Izrada 3D strateške igre u stvarnom vremenu u programskom alatu Unity

Hrvoje, Hodak

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Hodak

IZRADA 3D STRATEŠKE IGRE U STVARNOM VREMENU U PROGRAMSKOM ALATU UNITY

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Hodak

Matični broj: 43952/15–R

Studij: Informacijsko i programsko inženjerstvo

IZRADA 3D STRATEŠKE IGRE U STVARNOM VREMENU U PROGRAMSKOM ALATU UNITY

DIPLOMSKI RAD

Mentor/Mentorica:

Doc. dr. sc. Mladen Konecki

Varaždin, lipanj 2022.

Hrvoje Hodak

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski 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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je izrada trodimenzionalne strateške računalne igre u stvarnom vremenu u programskom alatu Unity, odnosno opis procesa izrade jedne takve igre. Objasnit ćemo što su to strateške igre, navesti i objasniti njihovu podjelu te neke primjere takvih video igara, a zatim ćemo opisati alat Unity u kojem ćemo izraditi igru te prikazati osnovne mehanike strateških igara u stvarnom vremenu kao što su kreiranje jedinica i zgrada, davanje naredbi, sustav prikupljanja i potrošnje resursa te borba između jedinica. Također, tekstualnim opisom objasnit ćemo načine na koji su oni implementirani u praktičnom primjeru uz ključne dijelove programskog koda koji su važni za svaku mehaniku.

Ključne riječi: računalne igre, strateška igra, strateška igra u stvarnom vremenu, Unity, C#,

Sadržaj

1. Uvod

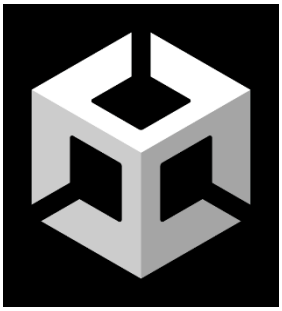
Video igre su vrsta igara koje se mogu igrati na računalu, igraćoj konzoli ili mobilnom telefonu. Od samih početaka prvih video igara koje su napravljene sedamdesetih godina prošlog stoljeća pa sve do danas, industrija izrade video igara u stalnom je porastu. Vrijednost tržišta video igara veća je od 220 milijardi američkih dolara a taj broj svakodnevno raste [4] [5]. Upravo zbog tako velikog i brzorastućeg tržišta, danas možemo naći razne tipove video igara. Igre za jednog ili više igrača, igre u prvom ili trećem licu, igre za osobno računalo, igraću konzolu ili mobilni uređaj i slično. Također, postoje i razne kategorije video igara: Akcijske, akcijsko-avanturističke, avanturističke, igre uloga (eng. role-playing games), simulacije, strateške igre, sportske igre, igre zagonetke te jednostavne (eng. idle) video igre [6]. U ovom radu, osvrnut ćemo se na jedan od ovih žanrova – strateške igre. Točnije opisat ćemo i objasniti glavne mehanike jednog od pod žanrova strateških igara – strateške igre u stvarnom vremenu (eng. Real-Time Strategy games). Također, glavne mehanike ovog pod žanra demonstrirat ćemo i na praktičnom primjeru kojeg ćemo izraditi u programskom alatu Unity koji je jedan od najpopularnijih alata za izradu video igara na svijetu.

2. Unity

Unity je višeplatformski „motor igre“ (eng. Game engine) kojeg je 2005. godine objavila danska tvrtka Unity Technologies. Tvrtka je osnovana 2004. godine u Kopenhagenu pod nazivom Over the Edge Entertainment kojeg je imala do 2007. godine. Njihovo sjedište danas se nalazi u Sjedinjenim Američkim Državama, u San Francisku [7]. Osim za izradu video igara, engine se danas koristi i u građevinskoj, filmskoj te autoindustriji.

U Unity engine-u je moguće raditi 2D i 3D video igre za osobna računala, igraće konzole i mobilne uređaje. Igre je moguće izraditi za mnoge platforme kao što su PC, Mac & Linux Standalone, Android, Universal Windows Platform, tvOS, PS4, iOS, PS5, Xbox One, WebGL i druge. Sam Unity je napisan u programskom jeziku C++, dok korisnici alata za izradu video igara koristi C#. Unity nema svoj uređivač koda pa je potrebno instalirati vanjski uređivač. Preporuka za to je Visual Studio ali se mogu koristiti i drugi.

Neki od najpoznatijih naslova izrađenih u Unityju su Temple Run, Dead Effect, Rust, Plague Inc: Evolved, Cities: Skylines, Pokémon Go, Among Us, RimWorld, Valheim i mnogi drugi [8] [9].



Slika 1: Unity logo (preuzeto 12. srpnja 2022. s https://unity3d.com/fr/legal/branding_trademarks)

U Unityju postoji mnogo različitih opcija koje pomažu kod izrade bilo kakve vrste igara. Kako bi korisniku rad bio jednostavniji, dobro je podesiti radnu okolinu na način da je ona pregledna i lako dostupna. Na slici 2 vidimo neke od najčešće korištenih prozora za vrijeme rada u alatu Unity. Prozori su označeni crvenim brojevima i svaki od njih će biti posebno objašnjen [10] [11].

Pod brojem 1 je prozor hijerarhije (eng. Hierarchy). U ovom prozoru nalazi se popis svih objekata koji su prisutni u sceni. Objekti se mogu proširiti ako imaju objekte-djecu ispod sebe. Odabirom svakog objekta dobijemo njegov detaljan pregled u prozoru inspektora.

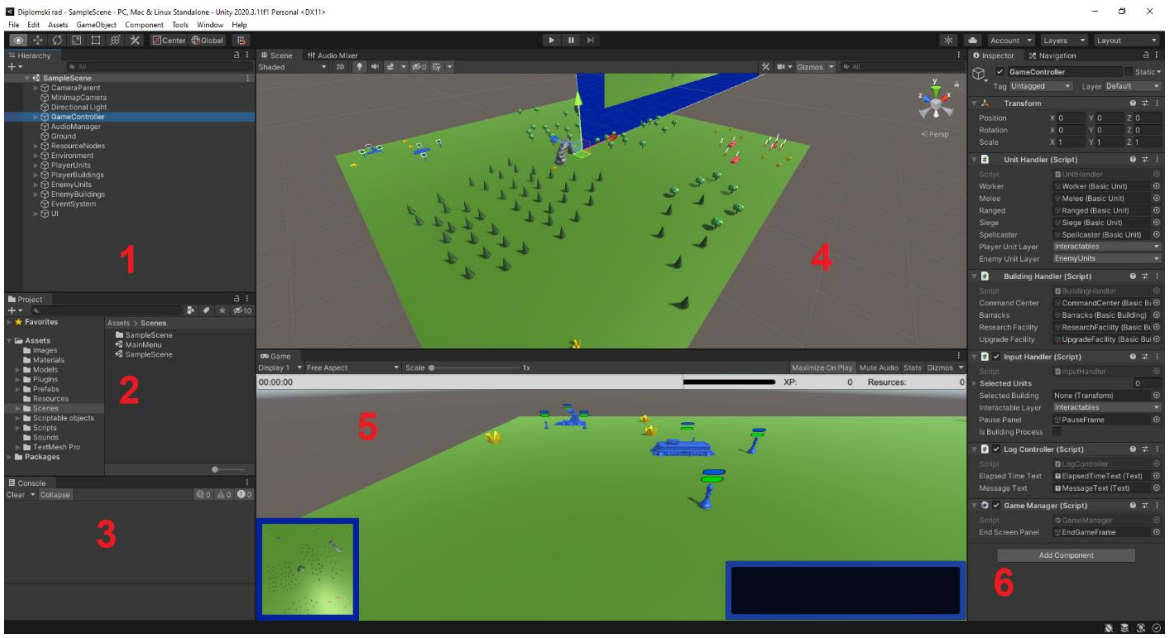
Zatim dolazimo na prozor projekta (eng. Project). Ovdje se nalaze sve datoteke koje koristimo u projektu sortirane po direktorijima. Radi lakšeg snalaženja, sve skripte (.cs datoteke) držimo u direktoriju „Scripts“, slike u direktoriju „Images“, predloške u direktoriju „Prefabs“ i tako dalje. Svaki direktorij je po potrebi dodatno podijeljen na pod direktorije radi lakšeg upravljanja projektom.

Nakon toga imamo prozor za konzolu (eng. Console). U ovom prozoru ispisuju nam se upozorenja koja se mogu dogoditi, greške u programskom kodu s objašnjenjem greške i linijom na kojoj se greška dogodila te ispis raznih poruka koje sam korisnik može dati na ispis, što može biti korisno prilikom testiranja određenih funkcionalnosti.

Sljedeći prozor je scena (eng. Scene). U ovom prozoru korisnik postavlja svoje objekte, pozicionira ih, rotira, skalira te općenito ima pregled na igraću scenu iz svih kuteva i udaljenosti. Pogled na scenu se također može postaviti kao 2D ako je to potrebno. Osim toga, mogu se i upaliti ili ugasiti razne „naprave“ (eng. gizmos) primjerice svjetla, sudarači ili druge stvari koje mogu pomoći korisniku.

Prozor za igru (eng. Game) igraču prikazuje kako igra zapravo izgleda. Po zadanoj vrijednosti, u prozoru za igru prikazuje se zaslon kojeg snima glavna kamera (eng. Main Camera). Pritiskom na gumb za igru (nalazi se na vrhu u sredini prozora) korisnik može isprobati igru u uređivaču, pauzirati trenutni okvir (eng. Frame) ili pomicati igru okvir po okvir radi testiranja.

U inspektor prozoru (eng. Inspector) korisnik ima pregled trenutno odabranog objekta. Odvje korisnik ima pregled svih komponenti dodanih na objekt kao i svih skripti te vrijednosti njihovih varijabli. Varijable se mogu urediti u editoru kako ne bi bilo potrebe za stalnim mijenjanjem programskog koda. Komponente se mogu uklanjati i dodavati po želji (gumb Add Component na dnu prozora).



Slika 2: Prikaz radnog okruženja u alatu Unity (slika autora)

3. Strateške video igre

Strateške video igre su žanr video igara u kojima je naglasak na kratkoročnom i dugoročnom planiranju te upravljanju jedinicama, zgradama, resursima te mnogim drugim komponentama igre kako bi se ostvarila pobjeda [1]. Iako su strateške igre danas najpopularnije na računalima ili konzolama, korijene vuku još od oko 2500-tih godina prije Krista kada su „popularne“ bile igre Senet (Egipat) i Go (Kina), a nešto kasnije i Šah (Indija, Europa) [38]. Prvom video igrom strategije smatra se igra Invasion koja je 1972. godine izašla za konzolu Magnavox Odyssey. 1983. godine izlazi znanstveno-fantastična igra Reach for the Stars koja je preteča 4X igara koje naglasak stavljaju na istraživanje, širenje, iskorištavanje i istrjebljenje (eng. explore, expand, exploit and exterminate). Prvom strateškom video igrom u stvarnom vremenu smatra se Herzog Zwei kojeg je 1989. izdao Technosoft. U toj igri igrač je upravljao kopnenim i zračnim jedinicama s ograničenim gorivom i municijom. 1991. godine Sid Meier i Bruce Shelley osmišljavaju Civilization, stratešku igru na poteze u kojoj igrači više nisu upravljali samo malim vojskama na bojnom polju, nego čitavom nacijom, njezinom ekonomijom, kulturološkim aspektima i mnogim drugim stvarima. 1992. godine Westwood Studios izdaje igru Dune II koja je postavila današnje standarde RTS igara i koja je postavila temelje današnjih velikih naslova kao što su Age of Empires, Command & Conquer, Starcraft i Warcraft [1] [39].

3.1. Strateške igre na poteze

Strategije na poteze (eng. Turn-based strategy – TBS) je vrsta igara u kojima svaki igrač izvodi akcije tokom svog poteza. [2]. Ovisno o igri, igrač ima pravo neometano povući jedan ili više poteza nakon čega mora čekati jednog ili više protivničkih igrača koji će unaprijed definiranim redoslijedom također povući svoje poteze. Nakon što svaki igrač izvrši svoje akcije, prvi igrač ponovno dolazi na red. U nekim igrama, igrači mogu istovremeno odigrati svoj potez, ali tada ne mogu izvršiti više akcija ili prijeći na sljedeći potez dok svi igrači nisu gotovi s trenutnim potezom. Najpoznatija strateška igra na poteze je šah u kojoj prvi igrač ima pravo pomaknuti jednu figuricu na jedno od dozvoljenih polja nakon čega mora čekati drugog igrača.

3.1.1. Civilization

Jedna od najpoznatijih i najpopularnijih strateških video igara na poteze je serijal Civilization koji je izašao 1991. godine. Igrač može igrati protiv jednog ili više protivnika, a cilj mu je ostvariti zadane ekonomske, vojne ili druge ciljeve kako bi pobijedio. Igrač može proširivati svoje carstvo izgradnjom novih gradova, trgovinom s drugim igračima ili vojnim intervencijama. Akcije se odvijaju na ploči prekrivenoj šesterokutima na koje igrač postavlja gradove, gradske distrikte i jedinice kojima može upravljati.

Slika 3: Civilization VI (slika autora)

3.2. Strateške igre u stvarnom vremenu

Strateške igre u stvarnom vremenu (eng. Real-time strategy – RTS) ne koriste mehaniku igranja na poteze već se sve akcije između igrača događaju istovremeno [3]. Dva ili više igrača nalaze se na mapi na kojoj igraju jedni protiv drugih (ili zajednički protiv drugih igrača) te im je cilj upravljanjem jedinicama, zgradama, resursima i tehnologijama doći do pobjede [12] [13]. Neke od RTS igara koje su bile inspiracija prilikom izrade praktičnog rada su Age of Empires, Starcraft, Stronghold, Total War, Warcraft i World in Conflict.

3.2.1. Starcraft

Starcraft je serijal strateških video igara u stvarnom vremenu kojeg je 1998. godine započela američka kompanija Blizzard Entertainment. Igra je smještena u futurističkom svijetu u kojemu igrač upravlja s jednom od 3 moguće rase: Terran (ljudi), Zerg (izvanzemaljska insektoidna rasa) i Protoss (napredna svemirska civilizacija). Svaka od tri navedene rase posjeduje određene zgrade i jedinice (zemljane i zračne) te koje svaka na svoj specifični način ima određenu prednost u odnosu na preostale rase. Igrač mora voditi brigu o broju i stanju zgrada i jedinica, količini i prihodu resursa (minerali i plin) te stanju zaliha koji određuje maksimalan broj jedinica koje igrač može koristiti u jednom trenutku. Igra ima sustav kampanje, odnosno misija koje igrač mora izvršavati kako bi uspješno završio igru, a također postoji i opcija igranja u više igrača u kojima je cilj uništiti sve protivničke zgrade kako bi pobijedili.

