Izrada 3D računalne igre u programskom alatu Unity

Razum, Bruno

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:947668

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-03-28**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bruno Razum

IZRADA 3D RAČUNALNE IGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bruno Razum

Matični broj: Z-45056/16–I

Studij: ***Primjena informacijske tehnologije u poslovanju***

IZRADA 3D RAČUNALNE IGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, studeni 2020.

Bruno Razum

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema završnog rada je izrada 3D računalne igre u alatu Unity. U radu se detaljno opisuje planiranje, dizajniranje i izrada igre. Isto tako opisani su alati za pisanje koda programskog jezika C#, Visual Studio te osobito alat Unity i njegove mogućnosti pomoću kojih se radi dizajn i izrada igre. U kratkim se crtama spominje povijest igara te se uspoređuje Unreal engine i Unity, jedni od najpoznatijih alata za izradu igara. Na kraju je predstavljen detaljan opis najvažnijih koraka pri razvoju igre „Tower defense“ uz pomno objašnjenje koda i same logike igre te prikaz neki bitnih dijelova programskog koda.

Ključne riječi: C#, Unity, 3D, dizajn, animacija, razvoj igre, obrana kule

Sadržaj

1. Uvod 1
2. Povijest videoigara 2
3. Unity 4
3.1. Povijest Unitya 4
3.2. Unity IDE 5
3.3. Skladište imovine 5
3.4. Multiplatfoma 6
3.5. Globalizacija 6
3.6. Unity licenca 7
4. Unity i ostali alati 8
5. Obrana kule 10
6. Unity Editor 12
7. Razvoj igre 17
7.1. Tip i cilj igre 17
7.2. Dizajn igre 18
7.3. Programiranje i logika igrice 18
7.3.1. Klasa Waypoints 18
7.3.2. Klasa EnemyMovement 19
7.3.3. Klasa WaveSpawner 20
7.3.4. Klasa Turret 21
7.3.5. Klasa Bullet 22
7.3.6. Klasa „Node“ 23
7.3.7. Klasa BuildManager 25
7.3.8. Klasa Enemy 26
7.3.9. Klase LivesUI, WavesUI, MoneyUI 27

7.3.10. Klase PauseMenu, GameOver, CompleteLevel 27
7.3.11. Klasa CameraController 28
7.3.12. Klasa MainMenu 29
8. Zaključak 31
Popis literature 32
Popis slika 34

1. Uvod

U današnje vrijeme tehnologija napreduje i raste nevjerojatnom brzinom. Zahtjevi koji se stavljaju pred tehnologiju sve su veći i složeniji te se očekuju kvalitetni, brzi, ali istodobno i jeftini proizvodi. Budući da to vrijedi za sve grane u IT-u, vrijedi onda i za *gaming* industriju. Tvrtke moraju za klijente razvijati igre s nevjerojatno velikim detaljima, dizajnom i grafikom. Kako bi se zadovoljili svi navedeni uvjeti, bilo je potrebno osmisliti moćne alate koji bi ubrzali proces izrade i podigli kvalitetu igre. Jedan od takvih alata je Unity.

Unity alat vrlo je jednostavan za upotrebu. Služi za izradu raznih imovina (eng. *Assets*), animacija, zvukova, za povezivanje skripta najčešće pisanih u C#, isto tako služi za pronalaženje već gotove imovine koje su ljudi razvili i nalaze se na internetu (Unity Assets Store), a što ubrzava proces razvoja i više vremena se može posvetiti kvaliteti koda.

Unity služi za izradu 2D i 3D igre, podržava različite platforme, poput mobilne, PC, Mac i Playstation. Budući da je tema ovoga završnoga rada izrada 3D igre, u nastavku će se detaljnije opisati razvoj, planiranje i dizajn igre, te detaljna analiza programskog koda koji pokreće igricu i omogućuje njenu funkcionalnost.

2. Povijest videoigara

Prva videoigra pojavljuje se 50-ih godina prošlog stoljeća, točnije 1958. godine kada je napravljena videoigra Tennis for Two koju je osmislio William Higinbotham. [1] Prva računalna igra je napravljena 1962. godine, a zvala se Spacewar, a njezini su autori su bili Dan Edwards, Peter Samson i Martin Graetz.[20]

1972. osniva se tvrtka Atari. Riječ je o tvrtki koja cijelo desetljeće dominira u *gaming* industriji. Tvrtka je razvila igricu *Pong* koja je postala globalno popularna. Nakon toga Atari prebacuje igru Space Invaders u stroj za video igre. Riječ je strojevima koji su radili na principu ubacivanja kovanice te su time dobili mladu publiku koja je svoj džeparac trošila na igrice. 1977. Atari razvija Atari 2600 home console koju je kupilo preko 30 milijuna ljudi.[2]

1983. godine sjevernoamerička industrija videoigara doživjela je veliki krah zbog prekomjerne zasićenosti tržišta konzolama, zbog velike konkurencije igara te zbog razvoja prekompliciranih i nekvalitetnih igara. Smatra se da je videoigra ET koju je razvila tvrtka Atari jedna od najgorih igra ikad stvorena.[2]

Situacija se preokrenula i *gaming* industrija počela se oporavljati kada je 1985. Nintendo došao u SAD i razvio i dandanas popularne videoigre Super Mario Bros, The Legend of Zelda i Metroid. Druge, a još uvijek danas popularne igre koje su se razvile 80-ih godina prošlog stoljeća su Tetris, Pac-man, SimCity i dr.[3]

Može se reći da su videoigre tijekom devedesetih godina prošloga stoljeća ušle u novu dimenziju. Tada su se, naime, počele proizvoditi videoigre u 3D, igrači su se mogli kretati u tri, a ne samo u dva smjera. Sony je 1994. razvio i dandanas najpopularniju konzolu, PlayStation. Neke od igara razvijene u to doba su Command & Conquer, Tomb Raider, Wolfenstein 3D koja predstavlja prvu FPS (eng. *First Person Shooter*) igru i jednu od najprodavanijih igara do današnjeg dana. 1993. godine Id Software je napravio FPS Doom koji je bio prijelomna točka u grafici i dizajnu za to vrijeme. Neki autori ističu problem negativnoga utjecaja tih igara tijekom 90-ih godina, osobito s obzirom na povećanje nasilja i ubijanja.[2][3]

Početkom 2000. Internet još uvijek nije bio dovoljno jak kako bi se ljudi mogli međusobno igrati *online*, pa su se organizirali tzv. LAN Party. Svatko bi donio svoje računalo te bi se računala povezala u lokalnu mrežu. Jedna od popularnijih igara bila je Counter Strike. 2004., kada je Internet postao dovoljno jak da se igranje prebaci na *online* razinu, pojavila se jedna od najpopularnijih igara svih vremena World of warcraft koja se igra i dandanas.[2]

Moćne grafike učinile su svjetove videoigara još realnijima. Zahvaljujući umjetnoj inteligenciji, protivnici se više ne ponašaju na isti način u svakoj situaciji, već svako ima svoju

reakciju na ono što se događa u igri. Stvorene su igre otvorenoga svijeta u kojima igrači sami istražuju izmišljene svjetove i mogu slobodno odrediti tijek igre.[2]

3. Unity

Unity je višeplatformski alat koji služi za razvoj igara, a koji je razvila kompanija Unity Technologies 2005. godine. Predstavlja jedan od najpoznatijih alata za kreiranje igara, besplatan je svim korisnicima i vrlo pristupačan ljudima koji se tek upuštaju u svijet kreiranja igara. [4] Više o njemu u nastavku.

3.1. Povijest Unitya

Sve je počelo u ranim 2000. kada su tri mlada programera skromnih financijskih mogućnosti, u garaži krenuli razvijati softver koji će 2005. godine biti prvi put predstavljen u Worldwide Developers konferenciji tvrtke Apple Inc i nekoliko godina kasnije postati jedan od najpopularnijih alata za razvoj video igara.[6].

U početku je cilj bio napraviti igricu, ali radeći na njoj postupno su shvatili da im zapravo treba alat koji bi olakšao procese pri izradi igara. Joachim Ante doselio se iz Njemačke u Dansku kod Nicholasa Francisa, te zajedno s Davidom Helgasonom krenuli su u izradu Unitya. Glavni razlog za uspjeh bio je to što je Unity bio dostupan i besplatan za sve samostalne *developere* koji nisu imali mogućnost plaćanja velikih svota za licencu. Zatim su osnovali kompaniju koja se nazvala Over the Edge Entertainment (OTEE). OTEE je uvidio da su online igre budućnost te su se okrenuli samo na razvoj alata za tu branšu. Nakon što je prva verzija Unitya bila pri kraju, shvatili su da im treba igra koja bi bila dobra reklama za proizvod. Sljedećih pet mjeseci uložili su u kreiranje prve igre napravljene u Unityu - GooBall.

