtualni setovi su stvoreni unutar Unreal-a, a zatim prikazani na velikim LED projekcijskim ekranima i atmosferskim sustavima osvjetljenja koji prate kretanje kamere oko glumaca i predmeta. Cjelokupni izgled prepoznat je kao prirodniji u odnosu na tipične efekte i omogućava kompoziciju kadrova u stvarnom vremenu, trenutno uređivanje virtualnih setova po potrebi i mogućnost snimanja više scena u kratkom razdoblju samo promjenom virtualnog svijeta iza glumca [15].

Slika br. 1: Izgled sučelja Unreal Engine-a 4 [16]

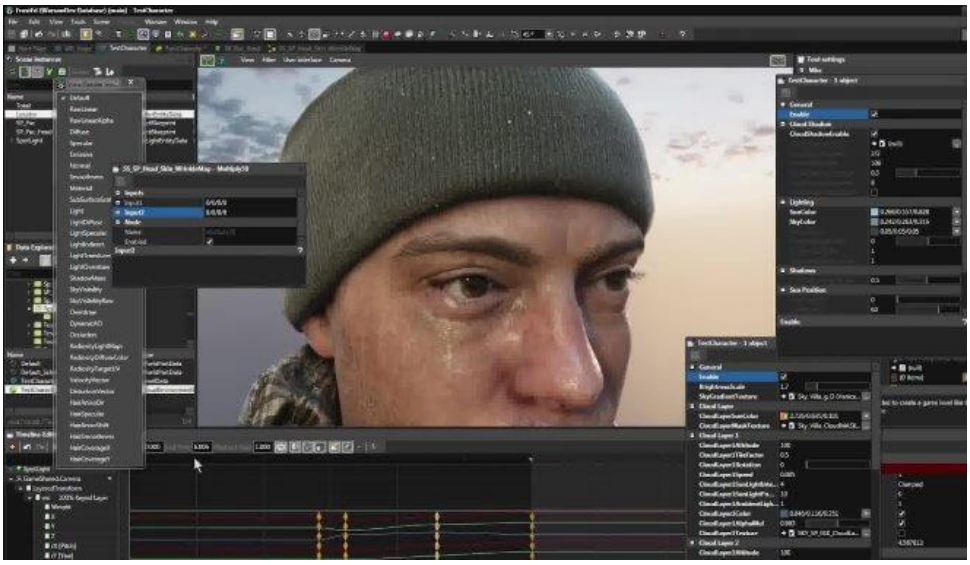
Frostbite

Frostbite je alat za razvoj videoigra, razvijen 2008. od strane DICE-a, dizajniran za korištenje na više platformi u sustavu Microsoft Windows, igraćim konzolama sedme generacije PlayStation 3 i Xbox 360 i igraćim konzolama osme generacije PlayStation 4 i Xbox One. Frostbite je izvorno služio isključivo u seriji videoigara Battlefield, ali kasnije biva proširen na druge pucačke videoigre u prvom licu i razne druge žanrove. Do danas je Frostbite ekskluzivan za videoigre u izdanju Electronic Arts-a. Jezgra ovog alata je pisana u C++ i C#-u [17].

Izlaskom igre Battlefield 4, krajem 2013., razvijen je Frostbite 3. U novoj verziji alata ažurirana su okruženja koja postaju puno dinamičnija nakon djelovanja igrača. Novom komponentom Destruction 4.0, poznatom pod imenom Levolution, igračevim djelovanjem na jednom dijelu područja je postalo moguće prouzročiti promjene za sve ostale igrače koliko god oni bili udaljeni. Korištenjem Frostbite-a 3 je nastavljen serijal Need for Speed, izlaskom nastavka Need for Speed Rivals [18] [19].

Prvom razvijenom igrom pomoću Frostbite 1, igranje je bilo podržano samo na konzolama, odnosno isključivo za PlayStation 3 i Xbox 360. Dvije godine kasnije izlaskom drugog seta igara koje su bile razvijene u verziji 1.5, dodano je proširenje za računalnu platformu. Do danas se nije puno toga promijenio što se tiče podrške ovog alata za neke druge platforme poput mobilne, za sada se pomoću Frostbite-a razvijaju igre isključivo za računala i konzole.

Alat Frostbite za razvoj videoigara je u vlasništvu Electronics Arts-a i sve videoigre koje su izdavane i koje će se izdavati su isto tako u vlasništvu EA-a ili njezinih sestrinskih poduzeća. Time je ovaj alat zatvoren za javnu uporabu, niti se izdaje licenca za korištenje drugim poduzećima. Sva poduzeća koje su u vlasništvu EA-a i razvijaju videoigre su dužna koristit njihov alat [20].



Slika br. 4: Izgled sučelja Frostbite 3

Zaključak

Različite su zamisli dostupnosti i upotrebe današnjih alata. Alat Frostbite nije uopće zamišljen za javnu upotrebu, a Electronics Arts, poduzeće koja posjeduje Frostbite, ne generira direktno profit preko njega. Za razliku od EA-a, Unity Technologies s Unity-jem i Epic Games s Unreal Engine-om imaju potpuno suprotnu zamisao uporabe svojih alata. Omogućivši besplatno korištenje svojih alata u nekomercijalne svrhe privukli su veliki broj novih korisnika, bilo to već iskusnih programera ili početnika. Takvim pristupom na tržište je počeo pristizati veliki broj novih videoigara, naravno sve videoigre koje su napravljene njihovim alatima su se naplaćivale ovisno o prihodu kojeg ostvare od prodaje.

Alati Unreal Engine i Unity prednjače i u drugim poljima, a ne samo u videoigrama. Unity je uvelike prepoznat u industriji koja se bavi razvojem umjetne, virtualne i mješovite stvarnosti, stoga se mnoga poduzeća odlučuju za Unity kao razvoju okolinu za svoj softver. Unreal Engine biva sve više prepoznat kao alat za razvoj animiranih i znanstveno-fantastičnih filmova, posebno nakon suradnje s Disney-em na seriji Star Wars: The Mandalorian.

Zaključno, o većini današnjih popularnih alata za razvoj videoigara ne treba govoriti isključivo u kontekstu razvoja videoigara, nego u kontekstu koji je puno širi od toga.

Reference

[1] Plarium, »What Was the First Video Game, Who Invented It and Why,« 2018. [Mrežno]. Dostupno: https://plarium.com/en/blog/the-first-video-game/. [Pokušaj pristupa 10 7. 2020.].

[2] B. Edwards, »Computer Space and the Dawn of the Arcade Video Game,« 2011. [Mrežno]. Dostupno: https://www.technologizer.com/2011/12/11/computer-space-and-the-dawn-of-the-arcade-video-game/. [Pokušaj pristupa 10 7. 2020.].

[3] P. S. Paul, S. Goon i A. Bhattacharya, »History and comparative study of modern game engines,« BioIT International Journals, pp. 245-249, 2012.

[4] Microsoft Azure, »What is middleware?,« bez dat.. [Mrežno]. Dostupno: https://azure.microsoft.com/en-us/overview/what-is-middleware/. [Pokušaj pristupa 12 7. 2020.].

[5] Studytonight, »Understanding Game Architecture,« bez dat.. [Mrežno]. Dostupno: https://www.studytonight.com/3d-game-engineering-with-unity/game-development-architecture. [Pokušaj pristupa 14 7. 2020.].

[6] S. Axon, »Unity at 10: For better-or worse-game development has never been easier,« 2016. [Mrežno]. Dostupno: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/. [Pokušaj pristupa 1 9. 2020.].

[7] Aleksandr, »Documentation, Unity scripting languages and you,« 2014. [Mrežno]. Dostupno: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/. [Pokušaj pristupa 2 9. 2020.].

