Izrada strateške igre u stvarnom vremenu u programskom alatu Unity

Baban, Deni

Master's thesis / Diplomski rad

2021

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:211:265523

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Deni Baban

IZRADA STRATEŠKE IGRE U STVARNOM VREMENU U PROGRAMSKOM ALATU UNITY

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Deni Baban

Matični broj: 44023/15–R

Studij: Informacijski sustavi

IZRADA STRATEŠKE IGRE U STVARNOM VREMENU U PROGRAMSKOM ALATU UNITY

DIPLOMSKI RAD

Mentor/Mentorica:

Dr. sc. Mladen Konecki

Varaždin, rujan 2021

Deni Baban

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

Ovaj rad se bavi izradom strateške igre u stvarnom vremenu (eng. Real Time Strategy) pomoću programskog alata Unity. Igra je zamišljena da se igra protiv varijabilnog broja agenata, tj. umjetne inteligencije sa osnovnim znanjem izgradnje i kontrole jedinica u igri. Igra sadrži sve klasične elemente strateške igre u stvarnom vremenu poput izgradnje građevina baze, treniranja jedinica za borbu, korištenje vještina ili magije, prikupljanje prirodnih resursa poput drva i zlata te mnoge druge. Mape na kojima se igra mogu biti nasumično generirane pomoću algoritma, a cilj igre je kompletno uništiti sve protivničke baze te zaštititi svoju.

Ključne riječi: Unity, strateška igra u stvarnom vremenu, algoritmi, programiranje, računalne igre

Sadržaj

1. Uvod

Diplomski rad se bavi izradom strateške igre u stvarnom vremenu između igrača te nekoliko autonomnih agenata koji također igraju protiv svih ostalih, a moguće je odabrati i borbu u timovima od nekoliko igrača. Razvoj ovakvoga tipa igre se jako temelji na sposobnostima korištenja svih aspekata objektno orijentiranog programiranja, kreacije raznih algoritama za neke od mnogih funkcionalnosti igre te kreativnosti prilikom grafičkog dizajniranja modela, okoliša i 2D grafike. Igra sadrži glavni izbornik iz kojega se pokreće borba sa odabranim parametrima i same mape na kojoj će se borba odvijati. Igrači počinju borbu samo sa nekoliko jedinica radnika te glavnom zgradom, a za svakog pojedinog igrača završava onda kada mu se unište sve zgrade i sve jedinice radnika tako da više ne može praviti nove građevine ili jedinice. Borba se također može igrati u timovima po nekoliko igrača zajedno, a pobjednik je onaj tim ili igrač koji ostane zadnji na mapi.

Inspiracija za ovaj rad su bile igre poput Warcraft 3 ili Starcraft 2 koje su jedne od vrlo popularnih primjera strateških igara sa svim klasničnim elementima koje će ovaj rad koristiti. Budući da sam proveo dosta vremena uživajući u navedenim igrama, ova tema diplomskog rada mi se činila zanimljivim i zabavnim odabirom, pogotovo zato što već imam dosta iskustva sa Unity alatom te razvojem igara u njemu.

2. Metode i tehnike rada

Za izradu ovoga rada je korišten popularni alat za izradu računalnih igara Unity. Prilikom izrade svojeg završnog rada sam također koristio Unity pa sam već dosta upoznat sa njegovim korištenjem i mogućnostima koje nudi. Koristio se programski jezik C# za pisanje Unity skripti uz pomoć Microsoftove razvojne okoline Visual Studio. Unity pruža vrlo jednostavno sučelje za rad i povezivanja skripti sa objektima unutar igre koje se u većini slučajeva može obaviti jednostavnim povlačenjem i spuštanjem drugih objekata, komponenata ili skriptni na određena mjesta u Unity Inspector prozoru za pojedini objekt.

Glavna dokumentacija za izradu ovoga rada je bila službena Unity dokumentacija sa njihovih web stranica, pojedine stranice koje razrađuju neke od specifičnih algoritama korištenih za izradu ovoga rada te materijali sa korisnim uzorcima dizajna. Iako imam već dosta iskustva sa nekim od osnovih mehanika igara, nisam se nikada susreo sa primjerice sakrivanjem udaljenih neprijateljskih jedinica (eng. Fog of War) što sam prije izrade morao detaljno proučiti i pronaći na internetu.

Izradu ovoga projekta sam podijelio u nekoliko glavnih faza koje se fokusiraju na neke od većih funkcionalnosti igre kao što su selekcija i kretanja jedinica, korištenje magije, izgradnja građevina i jedinica, umjetna inteligencija i slično. Jedan od glavnih djelova će biti upravljanje stanjem kursora koji obuhvaća nekoliko drugih funkcionalnosti, spaja ih i služi kao prijelaz između njih na način da lijevi klik miša može raditi puno različitih aktivnosti ovisići u kojem se trenutno stanju nalazi (Odabir jedinica, selekcija podječja ili jedinice za korištenje magije, odabir mjesta gradnje zgrade...). Nakon što se igra mogla normalno igrati, slijedi poboljšanje korisničkog sučelja sa grafičkim elementima i ikonama, funkcionalnošću prikaza jedinica na minimapi, implementacija Fog of War sustava, a nakon toga izrada glavnog izbornika koji bi pokretao igru.