Slika 4: Starcraft: Brood War (slika autora)

Zbog velike popularnosti, nešto kasnije izdano je i proširenje kampanje zvano Brood War koje je igračima donijelo nove misije kao i nove jedinice. 2010. godine tvrtka je izdala nastavak Starcraft II: Wings of Liberty koji je za razliku od originalne igre (koja je bila 2D) izrađen na 3D platformi. Nekoliko godina kasnije, napravljeni su i nastavci Starcraft II: Heart of the Swarm te StarCraft II: Legacy of the Void. Obje igre stekle su veliku popularnost zbog čega se i danas održavaju razni svjetski turniri na kojima su glavne nagrade pehari i veliki novčani iznosi.

Slika 5: Starcraft II: Wings of Liberty (slika autora)

3.2.2. Age of Empires

Age of Empires je serijal strateških video igara u stvarnom vremenu koji ima povijesnu tematiku. Prva igra serijala izašla je 1997. godine. Igraču je na raspolaganju bilo 12 različitih civilizacija smještenih u vremenu između kamenog i željeznog doba. Drugi dio izašao je 1999. godine te je igračima nudio 13 civilizacija smještenih u srednjem vijeku. Sljedeći nastavak izašao je 2005. je je sadržavao 8 civilizacija u vremenu od kasnog srednjeg vijeka do ranog modernog doba. Posljednji nastavak izašao je 2021. u kojemu je igračima na raspolaganju bilo 8 srednjovjekovnih civilizacija. Svaki nastavak je u međuvremenu dobivao i proširenja koja su igračima omogućila korištenje novih civilizacija, kako u misijama, tako i u igri protiv drugih igrača.

Slika 6: Age of Empires III: Definitive Edition (slika autora)

3.2.3.Total War

Total War je serijal strateških igara kojeg je 2000. godine započela britanska tvrtka Creative Assembly. Igra je kombinirala elemente TBS-a gdje igrač pomiče svoju vojsku, odrađuje diplomatske sporazume te izgrađuje carstvo i RTS-a u kojem igrač kontrolira svoje jedinice u bitkama. Borbe bi se često odigravale na velikim mapama i uz veliki broj jedinica. Igra je temeljena na povijesnim scenarijima (izuzev Warhammer nastavaka) te je svaki od nastavaka smješten u određeno povijesno razdoblje u kojemu su se odigravale različite bitke s različitim carstvima. Do sada je izdano 15 nastavaka igre: Shogun: Total War (2000.), Medieval: Total War (2002.), Rome: Total War (2004.), Medieval II: Total War (2006.), Empire: Total War (2007.), Napoleon: Total War, Total War: Shogun 2 (2010.), Total War: Rome II (2013.), Total War: Attila (2015.), Total War: Warhammer (2016.), Total War: Warhammer II (2017.), Total War Saga: Thrones of Britannia (2018.), Total War: Three Kingdoms (2019.), Total War Saga: Troy (2020.) i Total War: Warhammer III (2022.).

Slika 7: Rome: Total War (slika autora)

3.2.4.World in Conflict

World in Conflict je vojna strateška igra iz 2007. godine koju je razvila švedska tvrtka Massive Entertainment. Igra je smještana i vrijeme hladnog rata u fiktivnom scenariju u kojemu je Sovjetski Savez izvršio invaziju na Sjedinjene Američke Države. Cilj igrača je u određenim gradovima zauzeti strateški važne položaje (dijelove grada, benzinske crpke, mostove i slično) s vrlo limitiranim snagama. Dijelovi igre također se odvijaju i na teritoriju Rusije i Francuske.

Slika 8: World in Conflict (preuzeto 19. srpnja 2022. s https://www.gamespot.com/reviews/world-in-conflict-review/1900-6179012/)

3.2.5.Stronghold

Stronghold je RTS nastao 2001. godine od strane britanskog Firefly Studiosa. Igra je smještena u srednjevjekovno razdoblje u kojemu je cilj igrača ostavriti jaku ekonomiju, snažnu vojsku te uništiti protivničke dvorce u snage. Igra također ima i nekoliko nastavaka: Stronghold: Crusader (2002.), Stronghold 2 (2005.), Stronghold Legends (2006.), Stronghold Crusader Extreme (2008.), Stronghold 3 (2011.), Stronghold Kingdoms (2012.), Stronghold Crusader II (2014.), i Stronghold: Warlords (2021.).

Slika 9: Stronghold (Preuzeto 19. srpnja 2022. s https://www.hcl.hr/vijest/izradi-novi-stronghold-imamo-prve-detalje-188902/)

3.2.6.Warcraft

Warcraft je serijal kojeg je 1994. godine započela tvrtka Blizzard Entertainment. Igra je smještana u svijetu fantazije gdje igrač može igrati kao čovjek ili ork. Igra je također dobila i nastavke Warcraft II (uz proširenje „Beyond the Dark Portal“) te Warcraft III (uz proširenje „The Frozen Throne“) .

Slika 10: Warcraft III: Reforged (preuzeto 19. srpnja 2022. s https://playwarcraft3.com/en-us/)

4. Izrada strateške igre u stvarnom vremenu u alatu Unity

U praktičnom dijelu, napravit ćemo demonstracijsku verziju strateške igre u stvarnom vremenu. Za izradu ćemo koristiti programski alat Unity. Verzija alata u kojem je izrađena ova igra je 2020.3.11f1. Postoji nekoliko važnih mehanika koji se koriste u većini RTS igara, a to su:

- Kamera i minimapa – mnoge igre koriste fiksnu visinu i kut kamere koji pruža najbolji pogled na teren. No u nekim igrama, igrač sam može podešavati visinu, kut gledanja i rotaciju kamere pa ćemo to izvesti na ovom primjeru.
- Jedinice – ovisno o tematici igre, jedinice mogu biti razne. Razlika u jedinicama može biti u terenu po kojem se kreću (kopnene, pomorske, zračne…), po vrsti napada (bliska borba, dalekometna borba, opsadne jedinice…), po sposobnostima (vojska, radnici, liječnici, jedinice s posebnim moćima, skautske jedinice…) i mnoge druge. Za naš primjer koristit ćemo nekoliko osnovnih vrsta koje se najčešće pojavljuju u strateškim igrama
- Zgrade – također postoji razlika u igrama, ali većina strategija koristi zgrade pomoću kojih igrač može kupovati/graditi nove jedinice, nadograditi postojeće, istražiti nove sposobnosti za jedinice, braniti bazu i slično. U našem primjeru koristit ćemo zgrade za prikupljanje resursa i izgradnju novih jedinica.
- Resursi – gotovo svaka strategija ima određeni sustav prikupljanja i potrošnje resursa. U demonstrativne svrhe imat ćemo jedan resurs kojeg radnici pod kontrolom igrača moraju kopati kako bi igrač mogao kupovati nove jedinice i tako pojačati svoju vojsku.
- Cilj igre – ciljevi igre mogu biti razni: osvoji/zauzmi određeno područje, uništi protivnikovu vojsku/zgrade, skupi određeni broj bodova ili tehnološko dostignuće i slično. U našoj igri imat ćemo klasični cilj „Uništi sve protivničke zgrade“. Prvi igrač koji ostane bez ijedne zgrade je poražen, dok drugi igrač odnosi pobjedu.

4.1. Kamera

U ovoj igri koristimo jednu kameru i to Unityjevu glavnu kameru (Main Camera). Kamera se može pomicati i rotirati u svim smjerovima kako bi igrač imao bolji pregled situacije na terenu. Takav način pregleda igre inspiriran je igrama Total War i World in Conflict u kojemu se koriste slične mehanike [14] [16].

Kamera je postavljena u roditeljski objekt CameraParent koji na sebi sadrži skriptu *CameraMovement* zaduženu za prikupljanje korisničkog unosa kojim se kamera kreće i rotira po sceni. Kontrolne tipke za upravljanje kamerom su W (naprijed), A (lijevo), S (nazad) i D (desno). Pomicanjem kotačića miša kamera se može podizati i spuštati u odnosu na tlo. Pritiskom kotačića miša i pomicanjem miša, kamera se može okretati oko y osi (lijevo i desno) te x osi (gore i dolje). Držanjem lijeve tipke Shift igrač privremeno može ubrzati kretanje kamere po sceni.

Sljedeći blok programskog koda prikazuje implementaciju kretnje kamere po sceni. Prvo se pomoću GetAxis metode očitavaju podaci o pritisnutim tipkama i pomicanju miša. Sustavski je zadano da se parametar „Horizontal“ odnosi na strelice lijevo i desno te na tipke „A“ i „D“, dok se parametar „Vertical“ odnosi na strelice gore i dolje te tipke „W“ i „S“. Nakon toga provjeravamo nalazi li se kamera na najnižoj, odnosno najvišoj dozvoljenoj poziciji te se parametri podešavaju sukladno tome. Zatim kreiramo nove Vector3 objekte kojima prosljeđujemo informacije o brzini kamere, pomaku miša i unosu s tipkovnice za svaki smjer kretnje kamere. Kreiramo novi objekt u kojeg dodajemo sve smjerove kako bi dobili konačan smjer kretanja. Na kraju, taj smjer (varijabla „move“) dodamo objektu na kojem se nalazi (CameraParent) čime se traženi objekt pomakne.

```
float horizontalSpeed = transform.position.y * speed *
Input.GetAxis("Horizontal");
float verticalSpeed = transform.position.y * speed *
Input.GetAxis("Vertical");
float scrollSpeed = Mathf.Log(transform.position.y) * -zoomSpeed *
Input.GetAxis("Mouse ScrollWheel");


if (transform.position.y >= maxHeight && scrollSpeed > 0)
{
      scrollSpeed = 0;
}
else if (transform.position.y <= minHeight && scrollSpeed < 0)
{
      scrollSpeed = 0;
}


if (transform.position.y + scrollSpeed > maxHeight)
{
      scrollSpeed = maxHeight - transform.position.y;
}
else if (transform.position.y + scrollSpeed < minHeight)
```

```
{
      scrollSpeed = minHeight - transform.position.y;
}

Vector3 verticalMove = new Vector3(0, scrollSpeed, 0);
Vector3 lateralMove = horizontalSpeed * transform.right;
Vector3 forwardMove = transform.forward;
forwardMove.y = 0;
forwardMove.Normalize();
forwardMove *= verticalSpeed;

Vector3 move = verticalMove + lateralMove + forwardMove;
transform.position += move;
```

Rotacija kamere izvršava se nakon njezinog pomaka. Provjerava se je li pritisnut kotačić miša (parametar „2“ proslijeđen metodi GetMouseButton) te ako je, očitava pomak miša u odnosu na x i y os. Dobivene vrijednosti se množe s rotacijskom brzinom za svaku os i prosljeđuju novokreiranom Vector3 objektu „rotateValue“. Konačno, vrijednosti objekta rotateValue se oduzimaju od vrijednosti eulerAngles kako bi okretanje kamere pratilo stvarni pomak miša.

```
if (Input.GetMouseButton(2))
{
      float y = Input.GetAxis("Mouse X");
      float x = Input.GetAxis("Mouse Y");
      Vector3 rotateValue = new Vector3(x * rotationSpeedX, y *
rotationSpeedY, 0);
      transform.eulerAngles -= rotateValue;
}
```

4.1.1. Minimapa

Minimapa je još jedan važan element u strateškim igrama jer omogućuje igraču kratki pregled zbivanja na terenu kada je nije u mogućnosti pozicionorati glavnu kameru na druga mjesta. Minimape mogu upozoriti igrača na kretanja protivničkih jedinica i pružiti mu dragocjeno vrijeme za pravu reakciju [15].

Neke igre koriste dvodimenzionalne minimape koje ne prikazuju teren te koriste 2D simbole jedinica ili samo kružiće/kvadratiće različitih boja koji simboloziraju položaje jedinica na mapi. U ovom slučaju koristili smo klasičnu kameru koja prikazuje trenutno stanje terena iz

ptičje perspektive. Potavke kamere za minimapu su uglavnom jednake kao i one kod glavne kamere. Vrijednosti koje su različite su „Projection“ koju smo, u odnosu na glavnu kameru, s vrijednosti Perspective promijenili na Othrographic, „Size“ koja nam daje veće vidno polje kako bi pratili cijelu mapu te „TargetTexture“ za koju smo kreirali objekt tipa *RenderTexture* i podesili mu vrijednosti veličine (150x150 što je veličina objekta *RawImage* na kojem ćemo renderirati prikaz kamere) te DepthBuffer (postavljeno na NoDepthBuffer). Objektu RawImage smo pod parametar „Texture“ dodali RenderTexture kojeg smo kreirali te smo sliku postavili u donji lijevi kut zaslona radi preglednosti.

