Izrada dvodimenzionalnog grafičkog okruženja i razvoj interaktivne računalne igre

Prpić, Lucijan

SVEUČILIŠTE U ZAGREBU

GRAFIČKI FAKULTET

LUCIJAN PRPIĆ

IZRADA DVODIMENZIONALNOG GRAFIČKOG OKRUŽENJA I RAZVOJ INTERAKTIVNE RAČUNALNE IGRE

DIPLOMSKI RAD

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU

GRAFIČKI FAKULTET

LUCIJAN PRPIĆ

IZRADA DVODIMENZIONALNOG GRAFIČKOG OKRUŽENJA I RAZVOJ INTERAKTIVNE RAČUNALNE IGRE

DIPLOMSKI RAD

Mentor:
doc.dr.sc. Tibor Skala

Student:
Lucijan Prpić

Zagreb, 2021.

SAŽETAK

Posljedica sve većeg i bržeg razvoja industrije igara jest njena dostupnost širem krugu ljudi bez većih troškova izrade ili velikog budžeta potrebnog za njihovu distribuciju. Upravo se zbog te sve veće dostupnosti rađaju i novi načini primjene igara gdje focus ne mora biti samo na igrivoj vrijednosti već i na ostalim funkcijama koje mogu biti ukomponirane u ostalim granama različitih struka. Potencijal igara da budu nešto više se zadnjih godina eksponencijalno povećao zahvaljujući već spomenutoj većoj dostupnosti, tehnologiji i razvitku ostalih medija. Razvijanjem spomenutih tehnologija i medija dolazi do sve većeg razvoja pogonskih sklopova (eng. *game engine*). Jedan od tih pogonskih sklopova jest Unity. Korištenjem programa Unity svatko može (s određenim znanjem) i bez velikih troškova izraditi potpuno funkcionalnu igru. Razvojem multimedije kao i ostalih grafičkih alata (Photoshop, Illustrator, Clip studio paint) mogu se izraditi kreativni projekti s različitim svrhama koji imaju potencijala produbiti grafičku tehnologiju predstavljajući nove potencijalne smjerove u kojima se ona može kretati. Korištenjem navedenih programa stvaraju se elementi međusobno povezani logikom stvorenom programskim jezikom #C pomoću programa Visual Studio Code koji služi kao integrirano razvojno okruženje, te je partnerska aplikacija Unity-ju. Visual Studio Code ima ulogu platforme na kojoj se pišu kodovi za skripte korištene u Unity-ju prilikom razvoja igre. Istražit će se na koje se sve načine projekt može iskoristiti kako bi ispunio svoj puni potencijal u grafičkoj tehnologiji kao i njegova namjena na tržištu. Svaka faza izrade bit će potkrijepljena slikama izrade željenog projekta i objašnjena u dijelovima gdje je za svaki dio potrebno znanje različitih polja tehnologije te je svaki dio jednako bitan u razvoju dvodimenzionalnog grafičkog okruženja što će rezultirati funkcionalnom igrom. Od planiranja koncepta, izrade elemenata i programiranja, do izrade animacije i dijaloga potrebnih za potpuno igrače iskustvo. Cilj ovog rada jest stvoriti 2D grafičko okruženje koje osim iskustva same igre i priče daje igraču određene beneficije koje se mogu iskoristiti i u stvarnom svijetu napretkom kroz igru. Ovo grafičko okruženje poslužit će kao sredstvo koje može biti iskorišteno u svrhe promocije, kao i marketinških planova gdje se prolaskom spomenute igre dobiva popust pri kupnji određenog proizvoda. Isto tako se može iskoristiti kao i interaktivna galerija radova gdje se korisnik pritiskom miša može kretati po prostoriji i interaktirati sa slikama, rješavati zagonetke, i biti dio multimedijalnog svijeta.

Ključne riječi: Računalne igre, Unity 2D, Clip studio paint, Photoshop, Illustrator, Visual studio code, Korisničko sučelje

SADRŽAJ

1. UVOD

Danas je popularnost 2D indie igara sve više u porastu. Zbog zasićenja AAA igara na tržištu sve je veća potražnja za igrama koje svojom inovativnošću pokazuju da se zanimljiva igra može napraviti bez velikih financijskih ulaganja. Takve indie igre imaju ogroman potencijal u svojoj primjeni. Mogu se koristiti kao izvor zabave, sredstvo za učenje, te imati svakakve virtualne funkcije (portfolio, virtualna kolekcija, reklama, promocije…). Kroz ovaj rad se želi prikazati kako se kombinacijom više različitih softvera i tehnika može izraditi interaktivno 2D okruženje namijenjeno zabavi i primijeniti isto za razvoj računalne igre. Softverom Unity predstavit će se funkcionalnost korištenje digitalnih kreacija kako bi se napravila funkcionalna 2D indie igra koja će imati sve potrebne komponente i elemente nužne za užitak igranja, poput zvuka, izgleda, logike i priče. U ovom radu će se također istražiti više suvremenih revolucionarnih primjena koje bi mogle imati velikog potencijala u budućnosti. Određene igre već se koriste u psihologiji gdje služe kao vrsta terapije u liječenju ptsp-a, raznih anksioznih poremećaja i trauma, kao i simulacija događaja unosom određenih parametara što uvelike pomaže kod gradnje raznih objekata u građevinskoj industriji. Na utrkama konja više ne moraju sudjelovati pravi konji, već to može biti simulacija ovisna o raznim spontanim algoritmima prerušenim u računalno generirane konje na koje se ljudi mogu kladiti. Na taj se način i sama industrija igara na sreću razvija paralelno s novim spoznajama u igraćoj industriji. Medicina je još jedna važna grana na koju utječe razvoj igara. Pogonski sklopovi dosežu toliki visok stupanj razvoja da se ljudsko tijelo može reproducirati u potpunosti, do najsitnijih detalja. Simulacije mogu prikazati kako izgleda unutrašnjost ljudskog tijela pogođenog raznim bolestima gdje se tada može selektivno odabrati vrsta bolesti kako bi budući kirurg naučio ispravno postupati u određenom scenariju i učiti na svojim greškama bez opasnosti. Piloti, astronauti, vojnici, pa čak i umjetnici željni inspiracije mogu pokretom miša, kontrolera, ili čak pokretom ruke u virtualnoj stvarnosti (VR) zakoračiti u simulirani svijet koji ih priprema na stvarne događaje s kojima će se morati suočiti i isto tako im omogućiti okruženje gdje mogu činiti greške i učiti na njima kako bi u budućnosti nekome mogli spasiti život.

2. TEORIJSKI DIO

U ovome poglavlju će se pobliže objasniti značaj Indie igara te njihov utjecaj na igraču industriju kao i igre koje su tu industriju najviše izmijenile. Razmotrit će se psihološki faktori kod igara i što ih čini toliko privlačnima kao i tehnologija koja to sve omogućuje.

2.1.1. Indie igre

Indie igre pronalaze svoj početak na računalima, na čijoj su platformi mnoge i ostale. Većina indie igara započinje kao *Shareware*[1]. U ranim 90-ima, novi žanrovi indie igara kao i njihova zajednica dugo ostaju kao nepoznanica u mainstream-u, sve do polovice 21. stoljeća. Kada su indie igre dosegle određenu razinu popularnosti razvojni inženjeri počinju prepoznavati njihov potencijal te se većina njih odlučuje prebaciti na Indie tržište i započeti razvoj svojih vlastitih indie igara dok ostali početak svoje karijere vide u indie igrama alternativu sve više zakrčenom tržištem *mainstream* video igara. Bez obzira na početak indie igara, sve više i više proizvođača započinje rad na svojim neovisnim naslovima te njihova popularnost raste iz godine u godinu. Danas indie igre dominiraju tržištem, donoseći zaradu ponekad veću i od samih AAA igara. Indie igre se danas oslobađaju uzdi svojih skromnih početaka te se počinju proširivati i na ostale platforme (konzole, mobilne uređaje), s raznim virtualnim tržištima koji sada sadrže posebnu sekciju za indie igre ili jednostavno imaju indie igre u svojim katalozima. Indie razvojni inženjeri nastavljaju donositi svježe ideje u ustaljenom i repetitivnom tržištu te se ne vidi njihovo opadanje popularnosti u skorije vrijeme. Naprotiv, indie igrama s vremenom sve više raste popularnost. Bez obzira na žanr, sadržaj ili temu, indie razvojni inženjeri su predodređeni napraviti nešto posebno i inspirativno u budućnosti.

[1] podrazumijeva način distribucije proizvoda- najčešće software-a - na način kojem je kupcu omogućeno prethodno isprobavanje proizvoda prije nego isti bude plaćen. Javno je dostupan, i besplatan određeno vrijeme

2.1.2. Popularnost Indie igara

Velike kompanije igara često se nalaze u situaciji u kojoj imaju potrebu prilagođavati se svojim fanovima kod proizvodnje novih naslova čime su ograničeni što se tiče ideja i inovativnih rješenja. Suprotno tome, indie razvojni inženjeri nemaju takva ograničenja te su slobodni raditi što god im je na umu. Primarni primjer takve kreativne slobode jest igra Minecraft proizvođača Mojang izdana 2011. godine [1]

2.2 Pogonski sklop igre

Temelj svake igre jest pogonski sklop igre. Pogonski sklop je srž svake igre te je potreban kako bi se igra uspješno pokrenula. [2] Pogonski sklop igre je softver (eng. *software*) alat namijenjen smanjenju troškova i kompleksnosti igre. Ti softver alati stvaraju sloj apstrakcije te su povezani tako da funkcioniraju kao interoperabilne komponente koje se mogu zamijeniti ili proširiti s dodatnim komponentama. Pogonski sklopovi predstavljaju nevjerojatnu efikasnost smanjujući količinu predznanja potrebnog za izradu igre. Funkcionalnost takvih sklopova može se kretati od minimalne do potpune funkcionalnosti te tako daju veliku slobodu tvorcima igara (tvorac se može potpuno usredotočiti na pisanje koda u igri). Pogonski sklopovi daju veliku prednost u odnosu na izradu igre iz temelja što je potrebno Indie razvojnim inženjerima koji se samo žele usredotočiti na izradu igre. [3]

2.2.1. Pogonski sklopovi igre danas

Današnji moderni studiji za AAA igre kao što su Bethesda Game Studios i Blizzard Entertainment često imaju svoje vlastite pogonske sklopove. Bethesdina, naziva: Creation Engine, korištena je kako bi se izradila igra The Elder Scrolls V: Skyrim kao i Fallout 4. Blizzard ima svoj vlastiti pogonski sklop korišten u izradi igre World of Warcraft, kao i Overwatch. [3]

Skoro svaki originalni pogonski sklop igre svoj početak nalazi kao sklop napravljen samo za tu jednu igru. Nakon izlaska projekta na tržište, pogonski sklop može biti obnovljen (ponovno se koristiti) kada se upotrebljava za izradu sljedeće igre istog

razvojnog studija. Takav će se pogonski sklop možda morati unaprijediti kako bi ostao relevantan i ugrabio prednosti današnje tehnologije, no nikada se neće trebati početi iz temelja. [3]

Ako kompanija igara nema svoj vlastiti pogonski sklop, tada se tipično uzima otvoreni kod (eng. *open-source*) pogon, ili pogonski sklop licenciran od strane drugog proizvođača kao što je Unity. Proizvođačke firme znaju odvojiti programerske timove koji su posvećeni isključivo izradi pogonskih sklopova kao i njihovom optimizacijom.

2.2.2. Pogonski sklop *Unity*

Unity je ekstremno popularan pogonski sklop koji nudi veliku prednost nad ostalim pogonskim sklopovima danas dostupnima na tržištu. Nudi vizualni tijek rada (eng. *workflow*) s vuci i spusti (eng. *drag and drop*) funkcijama te podržava skriptiranje u C# jeziku. Unity podržava 2D i 3D grafiku, te set sofisticiranih alata prilagođenim korisniku (eng. *user friendly*) sa svakom sljedećom inačicom. Unity ima par razina kod kupnje licence, te je besplatan za projekte sa zaradom do 100000 dolara. Nudi izradu igara za više platformi te iskorištava grafički API[2] specifičan arhitekturi sustava, uključujući Direct3D[3], OpenGL[4], Vulkan[5], Metal[6], i nekoliko ostalih.[12] Unity teams nudi kolaboraciju projekata i na oblačnom računalstvu (eng. *Cloud*). Unity je od svog nastanka 2005. zaslužan za razvoj tisuća desktop, mobilnih te konzolnih igara i aplikacija. Neke od igara koje su nastale na bazi Unity pogonskog sklopa su: Thomas Was Alone (2010), Temple Run (2011), The Room (2012), RimWorld (2013), Hearthstone (2014), Kernal Space Program (2015), Pokémon GO (2016), i Cuphead (2017). [3]

---

[2] skup određenih pravila i specifikacija koje programeri slijede tako da se mogu služiti uslugama ili resursima operacijskog sustava ili nekog drugog složenog programa kao standardne biblioteke rutina (funkcija, procedura, metoda), struktura podataka, objekata i protokola.

[3] Direct3D je grafičko programsko sučelje za programiranje za Microsoft Windows. Dio DirectX-a, Direct3D koristi se za prikazivanje trodimenzionalne grafike u aplikacijama gdje su performanse važne, poput igara.

[4] OpenGL je multi-jezično, više platformsko aplikacijsko sučelje za prikazivanje 2D i 3D vektorske grafike. API se obično koristi za interakciju s grafičkom procesorskom jedinicom, kako bi se postiglo hardversko ubrzano prikazivanje.

[5] Vulkan je API s više platformi, otvoreni standard za 3D grafiku i računarstvo. Vulkan cilja 3D grafičke aplikacije visokih performansi u stvarnom vremenu, poput video igara i interaktivnih medija.

2.3 Faktori koji čine dobru igru

Industrija igara je jedna od najprofitabilnijih industrija prešavši filmsku industriju. Kada se gleda profitabilnost, smještena je jedanaesta na svijetu i prva u Latinskoj Americi. U ovom tekstu će se proći razlozi privlačnosti tih igara. Tržište igara se dijeli na pet kategorija: online, mobilne igre, konzolne igre, PC igre, i internetske igre. Konzolne i PC igre imaju puno veću razinu kompleksnosti i težine, dok ležerni (eng. casual) igrači preferiraju jednostavnije i lakše igre. [8] Upravo zbog tog razloga postoje različiti tipovi igara, s različitim zadacima i preprekama, zbog kojih je bitno razumjeti razlog koji motivira potrošače za kupnju igre koja njima najviše odgovara.

2.3.1. Glavni razlozi zašto potrošači kupuju igru

Igrači provode sate i sate igrajući razne igre. Što je razlog njihovog učestalog vračanja tim iskustvima? Istraživači diljem svijeta godinama mjere utjecaj igara društvo (ohrabljivanje ili obeshrabrivanje za nasiljem, inspiracija za kreativnost, ili poticanje lijenosti).[11] [12]U sljedećem tekstu razmotrit će se razlozi koji utječu na želju za igrom.

1.) Potrošači potražuju igre radi stimulacije osjećaja, u čemu su računalne igre jako dobre. Igrači su generalno na oprezu i usredotočeni na igru koju igraju. Njihovi umovi su usredotočeni samo na zadatak pred njima, što im kratkoročno oduzme pažnju od ostalih podražaja. Ovakvo stanje svijesti može podići razinu adrenalina, krvnog tlaka i lupanja srca. Svi ovi simptomi su rezultat osjećaja uzbuđenja koji igrači osjećaju u tom trenutku.

2.) Motivacija je slijedeći razlog zbog kojeg se potrošači upuštaju u određenu igru. Motivacija za ispunjenjem izazova postavljenog od strane igre kako bi ispunili određene ciljeve koja ih igra traži. Izazov je veliki dio svake igre i jedan od glavnih razloga interakcije između igrača i igre. Izazov mora biti balansiran kako ne bi uništio motivaciju igrača za igranjem, dok pre lagane igre stvaraju dosadu. U prvim igrama glavni sastojak bio je izazov. Dosta proizvođača se trudilo kako bi napravili nemoguću igru zbog koje bi igrač, nakon prelaska takve igre, dobio osjećaj postignuća zbog toga što je jedan od rijetkih koji je to uspio.

3.) Izazov i kompetitivnost također pozitivno utječu u namjeri za igranjem. Igre se koriste kako bi odvratili pažnju od nečega, i zbog toga se na njih gleda kao alat za relaksaciju, kao i bijeg od svakidašnjih problema i obaveza. Učenjaci ih nazivaju diverzijom koja je najefektivnija u ljudima koji traže bijeg od svakidašnjih zadataka i smanjenje stresa u formi zabave. Upravo zbog tog efekta privlačenja pažnje, i oslobođenja od svakidašnjih zadataka neki ljudi gledaju na igre kao ovisnost. Ovakva konstrukcija ima veći utjecaj na starije igrače za razliku od mlađih što čini veliki faktor u razlikovanju različitih profila igrača. Ovaj uvjet je baza sljedećeg razloga.

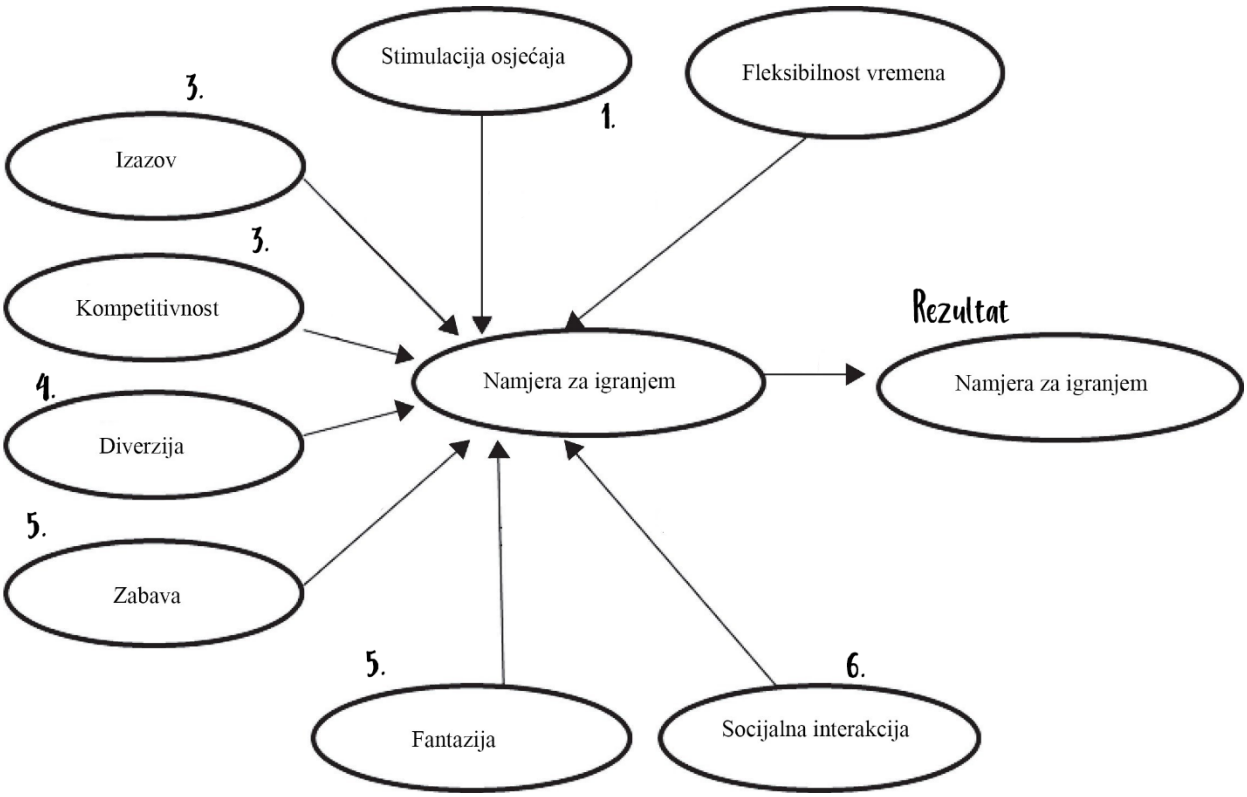
4.) Diverzija pozitivno utječe na intenciju za igrom. Igre se igraju radi zabave, osjećaja postignuća i užitka. Igranje i kontroliranje lika u igri, ili osjećaja posebnosti može biti glavni razlog utjecaja igre na tržištu. Očito je da igre koje ne zabavljaju neće biti privlačne, no bitno je pretpostaviti da je zabava jako subjektivan pojam koji se drastično mijenja od osobe do osobe zato što se bazira na odnosu između igrača i igre.

5.) Zabava pozitivno utječe na intenciju za igrom. Mogućnost kreiranja imaginarne uloge je prepoznato u literaturi kao potencijalna motivacija za igrom. Mogućnost ispunjenja snova, kao što je biti superjunak, voziti formulu 1, ili jednostavno proživljavati drukčiji život kroz *role play* (igranje uloga) igre čini ih jako privlačnima zbog toga što igrači mogu ispuniti svoje fantazije. Kroz fantaziju, nudi se bijeg iz stvarnog života.(5) Još jedan jako bitan dio onoga što igre nude jest mogućnost kreacije vlastitih okruženja ili priča koje mogu nalikovati onima iz Hollywoodskih filmova. Tako igre služe kao alat za usmjeravanje mašte i kreativnosti.

6.) Socijalna interakcija također je prepoznata kao važna motivacija za igrače, koji cijene dijeljenje iskustva s ostalim ljudima, uspostavljanje veze, prijateljstva, kao i osjećaj pripadanja određenoj zajednici. Drugim riječima, interakcija s ostalim igračima motivira igrača, što pokazuje kako je socijalna interakcija važan faktor koji se mora razmotriti prilikom razvoja igre. Ovim se dokazuje kako socijalna interakcija u igrama nadilazi samu igru u smislu informacija koje se dijele između dva igrača. U ovom primjeru se vidi kako igre u puno slučajeva mogu premašiti svoju prvotnu namjenu što će se u ovome radu dodatno istražiti. Socijalna interakcije je najizraženija tamo gdje ima kompetitivnosti, iako nadilazi samu kompetitivnost zato što nekada igrači moraju raditi zajedno kako bi prošli određenu prepreku. Ovaj sistem stvara uvjete koji sadržavaju i kompetitivnost, kao i

kooperaciju. Ukratko, može se zaključiti kako je socijalna interakcija i komunikacija među igračima ključan faktor u izradi igre.