Nakon što je igra bila napravljena, još su mi samo bili potrebni 3D modeli kao i njihove animacije kretanja, napadanja i korištenja vještina. Modele sam skidao sa Unity Store-a, dok sam razne 2D grafičke elemente i ikone napravio pomoću GIMP alata ili preuzeo sa interneta. Finalni korak je testiranje, poliranje i poboljšavanje igre na konačnu fazu prije nego se krene pisati dokumentacija.

3. Unity

Ovo poglavlje se bavi opisivanjem samoga alata u kojemu će se igra izrađivati. Unity je alat razvijen od strane kompanije Unity Technologies i napravljen je 2005. godine. Dostupan je na Windows operacijskom sustavu, MacOS i Linuxu, a aplikacije napravljene u njemu se mogu izvesti i na Android iOS, PlayStation, razne VR platforme i mnoge druge. Iako je primarno napravljen za izradu 3D aplikacija, Unity također podržava dvodimenzionalne igre. Unity alat ima besplatnu verziju kao i verziju koja se plaća i obavezna je ukoliko kreator aplikacija kroz njih zaradi više od određene svote novca tjekom jedne godine. Za ovaj rad se koristi besplatna verzija alata što se može prepoznati po Unity logu prilikom paljenja igre koji se automatski dodaje na sve aplikacije u besplatnoj verziji (Unity, Wikipedia).

Programski jezik u kojemu se pišu Unity skripte je C# za kojega je također potrebno imati Microsoft Visual Studio, dok se prije par godina moglo koristiti ugrađeno MonoDevelop okruženje. Unity u sebi sadrži PhysiX Engine koji služi za upravljanje fizikom u igrama, iako se fizika baš toliko i ne koristi u strateškim igrama, PhysiX može poslužiti za stvaranje raznih vizualnih efekata kao sjene ili odrazi. Novije Unity verzije su uvele Shader Graph, tj. grafičko sučelje za pisanje shader-a koji se onda mogu koristiti u igri i dio je Universal Render Pipeline-a Unity Engine-a. Uvoz vanjskih resusra poput skripti, 3D modela, zvuka ili grafike je vrlo jednostavno te se može konvertirati u druge formate kroz sam alat. Unity Store više nije integriran u sam alat već mu se pristupa isključivo putem web stranica, a popis kupljenih stvari se onda može vidjeti kroz Unity Import Manager u alatu. Dokumentacija za korištenje Unity alata, kao i za pisanje skripti, je dosta kvalitetno napisana te se nalazi na službenim Unity stranicama kao i unutar samoga alata.

Također je bitno napomenuti da sam ovaj rad prvobitno zamislio kao multiplayer igru sa mogućnosti povezivanja do 8 igrača, ali Unity sustav umrežavanja (UNet) je izgubio podršku još 2019. godine, dok se ove 2021. godine planira komplentno ukloniti. Postoje razni sustavi razvijeni od strane ostalih Unity korisnika kao zamjena za UNet (Mirror, Photon, DarkRift 2), no ipak sam odlučio napraviti igru sa autonomnim agentima nego koristiti nepoznate okvire za umrežavanje.

4. Cilj i terminologija igre

Kao i u mnogim drugim RTS igrama, cilj je eliminirati sve protivničke igrače iz igre na način da im se unište sve središne građevine kao što često znaju biti dvorac, vojna sjedišta i slično te uništiti sve jedinice sa sposobnostima izgradnje novih građevina što ostavlja igrača u stanju u kojem ne može sagraditi nove zgrade jer nema jedinica za izgradnju niti napraviti nove jedinice koje bi to onda mogle raditi. Kako bi zaštitili igru od mogućnosti beskonačnog igranja, uvode se ograničeni resursi po mapi kojih će u određenom trenutnu sigurno ponestati te prisiliti kraj. Igra koristi neke od standardnih termina korištenim u RTS ili sličnim žanrovima igre koji su objašnjeni u nastavku:

- Jedinica (eng. Unit) – Glavna stavka igre koja predstavlja zgrade i vojsku igrača.
- Životni poeni – Resurs svake jedinice koji predstavlja koliko štete mogu primiti prije nego se uklone iz igre. Pasivno se regenerira kroz vrijeme.
- Mana – Resurs svake pojedinačne jedinice koji često služi za plaćanje prilikom korištenja vještina. Regenerira se tjekom vremena slično kao i životni poeni jedinice.
- Vještina (eng. Skill) – Razne mogućnosti jedinica koje se mogu koristiti plaćanjem svoje cijene u resursima ili korištenjem mana poena.
- AoE (eng. Area of Effect) – Područje koje će biti zahvaćeno nekim od efekata.
- FoW (eng. Fog of War) – Efekt sakrivanja dijelova mape koji su izvan vidokruga vlastitih i prijateljskih jedinica
- Upkeep – Količina hrane potrebna za održavanje i treniranje novih jedinica. Jedinice troše hranu, dok farme i glavna zgrada generiraju hranu.
- Cooldown – Vrijeme koje je potrebno za ponovno korištenje pojedine vještine nakon što je ona jednom aktivirana. '