Slika 1: Prva Unity igra GooBall [15]

Nakon što su razvili igricu, koristili su je kako bi otkrili greške koje alat ima i elemente koje još treba dodati. Novac koji su zaradili igricom iskoristili su kako bi proširili tim i osmislili prvu konačnu verziju Unitya, a koja je izašla u šestom mjesecu 2005. godine.

Prva javno dostupna verzija Unity podržavala je samo Mac, a s verzijom 1.1 Unity je počeo podržavati izvoz u Windows operacijski sustav i internet preglednike. Isto tako, Unity je podržavao dodatke (eng. *plugins*) koji su omogućavali *developerima* dodatne mogućnosti; dodaci su bili programirani u C++ kao i sam Unity.

Kako je u to doba počelo masovno razvijanje pametnih telefona, Unity je odlučio razviti verziju Unitya koju će podržavati iPhone. 2009. godine s verzijom 2.5 Unity je počeo podržavati Windowse.

Unity je bio osmišljen kako bi podržavao izradu 3D igara, međutim 2013. godine izašla je verzija koja je počela podržavati dugo očekivane 2D igrice. [5]

3.2. Unity IDE

Za Unity se obično kaže da je motor igre (eng. *game engine*), međutim Unity je i IDE (eng. *integrated development environment*), integrirano razvojno okruženje koje predstavlja sučelje gdje imamo pristup svim alatima koji su potrebni za razvoj igre. Korisnici mogu koristiti opciju *drag and drop* za stvaranje jednostavnih elemenata. Postoji navigacija za sve datoteke koje čine jednu igru. Stvaranje animacija, otkrivanje sudara, 3D prikaz nikad nije bio lakši. [6]

Unity podržava tri različita programska jezika za pisanje skripta u IDE, Javascript, C# i Boo. Unity je pisan u C++.

3.3. Skladište imovine

Pred kraj 2011. godine kompanija Unity Technologies obznanila je plasiranje tržnice za Unity *developere Unity Asset Store*, skladište imovine. Skladište je ugrađeno u IDE, te *developeri* mogu kupovati, preuzimati besplatne ili stavljati svoje imovine a da pritom ne izlaze iz Unitya. Većinu prihoda uzima autor imovine, a dio uzima Unity. Procjenjuje se da ljudi koji su razvili jednu od imovina koje se nalaze pri samom vrhu prodaje mogu lagodno živjeti samo od te prodaje.

Glavni razlog zašto je nastao *Unity Asset Store* je taj kako bi se uštedilo vrijeme *developera* pri izradi raznih imovina i kako bi se više vremena moglo utrošiti na kvalitetu igre.

3.4. Multiplatfoma

Jedan od glavnih razloga zašto je Unity postao tako popularan je taj što podržava velik broj platformi. Potrebna su doslovno dva klika kako bi igricu pokrenuli odnosno prebacili na neku drugu platformu. Glavna četiri razreda koja podržava Unity su konzola, mobitel, internet i računalo. Tu naravno spada nama svima poznati iOS, Android, Xbox, Windows, Mac OS X i Linux.[5]

U 2010-ima Unity Technologies je počeo koristiti svoj alat, 3D platformu u filmsku i automobilsku industriju. Prvi film koji je koristio Unity bio je Adam, film o robotu koji je pobjegao iz zatvora. Nakon toga Unity je počeo surađivati s Neillom Blomkampom i nastala su dva filma napravljena računalom Adam: The Mirror i Adam: The Prophet. [7]

2017. godine Unity se fokusirao samo na snimanje filmova s novim Cinemachine alatom. [7] Cinemachine je alat za pametne, dinamične kamere koje omogućuju stvaranje najboljih snimki na temelju scene i interakcije, omogućuju podešavanje, ponavljanje, eksperimentiranje i imitiraju ponašanje kamere u stvarnom vremenu. Cinemachine je isto tako nagrađen Emmy-award nagradom.[8]

Zatim, 2018. godine Disney televizija plasira tri kratka crtića, Baymax Dreams koja su nastala alatom Unity. [17] Proizvođači automobila koriste Unity tehnologiju za stvaranje modela novih vozila u virtualnoj stvarnosti, izgradnju virtualnih linija za montažu i obuku radnika. [18] Tvrtke poput DeepMind, Alphabet Inc. koriste Unity za treniranje i razvijanje umjetne inteligencije. [19]

3.5. Globalizacija

Sve je počelo u malenom stanu u Danskoj, a sada Unity predstavlja globalnu kompaniju. Glavno središte nalazi se u San Franciscu, a ostali su uredi po cijelome svijetu, u Kanadi, Kini, Danskoj, Njemačkoj, Koreji, Japanu itd. Za manje od pet godina dostigli su brojku od 60 zaposlenih. Najveće prihode zarađuju u Americi, negdje oko 20%, 50% pripada ostalim zemljama, a 30% zarađuju od raznih aplikacija koje nisu vezane za *gaming* industriju. Trenutno Unity zapošljava preko dvije tisuće ljudi po cijelome svijetu, od Amerike, pa sve do Azije i Europe.[5] Unity ima preko sto milijuna mjesečno aktivnih i registriranih korisnika.[9]

3.6. Unity licenca

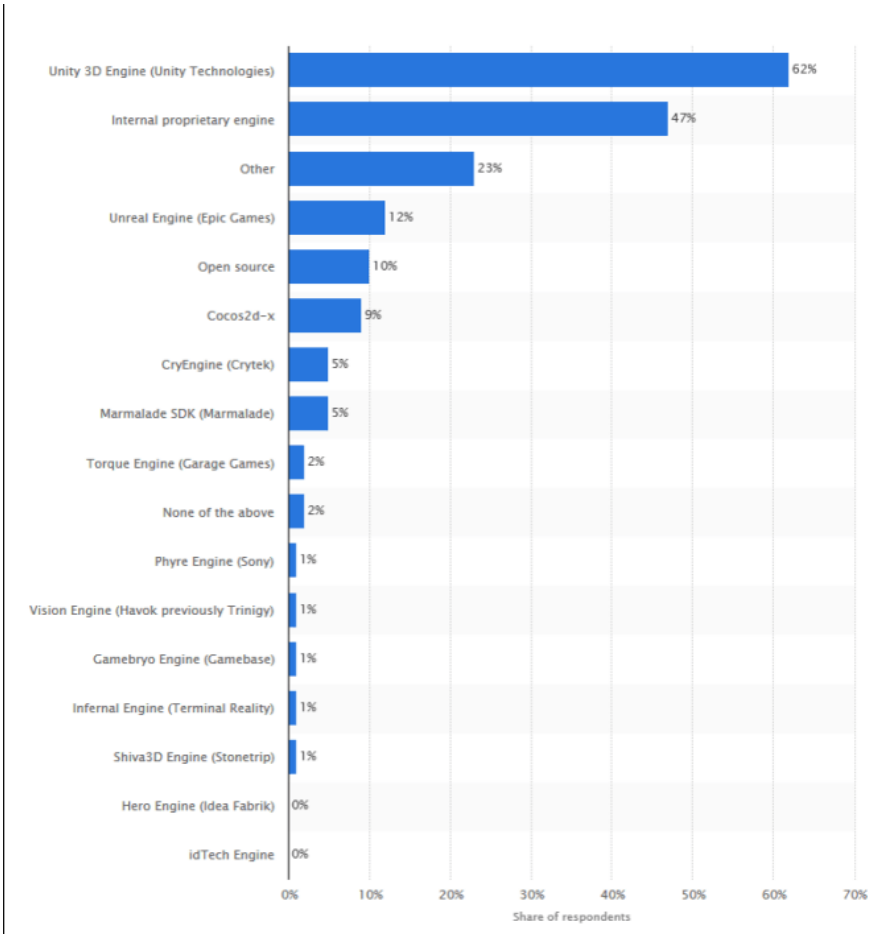
Postoji Unity licenca koja se plaća 1500€ mjesečno, no ona je potrebna kada tvrtka premaši profit od 100,000€. Ako je iznos ispod sto tisuća Unity se može koristiti besplatno i ima sve alate dostupne kao i Unity Professional opcija koja se plaća. Jedina razlika između besplatne licence i one koju se plaća je ta, da se prilikom pokretanja igrice koja je napravljena u Unityu pojavljuje na nekoliko sekundi logo Unitya i nemogućnost korištenja *Cache servera* koji je koristan pri srednjim i većim projektima zbog brzine unošenja imovine (eng. *assets*). [5]

4. Unity i ostali alati

Kako je u današnje vrijeme velika potreba za izradom igara, tako su se razvili i brojni drugi alati za izradu istih. Neki od popularnijih uz Unity su Unreal Engine, Cryengine. Pri odabiru pravog alata potrebno je voditi računa hoće li igra biti 2D ili 3D, na kojoj će se platformi sve igrica koristiti, kakve je veličine igrica koju radimo i mnogi drugi bitni faktori. Ipak trenutno najpopularniji alati su Unity i Unreal Engine te ćemo usporediti njihove mogućnosti i funkcionalnosti u nastavku teksta.