Slika 11: Prikaz terena za igru s minimapom u donjem lijevom kutu (slika autora)

4.2. Jedinice

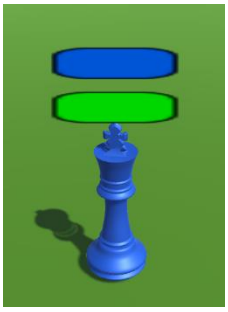
Jedinice su radna snaga i vojska koje igrač može micati po terenu i davati im naredbe kako bi ostavio prednost [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [35]. Za izradu ove igre, korišteno je 5 različitih tipova jedinica:

- Melee – jedinica namijenjena blizinskoj borbi
- Ranged – jedinica namijenjena borbi na daljinu
- Siege – jedinica namjenjena isključivo borbi na daljinu i protiv zgrada
- Spellcaster – jedinica s dodatnim sposobnostima (liječenje prijateljskih jedinica)
- Worker – radnička jedinica (skupljanje resursa)

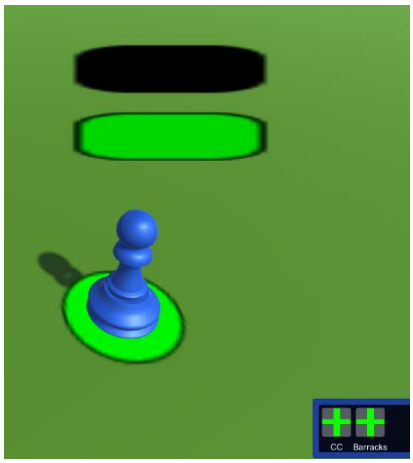
Slika 12: Jedinica tipa "Melee" (slika autora)

Slika 13: Jedinica tipa "Ranged" (slika autora)

Slika 14: Jedinica tipa "Siege" (slika autora)



Slika 15: Jedinica tipa "Spellcaster" (slika autora)



Slika 16: Jedinica tipa "Worker" s prikazom mogućih akcija u donjem desnom kutu (slika autora)

4.2.1. Statistike

Svaka jedinica sadrži skriptni objekt (eng. Scriptable Object) *BasicUnit* koji sadrži osnovne podatke svake jedinice: naziv jedinice, predložak za prijateljske/neprijateljske jedinice, ikona, tip jedinice, trajanje kreiranja jedinice te statistike po tipu jedinice. Statistike po tipu sadrže podatke kao što su zdravlje, šteta, brzina i slično. Sve informacije prikazane su u sljedećoj tablici:

Tablica 1: Statistički podaci za jedinice

Base (osnovni)		
Tip podatka	Naziv	Značenje
float	health	Maksimalno zdravlje jedinice
	armor	Otpornost na pretrpljenu štetu
	movementSpeed	Maksimalna brzina kretanja
	damage	Iznos štete na neprijatelja
	attackSpeed	Vremenski interval nakon kojeg ponovno može napasti
	attackRange	Maksimalna udaljenost za napad
	aggroRange	Udaljenost od neprijatelja na kojoj postaje agresivan
	energy	Maksimalna energija
	cost	Cijena kreiranje jedinice
Siege (opsadni)		
Tip podatka	Naziv	Značenje
float	splashDamage	Maksimalni iznos respršive štete
	minimumAttackRange	Minimalna udaljenost za napad

4.2.2. Predložak (eng. Prefab)

Svaki objekt jedinice sastoji se od jednog glavnog objekta i 4 osnovna objekta djece. Na roditeljskog objektu nalaze se komponente *Transform* (pozicija, rotacija i veličina), sudarači (eng. Colliders), *NavMeshAgent* (sustav za kretanje), *Rigidbody* (fizika jedinice), *AudioSource* (izvor zvuka) te skripte *PlayerUnit*/*EnemyUnit*, *IUnit* (samo za jedinice kojima upravlja igrač) te klasa specifična za za svaku vrstu jednice (npr. *Melee*). Također, jedinice imaju postavljenu oznaku „Unit“ te sloj „Interactables“ ako su igračeve jednicice, odnosno „EnemyUnit“ ako su neprijateljske. Objekti djece su:

- Highlight – zelena podloga koja prikazuje odabrane jedinice
- UnitStatDisplay – plave i zelene slike koje skaliraju ovisno o količini energije odnosno zdravlja koje jedinica ima. Također sadrži istoimenu skriptu koja upravlja obnavljanjem zdravlja i energije te primanjem štete. Također uklanja jedinicu iz igre ako jedinica nema zdravlja
- model jedinice – vizualni prikaz jedinice na sceni
- UnitSoundManager – sadrži i upravlja zvukovima stvaranja i umiranja jedinice

U klasi *UnitStatDisplay* nalazi se metoda HandleHealthAndEnergy koja upravlja zdravljem jedinica i zgrada. Objekt na kojem se nalazi skripta sadrži reference na prikaz trenutnog zdravlja i energije. Prikaz je napravljen pomoću dvije slike – crna slika koja je pozadina i zelena slika koja prikazuje zdravlje (odnosno plava koja prikazuje energiju). Slike su usidrene na lijevoj strani i skaliraju se tako da se proširuju na desno ovisno o tome koliko je trenutno zdravlje jedinice ili zgrade u odnosu na maksimalno moguće. Time se dobije efekt crnog „spremnika“ i zelene ispune. Na početku metode, dohvaća se referenca na glavnu kameru te se slike rotiraju prema kameri kako bi igrač uvijek imao pregled zdravlja jedinica. Zatim se u varijablama healthBar i energyBar, koje sadrže reference na slike, svojstvo fillAmount postavlja na vrijednost currentHealth / health, odnosno omjer trenutnog zdravlja i maksimalnog. Pozivamo metodu ChengeHealthBarColor koja mijenja boju slike na žutu ako je količina zdravlja manja od 66%, odnosno na crvenu boju ako je manja od 33%. Na kraju, provjerava se ima li jedinica ili zgrada preostalog zdravlja te ako nema, poziva metodu Die koja prvo zabilježi statističke podatke, uklanja jedinice iz prečaca za jedinice (eng. hotkeys) ako postoji te jedinice/zgrade iz trenutne kontrole, pušta zvuk umiranja te uništava objekt.

```
void HandleHealthAndEnergy()
{
      Camera camera = Camera.main;
      gameObject.transform.LookAt(gameObject.transform.position +
      camera.transform.rotation * Vector3.forward,
      camera.transform.rotation * Vector3.up);
```

```
        healthBar.fillAmount = currentHealth / health;
        energyBar.fillAmount = currentEnergy / energy;
        ChangeHealthBarColor();
        if (currentHealth <= 0)
        {
                Die();
        }
}

private void Die()
{
        if (isPlayerAsset)
        {
                if (isUnit)
                {
                        GameManager.GameStats.unitsLost++;
                        InputHandler.instance.selectedUnits.Remove(
                        gameObject.transform.parent);
                        InputHandler.instance.RemoveDestroyedUnitFromHotkey(
                        gameObject.transform.parent);
                        PlaySoundUnit("Death");
                        Destroy(gameObject.transform.parent.gameObject);
                }
                else
                {
                        GameManager.GameStats.buildingsLost++;
                        if (InputHandler.instance.selectedBuilding == this)
                        {
                                InputHandler.instance.selectedBuilding = null;
                        }
                        GameManager.instance.BuildingDestroyed(true);
                        PlaySoundBuilding();
                        Destroy(gameObject.transform.parent.gameObject);
                }
        }
        else
        {
                if (isUnit)
                {
                        GameManager.GameStats.unitsKilled++;
                        PlaySoundUnit("Death");
```

```
                Destroy(gameObject.transform.parent.gameObject);
            }
            else
            {
                GameManager.GameStats.buildingsDestroyed++;
                GameManager.instance.BuildingDestroyed(false);
                PlaySoundBuilding();
                Destroy(gameObject.transform.parent.gameObject);
            }
        }
}
```

4.2.3. Odabir jedinice

Jedinice možemo odabrati na tri klasična načina odabira jedinica u strateškim igrama. Prvi način je pritiskom lijeve tipke miša na jedinicu čime nam je pod kontrolom ta jedna jedinica. Drugi način je „povuci i pusti“ (eng. Drag and drop) gdje držeći lijevu tipku miša i povlačeći ga, crtamo pravokutnik na ekranu te puštanjem odabiremo sve prijateljske jedinice unutar granica tog pravokutnika. Posljednji način je korištenjem prečaca, odnosno kontrolnih grupa. Jedinice možemo dodati u 10 kontrolnih grupa i odabrati ih pritiskom na jednu od brojčanih tipki. Jedinice se proizvoljno mogu dodavati i uklanjati iz kontrolnih grupa.

Prva dva načina odabira jedinica riješili smo početnom provjerom je li lijeva tipka miša pritisnuta. Ako je, provjeravamo ima li korisnik interakciju s korisničkim sučeljem te prekidamo operaciju ako je to istinito kako ne bi odabirali jedinice dok se služimo izbornicima. Nakon toga, pohranjujemo poziciju miša u varijablu te ispaljujemo zraku (eng. Ray) iz kamere s trenutne pozicije miša. Ako je zraka pogodila prijateljsku jedinicu ili zgradu, što provjeravamo metodama AddedUnit i AddedBuilding koje vraćaju objekte jedinice ili zgrade ako je pogodak, odnosno null vrijednost ako nije, tada ih postavljamo pod trenutnu kontrolu. Ako nije, tada varijablu isDragging postavljamo kao istinitu jer smatramo da počinje povlačenje po ekranu te uklanjamo sve jedinice iz trenutne kontrole. Nakon toga dolazi provjera je li lijeva tipka miša puštena te ako je, za svaku prijateljsku jedinicu provjeravamo nalazi li se unutar nacrtanog pravokutnika, dodajući pod trenutnu kontrolu sve one koje se nalaze. Završno, varijablu isDragging postavljamo na laž.

```
if (Input.GetMouseButtonDown(0))
{
    if (EventSystem.current.IsPointerOverGameObject())
    {
```

```
            return;
      }
      mousePosition = Input.mousePosition;
      Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
      if (Physics.Raycast(ray, out hit, 1000, interactableLayer))
      {
            if (AddedUnit(hit.transform,
            Input.GetKey(KeyCode.LeftControl)))
            {
                  //...
            }
            else if (AddedBuilding(hit.transform))
            {
                  //...
            }
      }
      else
      {
            isDragging = true;
            DeselectUnits();
      }
}

if (Input.GetMouseButtonUp(0))
{
      foreach (Transform child in PlayerManager.instance.playerUnits)
      {
            foreach (Transform unit in child)
            {
                  if (IsWithinSelectionBounds(unit))
                  {
                        AddedUnit(unit, true);
                  }
            }
      }
      isDragging = false;
}

IUnit AddedUnit(Transform tf, bool canMultiselect = false)
{
```

```
	IUnit iUnit = tf.GetComponent<IUnit>();
	if (iUnit)
	{
		if (!canMultiselect)
		{
			DeselectUnits();
		}
		selectedUnits.Add(iUnit.gameObject.transform);
		iUnit.OnInteractEnter();
		return iUnit;
	}
	return null;
}

IBuilding AddedBuilding(Transform tf)
{
	IBuilding iBuilding = tf.GetComponent<IBuilding>();
	if (iBuilding)
	{
		DeselectUnits();
		selectedBuilding = iBuilding.gameObject.transform;
		iBuilding.OnInteractEnter();
		return iBuilding;
	}
	return null;
}
```

Ako jedinice odabiremo trećim načinom, putem prečaca, tada prvo provjeravamo je li pritisnuta tipka i je li ta tipka broj. Ako je broj, tada ga pohranjujemo u lokalnu varijablu i provjeravamo je li broj između 0 i 9. Ako je, provjeravamo je li također pritisnuta tipka „Q“ te ako je, sve jedinice pod našom trenutnom kontrolom dodajemo pod prečac pod pritisnutim brojem. Prečaci su definirani kao niz od 10 listi koje prihvaćaju objekte tipa *Transform*. Ako je uz broj pritisnuta tipka „E“, tada sve jedinice pod našom kontrolom uklanjamo iz kontrolne grupe pod pritisnutim brojem, ako se tamo nalaze. Ako je pak pritisnut samo broj, tada pod trenutno kontrolu postavljamo samo one jedinice koje se nalaze pod prečacem pod tim brojem.

```
if (Input.inputString != "")
{
	int number;
```

```
        bool isNumber = int.TryParse(Input.inputString, out number);
        if (isNumber && number >= 0 && number <= 9)
        {
            if (Input.GetKey(number.ToString()) && Input.GetKey(KeyCode.Q))
            {
                AddHotkeyUnits(number);
            }
            else if (Input.GetKey(number.ToString()) &&
            Input.GetKey(KeyCode.E))
            {
                RemoveHotkeyUnits(number);
            }
            else if (Input.GetKeyDown(number.ToString()))
            {
                SelectHotkeyUnits(number);
            }
        }
}
```

4.2.4.Naredbe

Svaka jedinica može dobiti određene naredbe koje može izvršiti. Naredbe se mogu dati jedinicama koje su pod trenutnom kontrolom igrača pritiskom na desnu tipku miša te će odabrana jedinica ili grupa jedinica izvršiti zadanu naredbu. Jedinica će izvršiti naredbu ovisno o tome nad kojim je objektom pritisnuta tipka. Ako je desna tipka miša pritisnuta na tlu, tada će se jedinica kretati prema tom mjestu. Ako je pritisnuta na neprijateljsku jedinicu, tada će krenuti u napad. Ako je jedinica tipa „Worker“, tada će pritiskom na resurs jedinica započeti sa skupljanjem tog resursa. Ako je jedinica tipa „Spellcaster“ i tipka je pritisnuta nad prijateljskom jedinicom, tada će jedinica započeti s liječenjem jedinice.

U programskom kodu prvo provjeravamo je li pritisnuta desna tipka miša i ima li igrač jedinice pod trenutnom kontrolom. Ako je to istinito, kreira se zraka iz kamere s pozicije miša te provjeravamo postoji li pogodak na neki objekt. Ako postoji, za početak provjeravamo imamo li u trenutno odabranim jedinicama radnike te im prekidamo trenutni rad ako ga obavljaju. Nakon toga, u varijablu layerHit pohranjujemo sloj (eng. layer) pogođenog objekta te u switch grananju dajemo naredbu ovisno i sloju na kojem se objekt nalazi. Slojevi u Unityju imaju nazive i brojčane vrijednosti pa tako provjeravamo sljedeće vrijednosti:

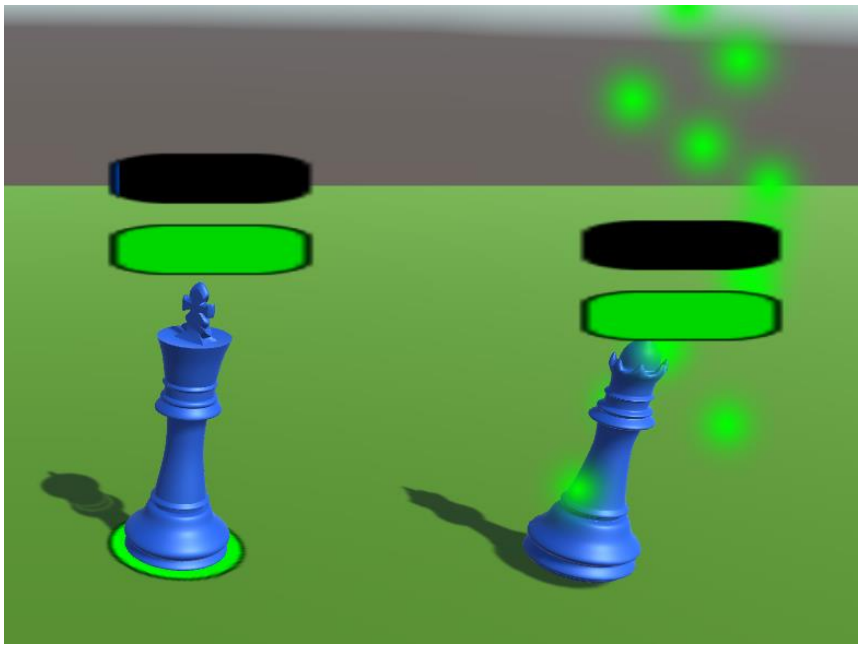
- 8: sloj prijateljskih jedinica – ako je pogođena prijateljska jedinica, provjeravamo imamo li samo jednu jedinicu pod trenutnom kontrolom i je li ta jedinica tipa „Spellcaster“. Ako

je, tada s njene komponente *Spellcaster* pozivamo metodu HealUnit kojom liječimo jedinicu na koju smo dali naredbu.