5. Kursor

Glavni način za izdavanje naredbi svojim jedinicama kao i kontrole kamere ili nekih djelova korisničkog sučelja je pomoću klikova miša te njegovog pomicanja. Pozicija kursora na ekranu kao i njegovo stanje određuje kako će se pojedini djelovi igre ponašati te koje akcije se provode prilikom lijevoga ili desnoga klika miša. Kursor se može pozicionirati na rubove ili u kutove ekrana kako bi se pomakla kamera u odabranom smjeru ili u dva smjera, tj. dijagonalno. Također je bitno dali je kursor pozicioniran na nekom od elemenata korisničkog sučelja umjesto prostora igre radi interakcije sa raznim funkcionalnostima korisničkog sučelja ili gumbima. Kursor također može promjeniti svoju ikonu na običan pokazivač ili metu za ciljanje. Neke od dodatnih funkcionalnosti kursora su isticanje obruba (*eng. Highlight*) jedinice preko koje se trenutno nalazi, isticanje prostora koji će biti obuhvaćen nekim od vještina ili prostora na kojemu će se sagraditi nova građevina. Mogućnosti vezane za elemente korisničkog sučelja će biti opisane u poglavljima vezanim za njih, dok sljedeće podpoglavlje opisuje sve ostale radnje moguće unutar prostora igre. Kamera u svim stanjima ima mogućnost pomicanja kamere prilikom pomicanja kursora na rubove ekrana, čiji kod se može vidjeti u nastavku. Važno je za napomenuti da se vektor smjera mora normalizirati zbog slučaja kada se kursor nalazi u jednom od kuteva i daje kameri brzinu u dva smjera (npr. gore i desno) koja će zbog zbrajanja vektora imati vrijednost $\sqrt{2}$ umjesto 1.

```
public float cameraMovementSpeed = 10f;
public int cameraEdgePanPixels = 40;

void CameraEdgePan()
    {
        Vector3 movementDirection = Vector3.zero;

        if (Input.mousePosition.x >= Screen.width - cameraEdgePanPixels
              && mainCamera.transform.position.x < terrain.terrainData.size.x)
                    movementDirection += Vector3.right;
        else if (Input.mousePosition.x <= cameraEdgePanPixels &&
              mainCamera.transform.position.x > 0)
                    movementDirection += Vector3.left;

        if (Input.mousePosition.y >= Screen.height - cameraEdgePanPixels &&
              mainCamera.transform.position.z + anglePositionDisplacement <
              terrain.terrainData.size.z)
                    movementDirection += Vector3.forward;
        else if (Input.mousePosition.y <= cameraEdgePanPixels &&
              mainCamera.transform.position.z + anglePositionDisplacement > 0)
                    movementDirection += Vector3.back;

        if (movementDirection != Vector3.zero)
            mainCamera.transform.position += movementDirection.normalized *
              cameraMovementSpeed * Time.deltaTime;
    }
```

5.1. Stanja i akcije

Kursor se može nalaziti u jednom od četiri definirana stanja prilikom rada u prostoru igre: Default, AOE, Building i Target. Za implementaciju funkcionalnosti upravljanja stanjima kursora i njihovih akcija je korišten State uzorak dizajna tako da postoji sučelje ICursorState koje sadrži osnovne mogućnosti korištenja miša, a pojedina klasa stanja mora definirati te klasa MouseController koja služi kao kontekst za upravljanje promjenama stanja i čuvanju trenutno aktivnoga stanja.

```
public interface ICursorState
{
    void Default(MouseController context);
    void OnLeftMousePress(MouseController context);
    void OnLeftMouseRelease(MouseController context);
    void OnRightMouseClick(MouseController context);
    void OnEnter(MouseController context);
    void OnExit(MouseController context);
}
```

Svaka klasa stanja mora imati vlastitu implementaciju metoda navedenih u ICursorState sučelju, a ukoliko stanje nema definirano ponašanje za određenu akciju, onda se jednostavno ostavi prazna metoda i ignorira. Metode OnEnter i OnExit služe za podešavanja ikone kursora i ostalih varijabli za normalno funkcioniranje stanja. Moguće korisničke akcije i njihovi opisi u pojedinom stanju su dani sljedećom tablicom i definirani kao:

- Highlight – Isticanje obruba jedinice nad kojom se kursor trenutno nalazi
- Selection – Odabir jedne jedinice
- Selection Box – Odabir većeg broja jedinica držanjem i povlačenjem kursora
- AoE Preview – Prikaz prostora koji će biti obuhvaćen vještinom
- Building Preview – Prikaz prostora kojega će zgrada zauzeti
- Move – Slanje komande odabranim jedinicama za kretanje na poziciju kursora
- Attack – Slanje komande odabranim jedinicama za napadanje neprijateljske jedinice
- Confirm – Potvrda odabrane točke ili jedinice
- Cancel – Poništavanje radnje i prelazak u Default stanje

	Default	OnLeftMousePress	OnLeftMouseRelease	OnRightMouseClick
Default	Highlight Selection Box	Selection Selection Box (start)	Selection Box (end)	Move Attack
AOE	AoE Preview	-	Confirm	Cancel
Building	Building Preview	-	Confirm	Cancel
Target	Highlight	-	Confirm	Cancel

Tablica 1: Stanja i ponašanja kursora

Jedan od primjera implementacija metoda je Confirm akcija u Target stanju koja radi Raycast iz pozicije miša na ekranu prema prostoru igre te vraća prvi objekt na putu zrake. Za Raycast se može postaviti četvrti parametar layerMask koji ignorira jedan sloj objekata u igri (primjerice teren, jedinice ili resurse) ili pomoću bitovnog pomaka podesiti da ignorira sve osim odabranoga što su u ovom slučaju jedinice.

```
public void OnLeftMouseRelease(MouseController context)
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, 1 << 7))
    {
        if (hit.collider.gameObject)
            context.RecieveSingleTarget(hit.collider.gameObject.GetComponent<Unit>());
    }
}
```