[8] R. Fine, »UnityScript's long ride off into the sunset,« 2017. [Mrežno]. Dostupno: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/. [Pokušaj pristupa 2 9. 2020.].

[9] »Unparalleled platform support,« 2020. [Mrežno]. Dostupno: https://unity.com/features/multiplatform. [Pokušaj pristupa 2 9. 2020.].

[10] L. Matney, »With new realities to build, Unity positioned to become tech giant,« 2017. [Mrežno]. Dostupno: https://techcrunch.com/2017/05/25/with-new-realities-to-build-unity-positioned-to-become-tech-giant/. [Pokušaj pristupa 2 9. 2020.].

[11] D. Kahn, »Evolving the Unity Editor UX,« 2019. [Mrežno]. Dostupno: https://blogs.unity3d.com/2019/08/29/evolving-the-unity-editor-ux/. [Pokušaj pristupa 1 9. 2020.].

[12] S. Andrew, An Introduction to Unreal Engine 4, USA: CRC Press, 2017, p. ix.

[13] Unreal Engine, »Unreal Engine 4 Documentation,« bez dat.. [Mrežno]. Dostupno: https://docs.unrealengine.com/en-US/Platforms/index.html. [Pokušaj pristupa 5 9. 2020.].

[14] D. Takahashi, »Epic Games: Unreal Engine 5 will bring a generational change to graphics,« 2020. [Mrežno]. Dostupno: https://venturebeat.com/2020/05/13/how-epic-games-is-tailoring-unreal-engine-5-to-make-next-gen-graphics-shine/. [Pokušaj pristupa 4 9. 2020.].
[15] O. S. Good, »How Lucasfilm used Unreal Engine to make The Mandalorian,« 2020. [Mrežno]. Dostupno: https://www.polygon.com/tv/2020/2/20/21146152/the-mandalorian-making-of-video-unreal-engine-projection-screen. [Pokušaj pristupa 5 9. 2020.].
[16] Unreal Engine, »Playing and Simulating,« bez dat.. [Mrežno]. Dostupno: https://docs.unrealengine.com/en-US/GettingStarted/HowTo/PIE/index.html. [Pokušaj pristupa 4 9. 2020.].
[17] »Pros and Cons of The Frostbite Engine,« 2020. [Mrežno]. Dostupno: https://www.gamedesigning.org/engines/frostbite/. [Pokušaj pristupa 6 9. 2020.].
[18] Battlefield, »Battlefield 4: Official Frostbite 3 Feature Video,« 2013. [Mrežno]. Dostupno: https://www.youtube.com/watch?v=R9yVV6g3q7g&ab_channel=Battlefield. [Pokušaj pristupa 6 9. 2020.].
[19] B. Crecente, »Need For Speed Rivals is a living game, and a sign of things to come,« 2013. [Mrežno]. Dostupno: https://www.polygon.com/2013/8/22/4646854/need-for-speed-rivals-is-a-living-game-and-a-sign-of-things-to-come. [Pokušaj pristupa 6 9. 2020.].
[20] »EA forces all their studios to use the Frostbite Engine to develop on,« 2018. [Mrežno]. Dostupno: https://www.reddit.com/r/titanfall/comments/7c0hqo/ea_forces_all_their_studios_to_use_the_frostbite/. [Pokušaj pristupa 6 9. 2020.].
[21] J. Brodkin, »How Unity3D Became a Game-Development Beast,« 2013. [Mrežno]. Dostupno: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. [Pokušaj pristupa 1 9. 2020.].

Izrada multiplatformske igre pomoću razvojnog okvira Flutter

Goran Alković i Mladen Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
galkovic@foi.hr, mlkoneck@foi.hr

Sažetak

U ovom radu opisuje se proces razvoja multiplatformskih igara u razvojnom okviru *Flutter*. Ukratko se opisuje sam razvojni okvir i proces izrade nekoliko funkcionalnih primjera jednostavnih igara: prvih nekoliko nivoa dvodimenzinalnog platformera *Dangerous Dave*, klon igre *Tetris* i klon igre *Minesweeper*.

Ključne riječi: računalna igra, Flutter, višeplatformska igra, Dangerous Dave, Tetris, Minesweeper, programiranje, algoritmi

Uvod

Razvoj računalnih igara je oduvijek bio izazovan proces. No u novije vrijeme, razvojem tehnologije, primarno alata koji su specifično razvijeni kako bi se koristili u procesu razvoja računalnih igara, proces razvoja vlastite računalne igre je uvelike olakšan. Danas na tržištu imamo brojne igrače platforme: osobna računala, konzole, tablete, pametne telefone i dr. [6]. Budući da svaka platforma koristi svoju vlastitu tehnologiju, razvoj igara koje bi bile podržane na više platformi je bio dugotrajan i skup. S vremenom su se pojavili programski alati poput alata *Unity* ili *Unreal Engine* koji su omogućili izvoz igara na više različitih platformi [4].

Flutter je potpuno novi razvojni okvir koji je razvilo poduzeće *Google* koji omogućava razvoj višeplatformskih aplikacija svih vrsta, pa tako i računalnih igara [3]. Razvoj aplikacije putem ovog razvojnog okvira korisniku bi trebao omogućiti brži razvoj aplikacije kao i bolju optimizaciju. U ovom radu će se istražiti je li ovaj razvojni okvir adekvatan za razvoj jednostavnih višeplatformskih igara koje bi se mogle izvesti na osobno računalo (kao samostalna aplikacija i kao igra u Internet pregledniku) a i na mobilne uređaje.

Korištene tehnologije

U svrhu razvoja prototipa tri jednostavne računale igre korištenjem razvojnog okvira *Flutter*, u kojem se koristi programski jezik *Dart*, korišteni su i neki

drugi programski proizvodi za realizaciju. Korišten je i alat *Adobe XD* [1] za izradu vizualnih efekata i korišten je alat *Tiled* [5]koji se koristio za izradu razina klona igre *Dangerous Dave* koja se temelji na uzorcima (eng. *tile*).

Flutter je Googleov razvojni okvir za izradu izvorno-kompajliranih aplikacija za mobilne uređaje, stolna računala i Internet preglednike. Sa svojom dobro osmišljenom višeslojnom arhitekturom i deklarativnim principima razvoja korisničkog sučenja olakšan je razvoj aplikacija. Izvorno kompajliranje kôda omogućuje brže izvođenje, što je poželjna karakteristika za igre. Poželjno je imati iscrtavanje brzine 60 (ili više) sličica po sekundi (eng. *frames per second*). Unutar bogate kolekcije vanjskih biblioteka za razvojni okvir *Flutter*, nalazi se i razvojni okvir *Flame* [2] koji je korišten za razvoj jedne od prototipa igara. *Flame* je modularan, jednostavan za korištenje, oslanja se na „vremenom provjerene“ prakse razvoja igara, a zahvaljujući podršci zajednice neprestano se razvija i dobiva nove funkcionalnosti.

Alat *Tiled* je besplatan uređivač razina igre (eng. *level editor*). Izuzetno je lak za korištenje i zbog jednostavnosti i otvorenosti formata zapisa razina se pokazao kao odlično rješenje za proces dizajniranja razina za klon igre *Dangerous Dave*.