Na slici 1 može se vidjeti konceptualan okvir koji sadržava grafički prikaz teksta iznad gdje se prikazuje kako različiti faktori utječu na intenciju, to jest želju ili namjeru za igrom.[9][10]



Slika 1. Grafički prikaz intencije za igrom to jest zašto potrošači kupuju igre objašnjeno u tekstu iznad.Izvor(https://www.sciencedirect.com/science/article/pii/S0080210717301838#bib0245)

2.3.2 Primjena video igara u drugim područjima

U tekstu prije dalo se dokučiti kako igre imaju potencijalnu primjenu i u drugim područjima te mogu pružiti mnogo više od same zabave. U ovom dijelu navest će se ostale primjene igara od kojih je jedna primjena praktični dio ovoga rada.

2.3.3. Potencijal igara kao pedagoški alat za učenje

Dizajn video igara je u svojoj osnovi baziran na tome da bude izvor učenja. Mnoge igre započinju s praktičnim vodičem (eng. *tutorial*), to jest treningom koji služi kao uvod u igru i njezine alate. Igrač kroz igru prolazi razne strategije i taktike koje su s vremenom sve složenije. Igra suptilno prelazi iz treninga u složene prepreke gdje je bitno iskustvo i znanje stečeno u samoj igri. U osnovi igre uče igrače kako kritički razmišljati i postupati sa svakom situacijom u kojoj se nađu kako bi našli najbolje rješenje. Primjer takvog učenja je igra Starcraft (kao i njeni prethodnik Starcraft: Brood War). To su igre koje spadaju u kategoriju strategija u stvarnom vremenu (eng. *real time strategy*) gdje igrač mora priskrbiti razne resurse što mu omogućuje kupnju raznih letjelica, vojnika i zgrada kako bi strateškim razmišljanjem pobijedio neprijatelja. Kako bi to postigao, igrač mora znati raspolagati vremenom i resursima bolje i efektivnije od protivnika kako svaka zgrada, i borbena jedinica imaju svoju cijenu, svrhu, i vrijeme gradnje. Baš kao i igra Pokemon, jedinice imaju svoje prednosti i slabosti. Igrač mora konstantno ažurirati svoje strategije kroz igru bazirane na njihovim interakcijama s ostalim igračima, što zahtjeva jako puno planiranja i sposobnosti kritičkog razmišljanja koji se dobivaju iskustvom kroz vrijeme. Kao primjer može se uzeti podatak kako prosječni početnik u igri Starcraft radi manje od 100 akcija po minuti dok ih profesionalac napravi 400. Na slici 2 i slici 3 može se vidjeti primjer bitke s jedinicama u igri. Unatoč dokazima za edukacijske svrhe igara postoji zamjerka da se igre koriste za učenje igrača informacijama koje vrijede samo u toj igri i ne uče edukacijski bitan materijal koji bi mogao poslužiti u stvarnom svijetu. Dokaz kako i ta stavka nije točna jest nedavno istraživanje na području digitalno baziranog učenja koje se vršilo nad studentima povijesti kojima je bilo zadano igrati povijesnu simulaciju naziva Civilization 3 sa svrhom da studenti nauče o povijesti tako da igraju igru. Bilo bi potrebno spomenuti kako sama igra nije dizajnirana kao edukacijska igra.[4][5] U ovoj igri zadatak studenata bio je preuzeti kontrolu nad jednom civilizacijom, i napredovati kroz povijest odabrane civilizacije što uključuje razvoj tehnologije, sudjelovanje u simuliranim ratovima kao i raspolaganje ekonomijom njihovih carstava. Istraživanja su pokazala da igranje ove igre za edukacijske svrhe pomaže studentima koji imaju problema s tradicionalnim učenjem, iako studenti koji inače imaju dobre rezultate ipak preferiraju tradicionalne metode učenja.[5]

Slika 2. Prikaz igre Starcraft. Izvor (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4179712/)

Na slici 2 se može vidjeti prikaz igre *Starcraft* u fazi taktiziranja i izvršavanja taktike na polju. Ovdje se igrači koriste tehnikama mikromenadžmenta (eng. *micro management*) gdje se jedinicama u igri upravlja tako da se točno zna što koja jedinica radi. Općenito se u ovakvim igrama upravlja s grupom jedinica, no neke igre zahtijevaju mikromenadžment kako bi igrač postigao prednost nad drugim igračima.

Slika 3. Prikaz igre Starcraft. Izvor (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4179712/)

2.3.4. Primjena igara u svrhe planiranja, modeliranja i simulacija događaja u stvarnom životu

Igra *Sim City*[7], napravljena od strane Will Wrighta 1989 bila je jedinstveno hvaljena od strane geografa, urbanih planera, i profesora. Osim same igrive vrijednosti, s ovom serijom igara često je povezana i mogućnost edukacije. U Americi postoji nekoliko urbanih studija i vježbi gradskih planera koji koriste video igre (*Adams 1998, Squire 2004, Gaber 2007, Gordon & Koo 2008, Nesson & Nesson 2008*). Video igre mogu biti heuristički, kao i didaktički alati. Alati za modeliranje također sve više liče video igrama. Grafički modeli video igara koriste se kako bi unaprijedili njihova korisnička sučelja. Neki od softvera za modeliranje su se pretvorili u video igre zbog didaktičke svrhe (npr MacSim[8]). Igre slične Sim City-ju bliže su alatima za modeliranje zbog samog procesa, a ne zbog sličnog sučelja. Koriste slične modele zasnovane na jednostavnim spacijalnim modelima (gravitacija, difuzija, periferija, itd) koji zajedno dovode do razvijanja formi spacijalne organizacije mezo ili makro razine. Također nude dijagnostičke alate kako bi procijenili proces u tijeku i svoje prostorne rezultate. Igrač može interaktirati s fizičkom okolinom i destinacijom tih

[7] otvorena serija videoigara o gradnji gradova
[8] makroekonomski software za simulaciju

ćelija (geografski prostor), ali ne utječe na njihov daljnji napredak, to jest na broj aktivnosti u tim "ćelijama". Na slici 4 se može vidjeti prikaz konstrukcije zgrada u igri Sims, kao i prikaz njenog sučelja.

Slika 4. Dijagnostički alati u igri SimCity.
Izvor (https://www.lifewire.com/simcity-4-starting-new-city-840147)

2.3.5. Korištenje video igara u svrhu razvitka boljih algoritama za strojno učenje

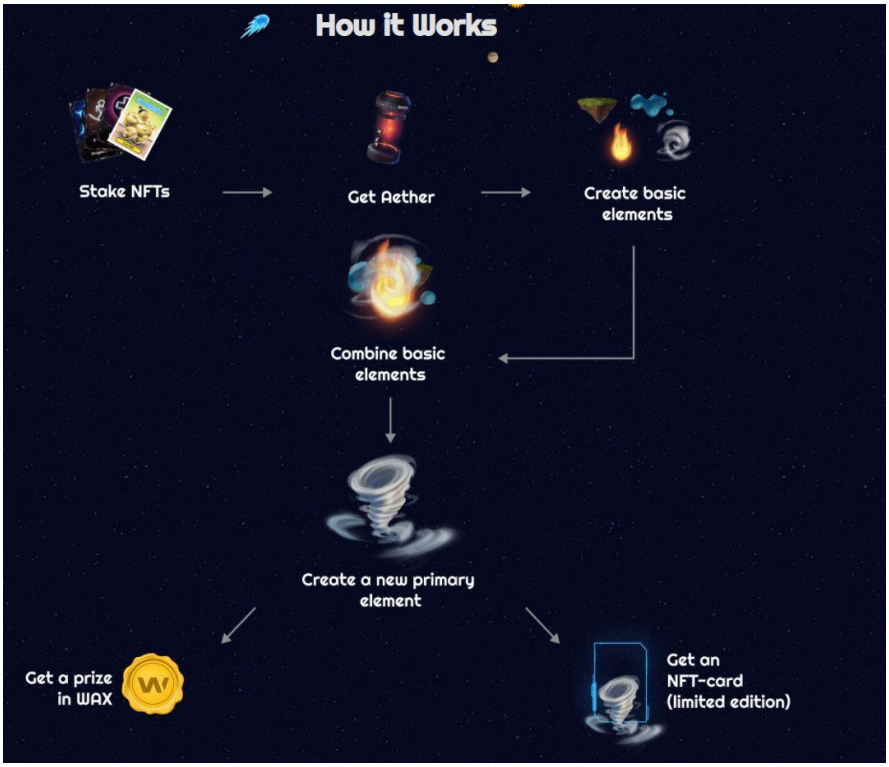
Jedan od polja strojnog učenja naziva pojačano učenje (eng. *reinforcement learning*) koje je kroz godine naraslo u popularnosti kada je poznati AI (umjetna inteligencija) razvijen od strane Britanske kompanije DeepMind naziva AlphaGo igrao protiv najpoznatijeg igrača igre Go Lee Sedol-a, dobitnika 18 svjetskih titula, i pobijedio ga 4 od 5 puta na Svjetskom prvenstvu 2019. Pojačano učenje tip je strojnog učenja koji omogućuje agentu učenje u interaktivnim okruženjima (igrače okruženje) koristeći se pogodak i promašaj (eng. *hit and miss*) taktikom iz vlastitih akcija i iskustava. Problem pojačanog učenja u AI-u može biti najbolje objašnjen kroz igre. Uzmimo igru Pakman (eng. Pac-man) gdje je cilj agenta jesti hranu u mreži (eng.*grid*) i u isto vrijeme izbjegavati duhove koji ga napadaju. Mrežni svijet (eng. *Grid world*) je interaktivno okruženje u kojem agent (u ovom slučaju Pakman) dobiva nagradu za konzumiranje hrane te kaznu ako bude ubijen od strane duhova (gubi igru). Stanja su definirana kao lokacije u mrežnom svijetu gdje je zajednički rezultat pobjeda u igri. Veza pojačanog strojnog učenja kroz igre je toliko neizbježna da nekoliko velikih imena u industriji kao što su DeepMind, Microsoft, OpenAI

i Unity već razvijaju svoja vlastita igrača okruženja koristeći otvoreni kod kako bi što više pridonijeli razvitku tog polja. Postoje četiri popularna okruženja razvijana od strane tih kompanija: DeepMind Lab, Project Malmo, OpenAI gym, Unity ML-Agents. Virtualno okruženje se također može koristiti kao virtualna galerija ili portfolio umjetnika. Moguća korist je razgledavanje galerije umjetnika i mogućnost kupnje njihovih radova. [6][7]

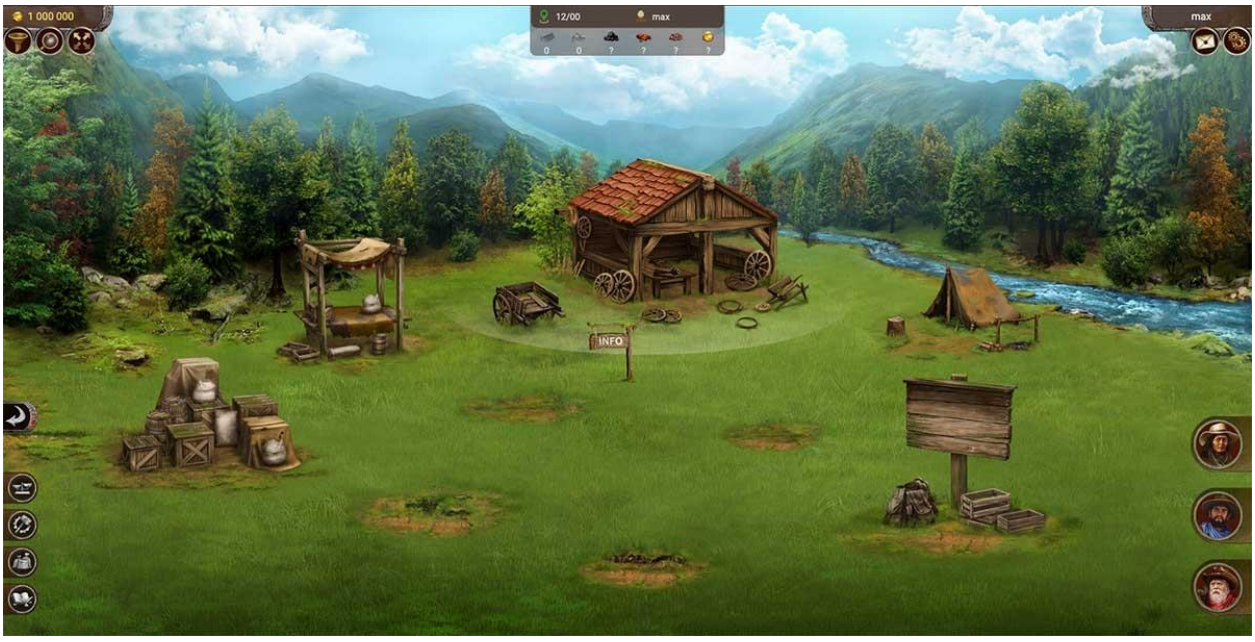
2.3.6. Igre i kriptovalute

Kriptovalute pružaju alternativniji način ulaganja i štednje od odlazaka u banku, te nude decentralizirani način raspodjele novaca. Sama tehnologija ulančanih blokova (eng. *blockchain*[9]) je fascinantna, no u ovom radu ona nije fokus, već izrada interaktivnog okruženja u formi 2D igre. Zašto ne iskoristiti interaktivni prostor kao mrežu spomenutu u tekstu prije kod Pacman igre, i imati valutu u igri koja služi kao i kriptovaluta spremna za korištenje u stvarnom svijetu. Ulančanih blokovi i kripto valute savršeno se prilagođavaju igraćoj industriji. Jednostavan razlog je taj što se time postiže veći stupanj sudjelovanja kao i angažmana za neki projekt. Kompanije poput Hedge Trade-a, Storma, i Metal Paya izvode velike implementacije ove tehnologije tako da predstavljaju razne varijacije prilagodbe igrama kao načina da se privuče veći broj korisnika. S dokumentiranim dokazom da prilagodba izradi igara pozitivno djeluje u drugim industrijama, samo je pitanje vremena kada će i svijet kripto (eng. *crypto*) valuta krenuti u istom smjeru. Slika 5 i Sika 6 prikazuju trenutno najuspješnije projekte (R planet i Prospectors).[7]

[9] podatkovni blokovi koji su povezani u jednosmjerni lanac, i u kojem svaka nova karika, odnosno blok, zavisi o vrijednosti prve starije karike. Kako to obično biva u informatici kad je nužna sigurnost i određena razina privatnosti, povezivanje blokova u lanac temeljeno je na kriptografiji.



Slika 5. Igra R-planet. Izvor. (https://rplanet.io/staking)



Slika 6. Igra Prospectors. Izvor (https://prospectors.io/)

3. PRAKTIČNA IZVEDBA

U ovom dijelu će se opisati cjelokupni postupak izrade igre počevši od stvaranja priče, planiranja, pisanja programskih kodova, izrade elemenata kao i kodiranja koje sve te elemente povezuje. Svaki korak je od velike važnosti kako se i najmanja greška može provući kroz cijeli projekt te na kraju rezultirati greškama (eng. *bug*) što zahtjeva nepotreban dodatan rad.

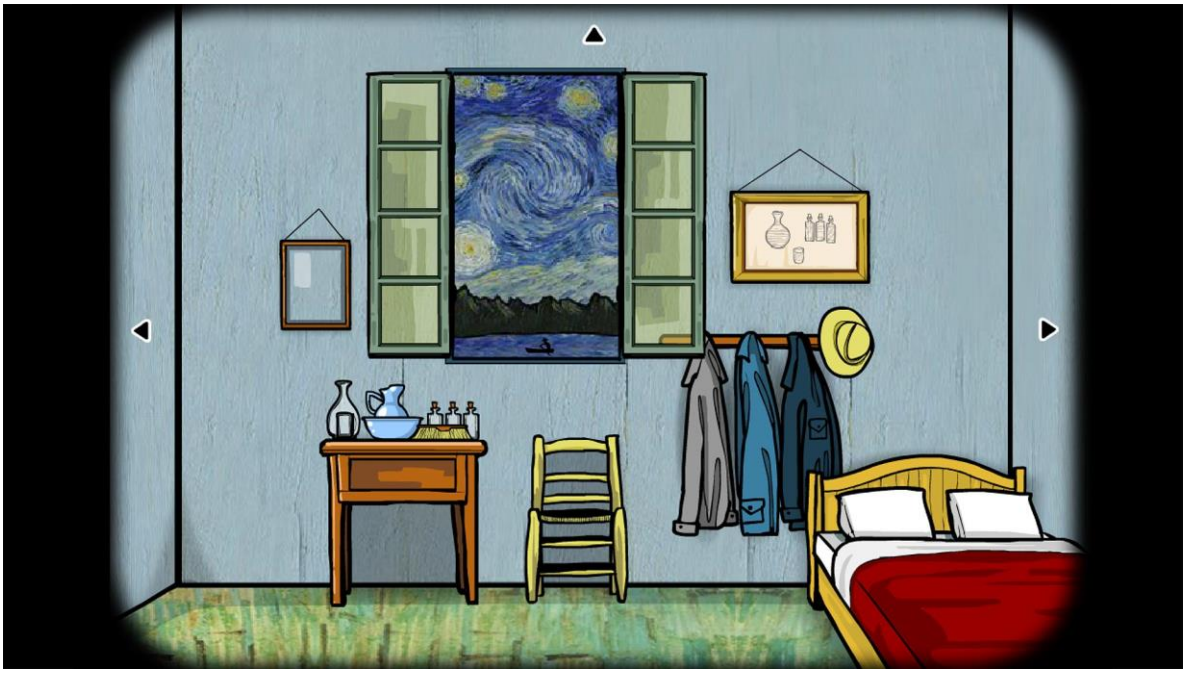
3.1. Postupak izrade originalne igre „Clavis“ korištenjem alata Unity

Na početku izrade igre potrebno je okvirno znati o čemu će se raditi, njen cilj i svrha. Kategorija ove igre je usmjeri i pritisni (eng. *point and click*) unutar koje su integrirane dvije mini igre kao zadatak koji igrač mora ispuniti kako bi postigao određeni napredak. Ideja i cilj igre jest skupiti dva dijela ključa koji, kada je spojen, može otključati ogrlicu zatočenog stvorenja i tim načinom pobijediti igru. Kako bi igrač uopće došao do prilike da zaigra jednu od tih dviju mini igara mora naći način kako bi se one aktivirale. Prvotna ideja ima priču, no ta ista priča može se modelirati tako da igrač mora proći set zagonetki kako bi dobio „kupon“, koji će mu poslužiti u stvarnom svijetu kao popust na određeni proizvod ili da dobije kod, koji kada unese dobije popust na sljedeću igru istog proizvođača ili ju dobije gratis.

Oprema: Računalo, Tablet, Papir i olovka, Unity software, Clip studio paint, Visual studio Code

3.1.2. Inspiracija

Bavljenje izradom igara pruža mogućnost kreativnog izražavanja ali isto tako i priliku za učenje raznih tehnika kao i tehnologija potrebnih za pretvaranje koncepta u realnost. Igre su kroz godine mijenjale svoju strukturu kako je tržište zahtijevalo, gdje su i prije spomenute Indie igre imale jako važnu ulogu u „pogurivanju“ novih ideja koje su obnovile industriju igara. Ispitivanjem tržišta te učenjem na tuđim greškama dobivamo sliku igrače industrije kakvu imamo danas. Prvi korak kod izrade svakog proizvoda jest proučavanje konkurencije (njihovih grešaka, kao i rješenja), reakcije tržišta na njihov proizvod, koji dijelovi su se mogli nadograditi a koji ne. Igra koja se izrađuje u ovom diplomskom radu inspirirana je tehnikom „usmjeri i pritisni *“* igara, to jest igara kojima igrač „klikom“ miša, ili pritiskom prsta (Android) istražuje različite objekte u nekom prostoru i tako napreduje u igri vlastitom logikom i razmišljanjem. Takve su igre najčešće vođene pričom i svojom jednostavnošću tjeraju proizvođača da igrače „zavede“ dobrom pričom i razvojem događaja. Mini igre (kratke igre najčešće integrirane u nekoj većoj igri, no mogu biti i samostojeće igre kao aplikacije) najčešće su dio ovih igara, gdje igrač mora njihovim rješavanjem doći do cilja, ili do predmeta i informacija koje će ga dovesti do većih razina igre. Na slici 7 i 8 se mogu vidjeti kadrovi iz igre Cube escape proizvođača Rusty lake koja je poslužila kao inspiracija za projekt. To je „usmjeri i pritisni“ avanturistička igra u kojoj se prati priča Dale Vandermeera, policijskog detektiva koji istražuje smrt žene koja se našla u misterioznom svijetu Rusty Lakea.