5.2. Označavanje jedinica

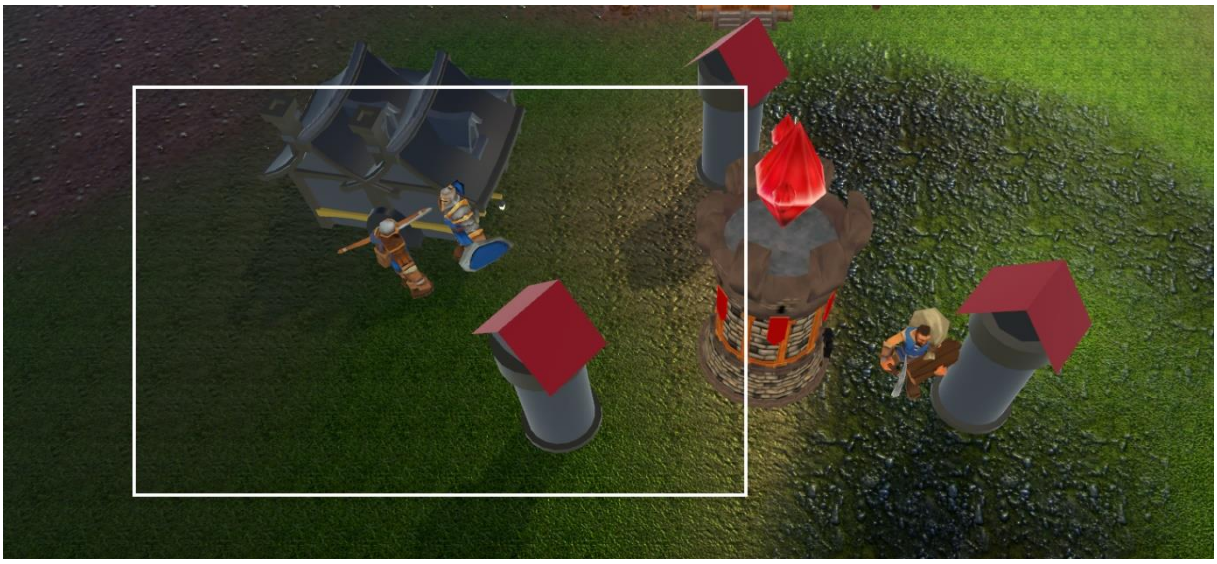
Jedinice se mogu označiti pojedinačno sa jednim klikom lijeve tipke miša ili držati tipku stisnutu kako bi se označio veći broj jedinica unutar odabranoga prostora te je moguće provesti ga isključivo u Default stanju kursora. Odabrane jedinice će biti prikazane na grafičkom sučelju te će se moći koristiti njihove posebne vještine, pokretati trening novih jedinica, izgradnje novih zgrada, poboljšanja svih jedinica i ostale akcije. U sličaju jednoga klika, moguće je samo napraviti Raycast vrlo sličan već objašnjenom u prethodnom poglavlju, dok za držanje pritisnutog miša zahtjeva malo drugačiji pristup. Prilikom pritiska miša je potrebno zabilježiti njegovu poziciju kao početna te nakon toga sačekati da se tipka otpusti što nam daje završnu točku jednako kao i područje selekcije. U međuvremenu je potrebno podešavati element grafičkog sučelja koji prikazuje područje odabira. Također je definirano minimalno odstupanje od početne pozicije miša koje se treba preći kako bi se odabir smatrao grupnim umjesto jednog klika.

```
Vector3 selectionBox = hit.point - selectionStart;
if (Mathf.Abs(selectionBox.x) > context.selectionBoxThreshold
    || Mathf.Abs(selectionBox.z) > context.selectionBoxThreshold)
{
    context.selectionBox.SetActive(true);
    RectTransform selectionTransform = context.selectionBox.GetComponent<RectTransform>();
    Vector3 boxStart = Camera.main.WorldToScreenPoint(selectionStart);
    selectionTransform.anchoredPosition = boxStart;

    float offsetX = Input.mousePosition.x - boxStart.x;
    float offsetY = Input.mousePosition.y - boxStart.y;
    selectionTransform.localScale =
        new Vector3((offsetX > 0 ? 1 : -1), (offsetY > 0 ? 1 : -1), 1);
        selectionTransform.sizeDelta = new Vector2(Mathf.Abs(offsetX), Mathf.Abs(offsetY));
}
else context.selectionBox.SetActive(false);
```

Grafički element odabira je jednostavna slika kvadrata sa bijelim rubom i transparentnom sredinom koji se smanjuje ili povećava u dvije dimenzije po potrebi. Važno je napomenuti kako Unity ne dozvoljava negativan broj za skaliranje dimenzije što znači da je potrebno provjeriti dali se početna pozicija miša nalazi ispred ili iza završne pozicije te ih obrnuti ukoliko je to potrebno kako bi se izbjegli negativno skaliranje.

```
public void OnLeftMouseRelease(MouseController context)
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, 1 << 6 | 1 << 7))
    {
        if (hit.collider.gameObject)
        {
            List<Unit> selectedUnits = new List<Unit>();
            Vector3 selectionBox = hit.point - selectionStart;
            if (Mathf.Abs(selectionBox.x) < context.selectionBoxThreshold &&
                Mathf.Abs(selectionBox.z) < context.selectionBoxThreshold)
            {
                if (hit.collider.gameObject.GetComponent<Unit>())
                    selectedUnits.Add(hit.collider.gameObject.GetComponent<Unit>());
            }
            else
            {
                Vector3 selectionBoxAbs = new Vector3(Mathf.Abs(selectionBox.x), 1,
                    Mathf.Abs(selectionBox.z));
                Collider[] overlapColliders =
                    Physics.OverlapBox(selectionStart + selectionBox / 2,
                    selectionBoxAbs / 2, Quaternion.identity, 1 << 7);
                foreach (Collider collider in overlapColliders)
                {
                    selectedUnits.Add(collider.gameObject.GetComponent<Unit>());
                }
            }
            context.SelectUnits(selectedUnits);
        }
        selecting = false;
    }
}
```