Razvoj igara - GameBox

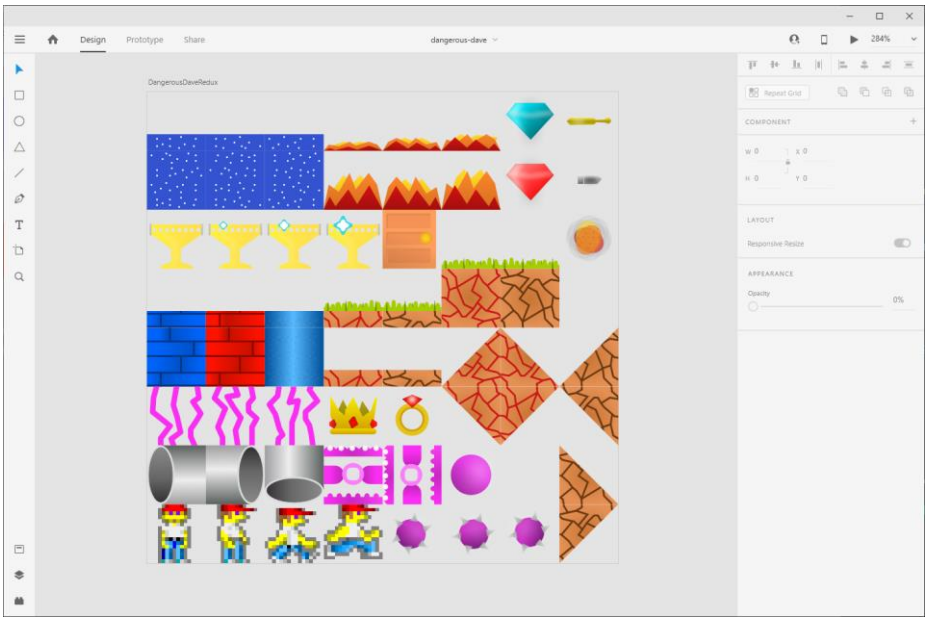
Inspiracija za razvoj prototipa jednostavnih igara bile su starije igre iz ere DOS operacijskog sustava (*Doom*, *Dangerous Dave*, *Tetris*, *Supapelx*, *Prehistorik* i dr.). Tematika je pomalo u „retro“ stilu. Odabrana su tri naslova na temelju kojih će biti izrađeni njihovi klonovi: dvodimenzionalna platformer *Dangerous Dave*, igra *Tetris* i klasična igra s Windows operacijskog sustava, *Minesweeper*. Te tri jednsotavne igre biti će objedinjene u jednu cjelinu koja je nazvana *GameBox*.

Slika br. 1: Logotip igre (vlastita izrada)

Razvoj klona igre Dangeorus Dave

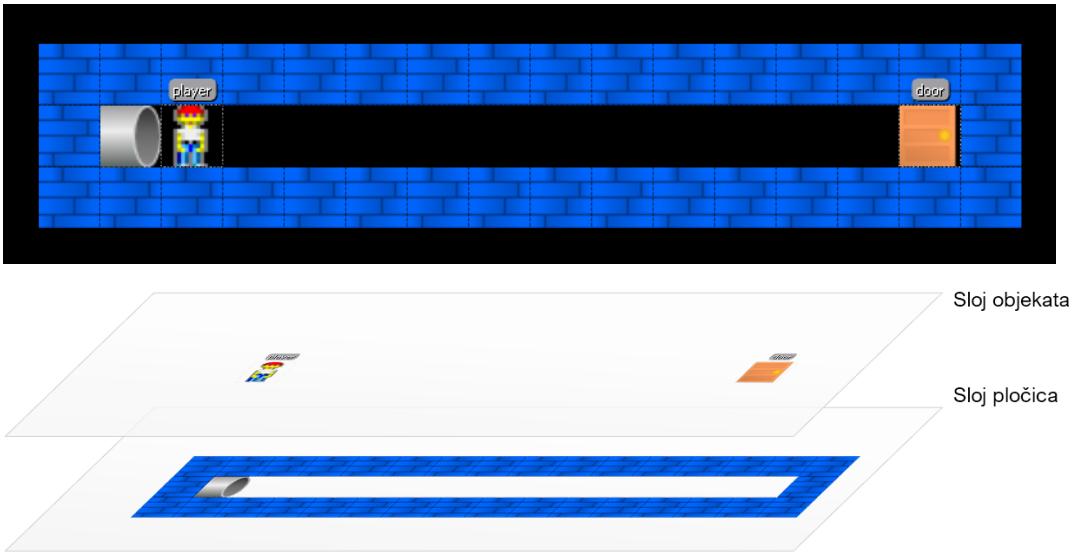
Budući da je igra Dangerous Dave igra koja ima više razina, prvo je bilo potrebno kreirati razine koje se želi uključiti u igru. Kao format odabran je *Tiledov* TMX format koji se temelji na XML-u. Naravno, kao što je već rečeno, same razine su dizajnirane i kreirane putem alata *Tiled*.

Nakon toga je putem alata Adobe DX kreiran set sličica (eng. *tileset*) koji se sastoji od osnovnih elemenata (statičnih i interaktivnih) koji se pojavljuju na svakoj razini igre.



Slika br. 2: Izrada tileseta (vlastita izrada)

Nakon što su izrađeni svi grafički elementi i nakon što su bili spremni za korištenje, kreirana je prva razina igre u kojoj igrač treba od ulazne točke razine (siva cijev) doći do izlaza razine (smeđa vrata). Svaka razina se sastoji od dva osnovna sloja: *sloja objekata* i *sloja pločica*. Sloj pločica je statičan i on se može učitati odjednom dok sloj objekata zahtijeva napredniju tehniku učitavanja.



Slika br. 3: Prikaz prve razine (vlastita izrada)

Za učitavanje *sloja objekata* bilo je potrebno napraviti dinamički sustav adaptivnog učitavanja sloja, koji ovisno o objektu dodaje potrebnu posebnu funkcionalnost istom i postavlja ga na pravo mjesto. Nakon što su svi objekti

iscrtani na sceni, potrebno je omogućiti osnovne funkcionalnosti igre: kretanje igrača, skakanje, pucanje, detekcija kolizije, sustav napredovanja kroz igru i sl.

Fizika u igri je relativno jednostavna. Za kretanje po X osi koristi se konstantna brzina dok se za kretanje po Y osi (skakanje) koristi varijabilna akceleracija. Igrač likom može upravljati putem zaslonskih kontrola (za mobilne uređaje) a također je moguće igru igrati korištenjem tipkovnice. U igri postoje statični i pokretni neprijatelji. Ako igrač dođe u koliziju s njima, tada gubi život i vraća se na početak razine na kojoj se nalazi. Igrač također ima mogućnost pucanja. Kada ispaljeni metak dođe u koliziju s pokretnim neprijateljima, tada se prikazuje animacija uništenja i neprijatelj se dealocira.

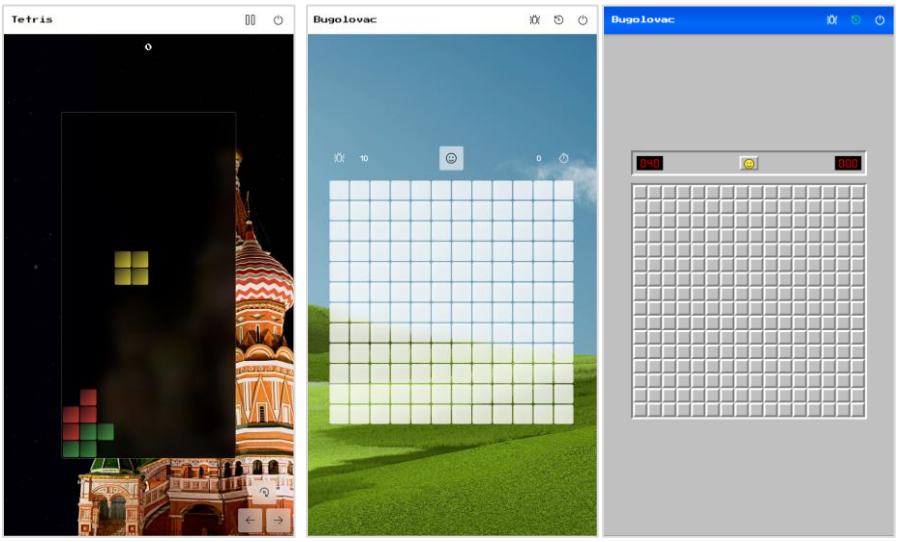
Slika br. 4: Dovršena igra (vlastita izrada)

Razvoj klona igara Tetris i Minesweeper

Osnovna mehanika igara *Tetris* i *Minesweeper* je zasigurno svima jako dobro poznata s obzirom da se radi o dva bezvremena klasika računalnih igara. U prvoj fazi razvoja bilo je potrebno odrediti podatkovni model igre za spremanje elemenata trenutnog stanja igre. To su *blokovi* i *atomi* kod klona igre *Tetris* i *ploča za igru* kod klona igre *Minesweeper* (koja je dobila domaći naziv *Bugolovac*).