Prva bitna stvar je programski jezik koji koriste ova dva alata. Unity je napisan u C++, a piše se u C#, dok je Unreal Engine pisan u C++ i kod se piše u C++, što je važna činjenica zato što je C# dosta jednostavniji i pristupačniji programski jezik za osobe koji se nisu prije susretale s programiranjem. U Unityu mogu raditi ljudi koji imaju veoma skromno zvanje u programiranju zato što na *Asset Store* postoji dodatak koji se zove PlayMaker koji sam odradi cijelu logiku igrice, dok kod Unreal Engine to nije moguće. Kada dolazimo do grafike igrice onda je tu u prednosti Unreal Engine čije su mogućnosti daleko veće. Pruža realniju grafiku pomoću dinamičke rasvjete, sjene izgledaju puno prirodnije, sitni detalji su puno preciznije napravljeni, što međutim dovodi do nedostatka, a on se sastoji u činjenici da nam je za pokretanje igre potrebno puno snažnije računalo.

Drugi element važan za osobe koje tek kreću u kreiranje igara je sučelje koje pruža alat. Unity je omogućio puno jednostavnije i *user-friendly* sučelje; postoje opcije „drag i drop“ te se dosta radnji može odraditi bez pisanja jedne linije koda.



Slika 2: Prikaz alata koji se najviše koriste [14]

Na slici iznad možemo vidjeti kako je Unity engine daleko najviše korišten alat, iz toga se nameće zaključak da ima uvjerljivo najveću zajednicu. Valja naglasiti da Unreal Engine, isto tako, ima isto dosta aktivnih korisnika, različitih foruma, raznih tečaja koji nam dosta pomažu pri razvoju.

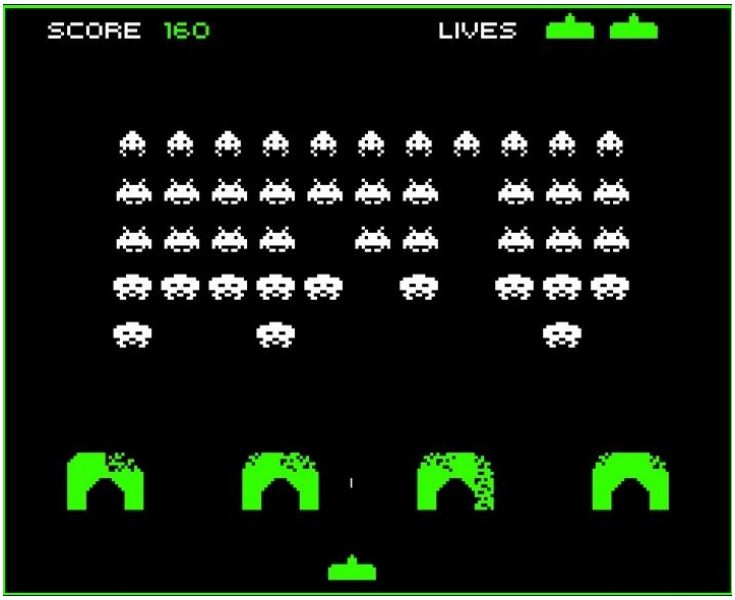
Što se tiče licenca, za Unity smo već spomenuli da postoji besplatna verzija i Unity Professional verzija koja se mora uzimati nakon što tvrtka zaradi preko sto tisuća dolara. Razlika među njima je veoma malena, u nekim slučajevima i zanemariva. Kod Unreal Engine imamo sličnu situaciju, besplatan je, no obveza plaćanja 5% od prihoda vrijedi u situaciji kada jedan od izrađenih proizvoda u Unreal Engine premaši iznos od tri tisuće dolara.

Ono što im je zajedničko i što ih čini najjačim alatima za izradu igara je to što podržavaju različite tipove igara kao što su 3D, 2D, Virtualna stvarnost, mobilne igre, *multiplayer* igre. Isto tako, pokrivaju skoro sve platforme Windows, macOS, iOS, Android, Playstation, Xbox i mnoge druge. Oba alata pokrivaju i neke druge industrije kao što je npr. filmografija.[11][12][13][14]

5. Obrana kule

Obrana kule (eng. *Tower defense*) je podvrsta strategijskih videoigara gdje je cilj obraniti kulu od neprijatelja, najčešće postavljanjem različitih obrambenih struktura koje se nalaze na samom putu ili pored puta gdje se neprijatelj kreće. Obrambene strukture automatski napadaju neprijatelje i nije potrebno njima upravljati.

Prva videoigra ovog tipa nastala je 1978., a zvala se Space Invaders. Cilj je bio obraniti od valova neprijatelja teritorij igrača koji je bio na dnu ekrana. Igrač je mogao koristiti štitove kako bi se zaštitio od neprijatelja ili ih omeo istim.



Slika 3: Space Invaders [16]

1980. Missle Command štitovima daje veću važnost, igrači su njima mogli ometati neprijatelje i svakim napadom je bilo sve više puteva kojim je neprijatelj mogao ići. Igrica je imala pokazivački uređaj, križić, što je bilo ispred svoga vremena i time su predvidjeli, odnosno najavili pojavu računalnih miševa.

Star Wars: The Empire Strikes Back koja je nastala 1982. godine od strane Parker Brother's popularizira ovakav tip igre. Uvedeno je više vrsta ometanja neprijatelja da ne dođe do kule. Kasnije, kada je počeo prelazak s arkadnih na osobno računalo, mnoge su je igre kopirale.

1990. godine nastala je igra Rampart. I ona se smatra prototipskom igrom Tower Defense. Obrambene strukture automatski napadaju neprijatelje, postoji više vrsta obrambenih

struktura, popravci kule su elementi koji se i danas uglavnom koriste pri izradi takvih videoigara. Iako je Rampart bio popularan, slične igre rijetko su se viđale sve do široko prihvaćenoga računalnog miša.

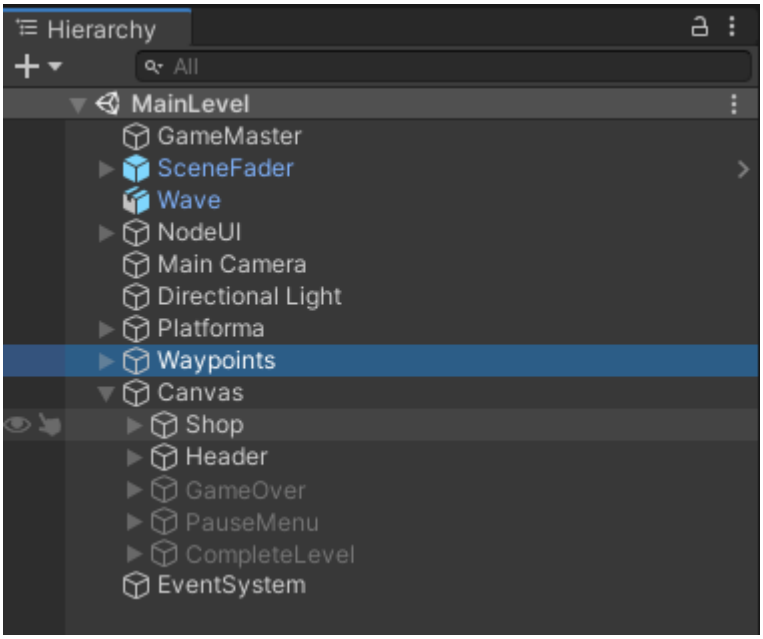
Kako je polako rasla popularnost PC igara, mnoge tvrtke su počele raditi ovakav tip igre za PC, pa tako i Warcraft III koji je izdao Element TD, a koja je bila svojedobno jedna od najpopularnijih igara ovakvog tipa.