Slika 7. Kadar iz igre Rusty lake. Izvor (http://www.rustylake.com/)

Slika 8. Kadar iz igre Rusty lake. Izvor (http://www.rustylake.com/)

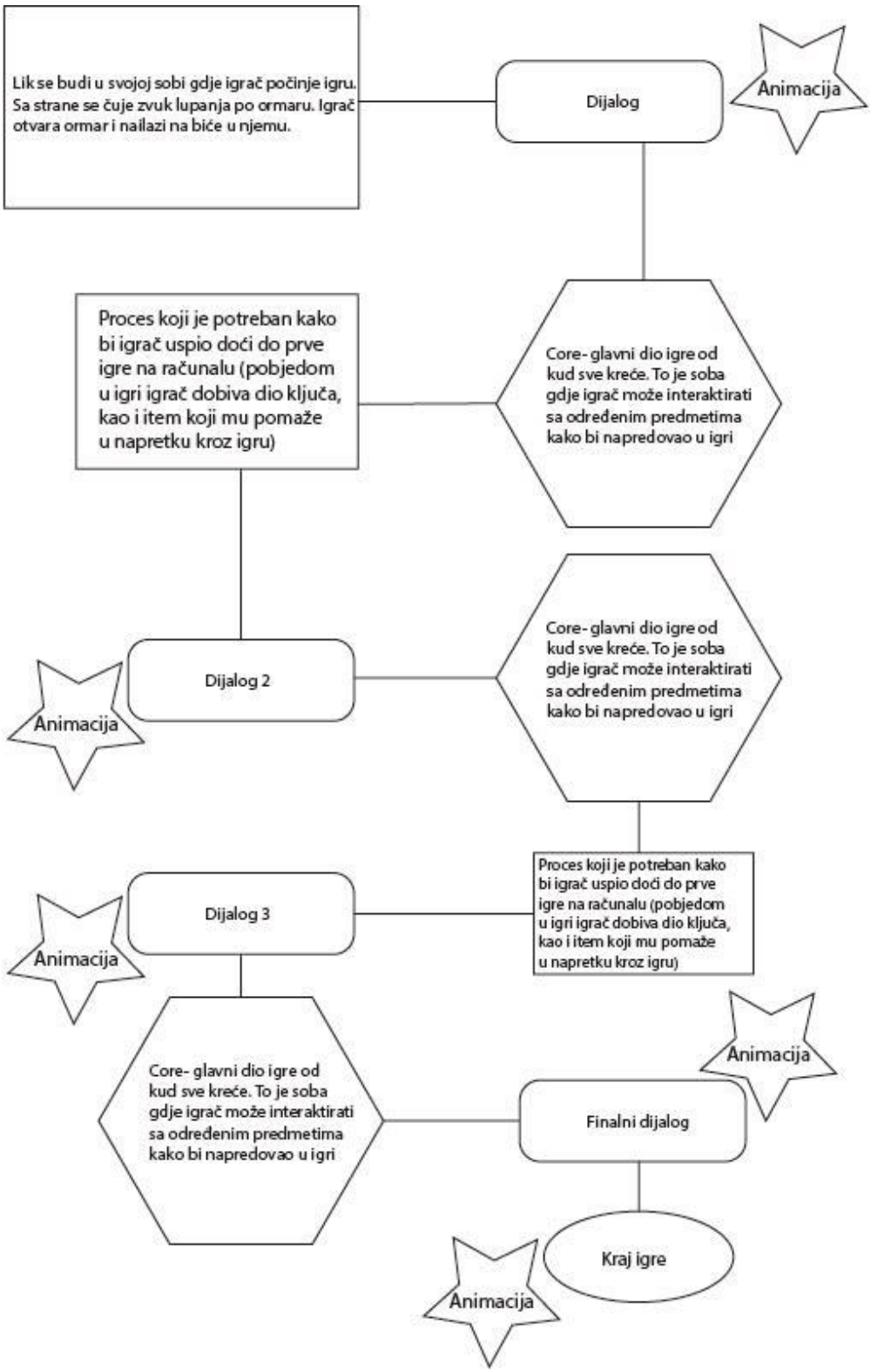
3.2. POSTUPAK IZRADE IGRE

3.2.1. Faza 1: storyboard, razrada i planiranje igre

Mehanika igre predstavljena je kao različite strane sobe gdje igrač može interaktirati s elementima (eng. *assets*) kako bi riješio zagonetku što bi ga dovelo do prve mini igre. Igra započinje događajem gdje zatočeno biće moli igrača da ga oslobodi, nakon čega kreće lutanje i istraživanje po sobi kako bi uspješno obavili zadatak. U igri se zamislio i određeni sistem natuknica (eng. *hint system*) koji bi produbio igru i dao igračima određenu pomoć. Ovakav koncept igre je idealan za stvaranje interaktivnog okruženja gdje se stvara potencijal korištenja takvog prostora u svrhe različite od one očite, samog igranja. U prošlom naslovu spomenule su se igre koje koriste tehnologiju ulančanih blokova u igrama kako bi dodatno motivirali ljude da kroz igru mogu zarađivati i učiti o ekonomiji. Ovaj projekt ima potencijala upravo za to, gdje različiti elementi u igri imaju neku vrijednost koja se može zamijeniti za prave novce ili kao određena vrsta kupona za korištenje u dućanima. Izradom storyboarda se, osim priče, dobiva i generalni izgled igre, likova, predmeta, i prostora. Storyboard, kao što se može vidjeti na slici 11 i 12, započinje crtanjem i pisanjem na papiru što se kasnije analizira te se donose određene izmjene. Na početku, dok je igra još uvijek u fazi izrade storyboarda, bitno je donositi mnogo promjena i puno razmišljati kako bi se točno znalo što treba raditi kako bi se izbjeglo donošenje promjena kasno u igri. Promjene kasno u igri zahtijevaju puno više posla nego da su se sve greške i željene promjene ispravile na početku. Uvijek će biti grešaka, no puno njih će se elidirati pravodobnim promjenama i organizacijom postavljenom na početku izrade projekta Na slici 9 i 10 se mogu vidjeti crteži izrađeni u fazi planiranja, koji su potom digitalizirani.

Slika 9. Prikaz slike iz igre. Izvor (Autorski rad)

Slika 10. Prikaz slike iz igre. Izvor (Autorski rad)



Slika 11. Prikaz razrade igre. Izvor (Autorski rad)

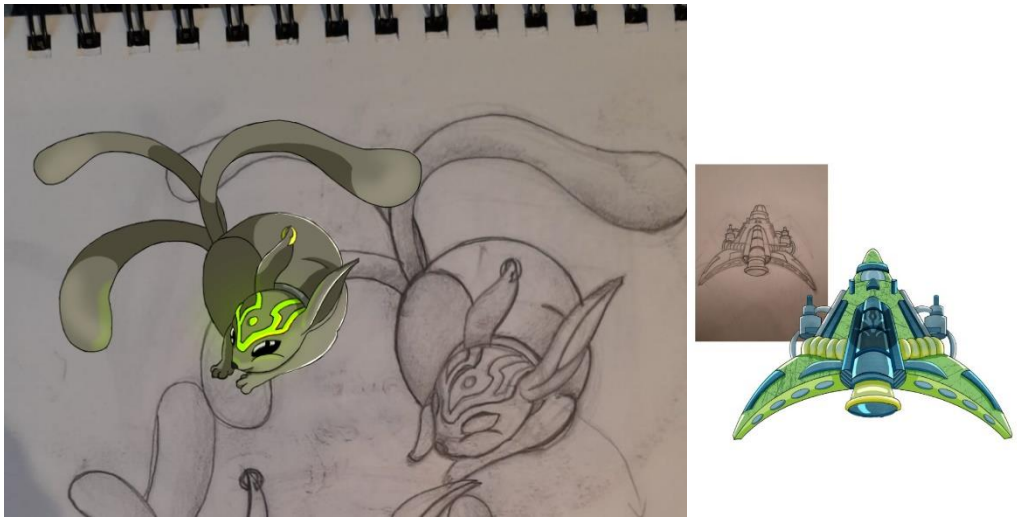
Slika 12. Storyboard igre. Izvor (Autorski rad)

3.2.2. Faza 2: izrada scena, likova i elemenata (eng. Assets).

Nakon storyboarda se kreće s izradom likova. Koristio se program Clip studio paint[10].Od skica do digitalne verzije, svaki asset bio je nacrtan, spremljen kao „.png“ i unesen u Unity.

a) Precrtavanje preko skice (eng. *inking*)

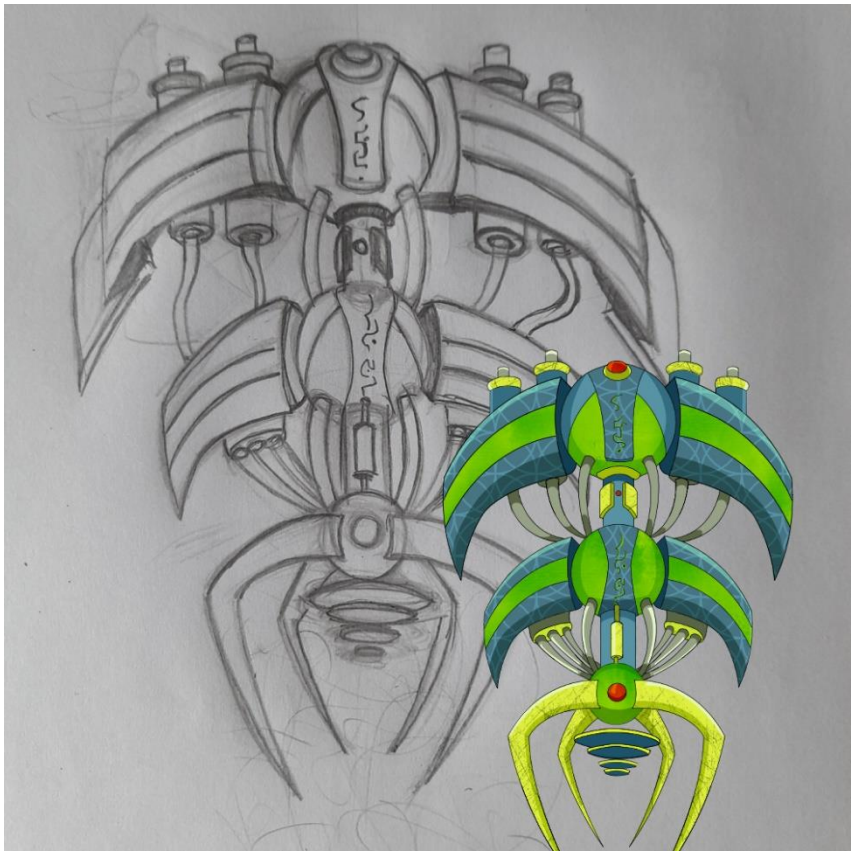
Crteži su uslikani fotoaparatom te je slika prenesena u program Clip studio paint gdje su aplicirane digitalne tehnike crtanja, obojenja i sjenčanja u različitim slojevima (eng. *layerima*) kako bi se u budućnosti moglo što više stvari izmijeniti ovisno o željama. Na slici 13 se može vidjeti prijelaz skice na digitalan rad koji će se potom koristiti u igri



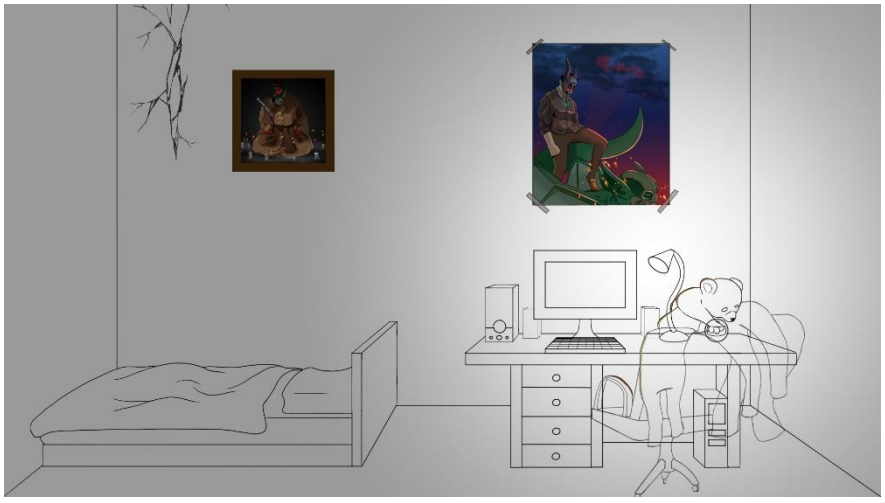
Slika 13. Precrtavanje preko slike korištenjem programa Clip studio paint. Izvor (Autorski rad)

Isto se može vidjeti na slikama 14, 15 i 16 gdje su prikazani crteži digitalizirani za potrebe igre. Neki koncepti se odbacuju na početku izrade tako da je potrebno biti temeljit i razmisliti koju će ulogu određeni koncept imati u samoj igri.

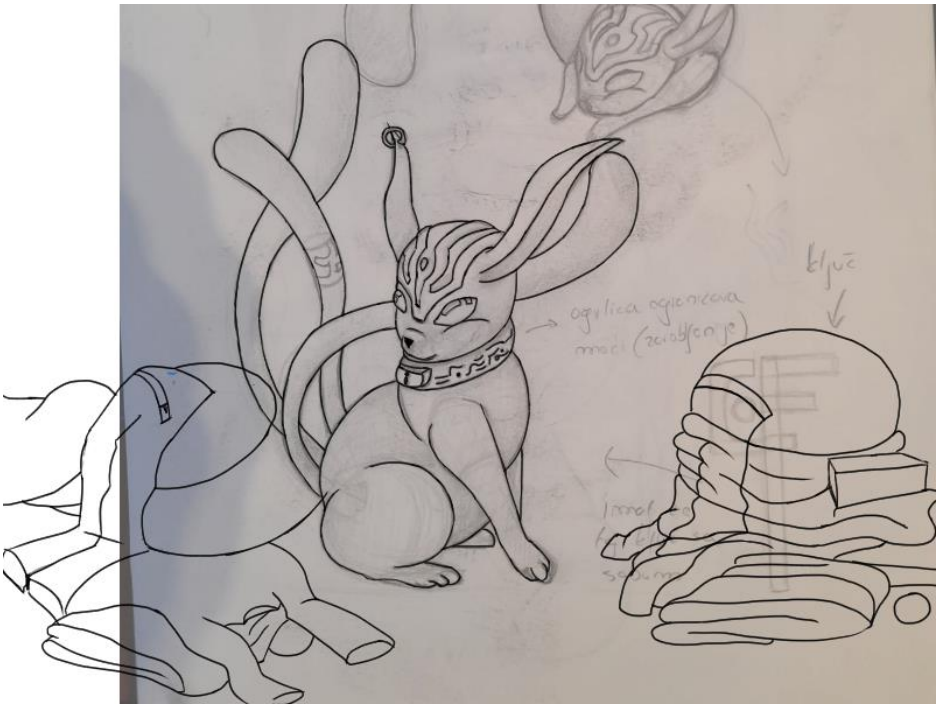
[10]Clip Studio Paint, u Japanu neformalno poznat kao Kurisuta, obitelj je softverskih aplikacija razvijenih od japanske kompanije za grafički softver Celsys. Koristi se za digitalno stvaranje stripova, općenitih ilustracija i 2D animacije.



Slika 14 Precrtavanje preko slike. Izvor (Autorski rad)



Slika 15. Crtanje linija prve sobe - kod crtanja prve strane sobe koristila se stvarna soba kao inspiracija te u ovom slučaju nije bio crtež na papiru. Izvor (Autorski rad)



Slika 16. Crtanje linija lika. Izvor (Autorski rad)

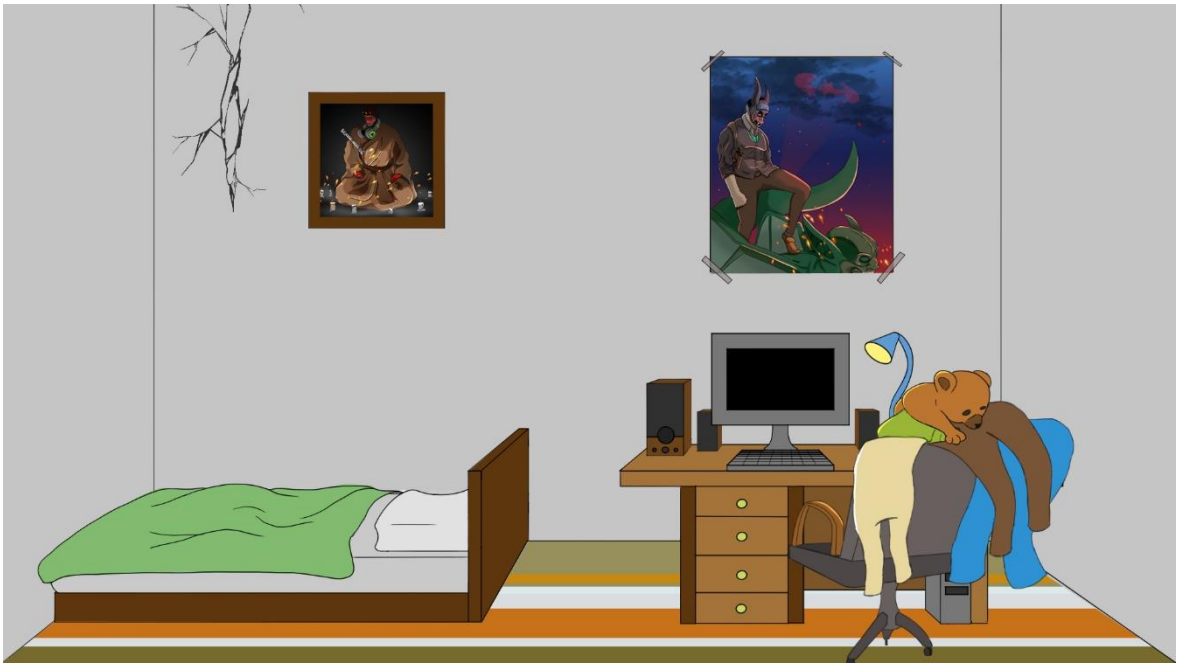
Slika 17. Crtanje linija lika. Izvor (Autorski rad)

b) Obojenje

Na slici 18, 19 i 20 može se vidjeti proces obojenja odabranih crteža koji će se koristiti u igri. Obojenje se radio klasičnim tehnikama u Clip studio paint programu. Svaka boja je bila u svom posebnom sloju, koristio se alat biča (eng. *whip tool*) kako bi se označile granice, te alat za punjenje (eng. *fill tool*) za ispunjenje tih granica, kao i dodatna manipulacija bojama koristeći Opcije miješanja (eng. *blending options*) U slikama se koristila posebna tehnika obojenja nazvana Obojenje u sivim tonovima (eng. *Grayscale*) gdje je postavljena siva boja kao temelj, te su ostale boje nanesene na nju. Ovom tehnikom postiže se prirodnija boja.

Slika 18. Obojenje sivim tonovima. Izvor (Autorski rad)

Slika 19. Dodavanje plošnih boja liku. Izvor (Autorski rad)

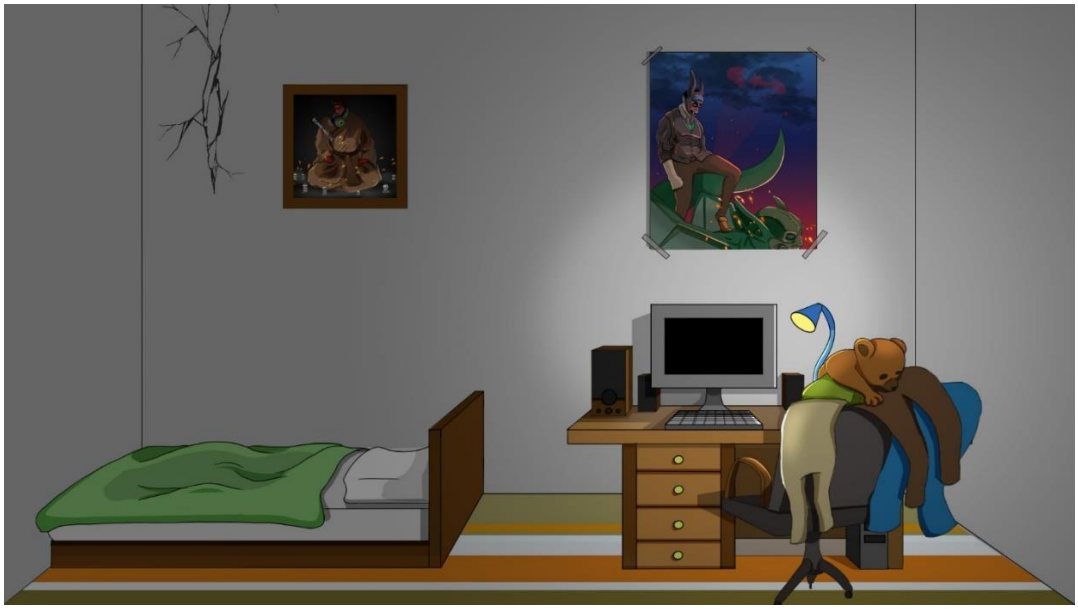


Slika 20. Dodavanje plošnih boja prvoj strani sobe. Izvor (Autorski rad)

c) Crtanje sjena i sjaja

Sjena i sjaj dobivaju se korištenje slojeva (eng. *layera*) u Clip studio paintu tako da se otvori nova datoteka u koju će se postaviti slojevi potrebni za postizanje efekta sjenčanja, kao i sjaja. Nakon što se otvore novi slojevi, postavlja im se postavka miješanja (eng. *blending option*). Za sjene postavljamo opciju množenja obojenja (eng.*multiply*), te za sjaj postavljamo opciju dodavanja obojenja (eng. *add*). Nakon odabira sloja za sjenu, odabire se određena četka (eng. *brush*), te se njome prolazi po slici. Lagano, koristeći pritisak olovke, prolazi se po dijelovima koji se žele zatamniti. Po potrebi se otvara dodatni sloj s istim postavkama na kojem se stvaraju jače sjene s jačim pritiskom olovke. Isti postupak se ponavlja i sa sjajem (lagani sjaj i grubi sjaj). Na slici 14 se može vidjeti kako izgleda sučelje tijekom ovog postupka. Na slici 21 je prikazan završni izgled scene koja će se koristiti u dijelu dijaloga gdje će igrač interaktirati s likom iz igre. Slika 22 prikazuje završni izgled jedne od 4 strana sobe gdje će se igrač moći kretati.