Kod klona igre *Tetris* kreirane su dvije osnovne klase: klasa *atomi* koja se odnosi na osnovne elemente od kojih su sačinjeni *blokovi*. Osim konfiguracije blokova, u klasi se nalaze i neke druge dodatne informacije poput boje bloka i sl. Nakon što je kreiran podatkovni model, bilo je potrebno implementirati osnovnu logiku igre: detekciju kolizije, generiranje blokova, prikaza na korisničkom sučelju, osnovnu logiku igre i sl. Kao i kod prethodne igre, igra se može igrati putem tipki na zaslonu ili putem tipkovnice. Igra *Bugolovac* ima

nešto jednostavniji podatkovni modela koji se sastoji od dvodimenzinalnog polja na kojem se generira potreban broj mina. Implementirane su osnovne standardne funkcionalnosti: otvaranje i prikaz praznih mjesta te broja mina na susjednim poljima. Igrač može postaviti zastavicu tamo gdje smatra da se nalazi mina. Nakon što uspješno označi sva polja s minama, igra se završava.



Slika br. 5: Sučelja Tetrisa i Minesweepera (vlastita izrada)

Zaključak

Flutter se pokazao kao izuzetno dobar razvojni okvir za razvoj jednostavnijih igara koje je moguće igrati na više platformi. Naravno, bez obzira na svu dodatnu pomoć, osnovnu programsku logiku igra i dalje mora sam kodirati i to je uvijek izazova proces. No, ovaj razvojni okvir u svakom slučaju proces izrade igre ubrzava. S obzirom na neprestani razvoj, ono što se može očekivati je da će ovaj razvojni okvir s vremenom postojati sve bolji te da će se sve lakše i brže moći doći do realizacije željene aplikacije.

Reference

[1] Adobe XD. (13. listopada 2020.). Dostupno na: https://www.adobe.com/products/xd.html
[2] Flame. (13. listopada 2020.). Dostupno na: https://flame-engine.org/docs/
[3] Flutter. (12. listopada 2020.). Dostupno na: https://flutter.dev/
[4] TheTool: The Best 15 Mobile Game Engines / Development Platforms & Tools in 2020. (12. listopada 2020.). Dostupno na: https://thetool.io/2018/mobile-game-development-platforms
[5] Tiled. (13. listopada 2020.). Dostupno na: https://www.mapeditor.org/
[6] Wikipedia: Video game platforms. (12. listopada 2020.). Dostupno na: https://en.wikipedia.org/wiki/Category:Video_game_platforms

Osnovne mehanike akcijsko avanturističke igre iz prvog lica

Matija Kralj i Mladen Konecki
Fakultet organizacije i informatike, Sveučilište u Zagrebu
m.kralj97@hotmail.com, mlkoneck@foi.hr

Sažetak

U ovom radu opisana je implementacija osnovnih mehanika akcijsko avanturističke igre iz prvog lica u programskom alatu Unity. Avanturistički dio mehanika igranja se odnosi na interakciju s okolinom i objektima u okolini dok akcijski dio uključuje interakciju s neprijateljima. Glavni fokus je na interakciji igrača s neprijateljima te programskoj logici ponašanja i reagiranja neprijatelja na igrača na temelj raznih podražaja koje neprijatelj „osjeća“.

Ključne riječi: računalna igra, Unity, NPC, umjetna inteligencija, algoritmi, programiranje

Uvod

Žanr akcijsko avanturističkih igara je izuzetno širok i uključuje razno razne igre različitih mehanika igranja. Avanturističke igre se temelje na situacijskim problemima i kod njih nije naglašen akcijski dio. Akcijske igre pak mehaniku igranja temelje na akciji, brzini i refleksima no nemaju izražen segment rješavanja problema. Sve ono što je između, što uključuje segmente jednoga i drugoga ulazi u ovu kategoriju igara [4].

Unity je alat koji služi za razvoj računalnih igara te je u potpunosti besplatan. To je zapravo pokretač igara (eng. *game engine*) koji nudi brojne funkcionalnost za razvoj kako dvodimenzionalnih tako i trodimenzionalnih igara, različitih žanrova. U njemu je moguće razvijati igre za mobilne uređaje, za miješanu i virtualnu stvarnost, za osobna računala i konzole, igre za web i mnoge druge platforme [3]. Programski jezik koji se koristi za razvoj igara u Unityju je C#.

Akcijsko avanturistička igra

A obzirom na širinu koju pokriva područje akcijsko avanturističkih igara, bilo bi dobro dobro definirati o čemu se točno radi kada se govori o ovom žanru. Naravno, ponekad je jako teško napraviti jasnu distinkciju između pojedinih žanrova kao što i mnoge igre ne ulaze specifično u jedan žanr zašto što možda

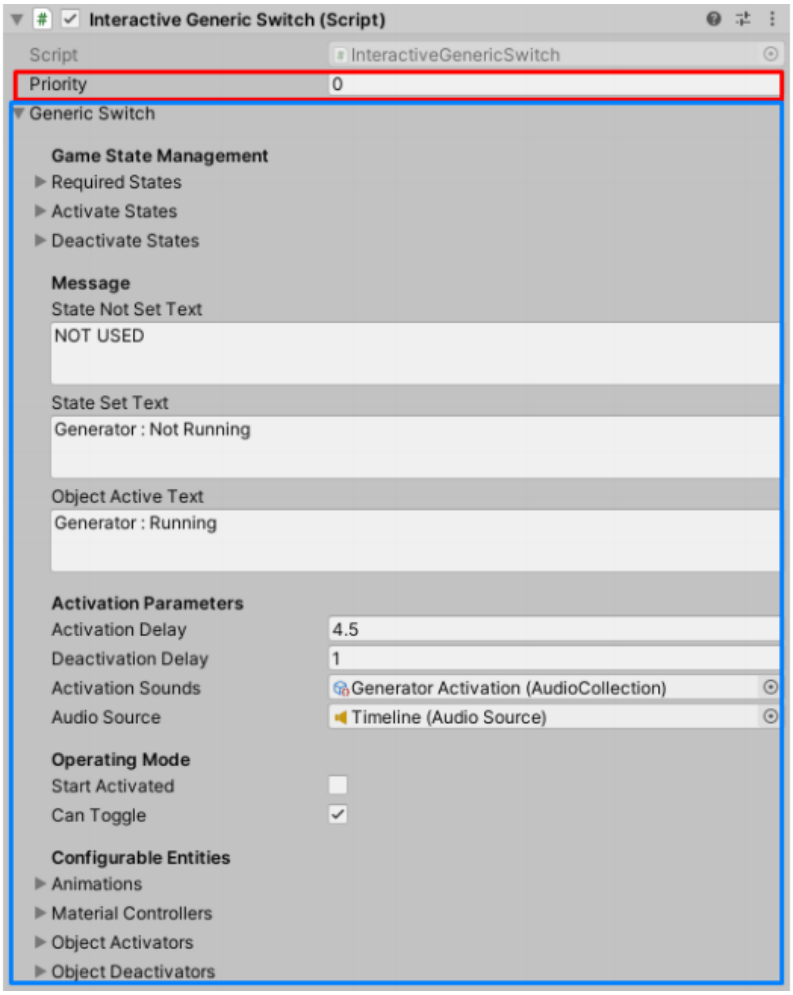
kombiniraju mehanike igranja iz više različitih žanrova. Na taj način su definirani brojni podžanrovi ovog tipa igara. Primjerice, akcijska avantura iz prvog lica čija je glavna specifična karakteristika način kretanja kamere tako da igrač igra igru iz vidnog polja glavnog protagonista igre [4]. Ili recimo patformska avanturistička igra: igra koja kombinira mehaniku igranja platformera s karakteristika avanturističkih igara [5]. Generalno, što se tiče mehanika igranja, akcijske avanture zahtijevaju fizičke vještine igrača koje su potrebne kod tipičnih akcijskih žanrova igara ali također uključuje aspekte avanturističkih igara: naglasak na priču, raznolikost likova, sustav inventara, složene dijaloge, rješavanje situacijskih problema, zagonetki i sl.