Najveću popularnost ovaj tip igre doživio je između 2007. i 2008. godine, i to najviše zbog velikog skoka Adobe Flash nezavisnih *developera* i pojave pametnih mobitela. Najpoznatija igra je bila Flash Element TD. Sve ove igre bile su izrađene u 2D.

3D u ovakvom tipu igre prvi put se pojavio 2010. godine s igricom Dungeon Defenders koja se u prva dva tjedna prodala u preko 250 tisuća kopija, a preko 600 tisuća do kraja 2011.

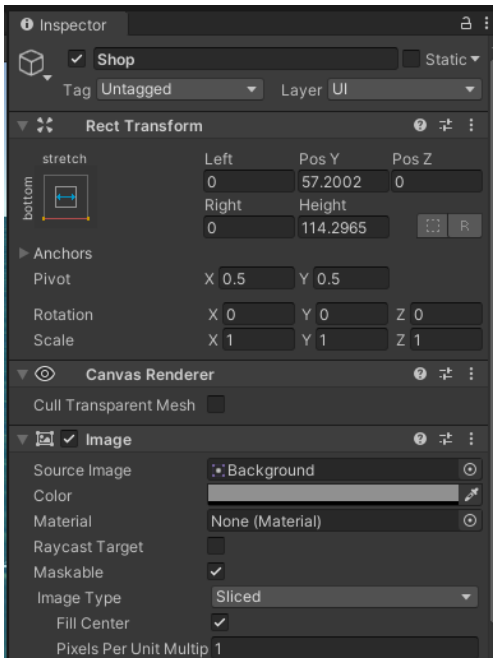
6. Unity editor

Kao što je više puta spomenuto, alat u kojem se radila igrica je Unity *editor*. *Editor* se sastoji od nekoliko bitnih dijelova. Počet ćemo s hijerarhijom: ona se nalazi na lijevoj strani i u njoj se nalaze svi objekti koji se trenutno nalaze na sceni.



Slika 4: Prikaz hijerarhije u Unityu

Omogućuje jednostavan i brz pristup svim objektima. Klikom na jedan od objekata otvara se s desne strane *Inspector*, dio u kojem se nalaze razni detalji samoga objekta poput lokacije, rotacije, veličine, skripte koje su povezane s objektom i mnoge druge stavke koje su vezane za, primjerice, dizajn objekta, slike itd.



Slika 5: Prikaz inspector dijela u Unityu

Dvostrukim brzim klikom na objekt u hijerarhiji uvećava nam se objekt na sceni što je često jako korisno kada želimo vidjeti nekakve detalje na objektu.

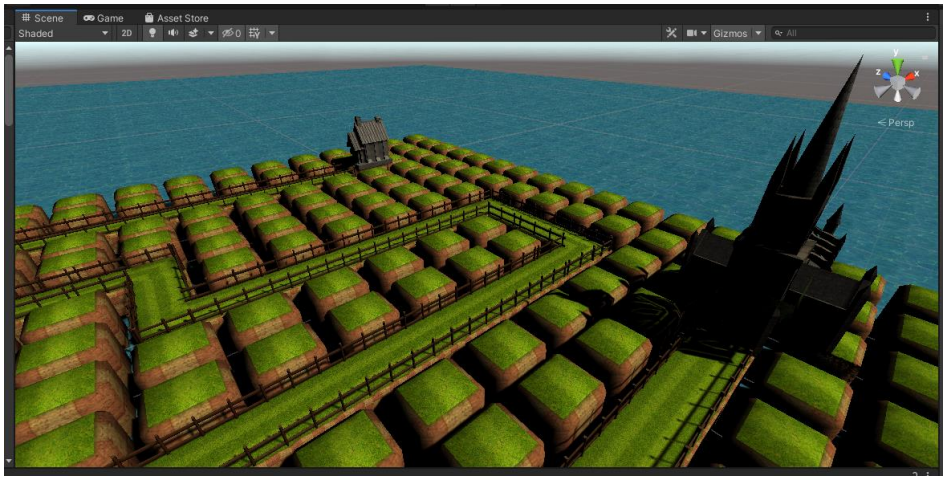
Slika 6: Prikaz najčešće korištenih tipka u Unityu

Iznad hijerarhije nalazi se nekoliko opcija koje su poprilično korisne pri izradi igre, a to su tipke za odabir objekta, za micanje objekta, za rotacije objekta, za uvećanje ili smanjivanje objekta i opcija za smanjivanje i uvećavanje objekta samo po x i y osi odnosno u 2D obliku dok z os ostaje nepromijenjena. Ta je opcija korisna kada, na primjer, koristimo *canvas* objekt koji nam je u našem slučaju služio za prikaz novca, broj života i dućan u kojem se mogu kupovati stupovi za obranu.

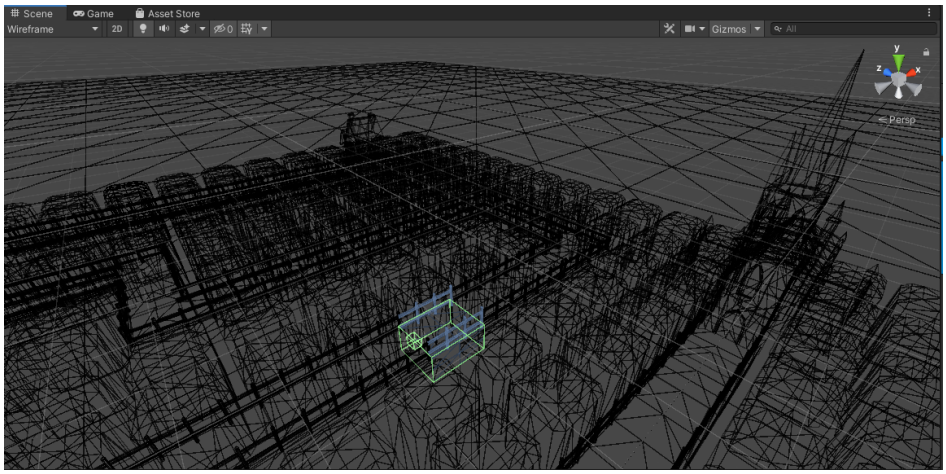
Ispod hijerarhije nalaze se sve datoteke koje su u projektu. Omogućuje jednostavan i brz pristup svim objektima, skriptama i imovini. Tu se nalaze objekti koji se ne moraju nužno nalaziti u hijerarhiji, nego se, na primjer, pozivaju programskim kodom, odnosno skriptama.

Slika 7: Prikaz svih datoteka u Unityu

Glavni dijelovi Unitya su prozori *Scene* i *Game*. Razlika između ovoga dvoga je ta što se kod *Scene* objekti mogu mijenjati i micati, dok kod prozora *Game* vidimo ono što se nalazi trenutno na sceni uz nemogućnost ikakvog uređivanja. Kod *Scene* isto tako imamo i opciju načina pregleda igre; to može biti ili 2D ili 3D. Pregled svih objekata, jedna je od zanimljivijih opcija *Wireframe* koja pomaže, primjerice, u poravnavanju objekta.