Slika 21. Dodavanje sjena, sjaja, i atmosfere liku. Izvor (Autorski rad)



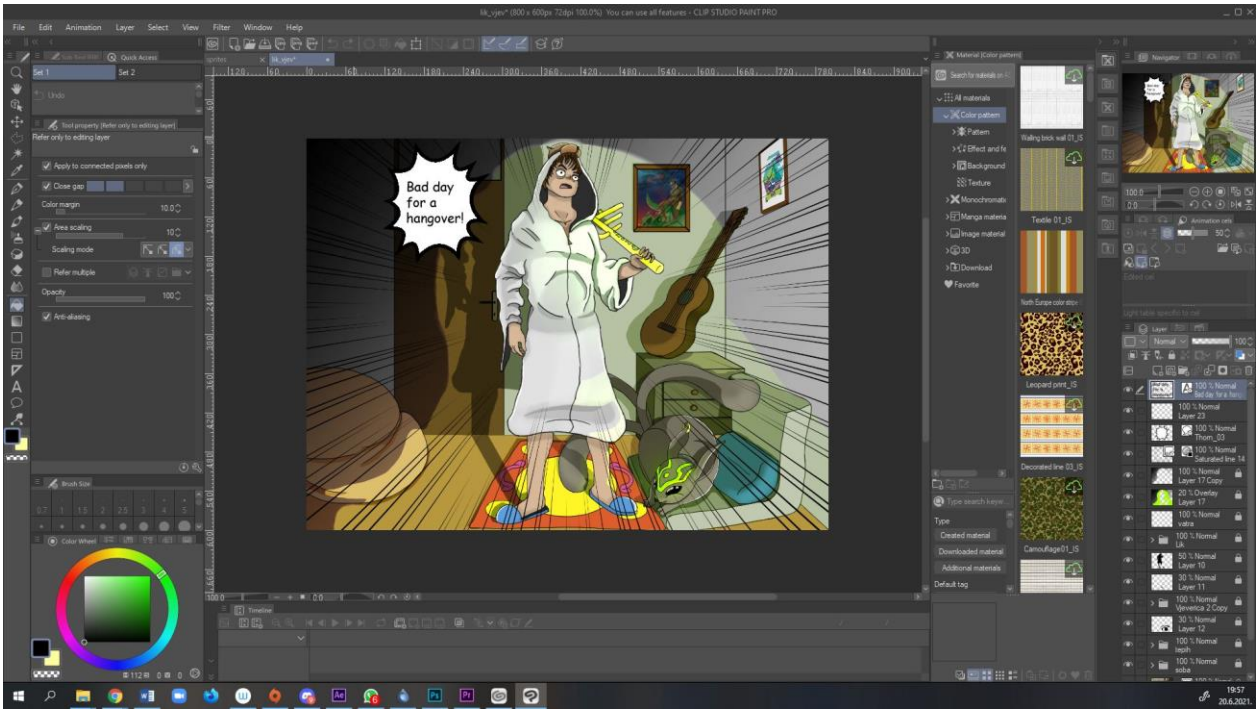
Slika 22. Dodavanje sjena, sjaja i atmosfere prvoj strani sobe. Izvor (Autorski rad)

d) Završno podešavanje

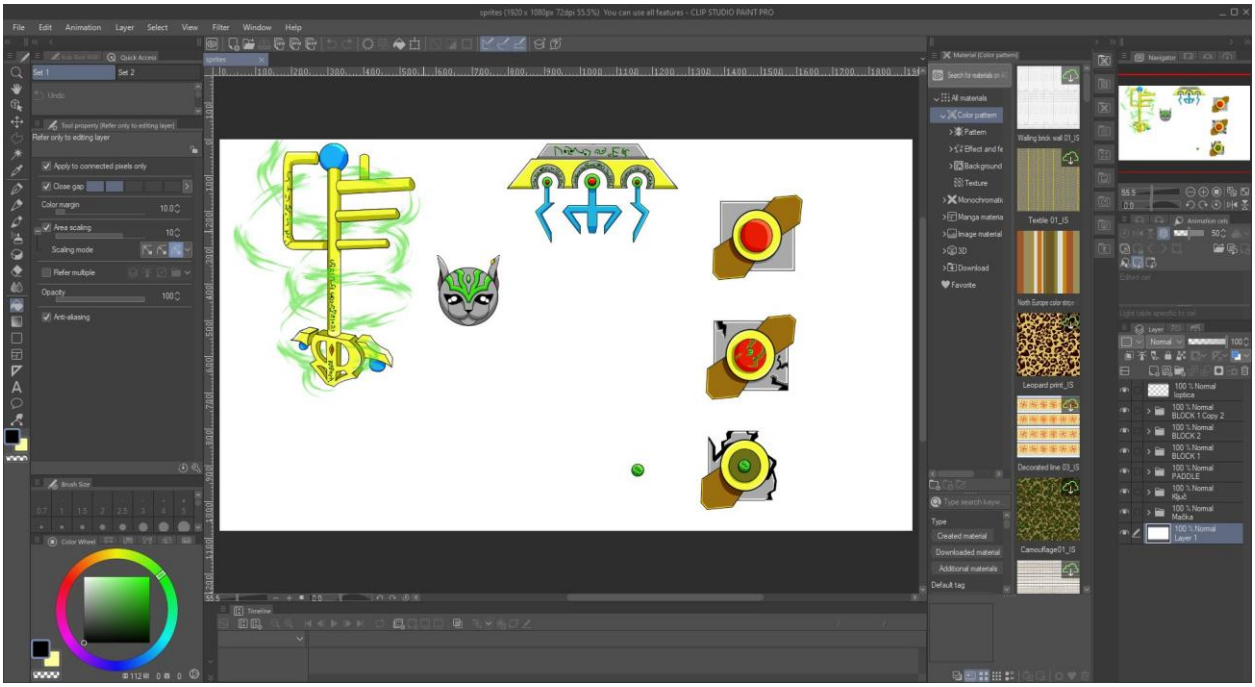
Slika 23 prikazuje izgled sučelja tijekom izrade završne scene prikazane na slici 21. Na desnoj strani mogu se vidjeti slojevi kojima su pridruženi različiti efekti kako bi dobili završni rezultat. Na lijevoj strani je prikaz alata koji uvelike pomažu kod manipulacije elemenata crteža kako bi se dobio traženi efekt.

Slika 23. Sučelje clip studio pro-a kod izrade lika. Izvor (Autorski rad)

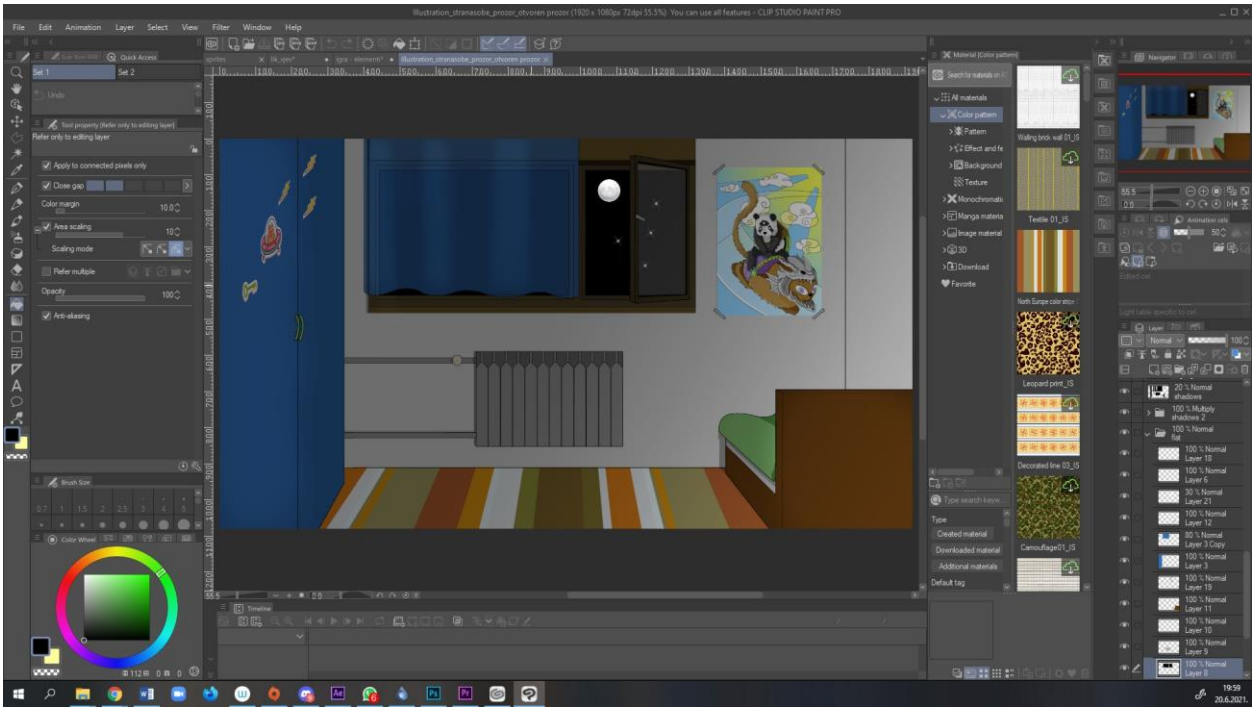
Na slikama 24, 25, i 26 prikazana su sučelja kod izrade ostalih elemenata. Kod Izrade samog projekta ih je bilo mnogo te je ovo samo par primjera procesa izrade.



Slika 24. Sučelje clip studio pro kod izrade ostalih likova. Izvor (Autorski rad)



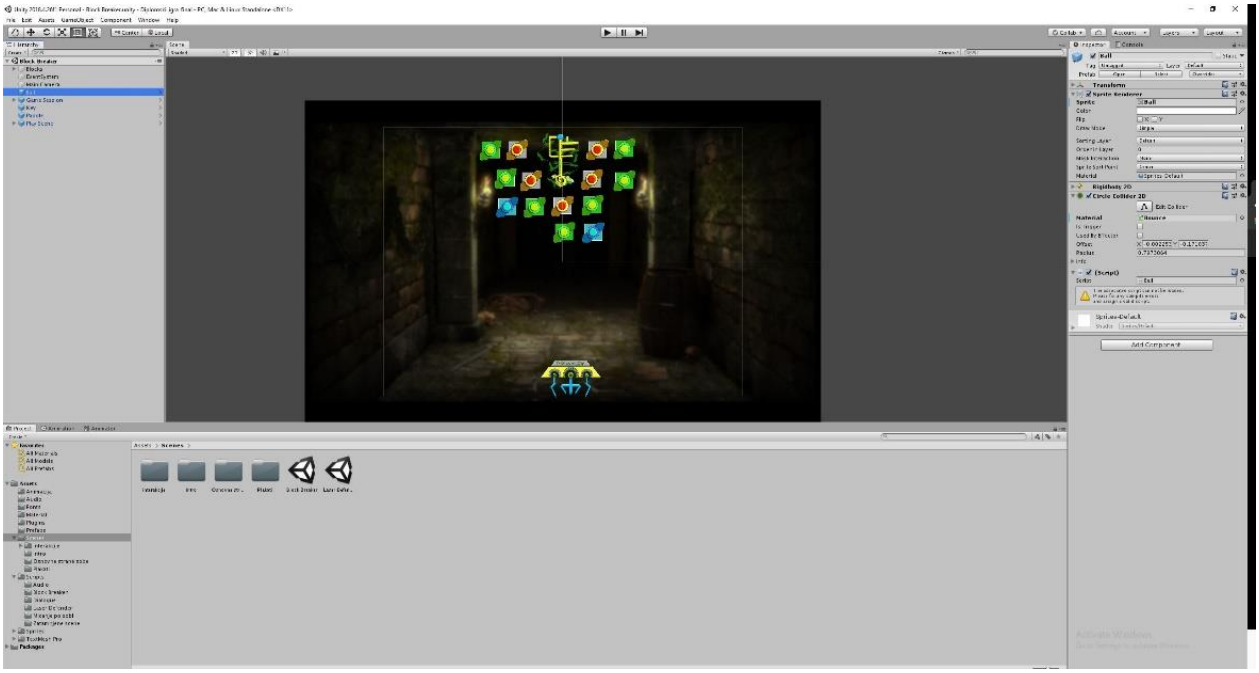
Slika 25. Sučelje clip studio pro-a kod izrade elemenata za igru (eng. *Assets*) Izvor (Autorski rad)



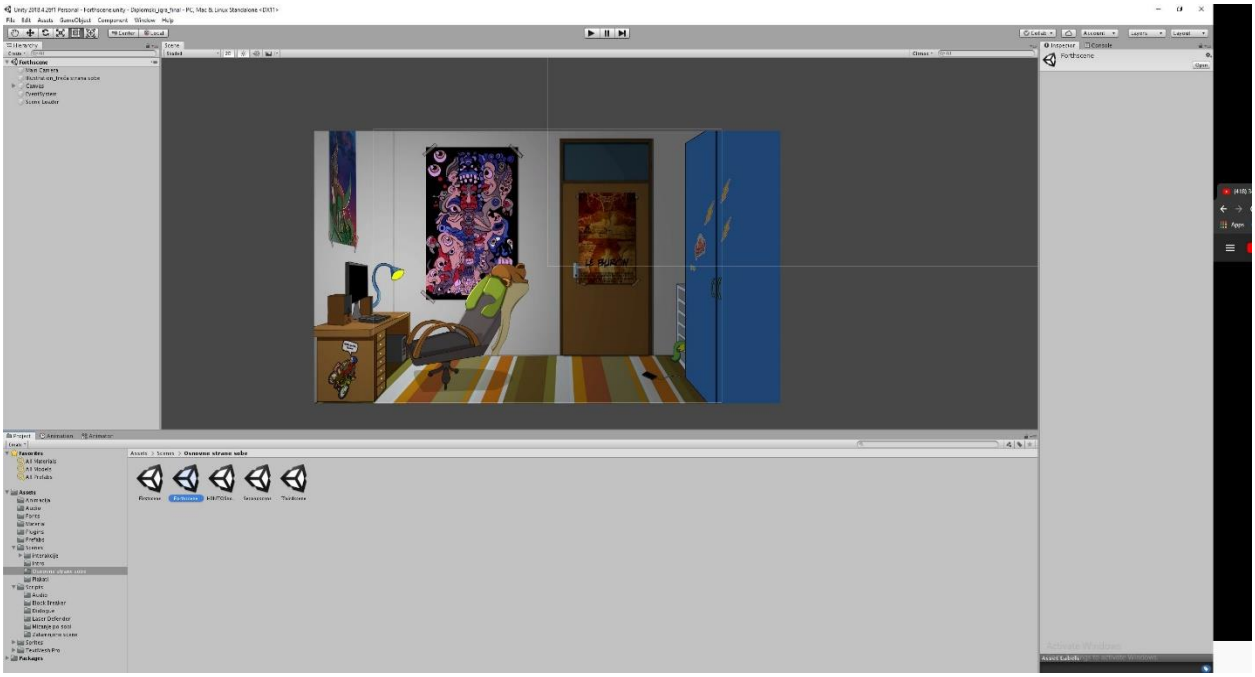
Slika 26. Sučelje izrade slijedeće strane sobe (Clip studio pro). Izvor (Autorski rad)

3.2.3. Faza 3: Rad u Unityju

Rad u Unityju započinje integriranjem svih digitalnih slika (eng. *sprites*) u program, napravljen je posebna datoteka za elemente, skripte, scene, zvučne podatke i animacije. Na slici 27 i slici 28 može se vidjeti prikaz sučelja Unity-ja kod implementiranja prije izraženih elemenata u program kako bi se moglo nastaviti s radom. U ovim postupcima jako je važno biti organiziran i dodavati nazive datotekama tako da se što efektivnije i brže obavi određena radnja. Besmisleno dodavanje imena datotekama može dovesti do nepotrebne potrošnje vremena što utječe na konačan rezultat. Potrebno je odvojiti zvučne datoteke, datoteke animacije, skripte, kao i slike te elemente igre.



Slika 27. Sučelje Unity-ja prilikom izrade prve „mini“ igre Blockbreaker. Izvor (Autorski rad)



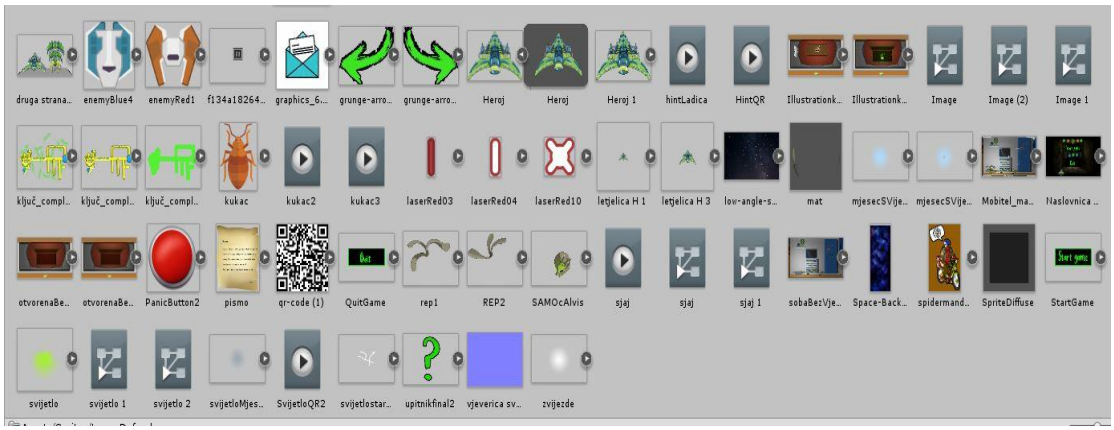
Slika 28. Interface Unity-ja prilokom izrade druge strane sobe. Izvor (Autorski rad)

3.3. Izrada igre u programu Unity

Nakon organiziranja datoteke i planiranja, kreće se sa radom na samom projektu. U slijedećim koracima pokazati će se potrebne radnje i tehnike kako bi se dobio određeni stil, I funkcionalnost igre, kao i objasniti funkcije određenih elemenata igre.

3.3.1. Element u igri

Igrači element (eng. *Asset*) je vizualan element u igri kojemu se dodaje fizika 2D svijeta, kao i granice kojima se može definirati mjesto kontakta gdje određujemo kako objekti (različiti elementi) interaktiraju i kako se ponašaju nakon što interaktiraju. Igraćim elementima se pridružuju slike elemenata (eng. *sprite*-ovi[11]). Također se trebao i izabrati font koji se koristio tijekom igre, to je bio *Mistery font*. Na slici 29 je prikazan sadržaj samo jedne od mapa gdje se nalaze i razne ostale datoteke koje se moraju organizirati kako ne bi došlo do zabune i teškog snalaženja tijekom izrade igre.



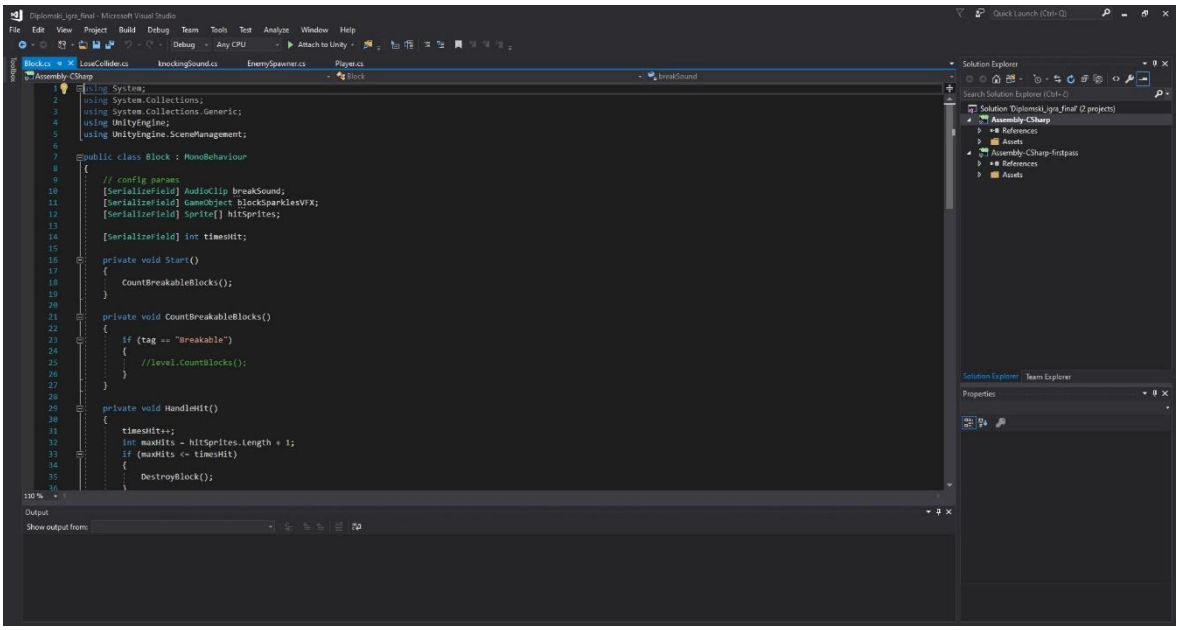
Slika 29. Sadržaj jedne od mape u Unity-ju. Izvor (Autorski rad)

[11] vizualne reprezentacije objekta, kako izgleda objekt koji interaktira s ostalim objektima

Slika 30. Prikaz igračih elemenata korištenih u igri. Izvor (Autorski rad)

3.3.2. Skripte

Skripte su napisane u programu C koristeći se software-om *Visual studio code*[12] u kojem su pisane već spomenute skripte koje određuju logiku ponašanja objekta kojem su dodijeljene. Skripte su tada dodavane igraćim objektima (gumbima, *assetima*), kao i univerzalnoj logici igre. Svaki igrači objekt ima vlastitu skriptu te se pritiskom određenog gumba pokreću funkcije definirane kodom. Na slici 31 prikazan je izgled sučelja *Visual studio code*-a u kojem su pisani kodovi skripti koje se tada vežu za igrače objekte ovisno o funkciji koja se treba izvršiti.