Interaktivnost u akcijsko avanturističkoj igri

U nastavku slijedi generalni opis razvoja osnovnih mehanika akcijsko avanturističke igre korištenjem programskog alata Unity. Svaki interaktivni objekt mora sadržavati *InteractiveItem* skriptu koja sadrži osnovne funkcionalnosti svakog interaktivnog objekta (aktivacija, dohvaćanje svojstva objekta i sl.). Također ja važno interaktivne objekte staviti u zaseban sloj (eng. *layer*). Specifično, interaktivni objekti mogu imati i svoje jedinstvenosti stoga će imati svoju zasebnu skriptu koja nasljeđuje odnosno specijalizira generalnu klasu interaktivnih objekata *InteractiveItem*. U nastavku slijedi primjer jedne takve klase: *InteractiveGenericSwitch*. Uz prikaz dijela koda, prikazana će biti i komponenta koju Unity kreira nakon kompilacije skripte.

```
public class InteractiveGenericSwitch : InteractiveItem {
   [SerializeField]
   private GenericSwitch _genericSwitch = new GenericSwitch();
   protected override void Start()
   {
      base.Start();
      _genericSwitch.OnStart();
   }
   public override string GetText()
   {
      if (!enabled)
         return string.Empty;
      return _genericSwitch.OnGetText();
   }
   public override void Activate(CharacterManager characterManager)
   {
      _genericSwitch.Activate(characterManager);
      if (_genericSwitch._coroutine != null)
         StopCoroutine(_genericSwitch._coroutine);
      _genericSwitch._coroutine = _genericSwitch.DoDelayedActivation();
      StartCoroutine(_genericSwitch._coroutine);
   }
}
```

Programski kod prikazuje klasu a ujedno i skriptu *InteractiveGenericSwitch* koja je zadužena za veliki broj promjena stanja kod aktivacije i deaktivacije interaktivnih objekata. Ovdje je jasno vidljivo kako ova klasa nasljeđuje funkcionalnosti *InteractiveItem* klase. Start metoda ove klase interaktivni objekt registrira u rječnik interaktivnih objekata u skripti koja je za to zadužena. Klasa GenericSwitch sadrži veliki dio programske logike koja je zadužena za promjenu stanja, dohvaćanja odgovarajućeg teksta za prikaz, pokretanje odgovarajuće animacije i sl.



Slika br. 1: Komponenta "InteractiveGenericSwitch" skripte

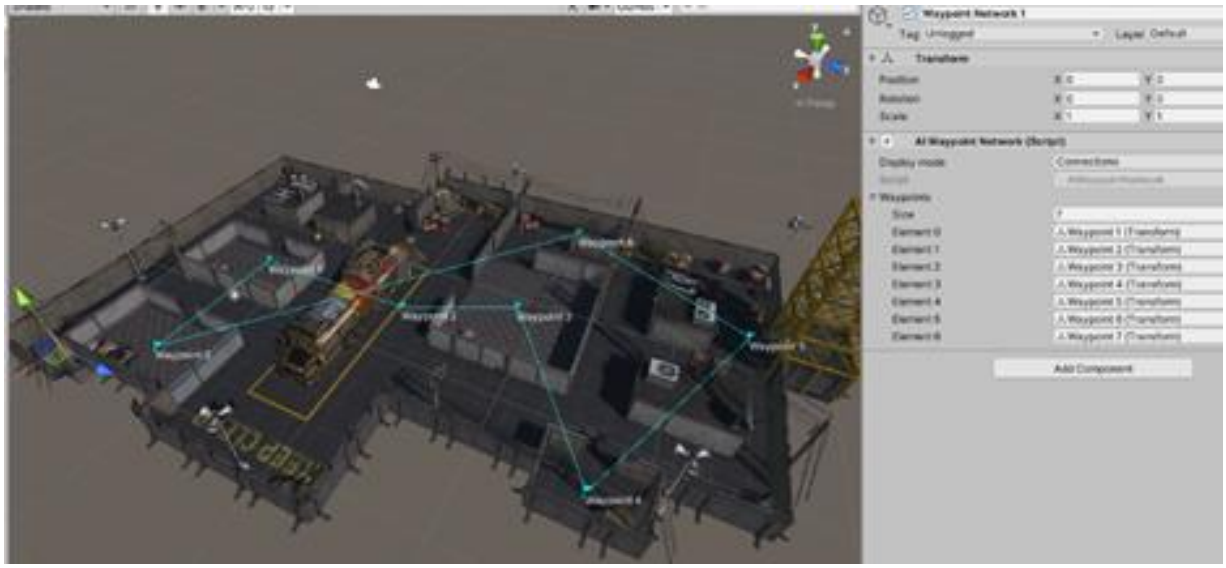
Na slici br. 1 je vidljiva komponenta koju kreira Unity. Crvenom bojom označen je dio vezan za *InteractiveItem* klasu (atribut prioriteta) a plavom dio vezan za *GenericSwtich* klasu. Skripta se sastoji od nekoliko dijelova: upravljanje stanjem igre, upravljanje porukama za igrača, parametri aktivacije, način rada, podesivi entiteti i dr.

Interakcija igrača s interaktivnim objektima

Kako bi igrač mogao vršiti interakciju s interaktivnim objektima potrebno je provjeriti nalazi li se igrač u neposrednoj blizini nekog interaktivnog objekta. To je implementirano korištenjem zrake (eng. *ray*) koja se projicira u sredini vidnog polja igrača [1]. Ako se zraka nađe u koliziji s nekim objektom koji se nalazi u interaktivnom sloju tada igrač ima mogućnost interakcije s interaktivnim objektom. Drugo ograničenje je i generalna udaljenost od tog objekta koja je podešena na neku prikladnu udaljenost za igrača. Nakon što se dogodi kolizija, dohvaća se identifikacijski broj (eng. *instance ID*) objekta koji je pogođen zrakom (ako takav postoji) [2]. Pomoću identifikatora provjerava se postoji li objekt s tim ključem u rječniku interaktivnih objekata. Ukoliko postoji tada je moguće vrlo jednostavno ostvariti odgovarajuću željenu interakciju.