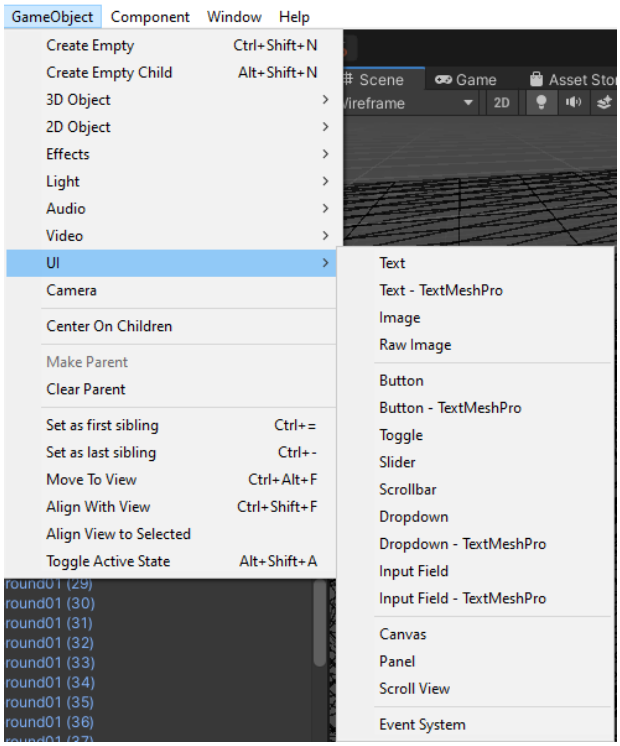


Slika 8: Scene View prozor



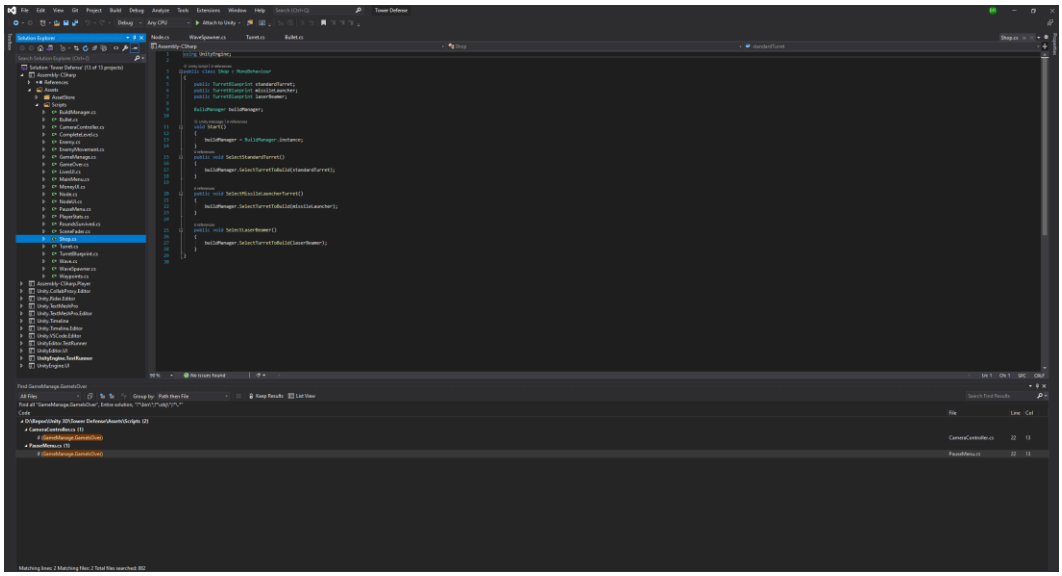
Slika 9: Wireframe View prozor

Jedna od najčešće korištenih opcija u Unityu je ubacivanje na scenu *GameObject*. Unity je omogućio korisnicima ubacivanje svih vrsta objekata: imamo prazne objekte, 3D objekte, kao što su, na primjer, kocka, valjak, kugla, različiti tereni, drveća, 2D objekti, razni efekti, svjetlost, kamera, video, zvuk. Isto tako imamo objekte za izradu korisničkog sučelja poput *Canvas*, „Panel“, slike, tekst i mnoge druge. Ovi elementi poprilično pomažu pojedincima koji su tek krenuli s razvojem igara, budući da je dosta elemenata već napravljeno te je jednostavno koristiti gotove objekte. Isto tako pod opcijom *Window* imamo popularni *Asset Store* s kojim je moguće ubaciti imovine koje smo prethodno kupili ili uzeli besplatno. Iako odabir besplatnih imovina nije prevelik, svejedno se može dosta toga korisnoga pronaći čime ćemo si uveliko olakšati izradu igara. Na slici ispod imamo prikaz nekih opcija koje nam omogućuje Unity *editor*.



Slika 10: Gotovi objekti koje nam nudi Unity

Posljednje stvar koja nije u Unity *editoru*, a imala je jako veliku ulogu pri izradi igre je Visual Studio. Visual Studio je jedan od mnogobrojnih IDE-a u kojem je moguće razvijati skripte koje se koriste u Unityu. U našem slučaju skripte su pisane u objektno orijentiranome programskom jeziku C#.



Slika 11: Visual studio, alat za pisanje C# koda

7. Razvoj igre

Razvoj igre se sastoji od nekoliko bitnih faza: prva faza je određivanje tipa i cilj igre, odnosno sama priča igre, druga faza je priprema dizajna i izgled igre, i treća je programiranje, pisanje koda i izrada cijele logike videoigre.

7.1. Tip i cilj igre

Tip igre je, kao što smo već naveli ranije, *Tower defense*, obrana kule od neprijatelja. Ovakav tip igre je odabran zato što je vrlo jednostavna za igranje, a u isto vrijeme jako zabavna. Ne traži puno resursa od računala za pokretanje i ovakav tip igre je bio vrlo popularan krajem prošlog desetljeća, a mnogima je privlačan i dandanas.

Cilj igre nije bilo teško odrediti budući da svaki ovakav tip ima jedinstven cilj, a to je obraniti utvrdu od neprijatelja. Neprijatelj se kreće po točno određenome putu, na kojemu ga pokušavaju zaustaviti razni stupovi. U današnje vrijeme klasične stupove su zamijenili razni izmišljeni likovi. Postoji više valova i tipova neprijatelja. Svaki novi val donosi nove i snažnije neprijatelje, svaka vrsta neprijatelja ima svoje posebnosti kao što su, na primjer, brzina, energija, brojnost itd.

Kao što svaki tip neprijatelja ima različite posebnosti, tako i svaki stup odlikuje neka posebnost. To može biti brzina kojom pucaju po neprijatelju, sposobnost da uspori neprijatelja ili da uništi odjednom više neprijatelja. Isto tako, svaki stup može bit nadograđen čime najčešće dobije na snazi ili brzini.

Kao što je već spomenuto, neprijatelj se kreće od točke A do točke B, najčešće se brani neka utvrda. Svaka takva utvrda ima određen broj života koji može izgubiti prije negoli se igra završi. Ako se ne potroše svi životi tijekom svih valova, to znači da smo pobijedili, u suprotnom smo izgubili.

Naša igra se sastoji od 3 različita stupa. Jedan je običan, napada po jednog protivnika s ne baš velikom količinom oštećenja; drugi je tzv *Missile Launcher*, njegova sposobnost je ta što ima jako veliki domet i radi *splash damage*, odnosno u isto vrijeme oštećuje više neprijatelja; treći stup ima sposobnost usporiti neprijatelja s jako velikim oštećenjem u sekundi. Svaki od navedenih stupova može se nadograditi čime se pojačavaju njihove sposobnosti koje posjeduju. Postoje 4 različita neprijatelja, razlikuju se u količini energije, brzini i količini novca koji daju nakon što ih se uništi. Novac koji se dobije uništenjem neprijatelja troši se na kupnju i nadogradnju stupova.

7.2. Dizajn igre

Iako dizajn igre ne izgleda složeno i igra se sastoji od samo dvije scene, početna i jedna razina, velika količina vremena potrošena je upravo na dizajn. Najveći je izazov bio odabrati odgovarajući model neprijatelja i stupova. Bilo je velikog ograničenja u pronalasku modela zato što su u obzir dolazili samo besplatni modeli sa *Asset Store*. Kada bi se i našao kakav model, on najčešće nije imao animacije. Osim stupova i neprijatelja, more, zelena podloga i razni efekti su preuzeti s trgovine.

Na početnoj sceni uz stup, pojavljuje se brod i palma, koji najavljuju kontekst u kojem će se odvijati priča. Na glavnoj sceni imamo otok koji se sastoji od malih blokova, gdje se nalaze stupovi i travnati put po kojem se kreće neprijatelj.

U igri se koriste razni vizualni efekti. Kod uništavanja neprijatelja pojavljuje se krv, eksplozije, vatra lasera. Prilikom prodaje stupova i nakon što neprijatelj umre pojavljuje se dim.