Slika 31. Primjer izrade jedne od skripti koristeći Visual studio code. Izvor (Autorski rad)

[12] *Visual Studio Code* je uređivač izvornog koda koji je izradio *Microsoft* za *Windows, Linux i MacOS*

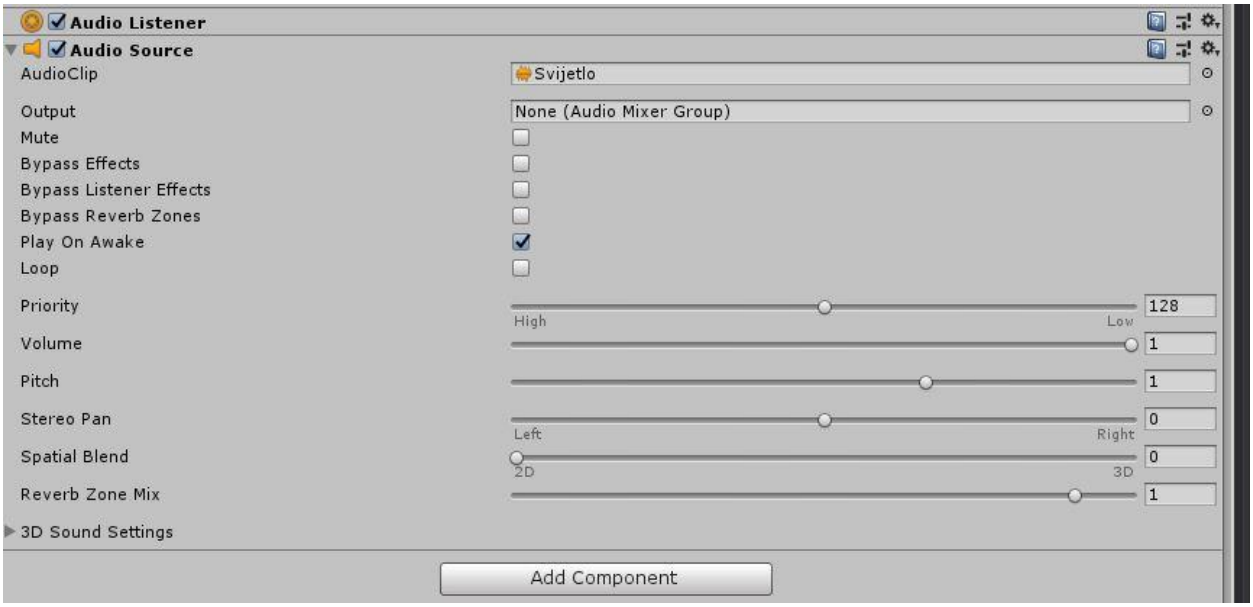
3.4. Scene

Scene u Unityju sadržavaju objekte igre. One mogu biti korištene kod izrade glavnog izbornika, individualnih razina, i svega ostaloga. U ovom slučaju koriste se kod izrade strana soba, kao i za svaku sljedeću stranu, dvije mini igre, pogled kroz prozor i početak dijaloga (sve je to bila zasebna scena). Svaka se scena može zamisliti kao zasebni svijet, ili unikatna razina. U svakoj sceni postavlja se okolina, zapreke, i ukrasi. Može se na njih gledati kao puzzle od kojih se gradi i dizajnira cjelokupna igra. Na slici 32 prikazan je izgled tih scena na sučelju Unity-ja..

Slika 32 Prikaz scena u Unity-ju. Izvor (Autorski rad)

3.3.3. Zvučni podaci

Zvučni efekti i muzika općenito ima veliku ulogu u igri kako bi što više privukli pažnju igrača. Muzika je alat kojim se može kontrolirati emocija i ustanoviti ton priče u igri. U filmu mnogi gledaju na zvučnu podlugu (eng. *SoundtrackI*) kao temelj svake scene. Ovo je još jedan dokaz sličnosti igre s drugim umjetnostima. Priroda igara je produbiti iskustvo koje pruža film, i pokušati ga unaprijediti. Korisnik ima priliku kontrolirati akcije omiljenog lika. Kod nekih igara igrač ima opciju isključiti zvuk, što razvojnim inženjerima daje dodatnu motivaciju da pokušaju muzikom privući igrača i zadržati ga u svijetu koji mu pruža igra. U ovom projektu muzika je imala minimalan utjecaj na igrača, ali je pokušano stvoriti mističnu atmosferu. Slično se postupalo i s odabirom fonta (*Mistery font*). Na slici 33 može se vidjeti sučelje gdje se objektima, ili cjelokupnoj igri dodaju zvukovi koji se tada podešavaju ovisno o primjeni. Zvučne datoteke se koriste i u skriptama gdje ih dodajemo različitim funkcijama u kodu. Slučaj prikazan na slici 33 opisuje zvuk koji se pokreće prilikom promjene scene (vežemo ga za kameru platna u programu). Slika 34 prikazuje stranicu s koje su spomenuti zvukovi bili skidani bez potrebe za plaćanjem.



Slika 33. Sučelje podešavanja zvuka. Izvor (Autorski rad)

Slika 34 Stranica za *download* besplatnih zvukova. Izvor (https://freesound.org/)

3.3.4. Animacija

Mogućnost animiranja nepokretnih slika postoji od 1800 ih godina. Moderna animacija s mogućnošću pripajanja na različite tehnologije čini industriju puno raznovrsnijom nego što je bila. Animacija u igrama nudi puno više mogućnosti i nema puno ograničenja kao što je slučaj kod npr. animiranog filma. Bez obzira na platformu igre nude galeriju različitih vrsta animacija, od cijelog igraćeg dijela igre (eng. *In-game*) videa do animacije različitih efekata i filmova unutar igre što ju čini igrom. FMV[13] može biti nacrtan rukom ili može biti CG[14]. Animacija je u igri generirana na način kao i svaka animacija u filmu ili u videu. S ograničenim ili nepostojećom ulaznom informacijom korisnika. FMV sekvence su većinom iskorištene u narativne svrhe, dok su animacije igraćeg dijela igre[15]

[13] FMV - *full motion video*
[14] CG-*Computer graphics*
[15] In game – tijekom igranja igre

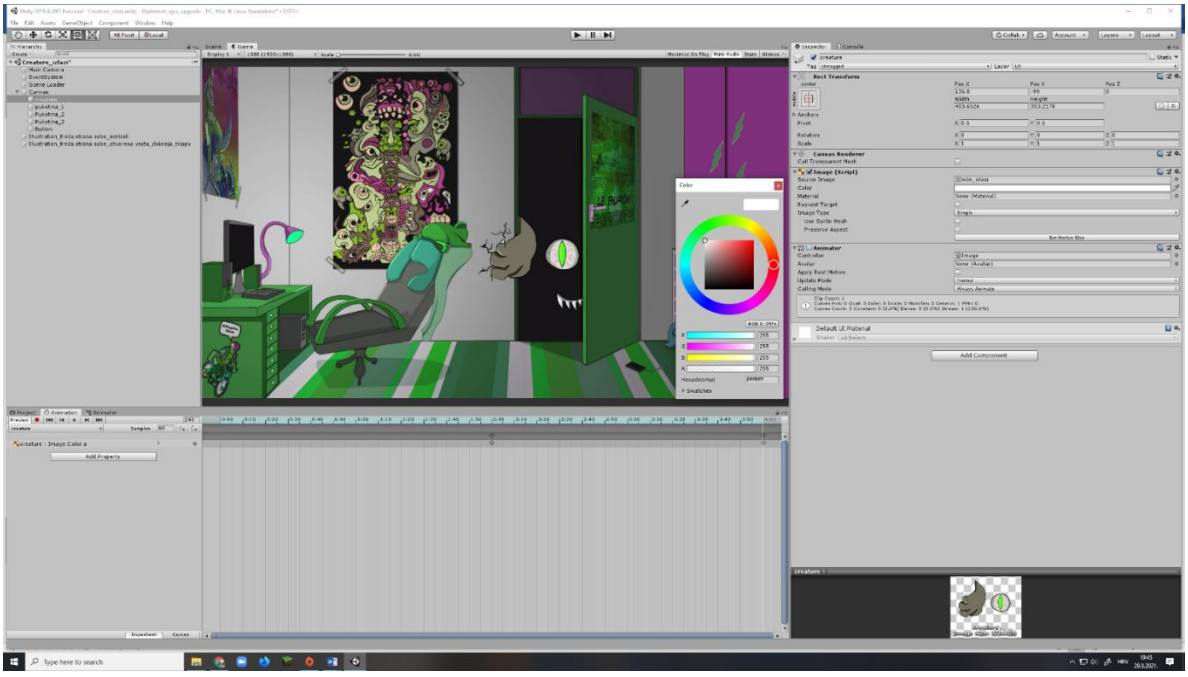
pravi izvor potencijala tog medija. Upravo u tim *in-game* animacijama autor može najviše pridonijeti igri u smislu njegove umjetničke poruke i stila. Talentirani autor ovdje može ispričati priču koristeći govor tijela lika s ograničenim grafičkim mogućnostima. Dok posuđivanje glasova i skriptirana priča imaju svoju ulogu, ipak je zadaća autora unijeti koliko osobnosti u projekt. Animacije u igri imaju puno pokreta tijela kao što su trčanje, hodanje, zamahivanje mačem. Puno njih je reciklirano kako bi se uštedjelo na vremenu. Igrač kontrolira lika, dok umjetnik ili autor ima zadatak stvaranja uvjerljivih pokreta u trenutku pritiska gumba na upravljaču. Ovo je primjer kako funkcionira jedan od principa animacije u igri, predviđanja pokreta (eng. *Anticipation*).

Predviđanje pokreta u tradicionalnoj i interaktivnoj animaciji je način na koji se lik fizički priprema na pokret što je relativno jednostavno uz pretpostavku da animator ima pripremljen storyboard te se može pripremiti. U igrama je na animatoru da iskoristi sve mogućnosti, alate i tehnike kako bi dobio zlatnu sredinu između reakcija lika za kojeg se spomenute animacije rade i postigao uvjerljiv pokret. Tehnička ograničenja dodatno kompliciraju proces, kao što je broj slika u sekundi (eng. *frame-rate*) određenog engine-a. Očekivanje u animacijama za video igre može biti duljine par sličica. Ako je broj sličica u sekundi mali, i ako je tih nekoliko kadrova preskočeno, igrač neće uopće vidjeti animaciju očekivanja, što smanjuje uvjerljivost animacije i njen utisak. Svi ti problem mogu biti riješeni kreativnim planiranjem, i skraćivanjem pokreta. Moderne igre koriste različite animacijske tehnike i stilove, iako mnogi koriste Mayu[16] i 3DS Max software[17]. U tradicionalnoj animaciji, focus je na akciji unutar kadra, no u igrama je moguće vidjeti događaje iz skoro svakog ugla što predstavlja univerzalan pristup te zahtjeva veću pozornost na detalje. Nakon planiranja, izrada slika za igru i logike se kreće s izradom igre. U ovoj igri animirani objekti, i efekti postignuti su izradom efekta u nekom od programa (Photoshop, Clip Studio Paint) gdje su razni dijelovi objekta rastavljeni po slojevima, i potom uneseni u Unity kao slika. Nakon što su se sve slike za igru unijele, kreće se s njihovim manipuliranjem u Unityju kako bi dobili snimani pokret (animaciju). Prvo se

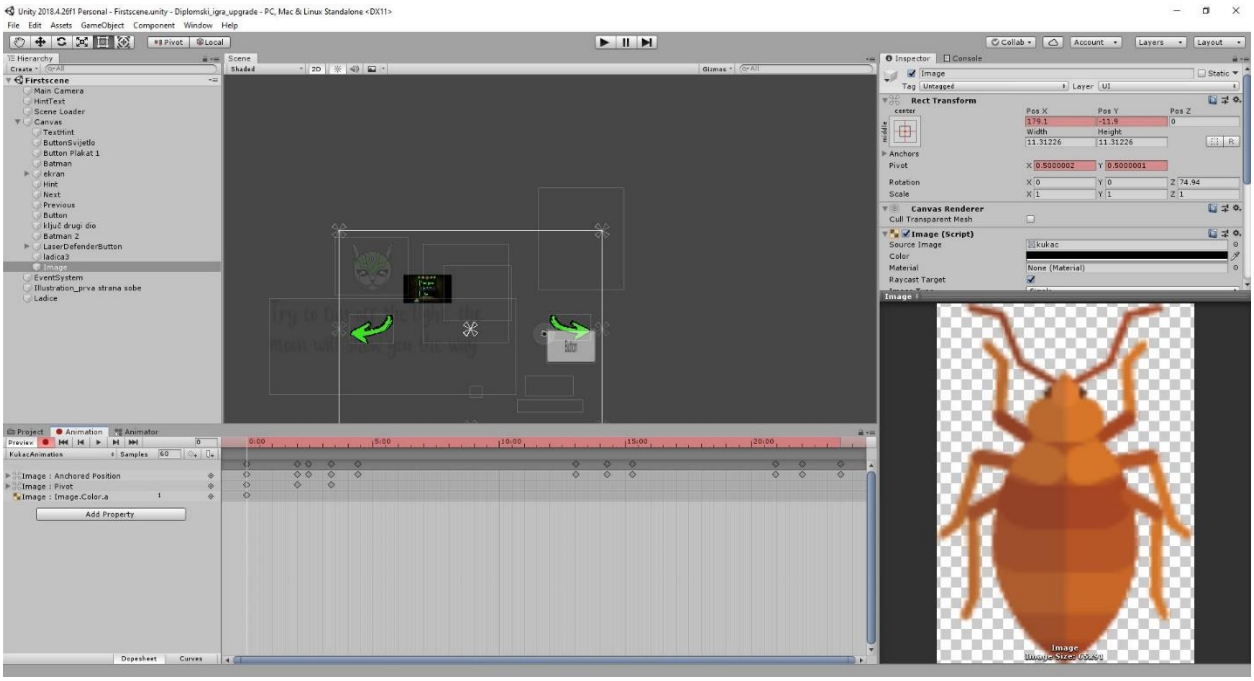
[16] Autodesk Maya, obično skraćeno samo Maya, je 3D računalna grafička aplikacija koja radi na Windowsima, macOS -u i Linuxu, izvorno je razvila Alias Systems Corporation, a trenutno je u vlasništvu i razvoju Autodeska.

[17] Autodesk 3ds Max, ranije 3D Studio i 3D Studio Max, profesionalni je 3D računalni grafički program za izradu 3D animacija, modela, igara i slika. Razvio je i producirao Autodesk Media and Entertainment. Ima mogućnosti modeliranja i fleksibilnu arhitekturu dodataka te se mora koristiti na Microsoft Windows platformi.

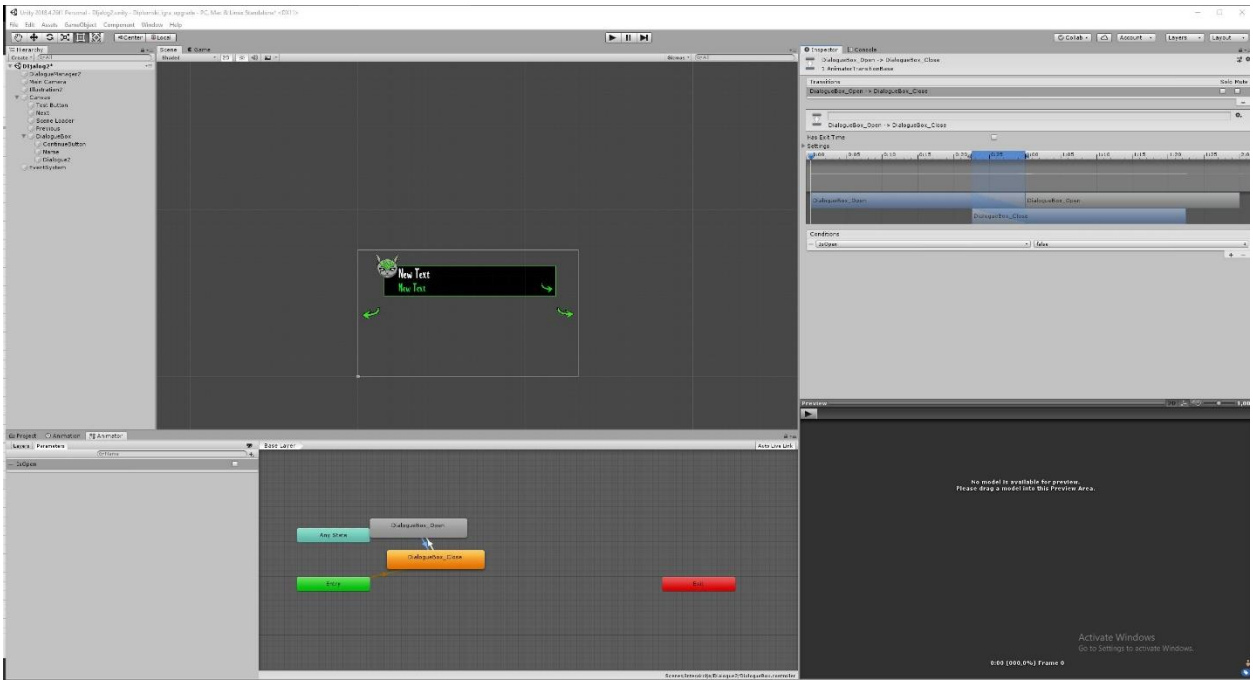
označio objekt, započelo se njegovo snimanje gdje se odabrao naziv animacije koja je tada spremljena u datoteku naziva „animacije“. Nakon snimljenog pokreta se pokreće igra kako bi se provjerilo da li je dobiven traženi efekt (kukac se giba po zidu). Na slici 36 je prikazano sučelje prilikom animiranja kukca Nakon dobivenog efekta, krenulo se s izradom još animacija gdje je za neke potreban kod ili funkcija u kodu kako bi se postavio proces pokretanja animacije pritiskom određenog gumba. Na slici 37 se može vidjeti kako se koristila animacija na način da se vidljivost (eng. *transparency*) smanjila na nulu te je putem animatora postavljen uvjet koji kaže da ako igrač pritisne prazni dio vrata, „biće“ koje izlazi iz vrata kreće dobivati sve manju i manju vidljivost kroz određeno vrijeme. Na slici 38 se može vidjeti prikaz animacije simbola po sobama napravljena istim načinom kao i dva primjera prije. Dolaskom igrača do određene faze u igri (definirano kodom) na zidovima se počinje izvršavati animacija simbola. Na toj istoj slici (Slika 38) mogu se vidjeti datoteke animacije i animatora. Animacija (1) je datoteka koja se automatski generirala, odnosno, koja je kreirana izradom animacije u sceni. Animator(2) jest logika animacije (kada će se određena animacija aktivirati u kojem dijelu igre, ili pritiskom određenog gumba).



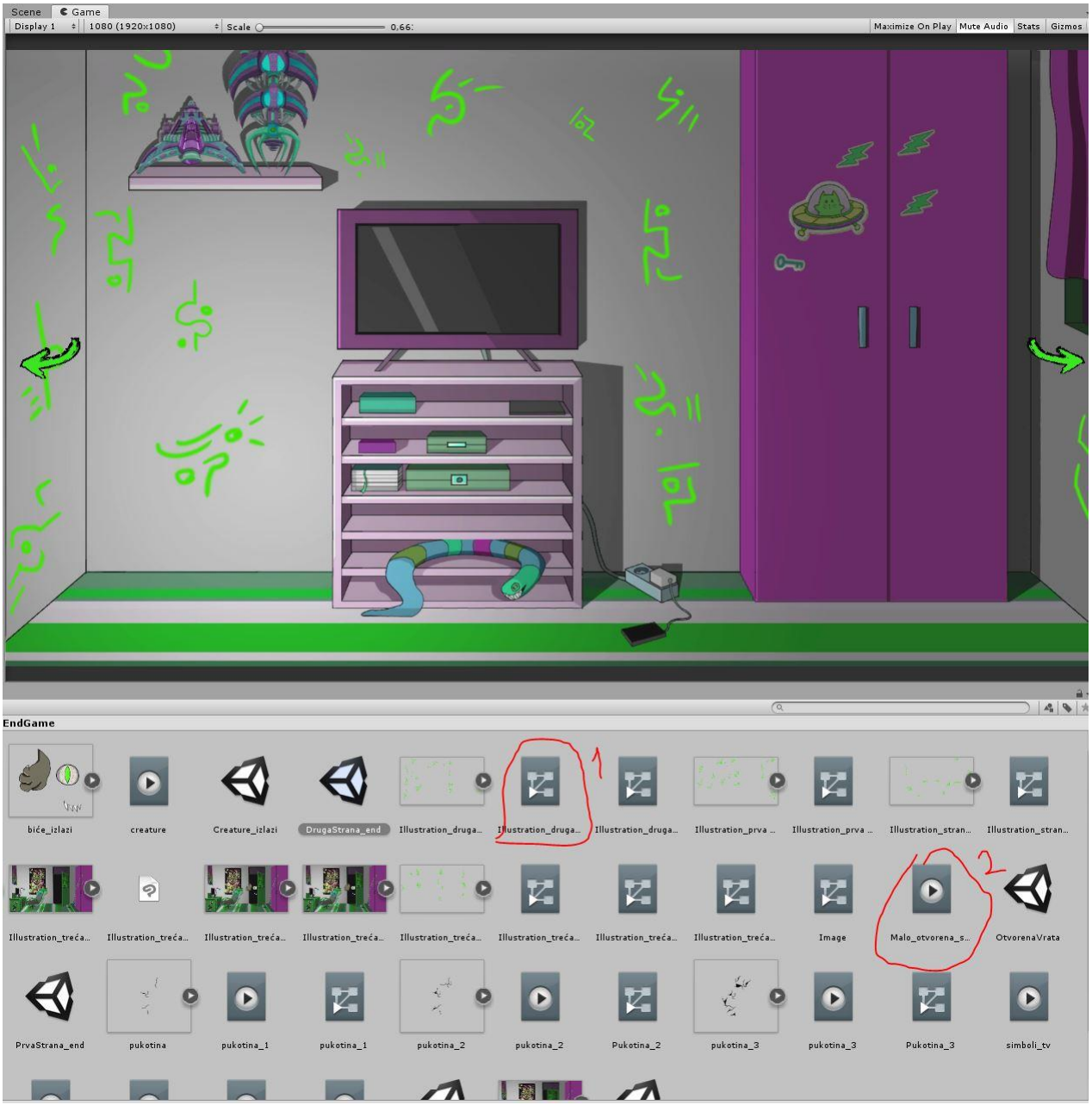
Slika 35. sučelje Unity-a kod izrade završne animacije. Izvor (Autorski rad)



Slika 36. Sučelje Unity-a kod izrade animacije kukca. Izvor (Autorski rad)



Slika 37. Prikaz rada u animatoru kod animacije dijaloga. Izvor (Autorski rad)



Slika 38. Prikaz datoteke animacije i animatora kod izrade animacije simbola. Izvor (Autorski rad)

3.3.5. Izrada elemenata i povezivanje scena

Nakon razrade i planiranja kreće se s izradom svih potrebnih asseta, pozadina i spriteova u Clip studio paint-u, te njihovo spajanja i slaganje u Unity-ju. Nakon postavljanja scena, započinje se s pisanjem skripti, stvaranjem logike same igre i dodavanjem istih skripti objektima (nekim objektima je dodan već od prije definiran kod za fiziku krutog tijela (eng. *rigid body*), gdje se utvrđivalo što će se dogoditi kada dva objekta dođu u dodir jedan s drugim). U nastavku ovoga teksta opisat će se logika ponašanja različitih objekata u igri putem skripti napisanih u Visual Studio Code-u[18] uporabom C programskog jezika.