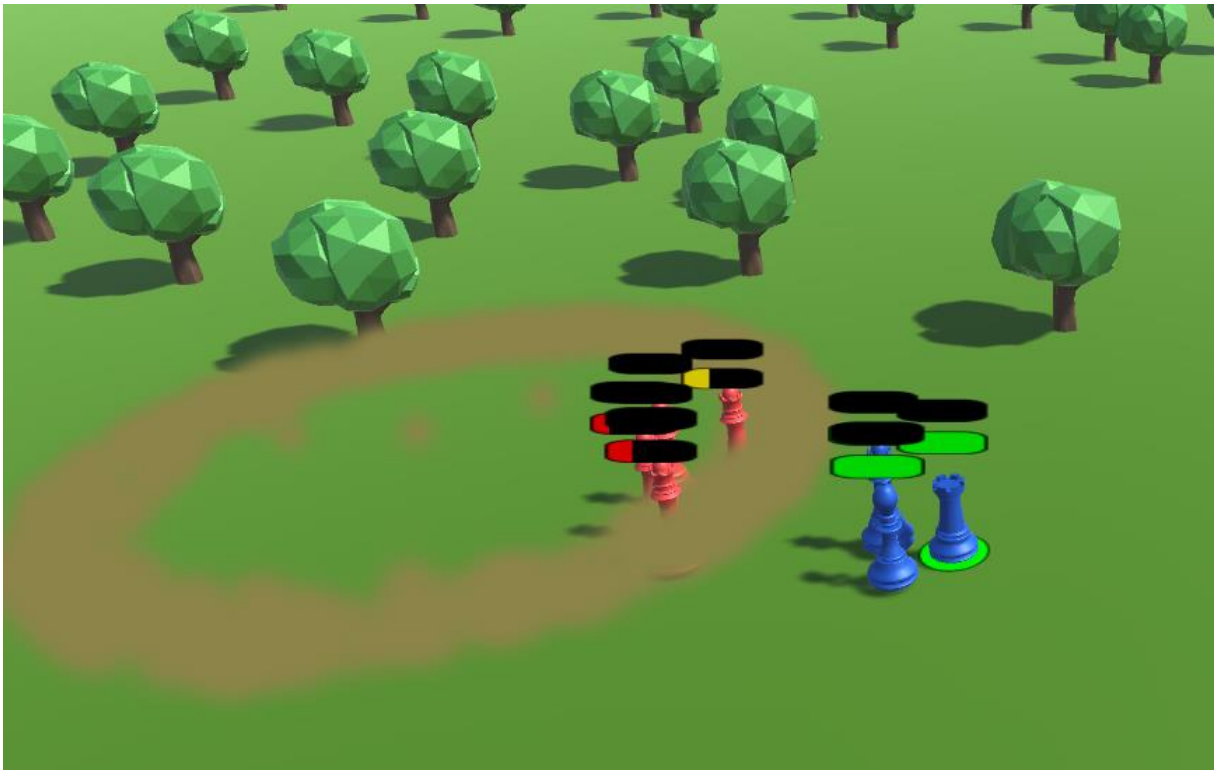
- 9: sloj neprijateljskih jedinica – ako je pogođena neprijateljska jedinica, tada za svaku jedinicu pod trenutnom kontrolom dohvaćamo njenu komponentu *PlayerUnit* iz koje pozivamo metodu SetEnemyTarget čime pogođenu jedinicu postavljamo kao „metu“ jedinicama pod trenutnom kontrolom igrača, odnosno šaljemo ih u napad na tu neprijateljsku jedinicu.
- 10: sloj resursa – ako je pogođen resurs, tada za svaku jedinicu pod trenutnom kontrolom provjeravamo je li tipa „Worker“ i ako je, s komponente *Worker* pozivamo metodu HandleWorking čime jedinicu šaljemo u skupljanje resursa sa zadanog resursnog čvora.
- Zadano: ako objekt koji je pogođen ne pripada niti jednom od ranije definiranih slojeva, tada sve jedinice pod trenutnom kontrolom uklanjaju svoje trenutne mete ako ih imaju te započinju kretnju prema mjestu na koje smo pogodili mišem.

```
if (Input.GetMouseButtonDown(1) && HaveSelectedUnits())
{
      Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
      if (Physics.Raycast(ray, out hit))
      {
            foreach (Transform unit in selectedUnits)
            {
                  PlayerUnit playerUnit =
                  unit.gameObject.GetComponent<PlayerUnit>();
                  if (playerUnit.unitType.unitType ==
            BasicUnit.UnitType.Worker)
                  {
                        playerUnit.GetComponent<Worker>()
                        .HandleWorking(false);
                  }
            }
            LayerMask layerHit = hit.transform.gameObject.layer;
            switch (layerHit.value)
            {
            case 8:
                  if (selectedUnits.Count == 1 &&
                  selectedUnits[0].GetComponent<Spellcaster>() != null)
                  {
                        selectedUnits[0].GetComponent<Spellcaster>()
                        .HealUnit(hit.transform);
```

```
                }
                break;
            case 9:
                foreach (Transform unit in selectedUnits)
                {
                    PlayerUnit playerUnit =
                    unit.gameObject.GetComponent<PlayerUnit>();
                    playerUnit.SetEnemyTarget(hit.transform);
                }
                break;
            case 10:
                foreach (Transform unit in selectedUnits)
                {
                    PlayerUnit playerUnit =
                    unit.gameObject.GetComponent<PlayerUnit>();
                    if (playerUnit.unitType.unitType ==
                    BasicUnit.UnitType.Worker)
                    {
                        playerUnit.GetComponent<Worker>()
                        .HandleWorking(true, hit.transform);
                    }
                }
                break;
            default:
                foreach (Transform unit in selectedUnits)
                {
                    PlayerUnit playerUnit =
                    unit.gameObject.GetComponent<PlayerUnit>();
                    playerUnit.hasAggro = false;
                    playerUnit.MoveUnit(hit.point);
                }
                break;
        }
    }
}
```



Slika 17: Jedinica tipa "Spellcaster" (lijevo) liječi jedinicu tipa "Melee" (desno) (slika autora)

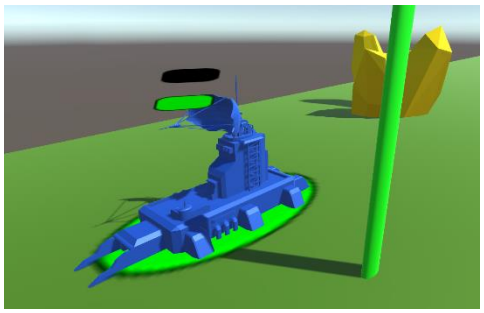


Slika 18: Borba između neprijatelja (crveni) i igrača (plavi) uz vidljivu rasipajuću štetu (eng. splash damage) u obliku smeđe kružnice koja se širi (slika autora)

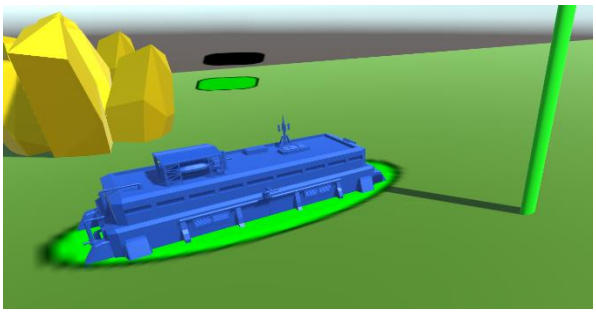
4.3. Zgrade

Zgrade su dio igračeve „imovine“ koju može kontrolirati, ali ih ne može micati po terenu [27] [28] [29] [30] [31] [32] [33]. Za ovu igru, koristili smo dvije vrste zgrada:

- CommandCenter – zgrada u koju se skupljaju resursi. Služi za proizvodnju jedinica tipa „Worker“
- Barracks – zgrada za proizvodnju vojnih jedinica („Melee“, „Ranged“, „Siege“, „Spellcaster“)



Slika 19: Zgrada tipa "CommandCenter" (slika autora)



Slika 20: Zgrada tipa "Barrcaks" (slika autora)

4.3.1. Statistike

Svaka jedinica sadrži skriptni objekt BasicBuilding koji sadrži osnovne podatke svake zgrade: naziv zgrade, predložak za prijateljske/neprijateljske zgrade, nacrt zgrade, ikona, tip zgrade, trajanje kreiranja zgrade, lokacija stvaranja jedinica te statistike po tipu zgrade. informacije prikazane su u sljedećoj tablici:

Tablica 2: Statistički podaci za zgrade

Base (osnovni)		
Tip podatka	Naziv	Značenje
float	health	Maksimalno zdravlje zgrade
	armor	Otpornost na pretrpljenu štetu

	cost	Cijena kreiranje zgrade

4.3.2. Predložak

U predlošku zgrade, objekt roditelja sadrži komponente *Transform*, *Collider*, *Rigidbody* te skripte *PlayerBuilding*/*EnemyBuilding* i *IBuilding* (samo za igračeve zgrade) Oznaka zgrade postavljena je na „Building“, a sloj na „Interactables“ za prijateljske zgrade, odnosno „EnemyUnit“ za neprijatesljske. Kao i kod jedinica, zgrada također sadrži nekoliko objekata djece:

- UnitStatDisplay – prikaz i upravljanje zdravljem zgrade
- Highlight – objekt koji se prikazuje kada je zgrada pod trenutnom kontrolom igrača
- SpawnMarker – objekt koji označava mjesto na koje dolaze novokreirane jedinice. Prikazuje se samo kada je zgrada pod trenutnom kontrolom igrača
- Model zgrade
- Sound – sadrži zvučni efekt uništenja zgrade. Također sadrži i sustav čestica

4.3.3. Kreiranje zgrade

Da bi igrač kreirao zgradu, prvo je potrebno odabrati jednog radnika. U desnom donjem kutu ekrana pojavljuju se dva gumba koji nude odabir zgrade za izgradnju. Pritiskom na gumb, prvo se stvara nacrt zgrade koji je bijele boje i proziran te služi igraču kao prikaz budućeg stanja polja za igru. Tek kada igrač pritisne lijevi gumb miša, zgrada je postavljena i spremna za korištenje.

Izgradnja zgrade započinje u klasi *Worker* metodom SpawnBuilding. Prvo provjeravamo ima li igrač dovoljnu količinu resursa za izgradnju zgrade. Ako ima dovoljno, instanciramo novi objekt nacrta (eng. Blueprint), u suprotnom ispisujemo poruku o manjku resursa.

```
public void SpawnBuilding(BasicBuilding building)
{
	if (building.baseStats.cost <=
	ResourceManager.instance.currentResources)
	{
		SpawnBlueprint(building.buildingBlueprintPrefab);
	}
	else
	{
		LogController.instance.ShowMessage("Not enough resources!");
```

```
        }
}

void SpawnBlueprint(GameObject blueprint)
{
        Instantiate(blueprint);
}
```

Nacrt zgrade zatim koristi svoju skriptu *Blueprint* za daljnju izgradnju. Po instanciranju nacrta dohvaćaju se komponente *BuildingPlacement*, *Renderer* i *AudioSource*, varijabla isBuildingProcess u klasi *InputHandler* se postavlja na istinu, puštamo prikladni zvuk te pozivamo metodu StickBlueprintToMouse koju poziva i u Update metodi. Ova metoda služi za pomicanje nacrta sukladno pomicanju miša. Prvo se ispaljuje zraka iz kamere s mjesta na kojemu se nalazi miš. Zatim provjeravamo jesmo li pogodili objekt tla i ako jesmo, ažuriramo x i z koordinate nacrta sukladno mjestu na kojemu se nalazi miš tako da je u ravnini s tlom.

```
void StickBlueprintToMouse()
{
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        if (Physics.Raycast(ray, out hit, 50000.0f, (1 << 7)))
        {
                transform.position = new Vector3(hit.point.x,
                prefab.transform.position.y, hit.point.z);
        }
}
```

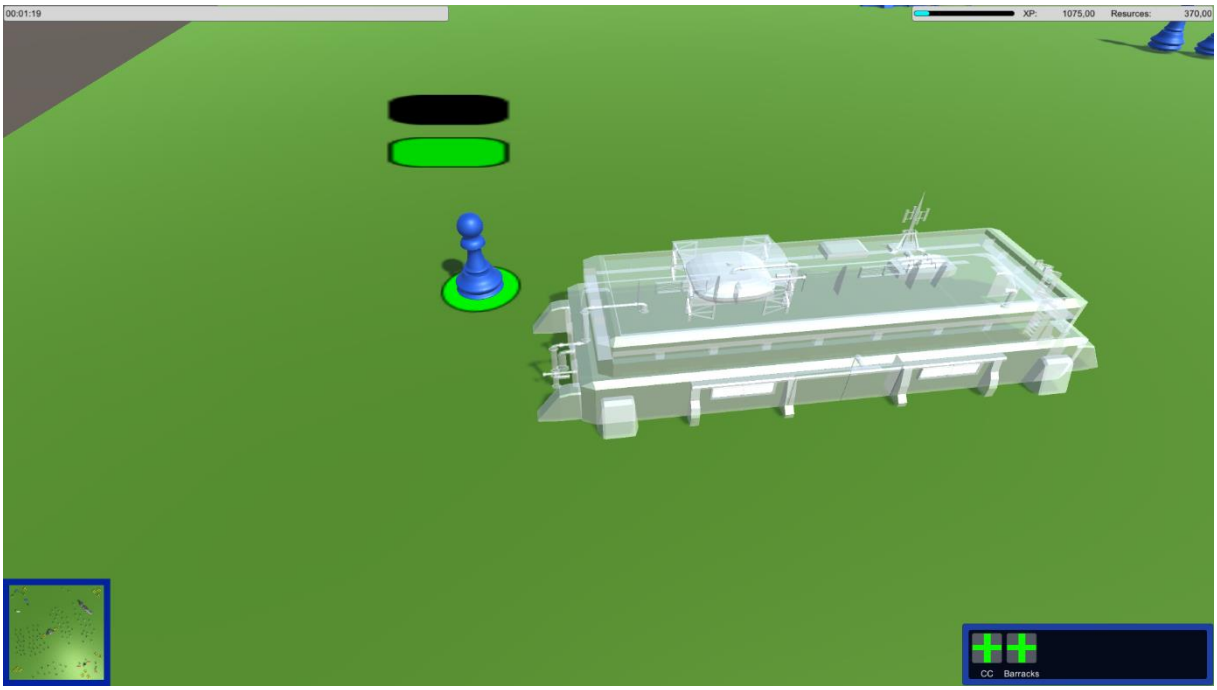
U nastavku Update metode provjeravamo može li igrač postaviti zgradu te sukladno mogućnosti, mijenjamo boju, odnosno materijal nacrta (bijelo-prozirno ako može, crveno-prozirno ako ne može). Nakon toga, slijedi provjera je li pritisnut lijevi gumb miša, to jest, želi li igrač postaviti zgradu na trenutno poziciju nacrta. Prvo metodom CanPlaceBuilding provjeravamo je li mjesto za izgradnju ispravno ili ne. Ako je, instanciramo novu zgradu na mjesto nacrta, oduzimamo cijenu zgrade od trenutne količine resursa koje igrač ima na raspolaganju, ažuriramo statističke podatke, smještamo zgradu na prikladno mjesto u hijerarhiji scene, mijenjamo varijablu isBuildingProcess u laž, zaustavljamo zvukove i uništavamo objekt nacrta. Inače ispisujemo poruku o nedozvoljenoj poziciji za gradnju zgrade.