7.3. Programiranje i logika igrice

7.3.1. Klasa Waypoints

Klasom *Waypoints* dobe se točke koje neprijatelj mora slijediti kako bi došao do kraja puta. Postavljeno je jedanaest različitih točaka na mjestima gdje neprijatelj mora promijeniti svoj smjer kretnje, točke se ispišu i spremaju u varijablu *points* i ta se varijabla koristi u klasi *EnemyMovement*

```
public class Waypoints : MonoBehaviour
{
    public static Transform[] points;

    void Awake()
    {
        points = new Transform[transform.childCount];
        for (int i = 0; i < points.Length; i++)
        {
            points[i] = transform.GetChild(i);
        }
    }
  }
```

7.3.2.Klasa EnemyMovement

Nakon što smo dobili točke i njihove pozicije gdje se treba kretati neprijatelj, u funkciji *Start()* postavljamo prvu točku iz liste, stoga što je lista točaka napravljena po redoslijedu kako idu točke na sceni, pa će neprijatelj nakon što se stvori odmah krenuti prema navedenoj točki. U funkciji *Update()*, koja se poziva za svaki učitani okvir (eng. *frame*) određujemo vektor od trenutne pozicije neprijatelja do pozicije sljedeće točke, te zatim pomoću *transform.Translate* pomičemo objekt određenom brzinom. Pomoću *Time.deltaTime* dobijemo kretanje objekta svake sekunde, a ne za svaki učitani okvir.

```
    private void Start()
    {
        startRotation = healthBar.rotation;
        enemy = GetComponent<Enemy>();
        target = Waypoints.points[0];
    }
    private void Update()
    {
        Vector3 direction = target.position - transform.position;
        transform.Translate(direction.normalized * enemy.speed *
Time.deltaTime, Space.World);
        if (Vector3.Distance(transform.position, target.position) <= 0.4f)
        {
            GetNextWaypoint();
            transform.LookAt(target);
            healthBar.rotation = startRotation;
        }
        enemy.speed = enemy.startSpeed;
         }
```

Nakon što dođemo do zacrtane točke poziva se funkcija *GetNextWaypoint()*, kako bi dobili novu odredišnu točku. Taj se proces ponavlja sve dok ne dođemo do zadnje točke u listi; ako neprijatelj dođe do zadnje točke to znači da je došao do tvrđave te se oduzimaju životi.

```
    void GetNextWaypoint()
    {
        if (wavepointIndex >= Waypoints.points.Length - 1)
        {
            EndPath();
            return;
        }
        wavepointIndex++;
```

```
        target = Waypoints.points[wavepointIndex];
    }
    void EndPath()
    {
        PlayerStats.Lives--;
        WaveSpawner.EnemiesAlive--;
        Destroy(gameObject);
       }
```

7.3.3.Klasa WaveSpawner

Klasom *WaveSpawner* stvaramo valove neprijatelja. Ono što je zanimljivo u ovoj klasi je korištenje koroutine. Kod pozivanja funkcija, svaka se funkcija izvrši u jednome okviru. Kako smo stvorili više neprijatelja pomoću jedne funkcije, postojala je opasnost da se svi neprijatelji stvore u jednoj točki u isto vrijeme, i jedan na drugome. Da bismo to izbjegli, poslužili smo se korutinom. Uporabom te funkcije postignuto je to da se nakon što se određeni dio koda izvrši unutar funkcije kratko sačeka te se kod nastavlja dalje izvršavati. Uz korutinu koristilo se formatiranje vremena za odbrojavanje novog vala neprijatelja. *Mathf.Clamp* se koristi kako bi se osigurali da vrijednost ne padne ispod nule.

```
    private void Start()
    {
        EnemiesAlive = 0;
    }
    private void Update()
    {
        if(EnemiesAlive > 0)
        {
            return;
        }
        if (waveIndex == waves.Length)
        {
            gameManager.WinLevel();
            this.enabled = false;
        }

        if (countdown <= 0f)
        {
            StartCoroutine(SpawnWave());
            countdown = timeBetweenWaves;
            return;
```

```
        }
        countdown -= Time.deltaTime;
        countdown = Mathf.Clamp(countdown, 0f, Mathf.Infinity);
        waveCountdownText.text = string.Format("{0:00.00}", countdown);
    }
    IEnumerator SpawnWave()
    {
        PlayerStats.Rounds++;

        Wave wave = waves[waveIndex];
        for (int i = 0; i < wave.count; i++)
        {
            SpawnEnemy(wave.enemy);
            yield return new WaitForSeconds(1 / wave.rate);
        }
        waveIndex++;
    }
```

7.3.4.Klasa Turret

Klasom *Turret* upravljamo ponašanje stupa. Funkcijom *LockOnTarget()* omogućuje se rotacija stupa prema neprijatelju, a za rotacije Unity koristi kvaternion. Nakon što dobijemo vrijednosti u obliku kvaterniona trebamo ih pretvoriti u Eulerove kutove. Eulerove kutove koristimo za rotaciju oko x, y, z osi. Sa *Quaternion.Lerp* omogućuje se glatko rotiranje stupa, bez naglih pokreta.

```
  void LockOnTarget()
    {
        Vector3 direction = target.position - transform.position;
        Quaternion lookRotation = Quaternion.LookRotation(direction);
        Vector3 rotation = Quaternion.Lerp(partToRotate.rotation,
lookRotation, Time.deltaTime * turnSpeed).eulerAngles;
        partToRotate.rotation = Quaternion.Euler(0f, rotation.y, 0f);
          }
```

Da bismo rotirali stup prema neprijatelju prvo je trebalo pronaći neprijatelje. To se radilo tako da se na svaki objekt koji je neprijatelj, stavljala oznaka *Enemy*. Zatim se prolazi kroz listu neprijatelja i traži se neprijatelj čija je udaljenost unutar dometa stupa i čija je udaljenost najmanja od stupa, te se taj neprijatelj stavlja za metu i poziva se funkcija *Shooot()* kojom se instancira novi objekt, *Bullet* tipa, te zatim izvršavamo kod u *Bullet* klasi.

```
            GameObject[] enemies =
GameObject.FindGameObjectsWithTag(enemyTag);
```

```
            float shortestDistance = Mathf.Infinity;
            GameObject nearestEnemy = null;

            foreach (GameObject enemy in enemies)
            {
                float distanceToEnemy =
Vector3.Distance(transform.position, enemy.transform.position);
                if (distanceToEnemy < shortestDistance)
                {
                    shortestDistance = distanceToEnemy;
                    nearestEnemy = enemy;
                }
            }
            if (nearestEnemy != null && shortestDistance <= range)
            {
                target = nearestEnemy.transform;
                targetEnemy = nearestEnemy.GetComponent<Enemy>();
                  }
```

7.3.5. Klasa Bullet

Klasom *Bullet* pomičemo metak nakon što smo ga instancirali u prethodnoj klasi. To radimo tako da oduzmemo udaljenost od točke gdje se nalazi metak i točke gdje se nalazi neprijatelj. Nakon toga provjeravamo je li udaljenost metka od neprijatelja manja od udaljenosti za koju ćemo pomaknuti metak u trenutnom okviru (eng. *frame*); ako je manja, to znači da smo već došli do neprijatelja, pa se poziva funkcija *HitTarget()* kojom se oštećuje neprijatelj i pozivaju se efekti eksplozije, te se zatim uništava metak pozivom *Destroy(gameObject)*. Također koristila se funkcija *transform.LookAt()* koja služi za rotiranje metka prema meti i *transform.Translate()* kojom se pomiče metak.

```
        var centerTarget = new Vector3(target.position.x, target.position.y
+ 5f, target.position.z);
        Vector3 direction = centerTarget - transform.position;
        float distanceThisFrame = speed * Time.deltaTime;
        deadTarget = centerTarget;

        if (direction.magnitude <= distanceThisFrame)
        {
            HitTarget();
            return;
        }
```

```
        transform.Translate(direction.normalized * distanceThisFrame,
Space.World);
             transform.LookAt(target);
```

U ovoj se klasi također odrađuje logika kada raketa, laser ili običan metak pogodi neprijatelja. Kod rakete je specifično da šteti u nekom radijusu, pa se stoga mora provjeravati postoje li u tom radijusu objekti s oznakom *Enemy*. Kada se pronađu takvi objekti svima se oduzima energija funkcijom *Damage()*, a kod lasera i običnog metka se preskače funkcija *Explode()* budući da oni rade štetu samo jednom neprijatelju, i odmah se prelazi na *Damage()* funkciju. Obje se funkcije pozivaju u prethodno navedenoj funkciji *HitTarget()*.

```
    void Explode()
    {
        Collider[] colliders = Physics.OverlapSphere(transform.position,
explosionRadius);
        foreach (Collider collider in colliders)
        {
            if (collider.tag == "Enemy")
            {
                Damage(collider.transform);
            }
       }
    }

    void Damage(Transform enemy)
    {
        Enemy e = enemy.GetComponent<Enemy>();

        if(e!= null)
        {
            e.TakeDame(damage);
```

7.3.6. Klasa „Node“

Klasom *Node* grade se i nadograđuju stupovi, radi se provjera postoji li već na nekom bloku stup, ima li igrač novaca za izgradnju stupova i promjena boje kada se mišem prođe po bloku.