Neprijatelji (agenti) i njihova konfiguracija

Kako bi igraču igra bila zabavna, a i izazovna, potrebno je napraviti uvjerljivu interakciju igrača s neprijateljima. Neprijatelji trebaju imati sofisticirano ponašanje kako bi pružili zanimljive načine i vrste interakcija. Jedna od osnovnih funkcionalnosti kod svake igre je mogućnost da se agenti kreću slobodno kroz neki definirani prostor. U Unityju je to lako moguće realizirati pomoću definiranja navigacijske plohe (eng. *Nav Mesh*) i definiranja navigacijskih agenata (eng. *Nav Agent*). Navigacijska ploha definira prostor po kojem se navigacijski agenti mogu kretati. Jedan oblik nasumičnog kretanja agenata po plohi je definiranje rute patroliranja agenata. Potrebno je definirati točke koje čine neku rutu patroliranja. Ruta može biti konstantna ali i nasumična. Na slici br. 2 prikazane su točke rute na definiranom prostoru.

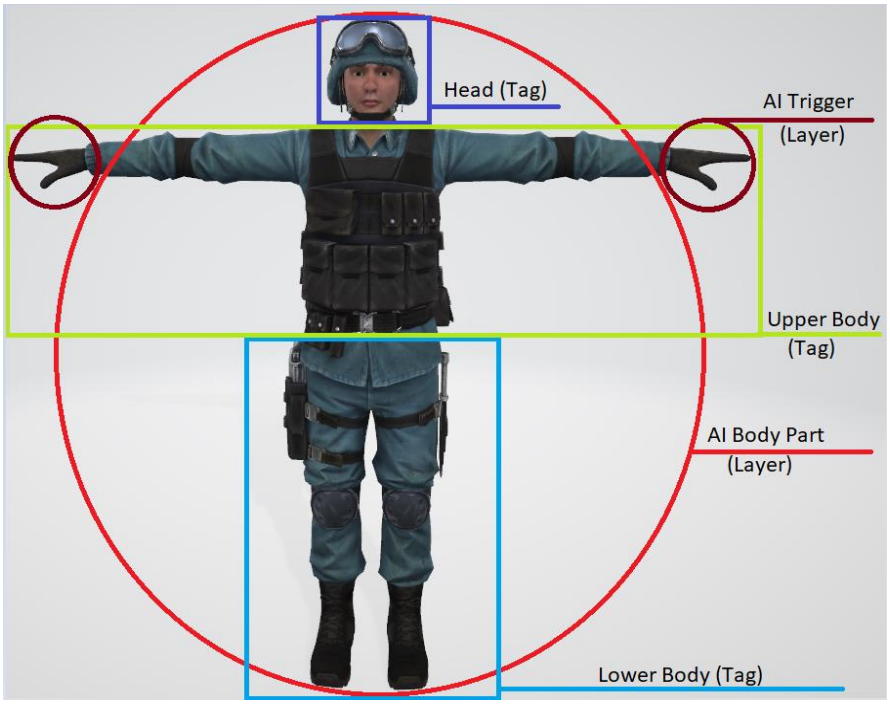


Slika br. 2: Ruta i točke rute

Osim samog kretanja, agenti su konfigurirani na način da svaki agent ima svoj objekt koji predstavlja senzor kojim „promatra okolinu. Svaki agent ima svoje atribute koji su podesivi na razini pojedine instance: inteligencija, zdravlje, granica nakon koje nastupa oduzimanje uda ili puzanje (ili potpuna nepokretnost), razina agresivnosti, širina kuta gledanja i sl. Agenti imaju i svoja stanja ponašanja. U svakom trenutku pojedini agent se može nalaziti u jednom od nekoliko definiranih stanja: patroliranje, bez posla, stanje pripravnosti, napadanje, potjera ili hranjenje.

Interakcija između igrača i neprijatelja

Mogućnost interakcije neprijateljskog agenta s igračem se može ostvariti na nekoliko različitih načina. Prvo što agent mora je na neki način detektirati igrača. Agent to može učiniti na nekoliko načina: vizualno (svijetljenje svjetiljke ili fizičko uočavanje igrača), služnim putem (koraci igrača ili skakanje, ispaljivanje metaka iz oružja) te miris (krv ranjenog igrača). Senzoru su implementirani na način da je definiran objekt koji nema nikakva vizualna i nalazi se u hijerarhiji agenta. Taj objekt sadrži komponentu *Box Collider* koja omogućava detekciju kolizije. Kada agentov senzor dođe u koliziju sa sličnim objektom koji se nalazi u hijerarhiji igrača tada agent zna da je igrač u blizini i omogućava se adekvatno reagiranje. Osim same detekcije agenti mogu napasti igrača tj. nanijeti mu štetu a jednako tako i igrač može nanijeti štetu agentu. Ovisno o tome kolika je razina oštećenost pojedinog dijela tijela agenta upravlja se razinom ranjenosti agenta. Na slici br. 3 vidljive su definirane regije tijela agenta.



Slika br. 3: Prikaz slojeva i oznaka dijelova tijela agenta

Gotovo cijelo tijelo agenta se nalazi u sloju *AI Body Part* osim ruku koje se nalaze u sloju *AI Trigger*. Razlog zašto je takva podjela je taj što rukama agent zadaje štetu igraču a ostatak tijela ne. Tijelo je dalje podijeljeno pomoću oznaka (eng. *tag*). Agent ima tri različite oznake: glava, gornji i donji dio tijela. Na temelju tih oznaka registrira se u koji dio tijela je agent pogođen i na temelj toga se mijenja njegovo zdravstveno stanje. Pogodak u glavu zadaje agentu veću količinu štete i može ga automatski usmrtiti. Pogodak u donji dio tijela može onesposobiti neprijateljevu mogućnost hodanja pa on počinje šepati ili čak i puzati ako se zada dovoljna količina štete. Na ovaj način se može implementirati sofisticiranija mehanika borbene interakcije između agenata neprijatelja i igrača. Ista stvar se naravno može primijeniti i na igraču. U slučaju da dobije udarac u glavu, igraču se zadaje veća šteta nego ako je udaren u ruku. Ako se igraču nanese šteta u nogu, mogućnost bi bila da igrač više ne može trčati, da mu se uspori hodanje i sl.

Na kraju ostaje spomenuti stanja agenata. Agenti počinju svoj životni vijek u stanju bez posla te nakon nekog vremena prelaze u stanje patroliranja i u njemu ostaju. Na dva načina agent može preći iz stanja patroliranja u neko drugo stanje: glad agenta i uočavanje igrača. Agent mijenja stanje ponašanja tako da prelazi u stanje pripravnosti te iz tog stanje odlučuje što će dalje učiniti. Primjerice, ukoliko agent primijeti svjetlost igračeve svjetiljke, on ostaje u stanju pripravnosti, okreće se oko sebe i pokušava otkriti izvor svjetlosti. Agent izlazi iz stanja pripravnosti tek ukoliko se svjetlost ugasi ili ako uspije pronaći izvor svjetlosti tj. igrača. Na brzinu pronalaska izvora svjetlosti utječe definirana razina inteligencije agenta. Kada agent opazi igrača, prelazi u stanje potjere. Ukoliko se dovoljno približi igraču u stanju potjere tada može promijeniti stanje u napadanje igrača. Ako igrač uspije pobjeći agentu tada se agent može vratiti u stanje pripravnosti.

Zaključak

Kod računalnih igara vrlo je bitna razina sofisticiranosti implementacije neprijateljskih agenata jer ako se agenti ne ponašaju dovoljno „inteligentno“ to će zasigurno pokvariti dojam o kvaliteti igre. Jednako tako, implementacijom nešto sofisticiranijih mehanika borbene interakcije moguće je dobiti zanimljivu raznolikost kako svaki susret s neprijateljem agentima ne bi izgledao jednak. Naravno, to zahtjeva dodatno animiranje i skriptiranje no kada je borbena interakcija u centru igre onda bi zasigurno trebalo obratiti pažnju na taj segment.