U slučaju da je igrač prije potvrde izgradnje pritisnuo desni gumb miša, tada mijenjamo varijablu isBuildingProcess u laž, zaustavljamo zvukove i uništavamo objekt nacrta.

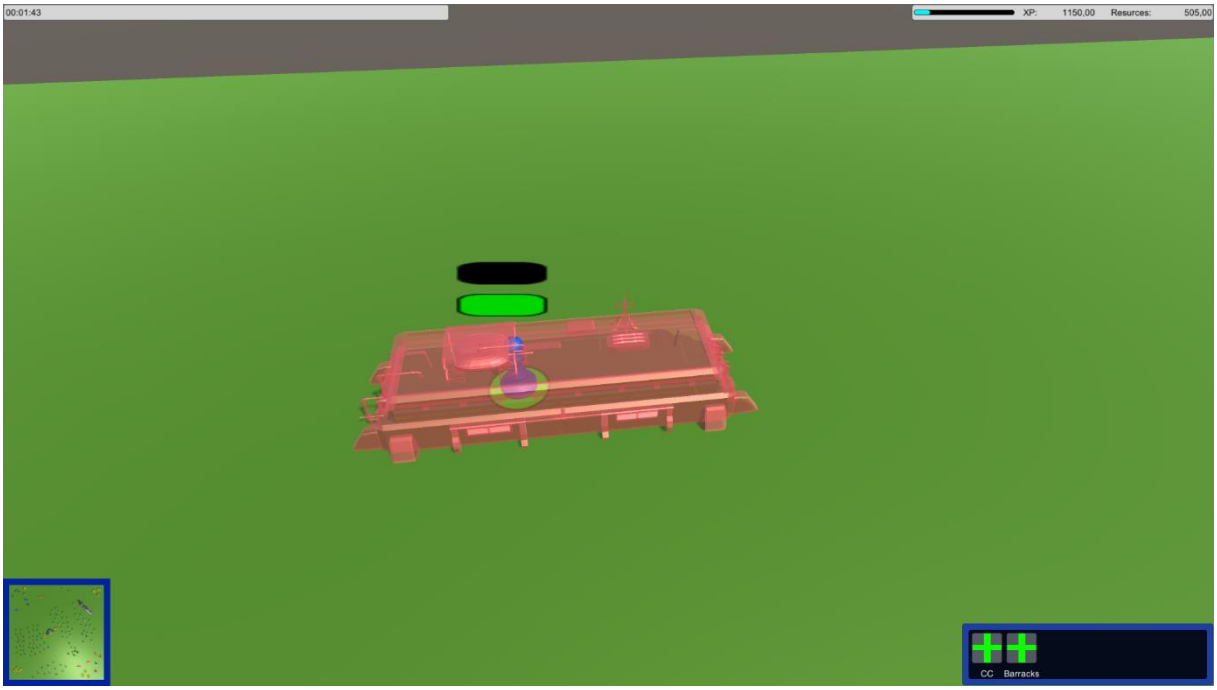
```
StickBlueprintToMouse();
if (CanPlaceBuilding())
{
      SetBlueprintMaterial(blueprintMaterial);
}
else
{
      SetBlueprintMaterial(blueprintInvalidMaterial);
}
if (Input.GetMouseButton(0))
{
      if (CanPlaceBuilding())
      {
            Vector3 position = new Vector3(transform.position.x,
            prefab.transform.position.y, transform.position.z);
            GameObject building = Instantiate(prefab, position,
            transform.rotation);
            PlayerBuilding playerBuilding =
            building.GetComponent<PlayerBuilding>();
            ResourceManager.instance.SubtractResource(
            playerBuilding.baseStats.cost);
            LogController.instance.ShowMessage(
            $"{playerBuilding.buildingType.buildingName} added to the
            construction queue.");
            GameManager.GameStats.buildingsConstructed++;
            ActionFrame.instance.SetBuildingParent(building);
            InputHandler.instance.isBuildingProcess = false;
            StopPlayingSound();
            Destroy(gameObject);
      }
      else
      {
            LogController.instance.ShowMessage("Invalid location!");
      }
}
if (Input.GetMouseButtonDown(1))
{
      InputHandler.instance.isBuildingProcess = false;
      StopPlayingSound();
      Destroy(gameObject);
}
```

Klasa koja provjerava ispravnost lokacije je *BuildingPlacement* i također se nalazi na nacrtu zgrade. Ova klasa sadrži listu sudarača i oznaka (eng. tags). Prilikom svakog sudara s drugim objektom, provjeravamo sadrži li lista oznaka istu oznaku kao i objekt s kojim se nacrt sudario. Ako sadrži, dodaje ga u listu sudarača. Prilikom izlaska iz sudarača objekta, također se radi ista provjera, ali ovaj put sudarač uklanjamo iz liste. Prilikom potvrde izgradnje, u klasi *Blueprint* se provjerava je li lista sudarača prazna kako bi se izgradnja mogla izvršiti.

```
[HideInInspector]
public List<Collider> colliders = new List<Collider>();
private readonly List<string> blockingTags = new List<string>(){
"Unit",
"Building",
"Resource",
"Environment"
};

private void OnTriggerEnter(Collider other)
{
      if (blockingTags.Contains(other.gameObject.tag))
      {
            colliders.Add(other);
      }
}

private void OnTriggerExit(Collider other)
{
      if (blockingTags.Contains(other.gameObject.tag))
      {
            colliders.Remove(other);
      }
}
```



Slika 21: Nacrt zgrade tipa "Barracks" (slika autora)



Slika 22: Nacrt zgrade tipa "Barracks" kada izgradnja nije dozvoljena (slika autora)

4.3.4. Stvaranje jedinica

Sve jedinice mogu se stvarati isključivo interakcijom sa zgradama. Odabirom zgrade pod trenutnu kontrolu, u donjem desnom kutu ekrana prikazat će se gumbi s jedinicama koje

je moguće stvoriti u odabranoj zgradi. Pritiskom na gumb, cijena jedinice bit će oduzeta od količine resursa koju igrač posjeduje, a jedinica će se dodati u red čekanja. Ukoliko igrač nema dovoljno resursa, ispisat će se poruka o manjku resursa.

Pritiskom na gumb, u klasi *PlayerBuilding* poziva se metoda SpawnUnit kojoj se prosljeđuje jedinica koju se želi stvoriti. Provjerava se ima li igrač dovoljnu količinu resursa te ako ima dodaje vrijeme kreiranja jedinice u listu spawnQueue i predložak jedinice u listu spawnOrder. Također ispisuje i prikladnu poruku korisniku. Poruka se ispisuje u slučaju da igrač nema dovoljno resursa za kreiranje jedinice. Provjera `if (spawnQueue.Count == 1)` služi kako bi se kreiranje jedinica pokrenulo samo jednom, budući da je riječ o rekurzivnoj funkciji. Ako je provjera istinita, iz klase *ActionTimer* poziva se metoda SpawnQueueTimerPlayer.

```
public void SpawnUnit(BasicUnit unit)
{
      if (unit.baseStats.cost <= ResourceManager.instance.currentResources)
      {
            spawnQueue.Add(unit.spawnTime);
            spawnOrder.Add(unit.cubePrefab);
            LogController.instance.ShowMessage(
            $"{unit.unitName} added to the building queue.");
            ResourceManager.instance.SubtractResource(unit.baseStats.cost);
      }
      else
      {
            LogController.instance.ShowMessage("Not enough resources!");
      }

      if (spawnQueue.Count == 1)
      {
            ActionTimer.instance.StartCoroutine(
            ActionTimer.instance.SpawnQueueTimerPlayer(this));
      }
      else if (spawnQueue.Count == 0)
      {
            ActionTimer.instance.StopAllCoroutines();
      }
}
```

SpawnQueueTimerPlayer metoda upravlja redom čekanja kod kreiranja jedinica. Prvo se provjerava ima li još jedinica u redu čekanja te ako ima, zove metodu WaitForSeconds koja odgađa daljnje izvođenje koda onoliko dugo koliko je trajanje kreiranja sljedeće jedinice u redu čekanja. Poziva se metoda SpawnObject koja instancira sam objekt jedinice na sceni. Nakon toga se provjerava ima li još jedinica u redu čekanja te ako ima, metoda se ponovno poziva.

```
public IEnumerator SpawnQueueTimerPlayer(PlayerBuilding pb)
{
	if (pb.spawnQueue.Count > 0)
	{
		yield return new WaitForSeconds(pb.spawnQueue[0]);
		pb.SpawnObject();
		if (pb.spawnQueue.Count > 0)
		{
			StartCoroutine(SpawnQueueTimerPlayer(pb));
		}
	}
}
```

Metoda SpawnObject se koristi za stvaranje samog objekta jedinice u sceni. Objekt se prvo instancira. Nakon toga se ažuriraju statistički podaci igre. Slijedi puštanje zvuka stvaranja jedinice, postavljanje objekta u hijerarhiju scene i zadavanje naredbe jedinici za kretanje na mjesto stvaranja. Na kraju se jedinica i trajanje kreiranja jedinice uklanjaju iz reda čekanja.

```
public void SpawnObject()
{
	GameObject spawnedObject = Instantiate(spawnOrder[0],
	spawnPoint.transform.parent.position + buildingType.spawnLocation,
	Quaternion.identity);
	GameManager.GameStats.unitsBuilt++;
	PlayerUnit playerUnit = spawnedObject.GetComponent<PlayerUnit>();
	UnitStatDisplay usd =
	spawnedObject.gameObject.GetComponentInChildren<UnitStatDisplay>();
	usd.PlaySoundUnit("Spawn");
	playerUnit.transform.SetParent(GameObject.Find("Player" +
	playerUnit.unitType.unitType.ToString()).transform);
	playerUnit.MoveUnit(spawnPoint.transform.position);
	spawnQueue.Remove(spawnQueue[0]);
	spawnOrder.Remove(spawnOrder[0]);
}
```

4.4. Resursi

U svakoj strateškoj igri, igrač je na neki način ograničen pri izgradnji vojske zalihama resursa. Bilo to kopanje ruda, sječa drveća, skupljanje hrane ili jednostavno količinsko ograničenje pri kupovini vojne opreme, igrač mora iskoristiti dostupne resurse na najbolji mogući način.

U našem praktičnom primjeru, postavili smo jedan resurs (zlatna ruda) kojeg igrač mora kopati i čijim zalihama može plaćati izgradnju svojih jedinica i zgrada kako bi pobijedio [34].

Objekt ResourceNode kojeg igrač kopa sadrži komponente *Transform*, *Collider* i skriptu *Resource*. Oznaka i sloj postavljeni su na vrijednost „Resource“. Resurs ima i objekt dijete koji sadrži model resursa.

U *Resource* klasi imamo dva float atributa: javni amount koji predstavlja trenutnu količinu resursa u resursnom čvoru i privatni resourcePerScale koji predstavlja količinu resursa potrebnu u čvoru da bi objektu povećali veličinu kako bi imali dojam da se resurs „iskopava“. U klasi imamo i dvije metode. Javna metoda GatheredResources koja kao argument prima maksimalnu količinu resursa koje radnik može kopati. Ovdje provjeravamo ima li dovoljno resursa za iskopati te ako ima, radnik kopa maksimalnu moguću količinu. Ako nema, radnik će iskopati onoliko resursa koliko ih je ostalo u čvoru. Zatim pozivamo metodu ScaleResourceObject u kojoj skaliramo veličinu resursa ovisno o tome koliko je resursa ostalo u čvoru.

```
public float GatheredResources(float gatheringAmount)
{
      float gatheredAmount = gatheringAmount > amount ?
      amount : gatheringAmount;

      amount -= gatheringAmount > amount ?
      amount : gatheringAmount;

      ScaleResourceObject();

      return gatheredAmount;
}

void ScaleResourceObject()
{
      float scale = Mathf.Floor(amount / resourcePerScale) + 1;

      transform.localScale = new Vector3(scale, scale, scale);
}
```

Radnička jedinica (Worker) može imati četiri stanja: GoingToResource (odlazak to resursnog čvora), Gathering (kopanje resursa), ReturningCargo (nošenje resursa do zgrade) i Other (kada nije zadana naredba kopanja). Samo kopanje izvršava se od strane radničke jedinice u klasi *Worker*. Naredba za kopanje izvršava se u metodi HandleWorking u kojoj radnik dobije informaciju o resursnom čvoru s kojeg će kopati te informaciju o zgradi CommandCenter u koju će odnijeti iskopane resurse. Metoda koje se poziva nakon toga je ChangeWorkerState u kojoj se mijenja stanje radnika. Ovisno o primljenom parametru, u ovoj metodi se pozivaju druge metode kao što su GatherResource, MoveUnit i StopUnit. Stanja se mijenjaju na sljedeći način:

- Gathering – radnik je došao do resursnog čvora. Poziva se korutina (eng. Coroutine) GatherResource. U toj metodi zaustavljamo radnika, aktiviramo dustParticles objekt (vizualni prikaz prašine kod kopanja) i puštamo zvuk kopanja. Zatim odgađamo izvođenje koda na dvije sekunde (radi dojma kopanja). Nakon toga zaustavljamo puštanje zvuka, potavljamo vrijednost tereta kojeg je iskopao, provjeravamo koliko je resursa ostalo (ako ih više nema, uništavamo resursni čvor), aktiviramo cargoGO objekt (vizualni prikaz tereta kod radnika) i deaktiviramo dustParticles objekt.
- GoingToResource – radnik ide prema resursnom čvoru. Poziva se metoda MoveUnit iz klase *PlayerUnit* (*EnemyUnit* ako je neprijateljska jedinica) kojoj se prosljeđuje pozicija zadanog resursnog čvora.
- ReturningCargo – radnik nosi iskopani resurs do zgrade. Ponovno se poziva metoda MoveUnit, ali sada je parametar pozicija zadane zgrade.
- Other – radnik je iskopao resurs. Poziva se metoda StopUnit. Također prijelazi u stanje Other ako je dobio naredbu za kretanje na neku drugu lokaciju ili za napad.