OnMouseDown() je ugrađena funkcija od Unitya i ona se poziva kada mišem kliknemo na blok. U funkciji se provjerava postoji li već na odabranom bloku stup, ako ne postoji onda pustimo gradnju stupa, a ako postoji poziva se *SelectNode()* iz klase *BuildManager* koja onda provjerava je li taj blok već odabran, ako je, onda se odabir poništava, a ako nije onda se blok

odabire i iznad njega iskače mala ploča na kojoj igrač može odabrati hoće li prodati ili nadograditi stup.

```
    void OnMouseDown()
    {
        if (EventSystem.current.IsPointerOverGameObject())
            return;

        if (turret != null)
        {
            buildManager.SelectNode(this);
            return;
        }

        if (!buildManager.CanBuild)
            return;
        BuildTurret(buildManager.GetTurretToBuild());
          }
```

Funkcije *BuildTurret()* i *UpgradeTurret()* su dosta slične. Jedina razlika je u tome što kod nadogradnje, odnosno *UpgradeTurret()* funkcije, prije nego što sagradimo novu verziju moramo uništiti prvu verziju stupa i postaviti varijablu *isUpgraded* na istinitu vrijednost kako bi u budućnosti znali da je stup nadograđen. Kod ovih funkcija isto tako se mora oduzeti novac koji igrač posjeduje, nakon što se izvrši kupnja ili nadogradnja.

```
    public void UpgradeTurret()
    {
        if (PlayerStats.Money < turretBlueprint.upgradeCost)
        {
            return;
        }
        PlayerStats.Money -= turretBlueprint.upgradeCost;
        Destroy(turret);
        GameObject _turret =
(GameObject)Instantiate(turretBlueprint.upgradedPrefab, transform.position,
Quaternion.identity);
        turret = _turret;

        var offset = new Vector3(transform.position.x, transform.position.y
+ 3, transform.position.z);

        GameObject effect =
(GameObject)Instantiate(buildManager.buildEffect, offset,
Quaternion.identity);
```

```
        Destroy(effect, 5f);

        isUpgraded = true;
         }
```

7.3.7. Klasa BuildManager

Kao što smo već naveli, klasa *BuildManager* nam je potrebna kako bi provjerili ima li igrač novaca za kupnju i nadogradnju, za provjeru je li blok već odabran ili ne i jedna od bitnijih stavka je ta da prosljeđuje u *Node* klasu stup koji je igrač odabrao za izgradnju u dućanu, odnosno *Shop* klasi.

```
public void SelectNode(Node node)
{
    if(selectedNode == node)
    {
        DeselectNode();
        return;
    }
    selectedNode = node;
    turretToBuild = null;

    nodeUI.SetTarget(node);
}
public void DeselectNode()
{
    selectedNode = null;
    nodeUI.Hide();
}
public void SelectTurretToBuild(TurretBlueprint turret)
{
    turretToBuild = turret;
    DeselectNode();
}
public TurretBlueprint GetTurretToBuild()
{
    return turretToBuild;
      }
```

U klasi *Shop* nalaze se tri funkcije, za svaki stup jedna funkcija, koje se okidaju kada igrač odabere jedan od tri stupa. Te funkcije prosljeđuju tip odabranog stupa u *BuildManager* klasu.

7.3.8. Klasa Enemy

U klasi *Enemy* nalaze se osnovni podaci o neprijatelju. Unutar klase je pohranjena brzina, energija i vrijednost koja se dobije nakon što se uništi neprijatelj.

Funkcija *TakeDamage()* se poziva iz klase *Bullet* nakon što metak dođe do mete te se zatim oduzima energija neprijatelja za vrijednost koju ima metak.

Funkcija *Slow()* se koristi samo kod stupova koji koriste laser i ona usporava neprijatelja i na kraju imamo funkciju *Die()* koja se poziva ako je neprijatelj došao do nule s energijom, pribraja se vrijednost koja se dobije uništenjem neprijatelja i uništava se objekt *Enemy*.

```
    public void TakeDamage(float amount)
    {
        health -= amount;
        healthBar.fillAmount = health / startHealth;
        if(health <= 0 && !isDead)
        {
            Die();
        }
    }
    public void Slow(float pct)
    {
        speed = startSpeed * (1f - pct);
    }
    void Die()
    {
        isDead = true;
        PlayerStats.Money += worth;
        GameObject effect = (GameObject)Instantiate(deathEffect,
transform.position, Quaternion.identity);
        Destroy(effect, 5f);

        WaveSpawner.EnemiesAlive--;
        Destroy(gameObject);
          }
```

7.3.9. Klase LivesUI, WavesUI, MoneyUI

Ovo su primjeri vrlo jednostavnih klasa i one se zapravo koriste za korisničko sučelje, odnosno kako bi se prikazala vrijednost poput preostalih života, broj valova i redni broj vala na kojem smo trenutno i prikaz novca koji imamo na raspolaganju za kupnju ili nadogradnju stupova.

```
public class LivesUI : MonoBehaviour
{
    public Text livesText;
    void Update()
    {
        livesText.text = PlayerStats.Lives.ToString() + " Lives";
    }
      }
```

7.3.10. Klase PauseMenu, GameOver, CompleteLevel

Klasa *GameOver* se poziva kada igrač izgubi sve živote i pojavljuje se opcija prelaska na glavni izbornik ili pokušaj ponovnog igranja igrice. Isto je i s klasom *CompleteLevel*, osim što se ona poziva nakon što uspijemo preživjeti svih osamnaest valova neprijatelja. Mogućnost mijenjanja scena ostvaruje se pomoću *SceneManager.LoadScene()*. Klasa *PauseMenu* se poziva pritiskom tipke *Esc* ili *P*, osim što nudi iste mogućnosti kao i prethodne dvije klase, ona se može pozvati u bilo kojem trenutku, a kada se pozove koristi se od Unitya mogućnost zaustavljanja scene, a to se ostvaruje pomoću *Time.timeScale = 0f*. Ako želimo pokrenuti scenu vrijednost stavljamo na 1, a ako želimo ubrzati scenu onda stavljamo vrijednost 2.

```
    private void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape) || Input.GetKeyDown(KeyCode.P))
        {
            Toggle();
        }
    }
    public void Toggle()
    {
        if (GameManage.GameIsOver)
            return;
        ui.SetActive(!ui.activeSelf);
        if (ui.activeSelf)
        {
            Time.timeScale = 0f;
```

```
        } else
        {
            Time.timeScale = 1f;
        }
    }
    public void Retry()
    {
        Toggle();
        sceneFader.FadeTo(SceneManager.GetActiveScene().name);
    }
    public void Menu()
    {
        Toggle();
        sceneFader.FadeTo(menuSceneName);
          }
```

Kao što se može vidjeti u kodu iznad koristi se klasa *SceneFader* gdje se poziva funkcija *FadeTo* kojom se radi animacija prelaska s jedne scene na drugu i u njoj se poziva *SceneManager.LoadScene()*.