3.3.6. Izrada početnog zaslona igre

Početni zaslon je prvo što igrač vidi kada ulazi u igru te predstavlja jedan od najbitnijih elemenata u igri. U nekim slučajevima početni zaslon postavlja ton cjelokupne igre te se kao takav mora predstaviti igraču. Dizajn početnog zaslona je napravljen pomoću Photoshopa gdje se font koji se provlači kroz cijelu igru (*Mistery font*) postavio na crnu pozadinu te se slika sprema kao .png i uvozi se u Unity gdje se pomoću skripte postavlja željeni efekt. Na slici 39 se može vidjeti rezultat efekta (pomakom miša osvjetljavamo naslov, te je dodan gumb koji ima funkciju pokretanja igre)

[18] Programski jezik C spada u proceduralne programske jezike koji je razvijen u ranim 70-im godinama 20. stoljeća.

Slika 39. Početni zaslon igre. Izvor (Autorski rad)

3.3.7. Sadržaj skripte početnog zaslona

Sljedeća skripta prikazuje program koji omogućuje praćenje miša (eng. *mouse tracker*) kako bi se toj poziciji dodao izvor svijetla te putem toga omogućilo osvjetljavanje slike micanjem miša.

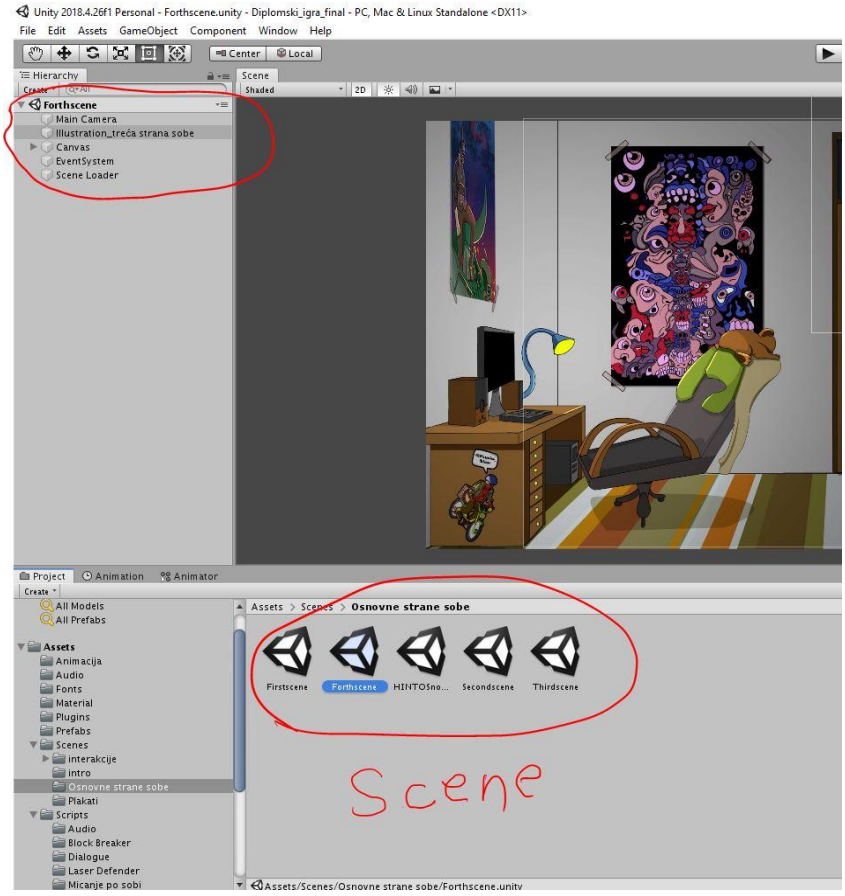
```
public class mouseTracker : MonoBehaviour{
    void Update(){
        Vector3 newPos = Camera.main.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y,
Mathf.Abs(Camera.main.transform.position.z - transform.position.z)));
        newPos.z = transform.position.z;
        transform.position = newPos;}}
```

Isječak koda 1. Tragač miša. Izvor (Autorski rad)

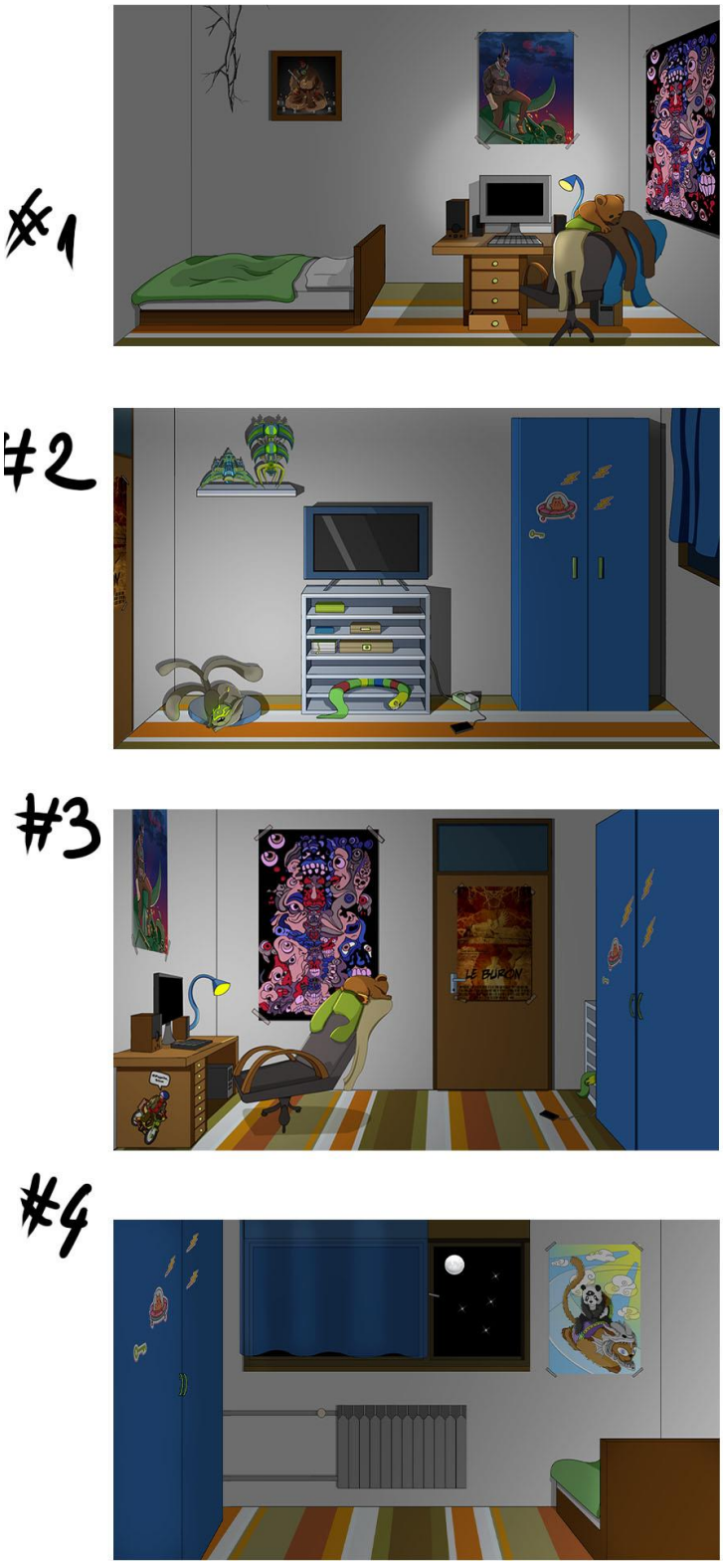
3.3.8. Povezivanje scena u svrhu kretanja po virtualnoj 2D sobi

Postupak je započeo izradom scene za svaku stranu sobe posebno (desni klik-→new scene). Nakon dodavanja 4 scene (jedna za svaku stranu), dodane su slike (svaka slika za jednu stranu sobe). Isto se može vidjeti na slici 41. Kada su sve slike ili strane sobe dodane, kreće se od jedne do druge scene te se stvara system događanja (eng. *event system*).

Platno (eng. *Canvas*) (polje koje imitira ekrane različitih formata gdje se može manipulirati pozicijom kamere te pozicionirati različite elemente igre ovisno o okviru). pokretač scena (eng. *scene loader*) (objekt kojemu se dodaje skripta s kodom, i koji se nakon toga dodaje igračem objektu, npr. kada igrač klikne na taj objekt, pokreće se izvršavanje koda). Na slici 40 se može vidjeti prikaz sučelja Unityja sa spomenutim grafičkim objektima.



Slika 40. Prikaz sučelja Unityja i scena. Izvor (Autorski rad)



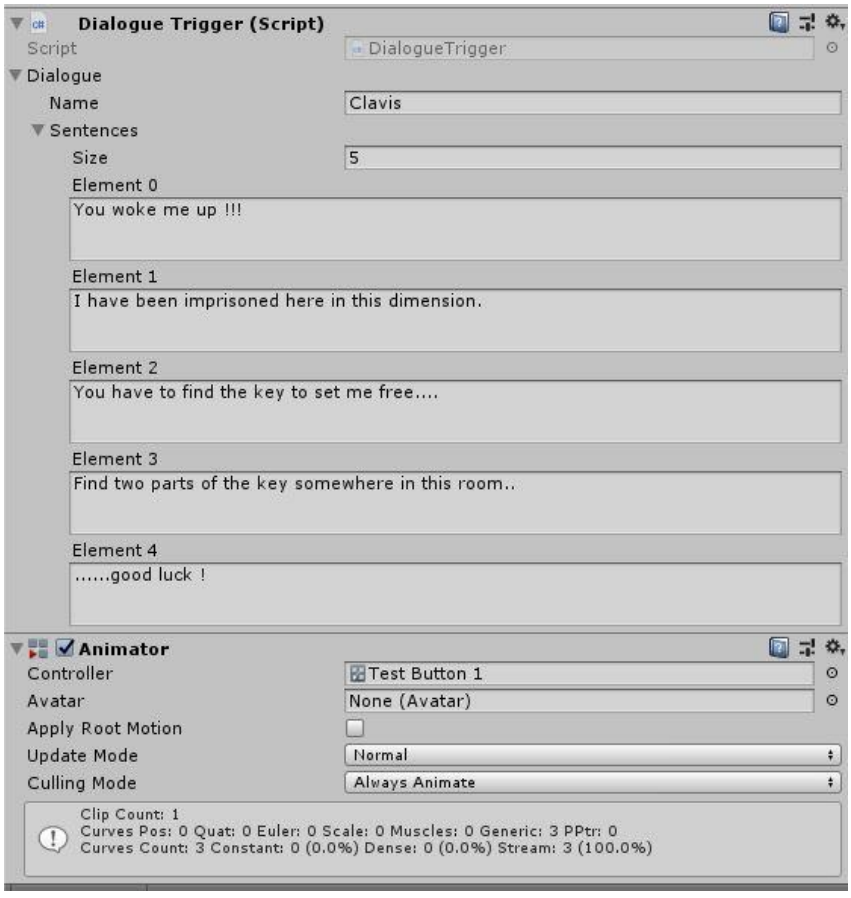
Slika 41. Strane sobe. Izvor (Autorski rad)

3.3.9. Izrada dijaloga i njegovo kodiranje u Visual studio code-u

Kod izrade dijaloga prvi korak bio je njihovo skriptiranje, to jest pisanje dijaloga kako bi poslužio pričanju priče i informirao igrača o daljnjim postupcima. U ovom dijelu pričat će se o koracima pomoću kojih se dobiva krajnji rezultat, tri dijaloga koji na neki način označavaju tri dijela igre.

3.3.10. Korak – skriptiranje dijaloga

Kod skriptiranja dijaloga je bitno u kratkim crtama objasniti zadatak kojeg igrač mora, kroz leću priče, ispuniti kako bi napredovao u igri. Na slici 42 prikazano je sučelje koje smo dobili definiranjem funkcije za serijalizaciju polja (stvaranje polja u inspektoru unutar kojih upisujemo rečenice koje će biti izrečene tijekom animacije dijaloga)



Slika 42. Prikaz sučelja dijaloga *u Unityju*. Izvor (Autorski rad)

Postupak kojim se dobilo sučelje prikazano na slici prije, kao i cijeli postupak izrade mehanizama dijaloga prikazan je u sljedećim koracima:

3.3.11. Izrada UI elemenata

Na početku se kreće s izradom UI elemenata koji se sastoje od okvira za dijalog, gumba koji omogućuje aktivaciju dijaloga, te gumba koji omogućuje progresiju teksta. Što se tiče teksta, unešen je jedan prostor za ime, a drugi za tekst dijaloga. Kao okvir je korištena slika izrađena u Clip sudio paint-u. Font koji je korišten kod izrade dijaloga isti je kao i kod ostalih tekstova (*Mistery Font*).

3.3.12. Izrada skripti

Sljedeći korak jest izrada skripti. Kreirana je nova skripta naziva *Dialogue Manager*. Također se stvara prazan objekt istog naziva na kojega se veže prije spomenuta skriptu. Skripta se otvara u Visual Studio Code-*u* gdje se piše kod.

podatkovni prostor *(*eng. *Dataspace*) koristi se kako bi se mogao iskoristiti redoslijed prema kojem će se dijalog kretati

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class DialogueManager : MonoBehaviour{
```

Isječak koda 2. Stvaranje reference za UI elemente

Nakon toga se definiraju varijable u koje će se kasnije (u Unity-ju) upisivati tekst:

```
public Text nameText;
public Text dialogueText;
```

Isječak koda 3. Definicija varijabli.

Definira se funkcija kako bi se mogao koristiti animator (animator je funkcija u Unity-ju pomoću koje se snimaju animacije korištene u igri).

```
public Animator animator;
```

Isječak koda 4. Definicija animatora.

```
private Queue<string> sentences;
void Start(){
sentences = new Queue<string>(); //inicijaliziranje funkcije}
```

Isječak koda 4. Učitavanje rečenica u sekvenci.

```
public void StartDialogue (Dialogue dialogue) {
```

Isječak koda 5. Pozivanje funcije koja se definira u skripti (*DialogueTrigger*)

```
animator.SetBool("IsOpen", true);
nameText.text = dialogue.name;
```

Isječak koda 6. Postavljanje uvjeta za animator

brisanje rečenica koje su prošle, dodavanje svih rečenica u red (eng. *Queue*), i stvaranje metode za prikazivanje rečenice kao i stvaranje uvjeta prema kojemu će program znati kada je kraj rečenice u dijalogu.

```
sentences.Clear();
foreach(string sentence in dialogue.sentences){
sentences.Enqueue(sentence); }
DisplayNextSentence();}
public void DisplayNextSentence (){
if (sentences.Count == 0)  // {
EndDialogue();
return;}
```

Isječak koda 7. Stvaranje uvjeta u kodu

Nakon toga se poziva korutina. Izrada korutine kako bi se rečenica ispisivala slovo po slovo i tako dobila animacija rečenice:

```
string sentence = sentences.Dequeue();
StopAllCoroutines();
StartCoroutine(TypeSentence(sentence));}
IEnumerator TypeSentence (string sentence){
dialogueText.text = "";
foreach (char letter in sentence.ToCharArray();{
dialogueText.text += letter;
yield return null;}}
```

Isječak koda 8. Izrada korutine

```
void EndDialogue(){
FindObjectOfType<DialogEnded>().SetDialogEnded(true);
animator.SetBool("IsOpen", false);}}
```

Isječak koda 9. Postavljanje uvjeta za animator

3.3.13. Izrada nove skripte naziva “Dialogue”

Sljedeća skripta će se koristiti kao objekt koji se prebacuje u *DialogueManager-u* kada bi se god trebao započeti novi dijalog tako da će ova skripta sadržavati sve informacije koje su potrebne za izvršavanje jednog dijaloga

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[System.Serializable]
public class Dialogue {
public string name;
[TextArea(3, 10)]
public string[] sentences;}
```

Isječak koda 10. Naredba za određivanje mogućnosti definiranja dužine rečenice kao i definiranje imena NPC-a s kojim se ulazi u dijalog