Prilikom odabira jedinica se poziva Physics.OverlapBox metoda koja pronalazi sve Collider objekte unutar odabranog područja te koristi layerMask za odabir samo onih objekata koji su označeni kao Units i dodaju se u listu odabranih jedinica igrača unutar Player klase.



Slika 1: Prikaz grafičkog elementa za odabiranje jedinica

5.3. Označavanje područja

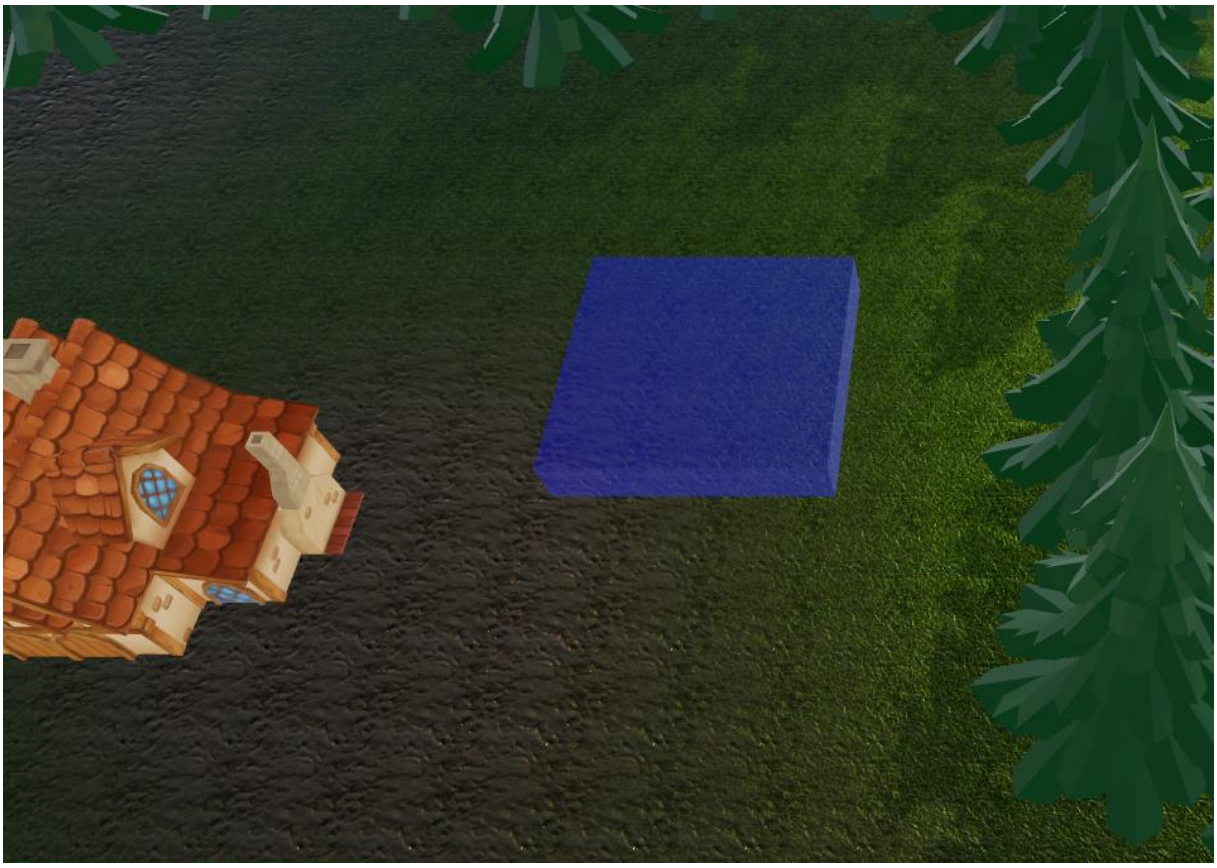
Kursor također sadrži mogućnosti označavanja većeg područja za izgradnju novih građevina ili kao prostora efekta nekih od vještina. Prilikom korištenja jedne od vještina koja zahtjeva AoE kao svoju metu, klasa Skill će od kontrolera zatražiti promjenu stanja kursora te se zapisati kao odredište za povratnu informaciju nakon korisničkog odabira. Stanja kursora AOE i Building u svom normalnom načinu rada konstantno prate gdje se pokazivač trenutno nalazi na samom terenu unutar igre te povlače objekt grafičkog prikaza na tu lokaciju. Moguće je specificirati i radius varijablu koja onda povećava ili smanjuje grafički element kako bi odgovarao stvarnom području koje će biti zahvaćeno.

```
public void OnEnter(MouseController context)
{
    context.objectDisplayBuilding.SetActive(true);
    context.objectDisplayBuilding.transform.localScale =
        new Vector3(context.buildingRadius, 1, context.buildingRadius);
}


public void Default(MouseController context)
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, 1 << 6))
    {
        if (hit.collider.gameObject)
            context.objectDisplayBuilding.transform.position = hit.point;
    }
}
```