7.3.11. Klasa CameraController

Klasa *CameraController* se koristi za micanje kamere mišem i tipkovnicom. Omogućeno je da pritiskom na tipkovnicu na jedno od slova „w“, „s“, „a“, „d“ pomiče se kamera naprijed, nazad, lijevo i desno. Isto tako kada mišem dođemo do ruba igrice kamera se miče i možemo sa *scroll* opcijom približiti ili udaljiti kameru od scene.

```
    void Update()
    {
        if (GameManage.GameIsOver)
        {
            this.enabled = false;
            return;
        }
        if (Input.GetKey("w") || Input.mousePosition.y >= Screen.height -
panBorderThickness)
        {
            transform.Translate(Vector3.forward * panSpeed *
Time.deltaTime, Space.World);
        }
```

```
        if (Input.GetKey("s") || Input.mousePosition.y <=
panBorderThickness)
        {
            transform.Translate(Vector3.back * panSpeed * Time.deltaTime,
Space.World);
        }
        if (Input.GetKey("d") || Input.mousePosition.x >= Screen.width -
panBorderThickness)
        {
            transform.Translate(Vector3.right * panSpeed * Time.deltaTime,
Space.World);
        }
        if (Input.GetKey("a") || Input.mousePosition.x <=
panBorderThickness)
        {
            transform.Translate(Vector3.left * panSpeed * Time.deltaTime,
Space.World);
        }
        float scroll = Input.GetAxis("Mouse ScrollWheel");
        Vector3 pos = transform.position;
        pos.y -= scroll * 1000 * scrollSpeed * Time.deltaTime;
        pos.y = Mathf.Clamp(pos.y, minY, maxY);
        transform.position = pos;
          }
```

7.3.12. Klasa MainMenu

Klasa *MainMenu* se koristi za scenu *MainMenu* i ona se pojavljuje kada igrač tek uđe u igricu i tamo mu se ponudi izbor ili igranje igrice ili izlazak iz iste. Klikom na *Play* poziva se klasa *SceneFader* kako bi nas prebacila na glavnu scenu, a pristikom na *Quit* poziva se *Application.Quit()* s kojim izlazimo iz igrice.

```
public class MainMenu : MonoBehaviour
{
    public string levelToLoad = "MainLevel";

    public SceneFader sceneFader;
    public void Play()
    {
        sceneFader.FadeTo(levelToLoad);
    }

    public void Quit()
    {
```

```
    Application.Quit();
}
  }
```

8. Zaključak

U današnje vrijeme razvoj igara napreduje nevjerojatno brzinom. Kako su zahtjevi klijenata sve zahtjevniji, a vremena sve manje zbog velike konkurencije, bilo je potrebno osmisliti alate koji bi ubrzali proces igre i podigli kvalitetu iste. Jedan od poznatijih takvih alata je upravo Unity.

Unity Editor vrlo moćan, a u isto vrijeme vrlo jednostavan alat za korištenje, prilagođen je svim razinama znanja od programera početnika pa sve do iskusnog programera. Sadrži mnoge elemente i alate koji su već gotovi i spremni za korištenje poput raznih 3D i 2D elemenata, neophodne stvari koje su potrebne za korisničko sučelje poput tipke, teksta, panela i mnogih drugih. Unity također ima gotove funkcije koje pružaju mnogobrojne mogućnosti koje se koriste u samom programiranju. Iako je ovo bio prototip igrice, u njoj se također koristio velik broj takvih funkcija i na taj način uštedjela golema količina vremena. Možemo, stoga, tek naslutiti koliko te funkcije dolaze do izražaja u složenim igricama. Upravo to čini Unity boljim od ostalih alata.

Također Unity ima veliku zajednicu aktivnih korisnika, pa ako se zapne na nekom dijelu koda vrlo je lako pronaći rješenje jer se već prije netko sigurno susreo s takvom situacijom i objavio na Internetu. Kako ima veliku zajednicu, razumljivo je da postoji velik broj gotovih imovina, od čega, naravno, velik dio se naplaćuje, ali se dosta toga može pronaći i besplatno. Sva imovina korištena u igrici je besplatna i pronađena na *Asset Store*. Sama cijena imovine nije prevelika, pa ako se ozbiljno ulazi u branšu za razvoja igara to ne bi trebalo predstavljati problem.

Obrana kule se prvi put pojavljuje sedamdesetih godina prošlog stoljeća, ali i dandanas je privlačna ljudima zbog jednostavnosti, a opet dosta velikog izazova. Ovaj je tip igre doživio vrhunac pojavom *flash* igara. 2010. godine pojavljuje se prva 3D igrica TD tipa, igra se zvala *Dungeon Defenders*.

Sam razvoj igre u okviru ovoga rada bio je dosta zanimljiv i izazovan, a u isto vrijeme veoma poučan. Bilo je brojnih prepreka vezanih za programski kod i dizajn igre, ali uz pomoć brojnih dokumentacija i tečaja sve su prepreke uspješno svladane. Izrada igre omogućila je i svladavanje osnova programskog jezika C#.

Popis literature

[1] October 1958: Physicist Invents First Video Game [Na internetu]. Dostupno: https://www.aps.org/publications/apsnews/200810/physicshistory.cfm#:~:text=In%20October%201958%2C%20Physicist%20William,Brookhaven%20National%20Laboratory%20open%20house. [16.11.2020]

[2] The history of video games [Na internetu]. Dostupno: https://blog.nationalmuseum.ch/en/2020/01/the-history-of-video-games/ [16.11.2020]

[3] Video Game History [Na internetu]. Dostupno: https://www.history.com/topics/inventions/history-of-video-games [16.11.2020]

[4] Unity - Multiplatform [Na internetu]. Dostupno: https://unity.com/features/multiplatform [16.11.2020]

[5] A History of the Unity Game Engine [Na internetu]. Dostupno: https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf [16.11.2020]

[6] How Unity3D Became a Game-Development Beast [Na internetu]. Dostupno: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ [16.11.2020]

[6] What is Unity? Everything you need to know [Na internetu]. Dostupno: https://www.androidauthority.com/what-is-unity-1131558/ [16.11.2020]

[7] How Neill Blomkamp and Unity are shaping the future of filmmaking with Adam: The Mirror [Na internetu]. Dostupno: https://www.theverge.com/2017/10/4/16409734/unity-neill-blomkamp-oats-studios-mirror-cinemachine-short-film [17.11.2020]

[8] Powering cameras for films and games [Na internetu]. Dostupno: https://unity.com/unity/features/editor/art-and-design/cinemachine [17.11.2020]

[9] Unity Technologies to Go Public in 2020 [Na internetu]. Dostupno: https://variety.com/2019/gaming/news/unity-technologies-ipo-report-1203135985/ [17.11.2020]

[10] Myths and Facts of the Unity Game Engine [Na internetu]. Dostupno: https://blog.theknightsofunity.com/myths-and-facts-of-unity-game-engine/ [17.11.2020]

[11] Unity VS Unreal – Which Engine Should You Choose? [Na internetu]. Dostupno: https://sundaysundae.co/unity-vs-unreal/ [18.11.2020]

[12] Unreal Engine vs Unity: Which is Better? [Na internetu]. Dostupno: https://www.gamedesigning.org/engines/unity-vs-unreal/ [18.11.2020]

[13] Unity vs. Unreal – Choosing a Game Engine [Na internetu]. Dostupno: https://gamedevacademy.org/unity-vs-unreal/ [18.11.2020]

[14] [Na internetu]. Dostupno: https://program-ace.com/blog/5-years-of-unity-vs-unreal/ [18.11.2020]

[15] GooBall [Na internetu]. Dostupno: https://connect.unity.com/p/gooball-6 [30.11.2020]

[16] The Original ‘Space Invaders’ Is a Meditation on 1970s America’s Deepest Fears [Na internetu]. Dostupno: https://www.smithsonianmag.com/science-nature/original-space-invaders-icon-1970s-America-180969393/ [30.11.2020]

[17] Disney Television Animation Launching 'Big Hero 6'-Themed Shorts [Na internetu]. Dostupno: https://www.hollywoodreporter.com/behind-screen/disney-television-animation-launching-big-hero-6-themed-shorts-1133450 [30.11.2020]

[18] How gaming company Unity is driving automakers toward virtual reality [Na internetu]. Dostupno: https://www.digitaltrends.com/cars/unity-automotive-virtual-reality-and-hmi/ [30.11.2020]

[19] How Google’s DeepMind will train its AI inside Unity’s video game worlds [Na internetu]. Dostupno: https://www.fastcompany.com/90240010/deepminds-ai-will-learn-inside-unitys-video-game-worlds [30.11.2020]

[20] FIRSTS: SPACEWAR! WAS THE WORLD'S FIRST VIDEO GAME [Na internetu]. Dostupno: https://www.syfy.com/syfywire/firsts-spacewar-was-the-worlds-first-video-game [30.11.2020]

Popis slika

Slika 1: Prva Unity igra GooBall [15] 4
Slika 2: Prikaz alata koji se najviše koriste [14] 9
Slika 3: Space Invaders [16] 10
Slika 4: Prikaz hijerarhije u Unityu 12
Slika 5: Prikaz inspector dijela u Unityu 13
Slika 6: Prikaz najčešće korištenih tipka u Unityu 13
Slika 7: Prikaz svih datoteka u Unityu 14
Slika 8: Scene View prozor 14
Slika 9: Wireframe View prozor 15
Slika 10: Gotovi objekti koje nam nudi Unity 16
Slika 11: Visual studio, alat za pisanje C# koda 16