3.3.14. Implementiranje koda

Izradom skripte su se definirale varijable, kao i uključile razne funkcije koje su potrebne za funkcioniranje logike u igri. Skripte i dalje nisu povezane pa prema tome nemaju zajedničku funkciju. Povezivanje se postižem izradom treće skripte koju smo nazvali *DialogueTrigger*. Ova će skripta „sjesti“ na objekt, i poslužit će kao pokretač dijaloga. Skripta se veže na gumb „dialogueTrigger“.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DialogueTrigger : MonoBehaviour{
public Dialogue dialogue;(Dialogue)
public void TriggerDialogue(){
FindObjectOfType<DialogueManager>().StartDialogue(dialogue); }}
```

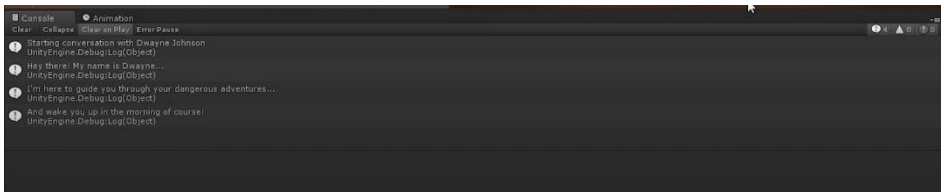
Isječak koda 11. Pozivanje funkcije za start dijaloga

Ovim kodom se dobiva interface u Unity-ju, gdje se može upisati ime lika s kojim igrač uspostavlja dijalog, te možemo dodati koliko god rečenica želimo. Također, ovim se postupkom lagano može izvršiti više dijaloga kroz igru. Mogućnost i potencijal *DialogueTriggera* je skoro neograničen zbog toga što bilo što može pokretati dijalog (dolazak igrača na određeno područje, pritisak određenog gumba, prelaženje jednog dijela igre, pomak lika na određenu stranu u sobi….itd).

3.3.15. Povezivanje skripti s njihovim gumbima

Selektira se „test gumb“, to jest gumb s kojim se pokreće dijalog. Postavlja se uvjet (*OnClick*) kako bi se definirao slijed funkcija koji se pokreće pritiskom tog gumba. Odabire se funkcija „*TriggerDialogue*“ koja se pokreće nakon pritiska gumba. Dvije skripte sada međusobno komuniciraju. Nakon toga se povezuje gumb koji služi za nastavak dijaloga (pritiskom na taj gumb pokreće se sljedeća rečenica u sekvenci). To se radi postupkom povuci i spusti gdje se skripta (za prikaz sljedeće rečenice) povezuje s gumbom za

nastavak. Kada je skripta povezana, određuje se funkcija koja će se tada započeti izvoditi, a to je funkcija naziva „DisplayNextSentence()" koja je definirana u skriptama prije. U ovom dijelu izrade jako je bitno pratiti konzolu programa u Unityju kojom se provjerava logika i utvrđuje funkcionalnost kao i ponašanje koda.



Slika 43. Prikaz konzole u Unity-ju. Izvor (Autorski rad)

Nakon toga se povezuje *DialogueManager* s varijablama imena i teksta kojeg će određeni lik izgovoriti (funkcija:„nameText.text = dialogue.name;")

3.3.16. Animacija aktivacije dijaloga, i njegovog zatvaranja (eng. *In and Out*)

U ovom dijelu ponovno se iskorištava postupak animacije kako bi se animirala sekvenca pokretanja dijaloga, i njegovog zatvaranja. Ovo se postiže selektiranjem „*DialogueBox-a*", nakon čega se selektira animator (*Window → Animation → Create*). Stvaraju se dva stanja (animacija otvaranja i animacija zatvaranja). Na slici 46 prikazano je sučelje kod izrade dijaloga, kao i prikaz konačnog izgleda na slici 47.

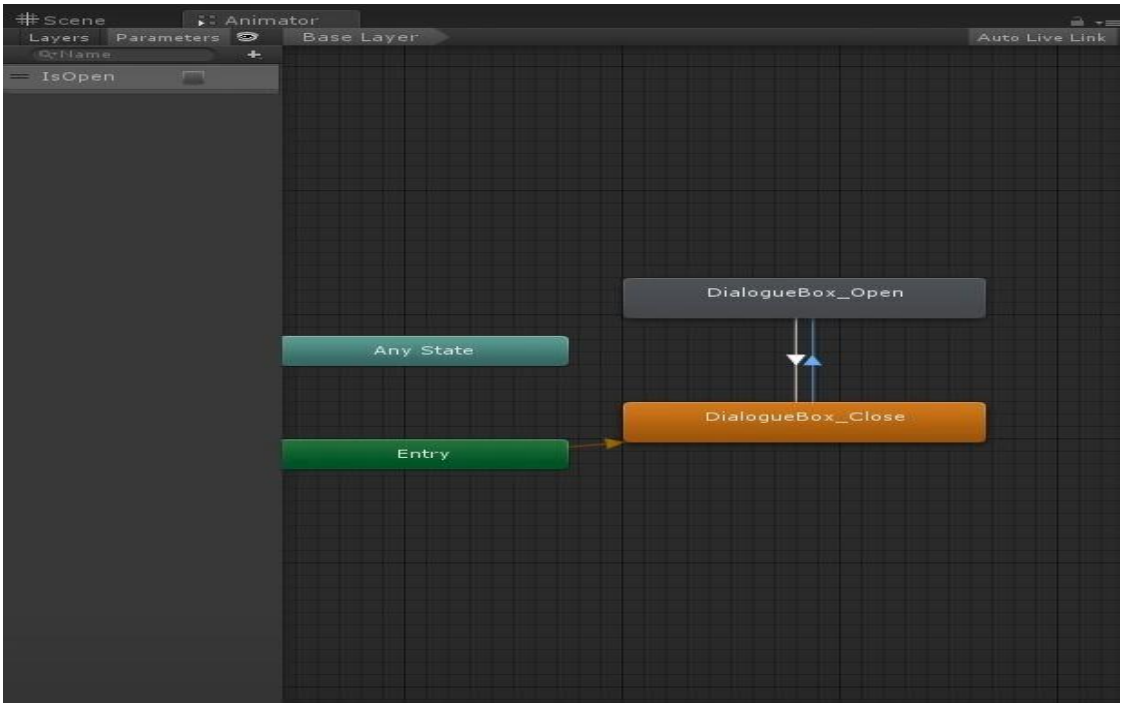
3.3.17. Animacija otvaranja

Animacija otvaranja dijaloga postiže se pritiskom na opciju „record" gdje pomoću „*Key frame-ova*" postavljena je početna i završna pozicija *DialogueBoxa* na kraju animacije. Ovaj proces je dosta sličan izradi animacije uz programima kao što je Adobe After effects[19]. Nakon što su obje pozicije, i njihova animacija definirana, pokreće se Animator, gdje se određuju eventi kojima se definira kada se koja animacija počinje

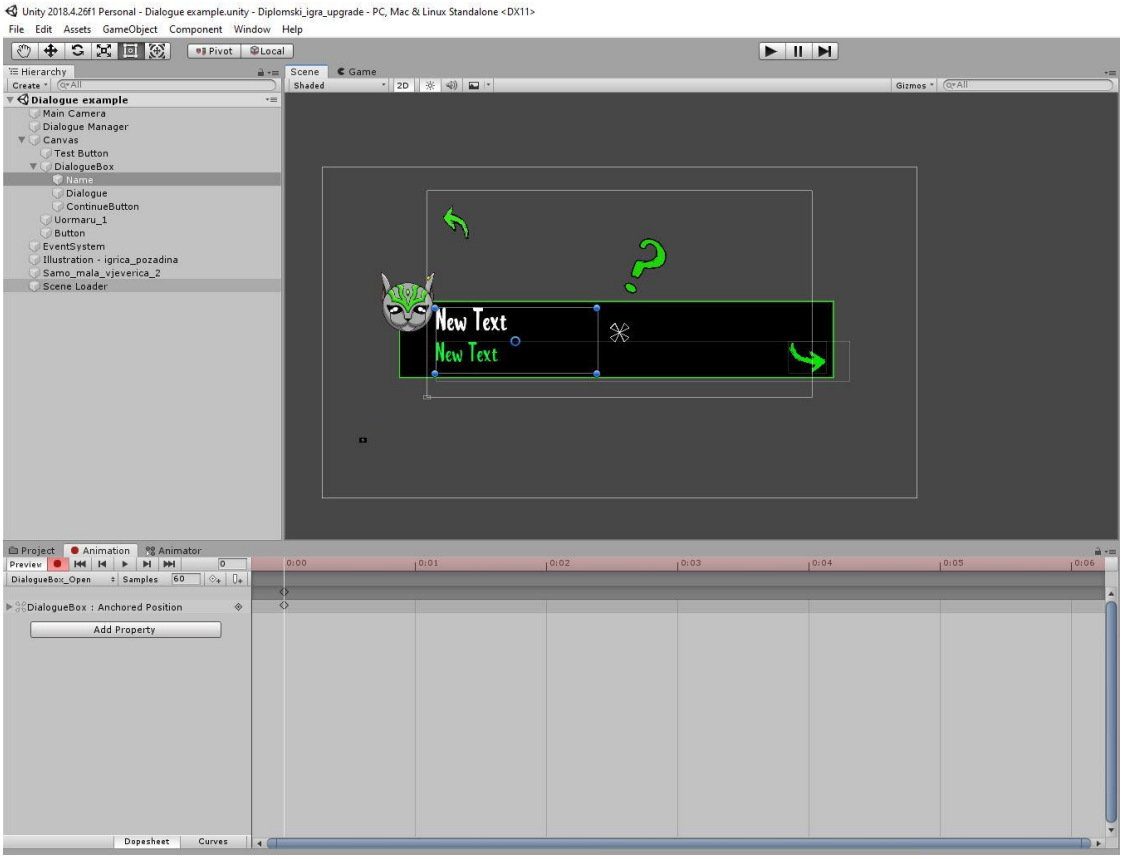
[19] Adobe After Effects, ili skraćeno After Effects, je program, razvijen i izdan od strane američke tvrtke Adobe Systems. Program se koristi za vizualne efekte, pokretne grafike i tipografiju. Također se koristi u postprodukciji kod izrade filmova i televizijskih produkcija.

izvršavati (sučelje Animatora liči na sučelje namještanja materijala u programu kao što je Blender). Ovdje se nalaze dva stanja koja su snimana (*DialogueBox_Open*, i *DialogueBox_Close*). Nakon toga se *DialogueBox_Open* postavlja kao početno stanje pritiskom desnog klika, te odabirom „*Set as Layer Default State*“. Nakon toga potrebno je postaviti parametar kojim se provjerava da li je *DialogueBox* trenutno otvoren nakon čega se dva stanja spajaju kako je prikazano na slici 45 (Uz spajanje mora se postaviti i uvjet koji kaže da će stanje prijeći iz *Closed* u *Open* samo onda ako je parametar *isOpen=true* , isto se tako odredi za *isOpen=false*). Ovim postupkom se dobiva nova varijabla u *DialogueManager-u* u koju se tada stavlja *DialogueBox* kao što se može vidjeti na slici 44.

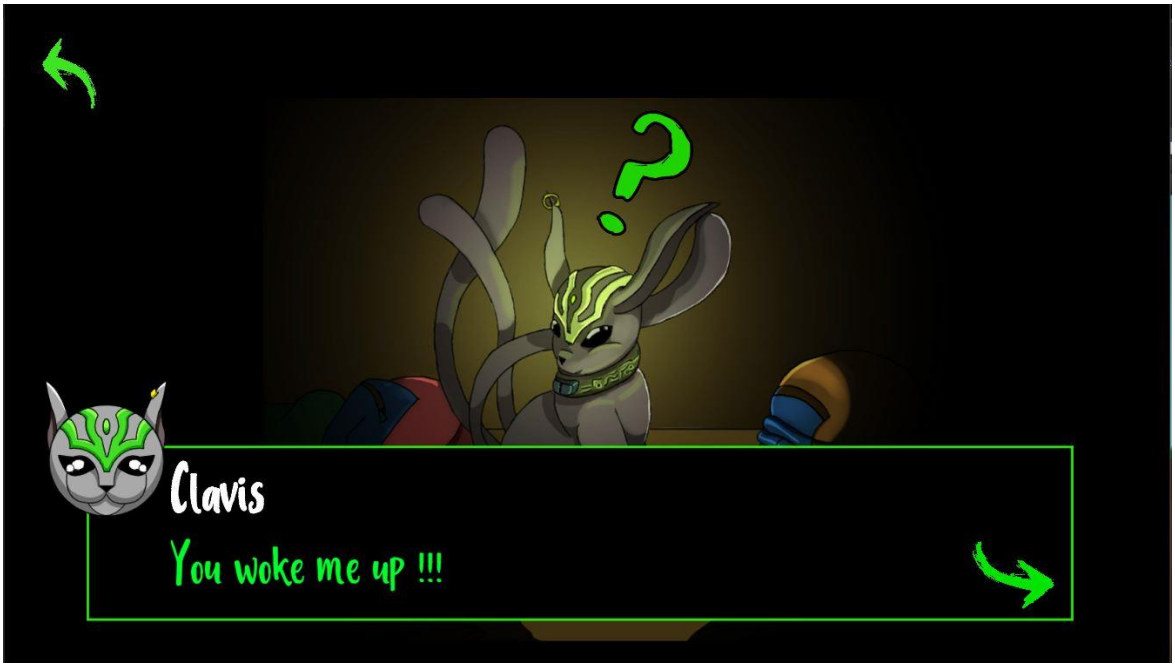
Slika 44. Inspector u Unity-ju. Izvor (Autorski rad)



Slika 45. Prikaz *Animatora* u Unity-ju. Izvor (Autorski rad)



Slika 46 Sučelje *Dialogue Box-a* u Unity-ju. Izvor (Autorski rad)



Slika 47. Prikaz dijaloga u igri**.** Izvor (Autorski rad)

3.3.18. označavanje scena i njihovo implementiranje u kod

U kodu su se morale definirati scene tako da su se definirale funkcije („LoadNextScene“, „LoadPrevScene“,“ Lspustanjerucke“, „Otvaranjeprozora“, „Nebo”), kao i brojač. Brojačem se uspostavila logika putem koje program zna na kojoj se sceni nalazi.

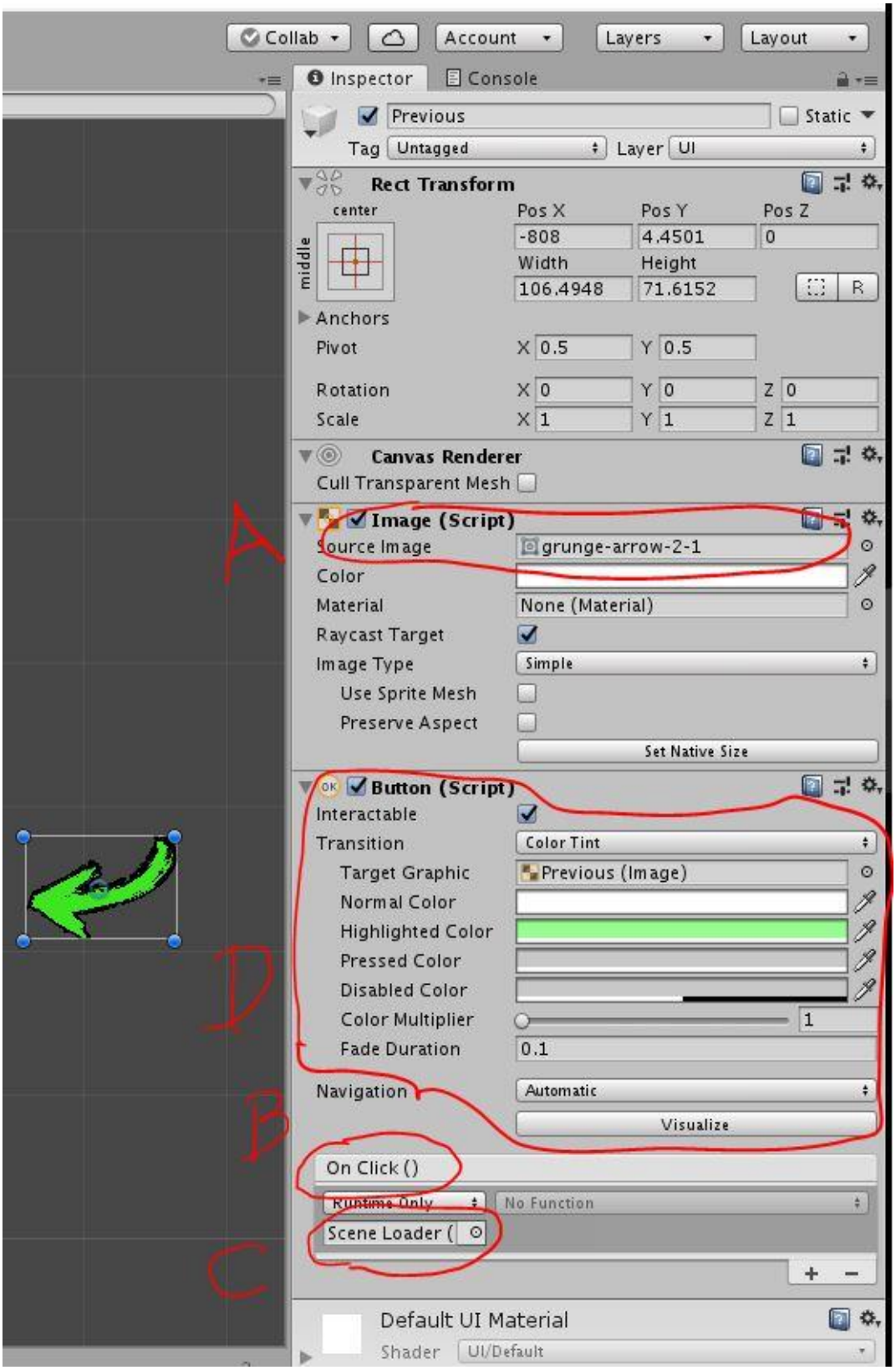
```
„int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
SceneManager.LoadScene(currentSceneIndex + 1);”
```

Isječak koda 12. Postavljanje brojača

Nakon toga se u Unity-ju pomoću graditelja scena (eng. *scene builder*) (Slika 48) definira redni broj svake scene te se one postavljaju u program što će kod daljnjeg programiranja služiti kao polazišna točka.

```
public void LoadStartScene(){
SceneManager.LoadScene(0);}
public void LoadPrevScene(){
SceneManager.LoadScene(1);}
public void Lspustanjerucke(){
SceneManager.LoadScene(10);}
public void Otvaranjeprozora(){
SceneManager.LoadScene(11);}
public void Nebo(){
SceneManager.LoadScene(12);}}
```

Isječak koda 13. Dodavanje funkcije različitim scenama



Slika 48. Inspector u Unity-u. Izvor (Autorski rad)

Slika 48 prikazuje sljedeći korak, a to je povezati skriptu s kodom i *Scene Loader*-om (objekt kojega smo stvorili u Unityju koji dodajemo UI objektu (u ovom slučaju gumb) kako bi dobio funkcionalnost u igri). Nakon toga *Scene Loader* povezujemo s gumbom koji također definiramo na način da odaberemo unaprijed definiranu funkciju koju nudi Unity ("On click()"). Pomoću toga postavljamo uvjet da se pritiskom guma kreće odrađivati kod u igri. (slika 46)

A – Dodali smo "sprite" (sliku) gumbu kako bi igra imala određen UI

B – korištenje integrirane funkcije "On Click()" kako bi se kripta krenula odrađivati kada igrač pritisne gumb.

C – mjesto na koji povlačimo skriptu s kodom (drag and drop)

D – ovdje manipuliramo izgledom gumba, ili bilo kojeg igraćeg objekta – boja, boja pritiska, boja kada kursor prijeđe preko objekta, itd.

Povezivanjem scena stvorila se određena dinamika kretanja po sobi, kao i interakcija s ostalim objektima (radili smo više scena gdje se za svaku interakciju scena morala promijeniti). Samo povezivanje scena je bio prvi korak, nakon toga se rad razdvojio na segmenta (izrada mini igara, izrada „hint" sistema, izrada dijaloga, dodavanje zvuka, programiranje same logike igre).

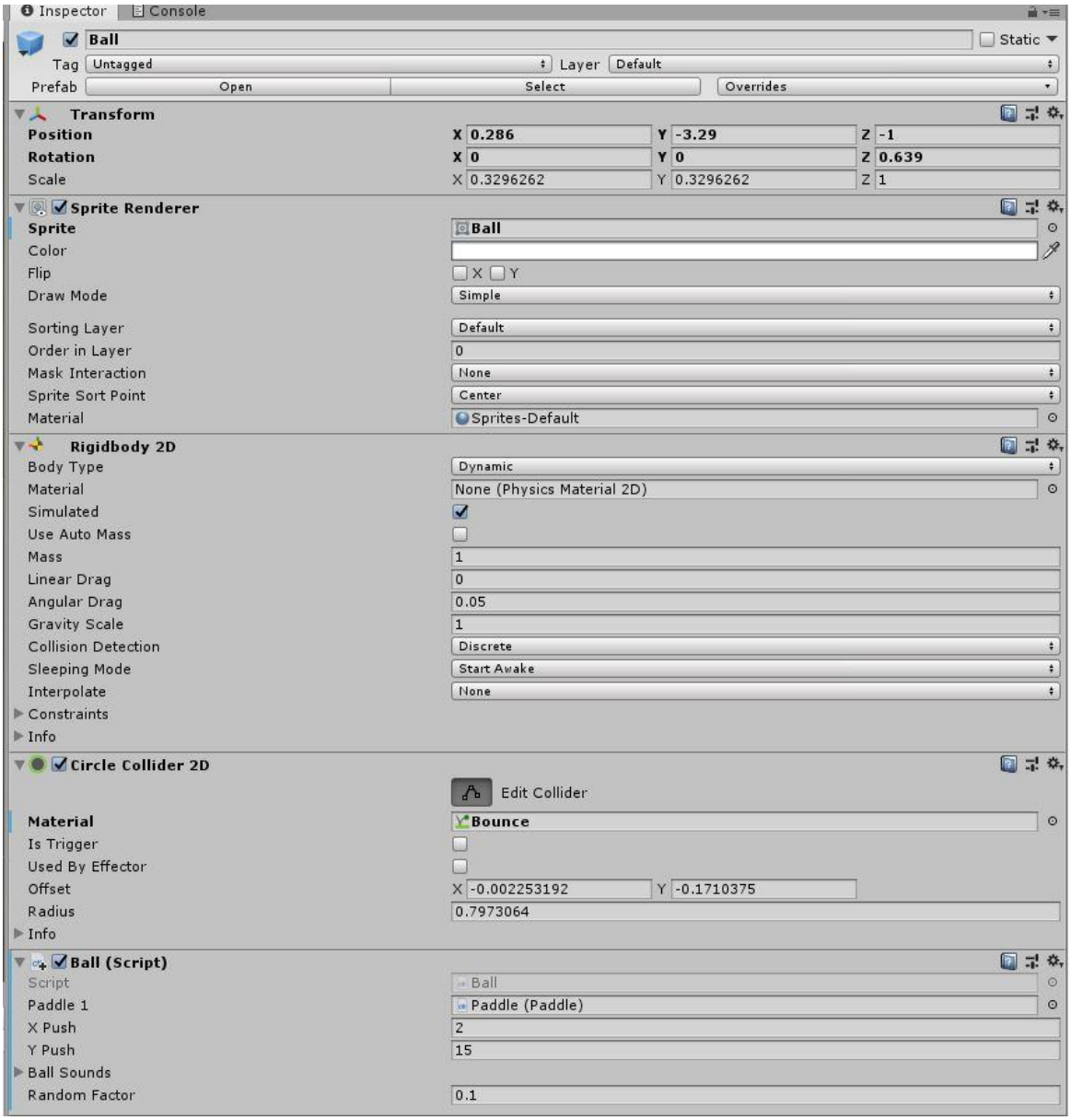
3.4. Izrada mini igara unutar igre

Mini igre kojima je svrha biti izazov igraču kako bi napredovao u igri. Igre su koncipirane tako da igrač prvo mora riješiti zagonetku kako bi otključao prvu igru, te isto to ponoviti za drugu igru. Poanta je skupiti dijelove ključa iz te dvije igre, i spojiti ga kako bi završio igru. Prva igra je naziva „*block breaker"*, a druga „*space invaders*".

3.4.1. Block Breaker

Kod block breakera trebali smo prvo izraditi sve potrebne igrače tehnikama crtanja i obojenja objašnjenih u prijašnjem tekstu te se tim objektima dodaje fizika, slike, i logika upotrebom skripti. Na slici 48 se može vidjeti prikaz inspektora kod izrade prve mini igre,

preciznije za izradu lopte u igri. Inspektorom dodajemo sliku igračem objektu, što je u ovom slučaju slika lopte. Bitno je dodati *Collider* (sudarač), kao i kruto tijelo objektu kako bi bilo moguće dodavanje fizike. Kruto tijelo se modificiralo kako bi što bolje prianjalo objektu. Ostale postavke su ostavljene kakve jesu



Slika 49. Inspektor za objekte u igri. Izvor (Autorski rad)

3.4.2. Lopta

Na početku se definiraju varijable kao i pozicija lopte te njeno pomicanje.

```
using UnityEngine;
using Vector2 = UnityEngine.Vector2;
public class Ball : MonoBehaviour{
```

Isječak koda 14. Definiranje javne klase i korištenja vektora.

```
[SerializeField] Paddle paddle1;
[SerializeField] float xPush = 2f;
[SerializeField] float yPush = 15f;
[SerializeField] AudioClip[] ballSounds;
[SerializeField] float randomFactor = 0.1f;
Vector2 paddleToBallVector;
bool hasStarted = false;
```

Isječak koda 15. Definiranje pozicije lopte i daske, te njihova serijalizacija (radi mogućnost promjene ulaznih podataka, radi kontroliranja)

```
AudioSource myAudioSource;
Rigidbody2D myRigidbody2D;
```

Isječak koda 16. Definiranje zvukova efekata u igri:

Pozivanje početka igre prije prvog kadra. Isto tako se postavlja ažuriranje stanja svaki sljedeći kadar

```
void Start(){
paddleToBallVector = transform.position - paddle1.transform.position;
myAudioSource = GetComponent<AudioSource>();
myRigidbody2D = GetComponent<Rigidbody2D>();}
```

Isječak koda 17. definiranje krutog tijela i zvukova kao funkcije

Isto tako se postavlja uvjet početnog stanja. Odnosno, za uvjet svakog sljedećeg ažuriranja. Uvjet konstatira da ako je stanje započelo, lopta mora biti zaključana za dasku s koje se "lansira"

```
void Update(){
if (!hasStarted){
LockBallToPaddel();
```

Isječak koda 18. zaključavanje lopte za dasku

```
LaunchBallOnMouseClick();}}

private void LaunchBallOnMouseClick(){
if (Input.GetMouseButtonDown(0)){
hasStarted = true;
myRigidbody2D.velocity = new Vector2(xPush, yPush);}}
```

Isječak koda 19. Lansiranje lopte pritiskom lijeće tipke miša

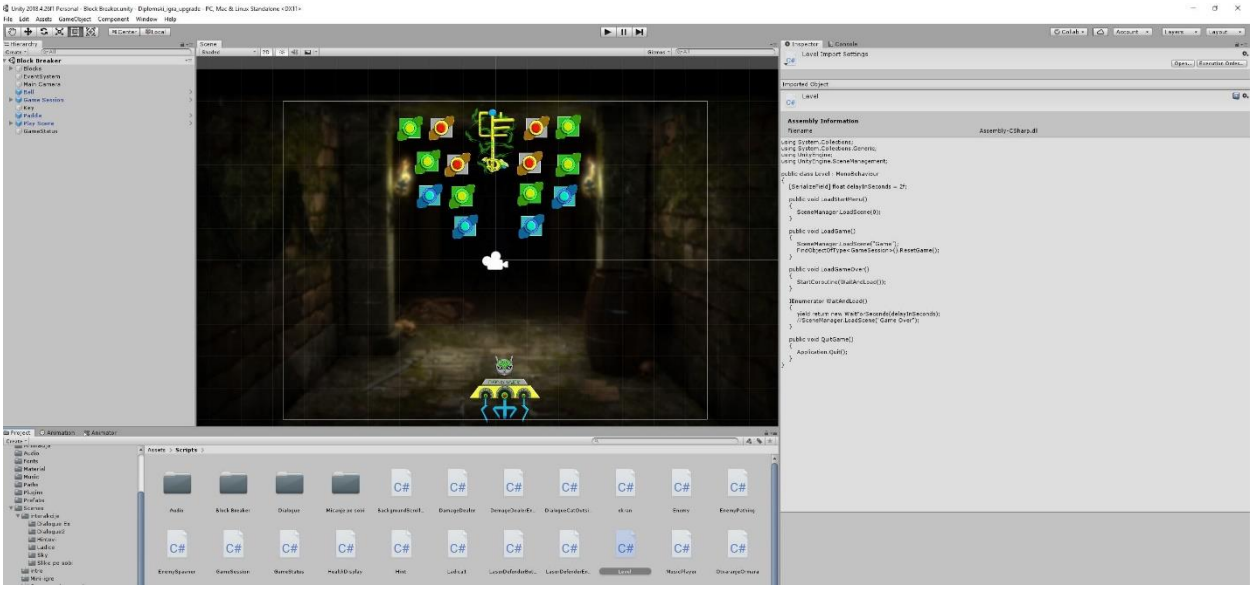
```
private void LockBallToPaddel()
{
Vector2 paddlePos = new Vector2(paddle1.transform.position.x,
paddle1.transform.position.y);
transform.position = paddlePos + paddleToBallVector;}
```

Isječak koda 20. Pozivanje funkcije kada lopta ponovo dođe u kontakt s daskom.