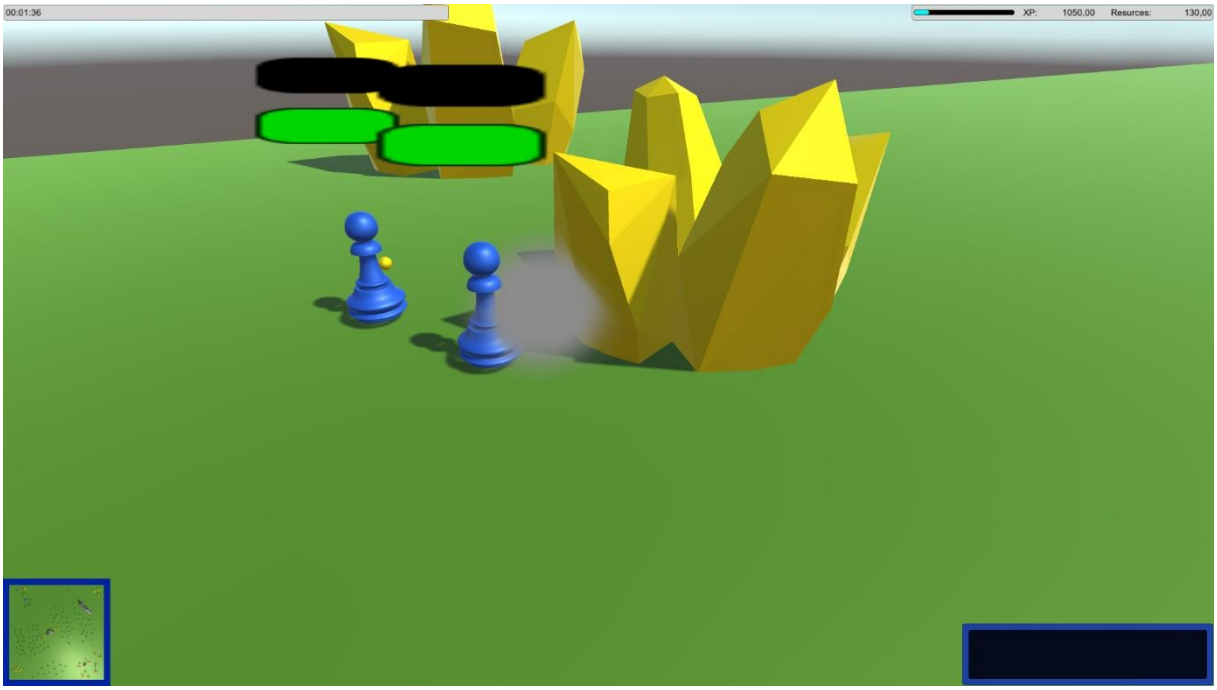
Jedinica radnika na sebi ima dodatni sudarač s uključenom oznakom isTrigger. Ovime se provjerava je li radnik došao do resursa ili zgrade čime se sukladno mijenja njegovo stanje.

```
void ChangeWorkerState(WorkerState workerState)
{
	state = workerState;
	switch (state)
	{
		case WorkerState.Gathering:
			StartCoroutine(GatherResources());
			break;
		case WorkerState.GoingToResource:
			if (isPlayer)
```

```
                playerUnit.MoveUnit(resourceNode.position);
                else
                enemyUnit.MoveUnit(resourceNode.position);
                break;
            case WorkerState.ReturningCargo:
                if (isPlayer)
                playerUnit.MoveUnit(commandCenter.position);
                else
                enemyUnit.MoveUnit(commandCenter.position);
                break;
            case WorkerState.Other:
                if (isPlayer)
                playerUnit.StopUnit();
                else
                enemyUnit.StopUnit();
                break;
        }
}

private void OnTriggerEnter(Collider other)
{
        if (other.gameObject.name.Contains("CommandCenter") &&
        state == WorkerState.ReturningCargo)
        {
            StartCoroutine(ReturnCargo());
            if (resourceNode)
            {
                ChangeWorkerState(WorkerState.GoingToResource);
            }
            else
            {
                ChangeWorkerState(WorkerState.Other);
            }
            return;
        }

        if (other.gameObject.layer == LayerMask.NameToLayer("Resource") &&
        state == WorkerState.GoingToResource)
        {
            ChangeWorkerState(WorkerState.Gathering);
            return;
```

```
	}
}

IEnumerator GatherResources()
{
	if (isPlayer)
		playerUnit.StopUnit();
	else
		enemyUnit.StopUnit();
	dustParticles.SetActive(true);
	PlaySound();
	yield return new WaitForSeconds(2);
	StopPlayingSound();
	cargo = resourceNode.GetComponent<Resource>()
	.GatheredResources(maxCargoAmount);
	CheckRemainingResouces();
	ChangeWorkerState(WorkerState.ReturningCargo);
	cargoGO.SetActive(true);
	dustParticles.SetActive(false);
}

void CheckRemainingResouces()
{
	if (resourceNode.GetComponent<Resource>().amount == 0)
	{
		if (!isPlayer && !EnemyManager.isRemovingResourceNode)
		{
			EnemyManager.isRemovingResourceNode = true;
			EnemyManager.instance.
			RemoveResourceNode(resourceNode.gameObject);
		}
		if (isPlayer)
		{
			if (resourceNode)
			{
				Destroy(resourceNode.gameObject);
				resourceNode = null;
			}
		}
	}
}
```

```
IEnumerator ReturnCargo()
{
	if (isPlayer)
		playerUnit.StopUnit();
	else
		enemyUnit.StopUnit();
	cargoGO.SetActive(false);
	yield return new WaitForSeconds(1);
	if (isPlayer)
		ResourceManager.instance.AddResources(cargo);
	else
		EnemyManager.instance.numberOfResources += cargo;
	cargo = 0;
}
```

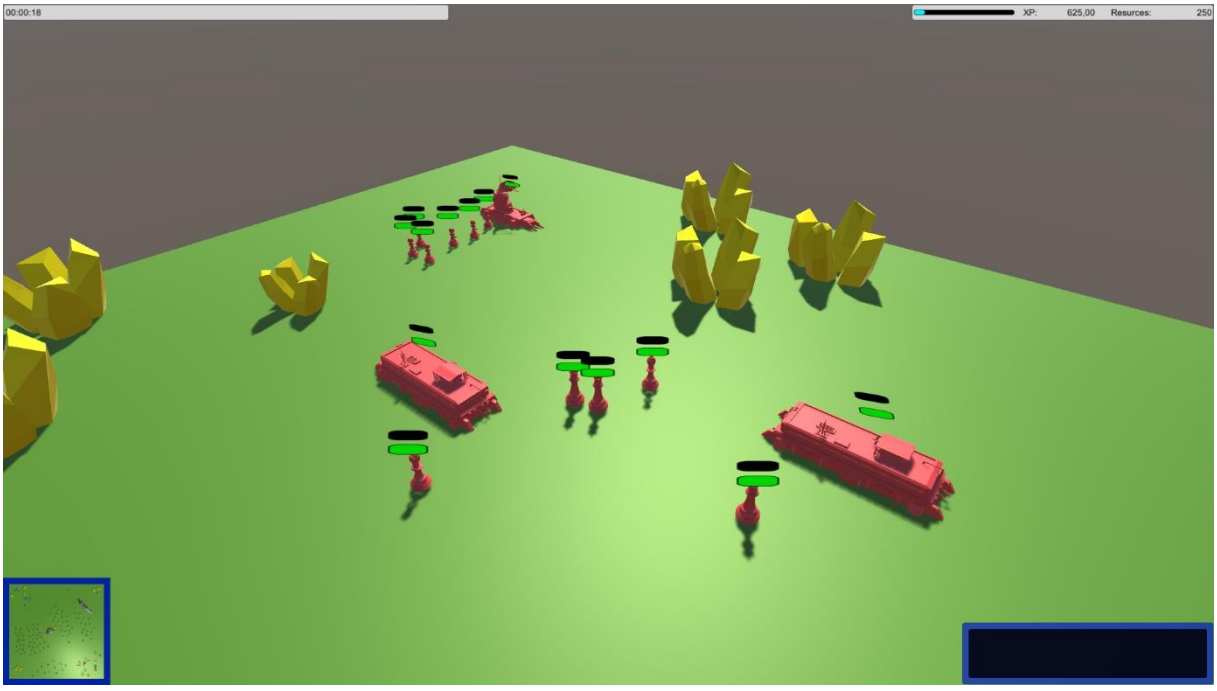


Slika 23: Resursi te radnici koji kopaju/odnose resurse (slika autora)

4.5. Neprijatelji

Za demonstracijske svrhe, napravili smo jednostavnog protivnika kojeg kontrolira računalo i protiv kojeg se igrač može boriti. Protivnik koristi isti set jedinica kao i igrač te ima jednaki zadatak kao i igrač – uništiti sve igračeve zgrade. Jedinice i zgrade neprijatelja sadrže komponente *EnemyUnit* i *EnemyBuilding* koje su slične igračevim *PlayerUnit* i *PlayerBuilding* uz manje preinake. Glavna klasa za igru računala je *EnemyManager* koja je u suštini „mozak“ računala i iz koje se zadaju naredbe zgradama i jedinicama.

Neprijatelju smo napravili tri jednostavne funkcije: kopanje resursa, staranje jedinica i slanje napada na igrača.



Slika 24: Neprijateljska baza (slika autora)

4.5.1. Kopanje resursa

EnemyManager a početku sadrži dvije liste: setResourceNodes i resourceNodes. Prva lista je definirana u Unityjevom editoru gdje smo stavili nekoliko najbližih resursnih čvorova u listu. Druga lista, koja je statička, na početku igre, točnije u Awake metodi, preuzima vrijednosti iz prve liste kako bi nam kasniji rad bio lakši. U Start metodi pozivamo metodu GatherResources. Za početak, iz varijable enemyUnits putem metode GetChild dohvaćamo prvi element koji sadrži sve radničke jedinice i pohranjujemo ga u varijablu enemyWorkers.

Zatim u foreach petlji prolazimo kroz svu djecu objekta enemyWorkers, dohvaćamo komponentu *Worker* za svakoga od njih i pozivamo metodu HandleWOrking kojoj prosljeđujemo prvi element liste resourceNodes. Također, na kraju ove metode varijablu isRemovingResurceNode postavljamo kao lažnu. Ovo nam je potrebno kada uklanjamo čvor kojeg smo „iskopali“ do kraja kako ne bi nehotice uklonili više čvorova iz liste nego je potrebno. Čvorovi koje su iskopali neprijateljski radnici uklanjaju se u metodi RemoveResourceNodes u kojima se ranije spomenuta varijabla isRemovingResourceNode postavlja na istinu, a zatim se uklanja prvi element liste resurceNodes, ponovno pozivamo metodu GatherResources koja svim neprijateljskim radnicima postavlja novi čvor kao zadani te na kraju uništavamo objekt netom „iskopanog“ resursnog čvora.

```
void GatherResources()
{
	Transform enemyWorkers = enemyUnits.GetChild(0);
	foreach (Transform worker in enemyWorkers)
	{
		Worker w = worker.GetComponent<Worker>();
		w.HandleWorking(true, resourceNodes[0]);
	}
	isRemovingResourceNode = false;
}

public void RemoveResourceNode(GameObject node)
{
	isRemovingResourceNode = true;
	resourceNodes.RemoveAt(0);
	GatherResources();
	Destroy(node);
}
```

4.5.2. Stvaranje jedinica

Računalo će stvarati jedinice ovisno o tome koliko jedinica ima pod trenutnom kontrolom. Radi demonstracije, zadano je da će računalo proizvoditi nove radnike ako ih ima manje od 5 i nove borbene jedinice ako ih ima manje 10.

Prvo se poziva metoda CheckUnitNumber koja broji jedinice pod kontrolom neprijatelja. Broj radnika provjerava tako da na objektu enemyUnits poziva metodu GetChild kojoj prosljeđuje parametar „0“ (radnici su prvi element) te zatimdohvaća parametar childCount i

pohranjuje ga u varijablu numberOfWorkers. Zatim varijablu numberOfUnits postavlja na vrijednost 0 te za svu preostalu djecu u objektu enemyUnits pozivamo metodu GetChild i njihov parametar chuldCount dodajemo varijablu numberOfUnits.