Reference

[1] Unity Documentation: Physics.Raycast (21. rujna 2020.). Dostupno na: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

[2] Unity Documentation: Object.GetInstanceID (21. rujna 2020.). Dostupno na: https://docs.unity3d.com/ScriptReference/Object.GetInstanceID.html
[3] Unity Multiplatform Support. (20. rujna 2020.). Dostupno na: https://unity.com/features/multiplatform
[4] Wikipedia: Action-adventure game. (20. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Action-adventure_game
[5] Wikipedia: Platform game – Platform-adventure game. (20. rujna 2020.). Dostupno na: https://en.wikipedia.org/wiki/Platform_game#Platform-adventure_game

Tehnologija virtualne i proširene stvarnosti u obrazovanju

Darjan Vlahov
Učiteljski fakultet, Sveučilište u Zagrebu
darjan.vlahov@gmail.com

Sažetak

Korištenje tehnologije virtualne i proširene stvarnosti iduća je stepenica među nastavnim sredstvima i pomagalima koja se koriste u školama i na fakultetima. Tehnološki napredak uvijek je korak ispred njegove implementacije u odgojno-obrazovni sustav ali je on neminovan i događa se kad tad, prije svega radi usklađivanja potreba tržišta rada za znanjima i vještinama koje će učenici i studenti trebati završetkom školovanja. Ovaj rad donosi pregled nekih svjetskih istraživanja na temu korištenja tehnologije virtualne i proširene stvarnosti u obrazovanju. Istraživanja prikazana u ovom radu samo su dio svjetskih istraživanja, ali obuhvaćaju pregled osnovnih tema kao što su prednosti i nedostatci korištenja, iskustva nastavnika u korištenju, dvojbe i izazovi s kojima se susreću kako nastavnici tako i učenici i studenti.

Ključne riječi: tehnologija virtualne stvarnosti, tehnologija proširene stvarnosti, obrazovanje

Uvod

Tehnologija je svojim napretkom uvijek pronalazila primjenu u odgojno-obrazovnom radu, kao i načine ulaska u ustanove predškolskog odgoja, osnovnoškolskog i srednjoškolskog odgoja i obrazovanja te ustanova visokoškolskog obrazovanja. Visokoškolsko obrazovanje često je bilo i predvodnik implementacije, a u velikom broju slučajeva i stvaratelj modernih informacijsko-komunikacijskih tehnologija.

Povijest nas često uči kako su upravo fakulteti bili rasadnik novih ideja korištenja moderne tehnologije. Implementacija informacijsko-komunikacijskih tehnologija u nastavni proces, bilo kao predmet poučavanja ili kao nastavno sredstvo i pomagalo, ima svoju važnost prije svega radi odgovornosti odgojno-obrazovnih ustanova u pripremi učenika i studenata za ravnopravno sudjelovanje u tržištu rada, koje je te tehnologije najčešće već prigrlilo. Upravo zbog toga najvažniji strateški dokumenti Republike Hrvatske, Europske unije i svijeta prepoznaju znanja i vještine rada sa i na modernim informacijsko-komunikacijskim tehnologijama kao jedno od prioritetnih

iskoraka u obrazovanju pojedinca radi produktivnog sudjelovanja u stvaranju društva blagostanja. Strategija Europa 2020. spominje korištenje informacijsko-komunikacijskih tehnologija kao preduvjet za pametan, održiv i uključiv rast.

Digital Education Action Plan kroz 11 aktivnosti upućuje zemlje kako osigurati podršku korištenju tehnologije i razvoju digitalnih kompetencija u obrazovanju. Dokument The 2018 International Computer and Information Literacy Study (ICILS) napominje kako se digitalne kompetencije ne stječu rođenjem u digitalnom svijetu jer postoji jaz u dostupnosti opreme i načinima njenog korištenja među zemljama Europske unije te se i učenici i nastavnici moraju ohrabrivati u korištenju tehnologije u školama. Integracija tehnologije po mjeri učenika, nastavnika i drugih zaposlenika u školama u Republici Hrvatskoj predviđena je i dokumentom Ministarstva znanosti i obrazovanja iz ožujka 2020. godine Strateški okvir za digitalno sazrijevanje škola i školskog sustava u Republici Hrvatskoj (2030.) koji proizlazi iz Strategije obrazovanja, znanosti i tehnologije koju je Hrvatski sabor usvojio 2014. godine. Još je mnogo domaćih i stranih strateških dokumenata, akcijskih planova i agendi koji prepoznaju važnost korištenja modernih informacijsko-komunikacijskih tehnologija u cijeloj odgojno-obrazovnoj vertikali. Korištenje tehnologije virtualne i proširene stvarnosti jedan je od posljednjih tehnoloških iskoraka u odgojno-obrazovnom sustavu. Upravo zbog toga sve češće postaje predmet interesa istraživača, posebno u kontekstu koristi od njene primjene u odgojno-obrazovnom radu.

Tehnologija virtualne i proširene stvarnosti

Oba su pojma poznatija prema svojim akronimima nego po punom nazivu. Akronim VR od engleskog naziva Virtual Reality označava virtualnu stvarnost, a proširenu stvarnost akronim AR od engleskog Augmented Reality. Bez obzira na svoje sličnosti drastična je razlika u ova dva pojma. Tehnologija virtualne stvarnosti podrazumijeva potpuno „uranjanje“ korisnika u 3D svijet koji ga okružuje, dok je korištenjem tehnologije proširene stvarnosti korisnik svjestan okoline u kojoj se nalazi, ali je ona „obogaćena“ računalno izrađenim elementima. Tehnologija proširene stvarnosti najveću je popularnost doživjela igricom za mobilne telefone Pokemon Go koja je ostvarila više od milijardu preuzimanja. Aplikacija je koristila kameru mobilnog telefona korisnika prikazujući prostor u kojem se on nalazi, a tehnologijom proširene stvarnosti pojavljuju se u tom prostoru Pokemoni koje korisnik „hvata gađajući ih“ na ekranu mobilnog telefona virtualnom lopticom. Fernandez (2017.) opisujući razliku između tehnologije virtualne i proširene stvarnosti ističe kako je virtualna stvarnost 10% stvarna i 90% virtualna, a proširena 75% stvarna i 25% virtualna. Autor upozorava kako ne smijemo smetnuti s uma da se cilj obrazovanja nije promijenio bez obzira na tehnologiju koju koristimo.

Poduzeće IDTechEx procjenjuje kako će 2030. godine tržište virtualne i proširene stvarnosti vrijediti 30 milijardi američkih dolara i to prije svega u gaming industriji.

Aplikacije koje se trenutačno mogu pronaći na tržištu, a namijenjene su ili se mogu koristiti u školama i na fakultetima, možemo podijeliti na one koje su izrađene kao dio nekomercijalnih projekta, financiranih javnim sredstvima i aplikacije izrađene od strane komercijalnih poduzeća koje ih pokušavaju monetizirati.

VR i AR u obrazovanju – pregled istraživanja

Liu, Kumar Bhagat, Gao, Chang i Huang (2017.) proučavali su broj radova na temu virtualne stvarnosti od 1995. do 2016. godine i zaključili kako je interes istraživača za tu temu bio eksponencijalan, a najviše su zanimanja pokazali autori iz Sjedinjenih Američkih Država, Ujedinjenog Kraljevstva i Tajvana, tehnološki među najnaprednijim zemljama svijeta. Autori upućuju istraživače na buđenje interesa prema temama implementacije tehnologije virtualne i proširene stvarnosti u odgojno-obrazovni sustavu i proučavanje implikacija tog procesa.