```
private void OnCollisionEnter2D(Collision2D collision)
{
Vector2 velocityTweak = new Vector2
(Random.Range(0f, randomFactor),
Random.Range(0f, randomFactor));
```

Isječak koda 21. Izrada funkcije koja definira slučajne vektorske smjerove kojima se lopta kreće nakon kolizije sa daskom.

Na slici 50 možemo vidjeti izgled igre u sučelju Unity-ja



Slika 50. Prikaz izrade block breakera. Izvor (Autorski rad)

Skripta kocke koja se razbija loptom

```
public class Block : MonoBehaviour{

[SerializeField] AudioClip breakSound;
[SerializeField] GameObject blockSparklesVFX;
[SerializeField] Sprite[] hitSprites;
[SerializeField] int timesHit;
```

Isječak koda 22. Definiranje serijalizacije polja (kako bi se ponovno mogli unjeti razni ulazni podaci koji su u ovom slučaju x i y smjer)

```
private void Awake(){
FindObjectOfType<GameStatus>().setGameObjectName("Key");}
```

Isječak koda 23. Definiranje funkcije koja određuje uvjet skupljanja ključa (koji se tada povezuje s ostalim skriptama (posljedica skupljanja ključa je otvaranje drugog dijaloga)

```
private void Start(){
CountBreakableBlocks();}

private void CountBreakableBlocks(){
if (tag == "Breakable"){
level.CountBlocks();}}

private void HandleHit(){
timesHit++;
int maxHits = hitSprites.Length + 1;
if (maxHits <= timesHit){
DestroyBlock();}
else{
ShowNextHitSprite();}}
```

Isječak koda 24. Postavljanje faze razbijanja blokova:

```
private void ShowNextHitSprite(){
int spriteIndex = timesHit - 1;
if (hitSprites[spriteIndex] != null){
GetComponent<SpriteRenderer>().sprite = hitSprites[spriteIndex];}else{
Debug.LogError("Block sprite is missing from array" + gameObject.name);}}
private void OnCollisionEnter2D(Collision2D collision){
if (tag == "Key"){
FindObjectOfType<GameStatus>().collectKey(true);
SceneManager.LoadScene(0);}
```

Isječak koda 25. Postavljanje “tagova”, ili uvjeta

```
if (tag == "Breakable"){
HandleHit();}}
private void DestroyBlock(){
PlayBlockDestroySF();
Destroy(gameObject);
level.BlockDestroyed();
TriggerSparklesVFX();}
private void PlayBlockDestroySF(){
```

```
FindObjectOfType<GameSession>().AddToScore();
AudioSource.PlayClipAtPoint(breakSound, Camera.main.transform.position);}
private void TriggerSparklesVFX(){
GameObject sparkles = Instantiate(blockSparklesVFX, transform.position,
transform.rotation);
Destroy(sparkles, 1f);}}
```

Isječak koda 26. Definiranje efekata nakon razbijenog bloka:

3.4.3. Daska

Slijedećom skriptom postiže se mogućnost pokreta daske u igri.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Paddle : MonoBehaviour{
[SerializeField] float minX = 4f;
[SerializeField] float maxX = 15f;
[SerializeField] float screenWidthInUnits = 16f;
```

Isječak koda 27. serijalizacija pokreta u igri

```
Ball theBall;
void Start(){
theBall = FindObjectOfType<Ball>();}
void Update(){
float mousePosInUnits = Input.mousePosition.x / Screen.width *
screenWidthInUnits - 8;
Vector2 paddlePos = new Vector2(transform.position.x, transform.position.y);
paddlePos.x = Mathf.Clamp(mousePosInUnits, minX, maxX);
transform.position = paddlePos;}
```

Isječak koda 28. Postavke granica pokreta u igri

Slika 51. Konačan prikaz prve mini igre. Izvor (Autorski rad)

3.4.4. Laser Defender

Micanje pozadine

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class BackgroundScroller : MonoBehaviour{
[SerializeField] float backgroundScrollSpeed = 0.5f;
Material myMaterial;
Vector2 offset;
void Start(){
myMaterial = GetComponent<Renderer>().material;
offset = new Vector2(0, backgroundScrollSpeed);}
void Update(){
myMaterial.mainTextureOffset += offset * Time.deltaTime;}}
```

Isječak koda 27. Ovom skriptom postižemo konstantno micanje pozadine kako bi dobili efekt micanja svemirskog broda kroz svemir.

Nanošenje štete

Prvo se, kao i kod lopte i daske, definira serijalizacija koja će u ovom slučaju predstavljati energiju letjelica koja u ovom slučaju iznosi 100

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DamageDealer : MonoBehaviour{
[SerializeField] int damage = 100;
public int GetDamage(){
return damage;}
public void Hit(){
Destroy(gameObject);}}
```

Isječak koda 28. Serijalizacija energije letjelica

Protivnici

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Enemy : MonoBehaviour{
[Header("Enemy Stats")]
[SerializeField] public float health = 500f;
[SerializeField] public int scoreValue = 0;
```

Isječak koda 29. Definiranje zdravlja protivnika, te koliko udaraca treba da bi se uništili

```
[Header("Shooting")]
public float shotCounter;
[SerializeField] public float minTimeBetweenShots = 0.2f;
[SerializeField] public float maxTimeBetweenShots = 3f;
[SerializeField] GameObject projectile;
[SerializeField] float projectileSpeed = 10f;
```

Isječak koda 30. Definiranje serijaliziranog polja

```
[Header("Sound Effects")]
[SerializeField] GameObject deathVFX;
[SerializeField] float durationOfExplosion = 1f;
[SerializeField] AudioClip deathSound;
[SerializeField] [Range(0,1)] float deathSoundVolume = 0.7f;
[SerializeField] AudioClip shootSound;
[SerializeField] [Range(0, 1)] float shootSoundVolume = 0.2f;
```

Isječak koda 31. Definiranje zvučnih efekata.

```
void Start(){
shotCounter = UnityEngine.Random.Range(minTimeBetweenShots,
maxTimeBetweenShots);}
void Update(){
CountDownAndShoot();}
private void CountDownAndShoot(){
shotCounter -= Time.deltaTime;
if(shotCounter <= 0f){
Fire();
shotCounter = UnityEngine.Random.Range(minTimeBetweenShots,
maxTimeBetweenShots);}}
```

Isječak koda 32. Postavljanje slučajnih interval “pucanja” protivnika

Definiranje kontakta krutog tijela glavne letjelice i laserskog metka protivnika, te što će se u tom slučaju dogoditi, kao i dodavanja zvučnog efekta svakom kontaktu. Isto tako se definira kada će se glavna letjelica uništiti. Ovdje je bitno postaviti kruto tijelo na igračevu letjelicu i protivničku letjelicu (slika 52).

Slika 52. Prikaz elementa u igri Laser defender. Izvor (Autorski rad)

Inicijalizacija laserskih metaka koji dolaze u kontakt s krutim tijelom, Kao i audio efekta koji se krene izvršavati kada se isti kontakt dogodi. Funkcija se naziva “Fire()”

```
private void Fire(){
GameObject laser = Instantiate(
projectile,
transform.position,
Quaternion.identity) as GameObject;
laser.GetComponent<Rigidbody2D>().velocity = new Vector2(0, -projectileSpeed);
AudioSource.PlayClipAtPoint(shootSound, Camera.main.transform.position,
shootSoundVolume);}
```

Isječak koda 33. Inicijalizacija laserskih metaka

Daljnje postavke uvjeta gdje se postavljaju “if” funkcije, te koliko štete mora primiti neko tijelo da bi se uništilo.

```
private void OnTriggerEnter2D(Collider2D other){
DamageDealer damageDealer = other.gameObject.GetComponent<DamageDealer>();
if (!damageDealer) { return; }
ProcessHit(damageDealer);}
private void ProcessHit(DamageDealer damageDealer){
health -= damageDealer.GetDamage();
damageDealer.Hit();
if (health <= 0){
Die();}}

private void Die()
{
FindObjectOfType<GameSession>().AddToScore(scoreValue);
Destroy(gameObject);
GameObject explosion = Instantiate(deathVFX, transform.position,
transform.rotation);
Destroy(explosion, durationOfExplosion);
AudioSource.PlayClipAtPoint(deathSound, Camera.main.transform.position,
deathSoundVolume);
```

Isječak koda 34. Postavljanje uvjeta za cjelokupnu igru

Postavljanje uvjeta koji utječe na cjelokupnu igru. Ako se glavna protivnička letjelica uništi, tada dolazi do promjene scene u glavnoj igri koja će voditi u daljnju progresiju (moći će se otvoriti ladica i sakupiti drugi ključ)

```
if (gameObject.tag == "BigBoy"){
FindObjectOfType<LaserDefenderEnded>().SetLaserDefenderEnded(true);
SceneManager.LoadScene(0);}}}
```

Isječak koda 35. Postavljanje uvjeta za cjelokupnu igru

Neprijateljsko kretanje

Nakon ponovnog definiranja klasa i „knjižnice funkcija“, stvaraju se smjerovi kretanja protivnika gdje se prema x i y parametrima pomoću uvjeta programira njihovo slučajno gibanje. Definiraju se varijable pozicija kao i njihove transformacije.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyPathing : MonoBehaviour{
WaveConfig waveConfig;
List<Transform> waypoints;
int waypointIndex = 0;
void Start(){
waypoints = waveConfig.GetWaypoints();
transform.position = waypoints[waypointIndex].transform.position;}
void Update(){
Move();}
public void SetWaveConfig(WaveConfig waveConfig){
this.waveConfig = waveConfig;}
void Move(){
if (waypointIndex <= waypoints.Count - 1){
var targetPosition = waypoints[waypointIndex].transform.position;
var movementThisFrame = waveConfig.GetMoveSpeed() * Time.deltaTime;
transform.position = Vector2.MoveTowards
(transform.position, targetPosition, movementThisFrame);
```

Isječak koda 36. Stvaranje smjerova kretanja protivnika

Također je postavljen uvjet uništavanja neprijatelja međusobno (ako interaktiraju jedan s drugim)

```
if (transform.position == targetPosition){
waypointIndex++;}}
else{
Destroy(gameObject);}}
```

Isječak koda 37. Postavljanje uvjeta za uništavanje protivnika

Stvaranje protivnika

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemySpawner : MonoBehaviour
{
[SerializeField] List<WaveConfig> waveConfigs;
[SerializeField] int startingWave = 0;
[SerializeField] bool looping = false;
IEnumerator Start(){
do{
yield return StartCoroutine(SpawnAllWaves());}
while (looping);}
private IEnumerator SpawnAllWaves(){
for(int waveIndex = startingWave; waveIndex < waveConfigs.Count; waveIndex++){
var currentWave = waveConfigs[waveIndex];
yield return StartCoroutine(SpawnAllEnemiesInWave(currentWave));}}
private IEnumerator SpawnAllEnemiesInWave(WaveConfig waveConfig){
for (int enemyCount = 0; enemyCount < waveConfig.GetTimeNumberOdEnemies();
enemyCount++){
var newEnemy = Instantiate(
waveConfig.GetEnemyPrefab(),
waveConfig.GetWaypoints()[0].transform.position,
Quaternion.identity);
newEnemy.GetComponent<EnemyPathing>().SetWaveConfig(waveConfig);
yield return new WaitForSeconds(waveConfig.GetTimeBetweenSpawns());
}}}
```

Isječak koda 38. Serijalizacija za definiranje broja protivnika i postavljanje instance protivnika

Stvaranje igrače sesije

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameSession : MonoBehaviour{
int score = 0;
int health = 0;

private void Awake(){
SetUpSingleton();}

private void SetUpSingleton(){
int numberGameSessions = FindObjectsOfType<GameSession>().Length;
if (numberGameSessions > 1){
Destroy(gameObject);}
else{
DontDestroyOnLoad(gameObject);}}
public int GetScore(){
return score;}
public int GetHealth(){
return health;}
public void AddToScore(int scoreValue){
score += scoreValue;}
public void ResetGame(){
Destroy(gameObject);}}
```

Isječak koda 39. Stvaranje igrače sesije

Označavanje stanja i zabilježavanje stanja igre (eng. Game status)

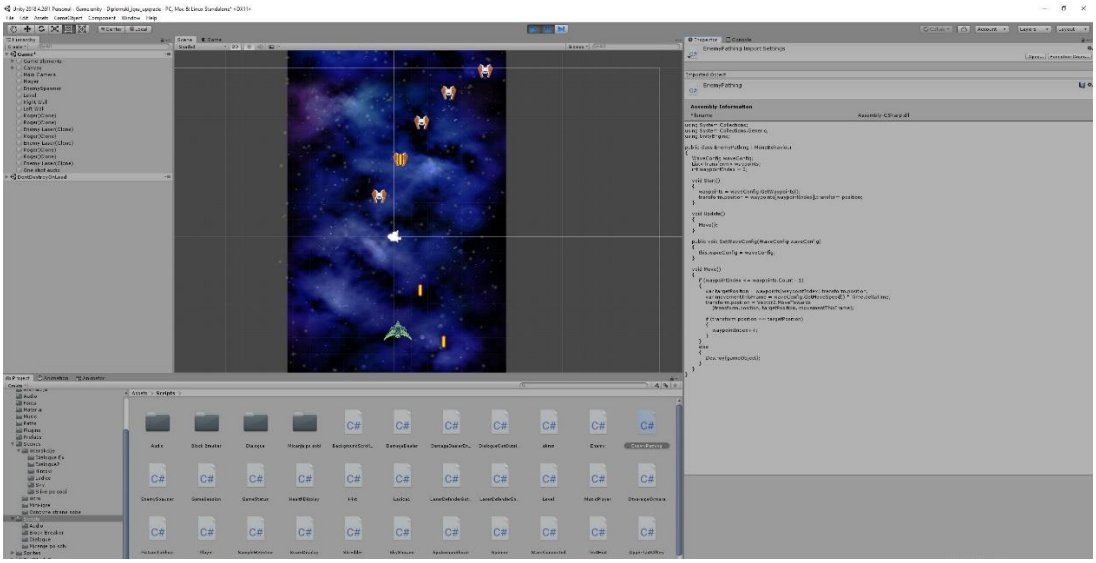
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameStatus : MonoBehaviour{
private bool _Key;
private string _gameObjectName;

void Awake(){
GameStatus[] gameStatus = FindObjectsOfType<GameStatus>();

foreach (GameStatus gameStatusI in gameStatus){
if (gameStatusI._gameObjectName == "Key"){
Destroy(gameStatusI);}
else{
DontDestroyOnLoad(gameObject);}}}

public void setGameObjectName(string gameObjectName){
_gameObjectName = gameObjectName;}

public void collectKey(bool Key){
_Key = Key;}
public bool getKey(){
return _Key;}}
```

Isječak koda 40. Postavljanje početnog statusa u igri

Prikaz zdravlja

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class HealthDisplay : MonoBehaviour{
Text healthText;
Player player;
void Start(){
healthText = GetComponent<Text>();
player = FindObjectOfType<Player>();}
void Update(){
healthText.text = player.GetHealth().ToString();}
```

Isječak koda 41. Postavljanje prikaza zdravlja kao i teksta koji to prikazuje



Slika 53. Laser defender. Izvor (Autorski rad)

Nakon izrade mini igara slijedi njihovo implementiranje u igru izradom skripti unutar kojih će biti definirana logika i razni uzročno posljedične veze i prekidači (Ako igrač pokrene "event" u igri, taj "event" tada pokreće sljedeću sekvencu). Na primjer postavila se sljedeća logika:

Igrač pritisne računalo → pokreće se prva mini igra (block breaker)→ Igrač je pokupio ključ→ otvara se mogućnost otvaranja prozora i pokretanje drugog dijaloga.

4. ZAKLJUČAK

Ovim diplomskim radom htjelo se istražiti kako se uporabom različitih programa, tehnika i programskih jezika mogu izraditi igru koja može poslužiti kao platforma različitih funkcija (portfolio, marketinški alat, promocija, sredstvo za zabavu, kao i ostale funkcije). Sama igra ima vlastitu priču koja se uvijek može nadograđivati (dodavanjem više soba, više mini igara koje donose više beneficija i dovode do veće funkcionalnosti). Prošavši sve faze, od stvaranja koncepta, crtanja, digitaliziranja crteža, izrade animacija, izrade i unošenja skripti potrebnih za funkcionalnost različitih objekata u Unityju, dodavanjem skripti na igrače objekte, dodavanje zvučnih efekata te povezivanjem svih tih različitih komponenti u jednu smislenu cjelinu dobio se gotov proizvod koji se može koristiti u više svrha i pružiti potrošačima određeno iskustvo igranja. Samim učenjem Unity-ja, Visual studio code-a, kao i istraživanjem tržišta te testiranjem proizvoda na potrošačima otkrile su se razne tehnike koje omogućuju izradu jedne ovakve igre kao i puno novih mogućnosti iskorištavanja tih tehnika u ostalim sferama tržišta. Istražilo se kako se igrača industrija mijenjala kroz povijest u ovisnosti o smjeni trendova te njen potencijal u budućnosti koji ju ne veže striktno za zabavu, već i za razvoj tehnologija, marketinških trikova, pa i mogućnosti zarade. Izrađeni projekt je tek u „alpha“ fazi što znači da je implementirana igrača funkcionalnost, te su igrači elementi (eng. *assetI)* parcijalno implementirani. Igra u „alpha“ fazi je igriva i sadržava sve glavne funkcije. Projekt ima potencijala te bi s dodatnom doradom mogao zaživjeti kao funkcionalna igra spremna za tržište, ili biti iskorištena kao sredstvo promocije različitih proizvoda te pridonijeti grafičkoj industriji kao još jedno sredstvo gdje bi dizajneri mogli pokazati svoje radove na jedan inovativan i zabavan način. Finalna igra se može naći na stranici https://sharemygame.com/upload gdje je stvorena zajednica razvojnih inženjera koji dijele međusobna iskustva i znanja te se eksponencijalno brže ostvaruje napredak individualca.

5. LITERATURA

1. https://www.technologyx.com/featured/rise-indie-games-quick-look-genre-history/ (15.6.2021.)
2. Oskar Morgenstern, john Von Neuman, Theory of Games and Economic Behavior 3rd Edition (26.8.2021.)
3. Jesse Schell, The Art of Game Design: A Book of Lenses 1st Edition (3.9.2021.)
4. Howard Raiffa, Games and Decisions: Introduction and Critical Survey (Dover Books on Mathematics) (7.7.2021.)
5. https://journals.sagepub.com/doi/full/10.1177/2056305119880177 (23.5.2021.)
6. Valencia-Garcia, Rafael (2016). Technologies and Innovation: Second International Conference (23.6.2021.)
7. Common game development terms and definitions | Game design vocabulary | Unity (15.7.2021.)
8. https://towardsdatascience.com/i-play-video-games-but-not-for-entertainment-c20d28d998bf (23.8.2021.)
9. Tan, James."Introduction: Unreal Engine (Canterbury Software Summit 2013 slides)"*(PDF)*. Unreal Engine. (14.7.2021.)
10. Daniel Sanchez-Crespo Dalmau and Daniel Sanchez-Crespo, Core Techniques and Algorithms in Game Programming (27.9.2021.)
11. https://www.teachthought.com/learning/why-people-play-video-games/(9.9.2021.)
12. https://www.sekg.net/gamer-psychology-people-play-games/(5.9.2021.)
13. William Spaniel, Game Theory 101: The Complete Textbook(23.8.2021.)
14. Presh Talwalkar, The Joy of Game Theory: An Introduction to Strategic Thinking(14.8.2021.)

15. Bruce Bueno de Mesquita, The Predictioneer's Game: Using the Logic of Brazen Self-Interest to See and Shape the Future by (28.8.2021.)

6. POPIS MANJE POZNATIH RIJEČI I POJMOVA

Igrači elementi – elementi igre kojima se pridružuju skripte koje određuju njihovo ponašanje.

AAA igre – igre izdavane od strane velikih kompanija.

Pogonski sklop – softver primarno dizajniran za izradu igara.

Pojačano učenje – učenje čija je svrha skupljati podatke bazirano na principu pobjeda i poraza na čemu se gradi okruženje.

Mainstream – najčešće prihvaćen način razmišljanja ili djelovanja. Ono što je pristupačno publici, što je u modi.

Strojno učenje - grana je umjetne inteligencije koja se bave oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju empirijskih podataka.

Softver - neopipljivi dio računala (engl.*soft* znači meko, nježno) u kojeg se ubrajaju programi i podaci koji se nalaze na računalu, ne uključujući operacijski sustav.

Razvojni inženjer - osoba koja razvija i oblikuje računalne sustave, razvija i održava mrežne stranice, razvija nove igre itd.

Mikromenadžment – u taktičkim igrama označava upravljanje pojedinca suprotno upravljanju cijele jedinice.

Usmjeri i pritisni tip igre – tip igre u kojoj igrač klikom miša ili pritiskom miša igra igru (istražuje, rješava zagonetke, itd.)

Cloud - Engleska riječ cloud znači ‘oblak’. U nazivlju se odnosi na ono što se odvija mrežom, a u novije se vrijeme odnosi na internet. Mogućnost iznajmljivanja jednoga poslužioca ili više njih te pokretanja različitih aplikacija na njima.

Otvoreni kod - softver čiji je izvorni kod i/ili nacrti (dizajn) dostupan javnosti na uvid, korištenje, izmjene i daljnje raspačavanje (primjeri: Firefox web preglednik, MediaWiki softver, Joomla).