Nakon brojanja jedinica dolazi njihovo stvaranje. Za to koristimo metodu SpawnUnits. Prvo provjeravamo imamo li manje od 5 radnika te ako je to istina, dohvaćamo zgrade tipa CommandCenter tako da na objektu enemyBuildings pozivamo metodu GetChild i prosljeđujemo joj parametar „0“ jer je CommandCenter prvi element i dohvaćeni objekt spremamo u varijablu enemyCommandCenters. Zatim, za svako dijete u navedenoj varijabli radimo provjeru imamo li dovoljno resursa za izgradnju jedinice te ako imamo, na tom objektu-djetetu dohvaćamo komponentu *EnemyBuilding* i pozivamo metodu SpawnUnit kojoj prosljeđujemo worker jedinicu iz klase *UnitHandler*. Nakon izrade radnika provjeravamo imamo li manje od 10 vojnih jedinica. Ako ih je manje, dohvaćamo neprijateljske zgrade tipa Barracks. zatim u switch grananju provjeravamo slučaj za unitSpawnCount % 2. Ovime raspoređujemo stvaranje jedinica tako da ravnomjerno proizvodimo jedinice za blisku i dalekometnu borbu. Ako imamo dovoljno resursa, ponovno pozivamo metodu SpawnUnit te brojač jedinica inkementiramo.

```
void SpawnUnits()
{
	if (numberOfWorkers < 5)
	{
		Transform enemyCommandCenters = enemyBuildings.GetChild(0);
		foreach (Transform commandCenter in enemyCommandCenters)
		{
			if (numberOfResources >=
			UnitHandler.instance.worker.baseStats.cost)
			{
				commandCenter.GetComponent<EnemyBuilding>()
				.SpawnUnit(UnitHandler.instance.worker);
			}
		}
	}
	if (numberOfUnits < 10)
```

```
	{
		Transform enemyBarracks = enemyBuildings.GetChild(1);
		foreach (Transform barracks in enemyBarracks)
		{
			switch (unitSpawningCount % 2)
			{
			case 0:
				if (numberOfResources >=
				UnitHandler.instance.melee.baseStats.cost)
				{
					unitSpawningCount++;
					barracks.GetComponent<EnemyBuilding>()
					.SpawnUnit(UnitHandler.instance.melee);
				}
				break;
			case 1:
				if (numberOfResources >=
				UnitHandler.instance.ranged.baseStats.cost)
				{
					unitSpawningCount++;
					barracks.GetComponent<EnemyBuilding>()
					.SpawnUnit(UnitHandler.instance.ranged);
				}
				break;
			}
		}
	}
}

void CheckUnitNumber()
{
```

```
        numberOfWorkers = enemyUnits.GetChild(0).childCount;
        numberOfUnits = 0;
        for (int i = 1; i <= 4; i++)
        {
                numberOfUnits += enemyUnits.GetChild(i).childCount;
        }
}
```

4.5.3. Slanje napada na igrača

Kako bi računalo poslalo napad, moraju se zadovoljiti određeni kriteriji koje provjeravamo u metodi CanAttackPlayer. Prvo provjeravamo imamo li manje od 10 jedinica te ako je to istina, metoda odmah vraća vrijednost false kako se napad ne bi pokrenuo. U suprotnom, za svaku zgradu pod kontrolom računala iz klase *Physics* pozivamo metodu CheckSphere i prosljeđujemo joj sljedeće parametre: building.position (pozicija trenutne zgrade za koju radimo provjeru), 20 (radijus u kojem radimo provjeru) i 8 (indeks sloja za igračeve jedinice). Ova metoda provjerit će nalazi li se u blizini neprijateljske zgrade igračeva jedinica te ako se nalazi, metoda CanAttackPlayer će odmah vratiti false i prekinuti daljnju provjeru. U slučaju da nisu uočene igračeve jedinice u blizini neprijateljskih zgrada, metoda vraća vrijednost true i napad računala će početi.

Računalo će narediti napad pozivom metode AttackPlayer. Za svaki tip jedinice dohvatit ćemo svako dijete njihovog roditeljskog objekta te svakom djetetu dohvatiti komponentu EnemyUnit i pozvati metodu MoveUnit kojem ćemo proslijediti rezultat metode GetPlayerBaseLocation.

U metodi GetPlayerBaseLocation iz klase *PlayerManager* dohvaćamo objekt playerBuildings i pohranjujemo ga u provremenu varijablu, nakon toga, prolazimo kroz svu djecu objekta playerBuildings (sadrži sve tipove zgrada) i za svaki tip zgrade pokušamo dohvatiti dijete čiju poziciju odmah vraćamo. Ovom metodom zapravo dohvaćamo poziciju prve neprijateljske zgrade u hijerarhiji te napad šaljemo na tu poziciju.

```
bool CanAttackPlayer()
{
        if (numberOfUnits < 10)
                return false;
        for (int i = 0; i < 2; i++)
```

```
	{
		foreach (Transform building in enemyBuildings.GetChild(i))
		{
			if (Physics.CheckSphere(building.position, 20, 8))
				return false;
		}
	}
	return true;
}

void AttackPlayer()
{
	for (int i = 1; i < 3; i++)
	{
		foreach (Transform unit in enemyUnits.GetChild(i))
		{
			unit.GetComponent<EnemyUnit>()
			.MoveUnit(GetPlayerBaseLocation());
		}
	}
}

Vector3 GetPlayerBaseLocation()
{
	Transform pb = PlayerManager.instance.playerBuildings;
	for (int i = 0; i < 2; i++)
	{
		foreach (Transform child in pb.GetChild(i))
		{
			return child.position;
		}
	}
	return approxEnemyBase;
}
```

4.6. Audiovizualni elementi

Zvučni i grafički elementi važan su dio svake strateške igre jer nam daju važne informacije u stanju na terenu. Zvukovi mogu pravovremeno obavijestiti igrača o važnim događanjima, dok kvalitetni vizualni elementi mogu dati precizne informacije o stanju jedinica [36] [37].

4.6.1.Zvukovi

U ovoj igri, dodani su zvukovi na važnije elemente i događaje u igri koji će igraču olakšati igračko iskustvo. Zvučni zapisi pokrenut će se prilikom konstrukcije zgrade, stvaranja i umiranja jedinca, borbe, kopanja resursa te na kraju igre. Unutar igre dodan je objekt AudioMixerController koji upravlja svim zvukovima u igri. Objekti koji emitiraju zvuk imaju komponentu AudioSource na kojoj smo mijenjali sljedeća važnija svojstva:

- Audio clip – zvuk koji će objekt proizvesti (.mp3 datoteka)
- Output – objekt preko kojeg puštemo zvuk. U ovom slučaju to je naš glavni AudioMixer
- Play on awake – puštanje zvuka čim objekt postane aktivan
- Loop – ponavljanje zvuka
- Volume – glasnoća zvuka
- Pitch – frekvencija zvuka
- Spatial blend – efekt zvuka u 3D prostoru

Kako bi pustili zvuk, potrebno je dohvatiti komponentu AudioSource, a zatim pozvati metodu Play. Zvuk možemo zaustaviti pozivom metode Stop.

```
audioSource = GetComponent<AudioSource>();

void PlaySound()
{
      audioSource.Play();
}

void StopPlayingSound()
{
      audioSource.Stop();
}
```

U sljedećoj tablici, nalazi se popis svih zvukova korištenih u igri. Korišteni zvukovi su formata .mp3. Svi zvukovi preuzeti su sa stranice ZapSplat.com.

Tablica 3: Zvukovi korišteni u igri

Originalni naziv	Naziv u editoru	Trajanje	Korištenje
ambience_construction_site_large_ground_zero_nyc_001	BuildingConstruction	0:58	Izgradnja zgrade (dok se koristi nacrt)
smartsound_HUMAN_CROWD_Military_Shout_German_Attention_02 (izrezan je samo dio zvuka koji je potreban za igru)	UnitSpawn	0:08 (0:01)	Jedinica je proizvedena
zapsplat_explosion_loud_punchy_powerful_with_fire_63285	BuildingExplosion	0:06	Uništenje zgrade
zapsplat_fantasy_magic_magical_ping_79214	Healing	0:03	Liječenje jedinice
zapsplat_foley_coconut_fall_from_tree_hit_ground_thud_003_73453	CannonballLaunch	0:01	Napad jedinice tipa Siege
zapsplat_horror_footstep_massive_distant_heavy_dinasour_monster_giant_003_13402	CannonballHit	0:01	Pogodak projektila jedinice tipa Siege
zapsplat_human_male_vocalisation_hit_pain_short_001_66499	Death	0:01	Smrt jedinice
zapsplat_impact_sword_swipe_into_large_wood_hit_001_43691	SwordHit	0:01	Napad jedinice tipa Melee
zapsplat_impacts_metal_portable_goal_frame_drop_concrete_002_83646	PickaxeHit	0:01	Kopanje resursa
zapsplat_multimedia_game_sound_retro_musical_fun_finish_end_musical_tone_78365	GameWon	0:02	Kraj igre - pobjeda

zapsplat_multimedia_game_sound_spooky_haunted_fail_negative_lose_life_78390	GameLost	0:04	Kraj igre - poraz
zapsplat_warfare_bow_arrow_fire_shoot_003_40988	ArrowFired	0:01	Napad jedinice tipa Ranged

4.6.2.Slike

Slike korištene u ovoj igri upotrijebljene su kako bi igrač imao jednostavnije i razumljivije korisničko sučelje. Simboli jedinica postavljeni su na gumbe za kreiranje novih jedinica koji se pojavljuju u donjem desnom kutu prilikom odabira zgrade. Slike koje su korištene su .png formata. Sve slike preuzete su sa stranice Icons-For-Free.com.

Tablica 4: Slike korištene u igri

Naziv	Korištenje
bishop	Gumb za kreiranje jedinice tipa Ranged
king	Gumb za kreiranje jedinice tipa Spellcaster
pawn	Gumb za kreiranje jedinice tipa Worker
queen	Gumb za kreiranje jedinice tipa Melee
rook	Gumb za kreiranje jedinice tipa Siege

4.6.3.3D modeli

Budući da je igra izrađena u tri dimenzije, tako smo za zgrade, jedinice, resurse i okoliš koristili 3D modele. Modeli su dodani objektima kao objekt-dijete. Nakon dodavanja modela, za svaku jedinicu podešeni su sudarači kako bi odgovarali modelu koji predstavlja jedinicu. Na modele su dodani materijali radi većeg korisničkog iskustva, ali i radi lakšeg snalaženja u igri. Tako su igračeve jedinice i zgrade plave boje, neprijateljske jedinice i zgrade crvene boje, a resursi zlatne. Okoliš je u kombinacijama zelene, sive i smeđe boje, ovisno o objektima koje

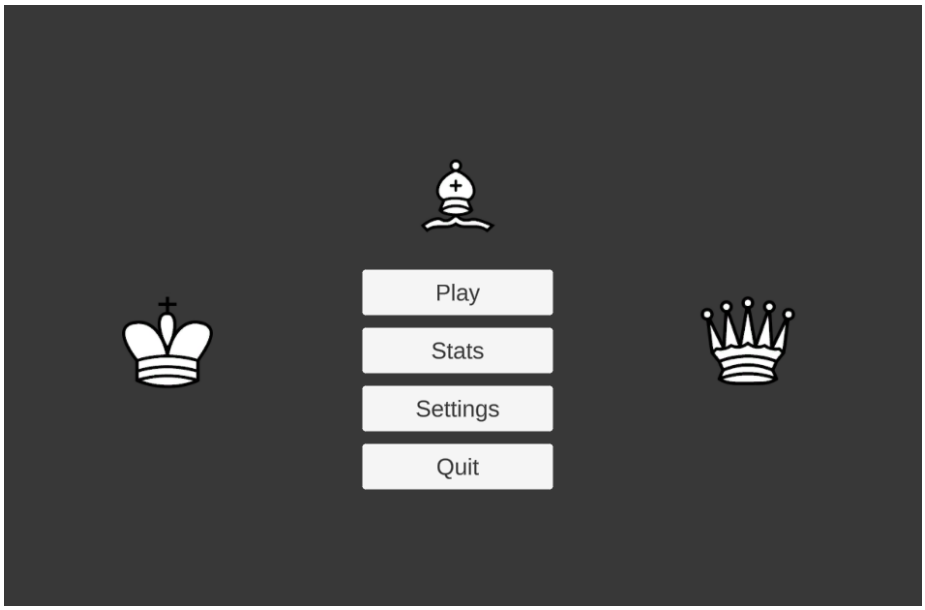
predstavljaju. Ovisno o modelu, formati su .3ds, .fbx i .obj. Modeli su preuzimani sa stranica Turbosquid.com, Shetchfab.com te Static.free3d.com.

Tablica 5: 3D modeli korišteni u igri

Izvor	Naziv	Format	Korištenje
Turbosquid.com	Bishop	.3ds	Jedinica tipa Ranged
Turbosquid.com	Dame	.3ds	Jedinica tipa Melee
Turbosquid.com	King	.3ds	Jedinica tipa Spellcaster
Sketchfab.com	krystal	.fbx	Resurs
Sketchfab.com	lowpolytree	.obj	Stablo
Turbosquid.com	pawn	.3ds	Jedinica tipa Worker
Turbosquid.com	Radar_	.obj	Zgrada tipa CommandCenter
Turbosquid.com	Rock_1	.obj	Kamen
Turbosquid.com	rook	.3ds	Jedinica tipa Siege
Turbosquid.com	service_building_	.obj	Zgrada tipa Barracks
Turbosquid.com	SmallArch_Obj	.obj	Kamen
Turbosquid.com	StoneArchLarge_Obj	.obj	Kamen
Static.free3d.com	tree	.fbx	Stablo

4.7. Izbornik

Prilikom pokretanja igre, otvara nam se glavni izbornik. Iz glavnog izbornika okrećemo novu igru, mijenjamo postavke igre, pratimo statistike dosadašnjih igara te gasimo aplikaciju. Izbornik se sastoji od 4 gumba: „Play“, „Stats“, „Settings“ i „Quit“.