Prethodni kod prikazuje metode klase CursorBuilding koja opisuje Building stanje kursora. Prilikom ulaska u stanje, aktivira se objekt za grafički prikaz te mu se podešava

veličina dobivena kroz radius varijablu. Unutar svoje Default metode, objekt se pomiče na točku terena gdje kursor pogađa, a prilikom lijevog klika miša se grafički prikaz gasi te se šalje povratna informacija sa odabranom lokacijom.

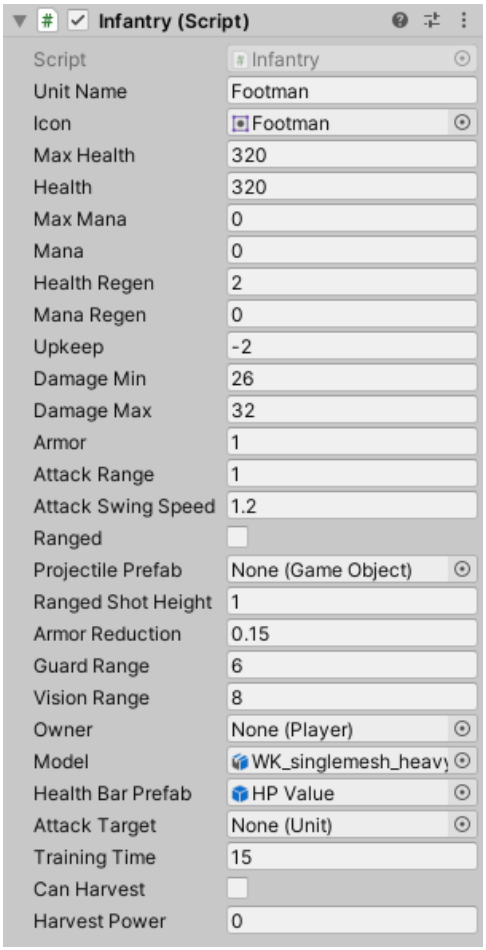
```
public void OnLeftMouseRelease(MouseController context)
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, 1 << 6))
    {
        if (hit.collider.gameObject)
            context.RecievePointTarget(hit.point);
    }
    OnExit(context);
}
```



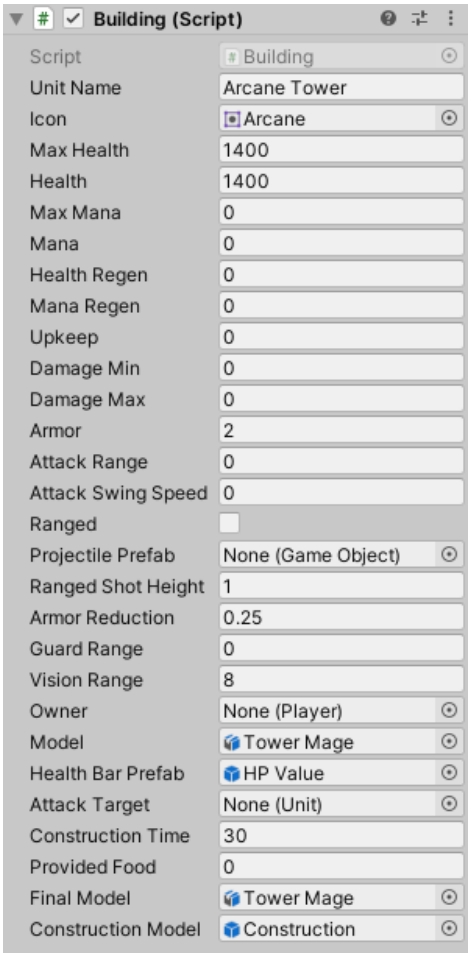
Slika 2: Grafički element kursora za odabir lokacije izgradnje

6. Jedinice

Jedinice u igri predstavljaju građevine i vojsku koju igrači kontroliraju tjekom borbe. Svaka jedinica ima svoje atribute koji se koriste tjekom igre kao primjerice životni poeni, jačina napada, brzina napada, brzina kretanja, naziv, sliku, popis vještina koje može koristiti, popis jedinica koje može trenirati i mnoge druge. Roditeljska klasa Unit sadrži glavne metode i atribute koje svaka jedinica posjeduje te koristi apstrakciju za implementaciju metoda koje se drugačije izvršavaju kod vojske u usporedbi sa zgradama. Klase Infantry i Building nasljeđuju klasu Unit te implementiraju potrebne apstraktne metode kao i određene virtualne metode ukoliko se one koriste. Neke od osnovnih metoda koje se nalaze u Unit klasi su metode za napadanje, kretanje, regeneraciju životnih poena, uništavanje svog objekta, skupljanje resursa te nekoliko ostalih. Ukoliko se podklasa ne koristi nekom od navedenih metoda, ne mora ju implementirati ili je jednostavno ostavi praznom.