Na temelju mnogobrojnih istraživanja Velev i Zlateva (2017.) procjenjuju kako je obrazovanje jedno od pet područja u kojima će tehnologija virtualne stvarnosti napraviti najveći prodor. S druge strane, autori Kavanagh, Luxton-Reilly, Wuensche i Plimmer (2017.) u svom radu ističu kako logistička pitanja vezana za implementaciju tehnologije virtualne stvarnosti te njena cijena i limitiranost sprječava veći prodor u obrazovni sustav, ali je neminovno da će se on dogoditi.

Većina je autora istraživala prednosti korištenja tehnologije virtualne i proširene stvarnosti u školama i na fakultetima te stavove nastavnika, učenika i studenata prije i nakon korištenja neke od aplikacija. Povećanje motivacije i angažiranosti studenata i konstruktivistički pristup učenju zabilježili su u svojim istraživanjima Martín-Gutiérrez, Efrén Mora, Añorbe-Díaz i González-Marrero (2016.), a do sličnog zaključka došao je i Boyles (2017.). Aplikacije za virtualnu stvarnost koje se koriste u obrazovanju proučavali su Kamińska, Sapiński, Wiak, Tikk, Haamer, Avots, Helmi, Ozcinar i Anbarjafari (2019.). Kritičkom analizom aplikacija autori su izdvojili prednosti poput stvaranja vizualizacija koju ni jedna druga tehnologija trenutačno ne omogućava, ali s druge strane tehnologija onemogućava fleksibilnost klasične učionice u kojoj učenici mogu postavljati pitanja i sudjelovati u raspravama.
Yildirim, Elban i Yildirim (2018.) su kroz intervjue s 25 studenata prve godine na kolegiju Povijest civilizacija zaključili kako je tehnologija virtualne stvarnosti korisna i dopadljiva za korištenje. Istraživanje Dominga i Bradleya

(2017.) pokazalo je da su svi studenti nastavničkog smjera umjetnosti imali pozitivno mišljenje o tehnologiji virtualne stvarnosti nakon korištenja, a Dyer, Swartzlander i Gugliucci (2018.) svojim su istraživanjem kod studenata medicinske struke pokazali kako ova tehnologija može pomoći u razvoju empatije povezane s radom s osobama starije životne dobi.

Stavove nastavnika fizike ispitivali su Pirker, Holly, Almer, Gutl i Winston Belcher (2019.). Istraživanje je pokazalo kako su nastavnici najviše zabrinuti za vrijeme koje je potrebno za pripremu tehnologije za rad i zbog toga su najviše zainteresirani za njeno povremeno korištenje. Isto tako postoji bojazan da se učenici više usmjere na samu tehnologiju, a manje na sadržaj koji moraju naučiti. Cho i Chun (2019.) na kraju svog istraživanja zaključuju kako je potrebno puno više stručnih usavršavanja za nastavnike u području korištenja tehnologije virtualne stvarnosti.

Kako se stvari i u tom području mijenjaju pokazuje i CARNET-ova konferencija za korisnike – CUC koja nekoliko zadnjih godina nastavnicima praktičarima nudi sadržaje vezane za korištenje tehnologije virtualne i proširene stvarnosti u učionicama. Praktičari kroz primjere dobre prakse iz svojih učionica na najbolji način šire prednosti ove tehnologije i mogućnosti njene primjene.

Zaključak

Implementacija tehnologije virtualne i proširene stvarnosti u obrazovanju neminovno je sljedeći iskorak u kontekstu korištenja nastavnih sredstava i pomagala u svakodnevnom radu u učionicama i fakultetima. Istraživanja pokazuju značajne prednosti, posebno u onim područjima u kojima vizualizacija predmeta predavanja nije moguća bez upotrebe tehnologije.

Potpuno uranjanje u svijet virtualne stvarnosti omogućuje učenicima i studentima uvid u stvarnost koja ih okružuje kao nikada do sada. Iako postoji bojazan nastavnika, posebno jer istraživanja pokazuju nedostatak stručnih usavršavanja, sve je više konferencija i stručnih usavršavanja na temu implementacije ove tehnologije u učionicama.

Istraživanja ističu problem usmjerenosti na samu tehnologiju, a ne sadržaj poučavanja, ali će češćim korištenjem i taj problem izostati. Motiviranost učenika za sadržaj poučavanja i uspjeh u savladavanju tog sadržaja zabilježen je u svim istraživanjima, a to i je osnovni cilj implementacije tehnologije virtualne i proširene stvarnosti u školama i fakultetima.

Reference

[1] Boyles, B. (2017). Virtual Reality and Augmented Reality in Education. Center for Teaching Excellence, United States Military Academy, West Point, NY.

[2] Cho, D. i Chun, B. A. (2019). Virtual Reality as a New Opportunity in Geography Education: From the teachers' perspectives in Korea.

[3] Domingo, J. R. i Bradley, E. G. (2017). Education Student Perceptions of Virtual Reality as a Learning Tool. Journal of Educational Technology Systems 46(3), 329-342.

[4] Dyer, E., Swartzlander, B. i Gugliucci, M. R. (2018). Using Virtual Reality In Medical Education To Teach Empathy. Journal of the Medical Library Association, 106 (4), 498-500.

[5] European Commission. Digital Education Action Plan (2021-2027).

[6] European Commission. The 2018 International Computer and Information Literacy Study (ICILS)

[7] Fernandez, M. (2017). Augmented Virtual Reality: How to Improve Education Systems. Higher Learning Research Communications, 7 (1), 1-15.

[8] Kaminska, D., Sapinski, T., Wiak, S., Tikk, T., Haamer, R.E., Avots, E., Helmi, A., Ozcinar, C., i Anbarjafari, G. (2019). Virtual Reality and Its Applications in Education: Survey. Inf., 10, 318.

[9] Kavanagh, S., Luxton-Reilly, A., Wuensche, B. i Plimmer, B. (2017). A systematic review of Virtual Reality in education. Themes in Science and Technology Education, 10(2), 85-119.

[10] Liu, D., Kumar Bhagat, K., Gao, Y., Chang, T-W. i Huang, R. (2017). A Bibliometric Analysis on Top Research Studies in the Last Two Decades. U D. Liu, C. Dede, R. Huang i J. Richards (Ur.). Virtual, Augmented and Mixed Realities in Education. Springer Singapore.

[11] Martín-Gutiérrez, J., Efrén Mora, C., Añorbe-Díaz, B i González-Marrero, A. (2016). Virtual Technologies Trends in Education. EURASIA Journal of Mathematics Science and Technology Education, 13 (2), 469-486.

[12] Ministarstvo znanosti i obrazovanja. Strateški okvir za digitalno sazrijevanje škola i školskog sustava u Republici Hrvatskoj.

[13] Pirker, J., Holly, M., Almer, H., Gutl, C i Winston Belcher, J. (2019). Virtual Reality STEM Education from a Teacher's Perspective. U iLRN 2019 London, Workshop, Long and Short Paper, and Poster Proceedings: from the Fifth Immersive (5 ed.). Verlag der Technischen Universität Graz.

[14] Velev, D. i Zlateva, P. (2017). Virtual Reality Challenges in Education and Training. International Journal of Learning and Teaching, 3(1).

[15] Yildirim, G., Elban, M. i Yildirim, S. (2018). Analysis of Use of Virtual Reality Technologies in History Education: A Case Study. Asian Journal of Education and Training, 4 (2), 62-69.

STRUČNA KONFERENCIJA

RAČUNALNE IGRE 2020

30. prosinca 2020.