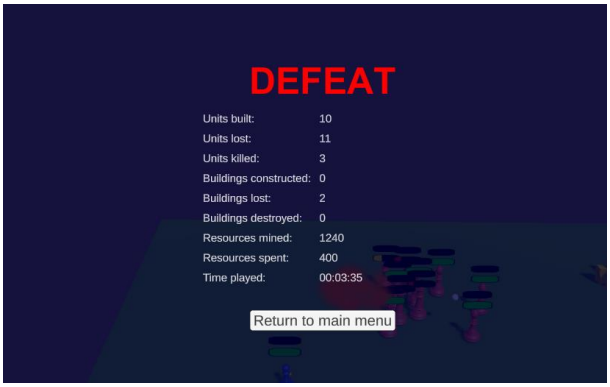
Slika 25: Glavni izbornik (slika autora)

4.7.1. Play

Pritiskom na gumb „Play“ zatvaramo scenu glavnog izbornika i otvaramo scenu igre. U samoj igri, u četiri kuta zaslona imamo četiri različita prozora. U gornjem lijevom kutu nalazi se štoperica koja prikazuje koliko vremena je proteklo od početka igre. Također, u nastavku prozora ispisuju se poruke o odavanju jedinica u red čekanja, nedovoljnoj količini resursa za izradu zgrada ili jedinica, neispravna pozicija za izgradnju zgrade i slično. U gornjem desnom kutu nalazi se prozor s trenutnim stanjem resursa kojima igrač može raspolagati. U donjem lijevom kutu nalazi se minimapa koja iz ptičje perspektiva prikazuje trenutno stanje na mapi. U donjem desnom kutu nalazi se prozor u kojem generiraju akcijski gumbi. Za zgrade će to biti jedinice koje se mogu kreirati, za radničke jedinice zgrade koje mogu izgraditi i slično. Igrač u bilo kojem trenutku igre može pauzirati igru pritiskom na tipku „P“ ili tipku „Esc“. Igru može nastaviti pritiskom na iste te tipke ili gumb „Resume“. Može se vratiti i u glavni izbornik pritiskom na gumb „Return to main menu“. Važno je napomenuti da se igra tada računa kao poraz koji ulazi u ukupnu statistiku igara. Po završetku igre, zaustavlja se vrijeme igre, pojavljuje se

prozirno-plavi zaslon s porukom o rezultatu igre, statistikom igre te gumbom za povratak u glavni izbornik.

Slika 26: Zaslon prilikom pobjede igrača (slika autora)



Slika 27: Zaslon prilikom poraza igrača (slika autora)

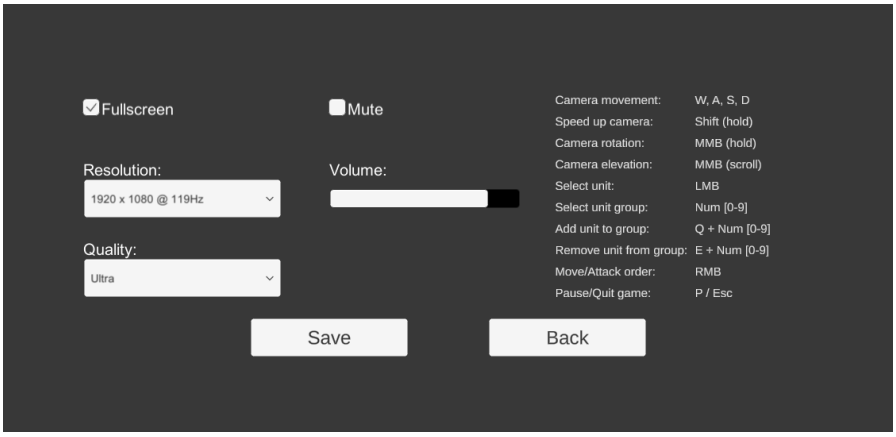
4.7.2.Stats

Pritiskom na gumb „Stats“ otvara nam se prikaz ukupnih statistika igrača. Ovdje igrač može vidjeti ukupne statistike svih do sad odigranih igara. Statistike sadrže: omjer pobjeda i odigranih igara, broj kreiranih jedinica, broj izgubljenih jedinica, broj uništenih jedinica, broj konstruiranih zgrada, broj izgubljenih zgrada, broj uništenih zgrada, količinu iskopanih resursa, količinu potrošenih resursa te ukupno vrijeme igranja. Ispod statistika, nalaze se gumbi „Reset“ za poništavanje svih statistika te gumb „Back“ za povratak u glavni izbornik. Sve postavke spremljene su u datoteku gameStats.json u aplikacijskom direktoriju igre.

Slika 28: Izbornik "Stats" (slika autora)

4.7.3.Settings

Gumbom „Settings“ otvaramo postavke igre. Postavke su podijeljene u tri kategorije. U lijevom stupcu nalaze se grafičke postavke, u sredini postavke zvuka, a s desne strane nalazi se popis kontrola u igri. Ispod postavki nalaze se gumbi „Save“ koji sprema trenutne postavke u datoteku settings.json u aplikacijskom direktoriju igre kako bi igrača sljedeći put dočekale željene postavke na računalu te gumb „Back“ za povratak u glavni izbornik.



Slika 29: Izbornik "Settings" (slika autora)

4.7.3.1. Grafičke postavke

Igrač ovdje može mijenjati postavke prikaza. Prva opcija je Fullscreen (eng. cijeli zalson). Ako je ova postavka upaljena, igra će biti prikazana preko cijelog zaslona, u suprotnom će igra biti u „prozorskom modu“ i neće biti prikazana preko cijelog zaslona. Sljedeća postavka je postavka rezolucije gdje se u padajućem izborniku prikazuju sve dostupne rezolucije za monitor na kojemu igrač trenutno igra. Rezolucije se nalaze u formatu „širina x visina @ frekvencija osvježavanja“. Preporuča se korištenje većih rezolucija radi boljeg korisničkog iskustva. Posljednja grafička postavka je postavka kvalitete u kojoj igrač ima nekoliko mogućnosti: Very low (eng. jako nisko), Low (eng. nisko), Medium (eng. srednje), High (eng. Visoko), Very High (eng. jako visoko) i Ultra. Na manjim postavkama kvaliteta slike i svjetlosnih efekata bit će niža (npr. na jako niskoj kvaliteti objekti neće bacati sjenu) dok će na većoj kvaliteti grafičke postavke biti bolje, ali u nekim situacijama to znači zahtjevnije korištenje hardverskih komponenti računala (na „Ultra“ kvaliteti uvijek će se vidjeti sjene koje bacaju objekti i slično).

4.7.3.2. Postavke zvuka

Zvučne postavke nude opcije „Mute“ za potpuno gašenje zvuka ako je postavka uključena, odnosno korištenje zvuka ako je isključena. Ako je postavka isključena koristi se zadana vrijednost postavke „Volume“. Postavka „Volume“ je klizač koji će pojačati ili smanjiti zvuk ovisno o tome koliko korisnik to tada pomicanjem klizača (Skroz lijevo za minimalnu glasnoću, skroz desno za maksimalnu).

4.7.3.3. Kontrole

Posljednji stupac prikazuje sve moguće kontrole u igri kojima se igrač može koristiti. Kontrole će također biti prikazane i u tablici dolje.

Tablica 6: Popis kontrola u igri:

Akcija	Kontrole tipke
Kretanje kamere (eng. Camera movement)	W, A, S, D
Ubrzanje kamere (eng. Speed up camera)	Shift (dok je tipka pritisnuta)
Rotacija kamere (eng. Camera rotation)	MMB (držanje koračića miša)
Visina kamere (eng. Camera elevation)	MMB (pomicanje kotačića kamere
Odabir jedinice (eng. Select unit)	LMB (lijevi gumb miša)
Odaberi grupu jedinica (eng. Select unit group)	Num [0-9] (brojčane tipke)
Dodaj u grupu jedinica (eng. Add unit to group)	Q + Num [0-9]
Ukloni iz grupe jedinica (eng. Remove unit from group)	E + Num [0-9]
Kreni/Napadni naredba (eng. Move/Attack order)	RMB (desni gumb miša)
Pauziraj igru/Izađi (eng. Pause/Quit)	P / Esc

4.7.4.Quit

Pritiskom na gumb „Quit“ zaustavljamo rad aplikacije.

5. Zaključak

U ovom radu naveli smo i opisali osnovne mehanike video igara strategije u stvarnom vremenu te napravili praktični primjer takve igre u programskom alatu Unity. Povećanjem tržišta video igara povećat će se i potreba za novim igrama kao i novim i zanimljivim mehanikama. Upravo zbog toga, potrebno je pratiti nove inovacije i standarde u igrama ovog žanra, ali i drugim žanrovima, kako bi se u budućnosti igračima pružilo novo i zanimljivo iskustvo igranja.

Također, dobro je igru napraviti tako da se na postojeću igru lako mogu dodati novi elementi, mehanike, komponente, jedinice i slično te pružiti mogućnost samim igračima za dodavanje vlastitih elemenata.

Prema predviđanjima mnogih stručnjaka, industrija video igara će i dalje rasti. Svake godine na tržištu se pojavljuju novi naslovi što samo pokazuje veliko zanimanje publike za video igre, a posebice strategije. Tome u prilog idu i česti turniri (npr. Starcraft) koji i dalje privlače veliko zanimanje medija.

Popis literature

[1] „Strategy video game“, Wikipedia. 10. lipanj 2022. Pristupljeno: 12. srpanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Strategy_video_game&oldid=1092541462

[2] „Turn-based strategy“, Wikipedia. 22. kolovoz 2021. Pristupljeno: 12. srpanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Turn-based_strategy&oldid=1040099408

[3] „Real-time strategy“, Wikipedia. 09. lipanj 2022. Pristupljeno: 12. srpanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Real-time_strategy&oldid=1092328379

[4] „Video game industry - Statistics & Facts“, Statista, 19. studeni 2021. https://www.statista.com/topics/868/video-games/ (pristupljeno 18. srpanj 2022.)

[5] „Video Game Market Size & Share Growth Report, 2030“. https://www.grandviewresearch.com/industry-analysis/video-game-market (pristupljeno 18. srpanj 2022.)

[6] „Ultimate List of Different Types of Video Games | 49 Genres & Subcategories“, iD Tech. https://www.idtech.com/blog/different-types-of-video-game-genres (pristupljeno 18. srpanj 2022.).

[7] „Unity Technologies“, Wikipedia. 16. srpanj 2022. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Unity_Technologies&oldid=1098464994

[8] „Unity (game engine)“, Wikipedia. 16. srpanj 2022. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1098555325

[9] Jon Brodkin, „How Unity3D Became a Game-Development Beast“, Dice Insights, 03. lipanj 2013. https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ (pristupljeno 18. srpanj 2022.)

[10] Victor G Brusca , Advanced Unity Game Development. Apress , 2021

[11] Nicolas Alejandro Borromeo, Unity 2021 Game Development. Packt Publishing, 2021

[12] T. Villar, „What Are Real-Time Strategy Games?“, MUO, 02. lipanj 2022. https://www.makeuseof.com/what-are-real-time-strategy-games-rts-games/ (pristupljeno 18. srpanj 2022.).

[13] „GameSpot Presents: A History of Real-Time Strategy Games“, 27. travanj 2011.

https://web.archive.org/web/20110427052656/http://gamespot.com/gamespot/features/all/real_time/ (pristupljeno 18. srpanj 2022.).

[14] Unity RTS - Camera Tutorial. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=pxS14VJ_eXQ&list=PLROHqCOfMPkNvsMiLFNol0NqRjl9GzwFW

[15] How to make a Minimap in Unity. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=28JTTXqMvOU

[16] „How to rotate my camera? - Unity Answers“. https://answers.unity.com/questions/1179680/how-to-rotate-my-camera.html (pristupljeno 18. srpanj 2022.)

[17] Unity RTS - Box Selection Tutorial. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=OL1QgwaDsqo&list=PLROHqCOfMPkNvsMiLFNol0NqRjl9GzwFW&index=3

[18] Control Units and Give Orders! (Unity RTS Tutorial). Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=mCIkCXz9mxI

[19] Unity RTS Movement Tutorial - How to Move Characters in Unity by Clicking | MOBA Control Tutorial. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=LoKNYlWWeSM

[20] How to use Unity NavMesh Pathfinding! (Unity Tutorial). Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=atCOd4o7tG4

[21] How to BUILD and LAUNCH an RTS Game in UNITY - ep2. RTS style UNIT SELECTION ft. Singletons. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=l5RpXGYyjK8&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=4

[22] How to BUILD and LAUNCH an RTS Game in UNITY - ep2.5 RTS style DRAG SELECTION ft Singletons & OnGUI. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=mNP80694C-o&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=5

[23] How to BUILD and LAUNCH an RTS Game in UNITY - ep3. Moving RTS Units ft NavMeshAgent. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=m9yk7sJjn7c&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=6

[24] How to BUILD and LAUNCH an RTS Game in UNITY - ep5. Enemy AI (Part1: Aggro) ft NavMeshAgent. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na:

https://www.youtube.com/watch?v=l0GhE9N0o9c&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=8

[25] How to BUILD and LAUNCH an RTS Game in UNITY - ep6. Enemy AI Part 2: COMBAT. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=BZMloxW7AbA&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=9

[26] How to BUILD and LAUNCH an RTS Game in UNITY - ep4 Customizing Editor & Units ft Scriptable Objects. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=QqQLgP1gvWw&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=7

[27] How to BUILD and LAUNCH an RTS Game in UNITY - ep8. Adding RTS Buildings in Unity. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=W5BTOemQsPI&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=11

[28] How to BUILD and LAUNCH an RTS Game in UNITY - ep9. Spawning Units ft. Inheritance (Part 1). Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=m9GebmmQMFU&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=12

[29] How to BUILD and LAUNCH an RTS Game in UNITY - ep9.5 SPAWNING RTS Units (Part 2) Ft UI HUD Creation. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=8zsrlDEJa7c&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=13

[30] How to BUILD and LAUNCH an RTS Game in UNITY - ep10 SPAWNING RTS Units (Part 3) Ft Coroutines. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=V6FHUDvd0Io&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=14

[31] How to BUILD and LAUNCH an RTS Game in UNITY - ep11. SET SPAWN LOCATION - ft nothing special. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=qz-ajVSJ7Y8&list=PLQPhaRCbpx5U0kcamApy727XC0v1bF0PK&index=15

[32] Unity RTS - Building Placement Tutorial. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=Omu0A4Mk5pE&list=PLROHqCOfMPkNvsMiLFNol0NqRjl9GzwFW&index=8

[33] Unity Tutorial: Building Placement. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=OuqThz4Zc9c

[34] Making Particles in Unity 2018.4! (Beginner Friendly Tutorial). Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=oPUxv-zClNA

[35] „Get numeric key - Unity Answers“. https://answers.unity.com/questions/420324/get-numeric-key.html (pristupljeno 18. srpanj 2022.).

[36] Simple Sound Manager (Unity Tutorial). Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=QL29aTa7J5Q

[37] Introduction to AUDIO in Unity. Pristupljeno: 18. srpanj 2022. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=6OT43pvUyfY

[38] „A Brief History Of Strategy Games | Starborne“, 07. prosinac 2020. https://starborne.com/post/a-brief-history-of-strategy-games (pristupljeno 19. srpanj 2022.)

[39] Fraser Brown, „The history of the strategy game“, PC Gamer, 24. prosinac 2018. Pristupljeno: 19. srpanj 2022. [Na internetu]. Dostupno na: https://www.pcgamer.com/the-history-of-the-strategy-game/

Popis slika

Popis tablica