Slika 3: Komponenta Infantry u prozoru inspektora



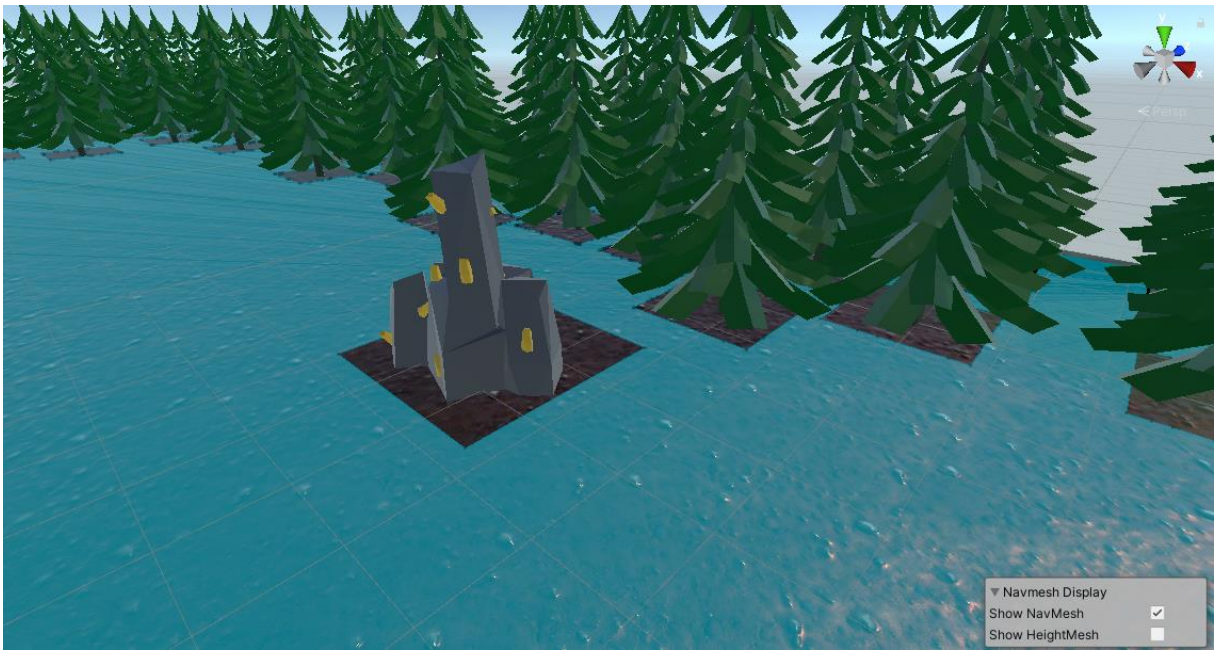
Slika 4: Komponenta Building u prozoru inspektora

6.1. Kretanje

Za kretanje jedinica se koriste ugrađene Unity komponente i sustav traženja puta i navigacije kroz NavMesh i NavMeshAgent. Unity ima prozor Navigation na kojemu je moguće generirati NavMesh, tj. označiti područja terena ili 3D objekata na kojima je moguće kretanje. Automatski označena područja inače budu ravne plohe ili određene neravnine kao uzvišenja i sniženja koja nisu previše strma, no moguće je i ručno podesiti područja ili parametre automatske generacije. Svaki objekt koji se može kretati u sebi sadrži NavMeshAgent komponentu pomoću koje se može zadati točka na NavMesh-u prema kojoj će objekt pronaći put između svih prepreka i početi se približavati sa zadanom brzinom. Objekti se mogu označiti kao statični što govori NavMesh generatoru da oni predstavljaju prepreku što se može koristiti za građevine ili drveće unutar igre. Objekti označeni kao statični se naravno ne mogu kretati iz čega proizlazi glavna mana ovoga ugrađenog sustava, a to je da agenti prilikom pronalaženja puta ne prepoznaju ostale agente te ih ne mogu obilaziti. Kako bi se agentu zadala odredišna točka, jednostavno se poziva metoda SetDestination(Vector3 point) nad komponentom NavMeshAgent, dok ResetPath() briše zadanu točku i agent ostaje na mjestu.

```
public override void Move(Vector3 point)
{
    agent.SetDestination(point);
}

public override void StopAction()
{
    attackTarget = null;
    targetResource = null;
    agent.ResetPath();
}
```

Navigacijski prozor unutar Unity alata pruža mogućnost prikaza trenutno generiranog NavMesh-a na odabranoj mapi zajedno sa svim prazninama stvorenim od strane objekata resursa te zgrada. Bitno je napomenuti kako se jednom generirani NavMesh neće sam ponovno izgenerirati nakon bilo kakvih izmjena na terenu, već je potrebno ručno pokrenuti generaciju čiji se podaci spremaju u posebnu datoteku na disku među ostalim datotekama projekta.



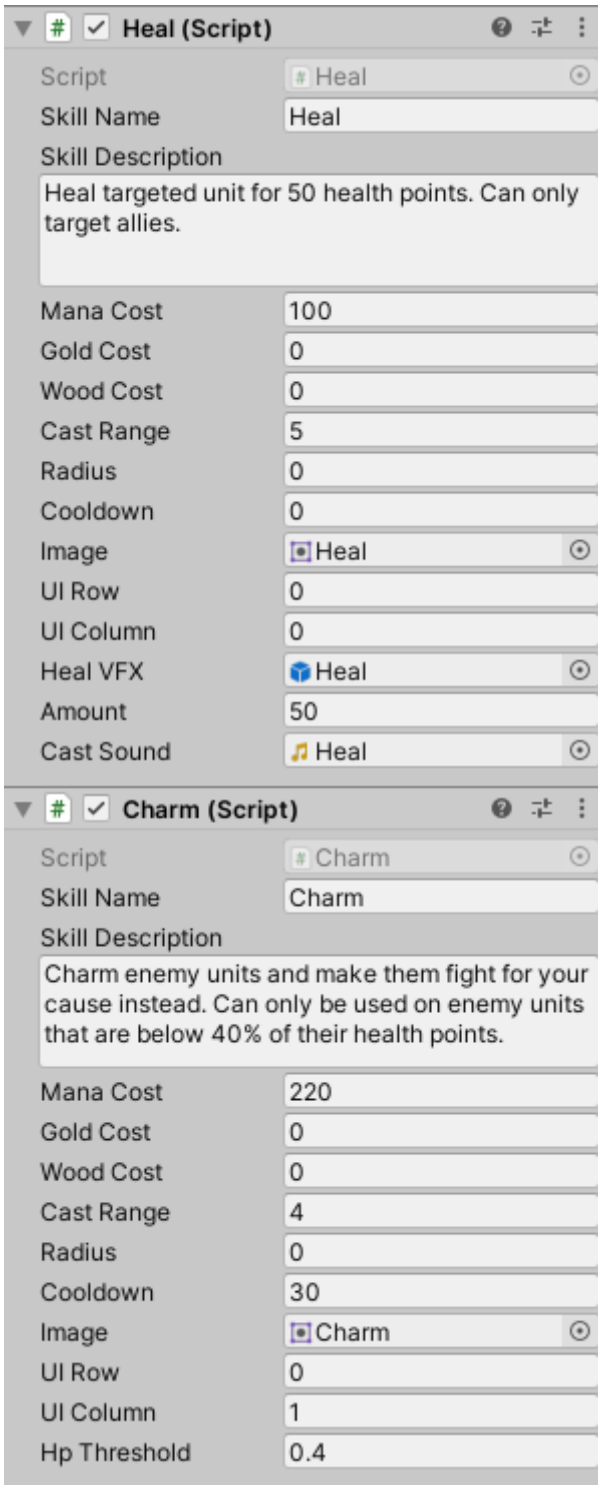
Slika 5: Generiranje NavMesh-a unutar Unity alata

6.2. Vještine

Većina jedinica u igra ima jednu ili više vještina koje može koristiti kako bi pomogla svojem timu ili sabotirala protivnike. Neke od osnovnih funkcionalnosti poput treninga jedinica ili izgradnje zgrada se također smatraju vještinama, dok primjerice “Mage“ jedinica može poslati vatrenu kuglu prema protivnicima i napraviti veliku štetu ili “Priest” koji može izlječiti određeni broj životnih poena na prijateljskim jedinicama. Svaka vještina nasljeđuje roditeljsku klasu Skill koja se sastoji od nekih osnovnih podataka kao što je tip mete (jedinica, točka na terenu, zgrada...), cijena mana poena potrebna za korištenje, vrijeme hlađenja (eng. Cooldown) prije nego se vještina ponovno može koristiti te neke osnovne metode prilikom aktivacije ili samoga korištenja. Budući da vještina može imati razne efekte, nije moguće napraviti generičku definiciju metode već je potrebno omogućiti svakoj pojedinačnoj vještiti da napiše svoju verziju metode što je ostvareno stvaranjem abstraktne Cast() metode. Jedan uzorak dizajna koji bi bio dosta koristan u ovome slučaju je Template Method koji se koristi za definiciju slijeda koraka u određenom algoritmu sa mogućnosti nadjačavanja nekih od koraka ili u ovome slučaju, obaveznog definiranja vlastitog. Slijedni koraci za svaki Skill su:

1. Provjera dali je vještina spremna za korištenje (Cooldown)
2. Traženje validne mete od kursora ukoliko je to potrebno
3. Trošenje resursa potrebnih za korištenje vještine ukoliko su dostupni
4. Izvršavanje efekta vještine
5. Aktivacija cooldown perioda

Korak 4 predstavlja abstraktni dio koji svaka podklasa mora implementirati kako bi obavljala svoj posao. Svaka vještina se može pridružiti jedinicama na način da se samo stave kao komponenta u inspector prozoru alata, nakon čeka će one pronaći i Unit komponentu te obaviti potrebna povezivanja. Sljedeća slika prikazuje inspector prozor kod “Priest” jedinice koji ima dvije različite vještine.



Slika 6: Različite Skill komponente u prozoru inspektora

Sljedeće kod prikazuje metode od svih koraka korištenja vještine osim četvrtoga koji se implementira posebno u podklasama.

```
public void ActivateSkill()
{
    if (castingUnit.mana >= manaCost && castingUnit.owner.gold >= goldCost &&
        castingUnit.owner.wood >= woodCost && !isDisabled)
    {
        if (targetingTypes.Contains(Targeting.None))
            Cast();
        else if (targetingTypes.Contains(Targeting.Infantry))
            RequestTarget(false, false);
        else if (targetingTypes.Contains(Targeting.Building))
            RequestTarget(true, false, radius);
        else if (targetingTypes.Contains(Targeting.Aoe))
            RequestTarget(false, true, radius);
        else if (targetingTypes.Contains(Targeting.Point))
            RequestTarget(false, true, 0);
    }
    else
    {
        if(isDisabled) castingUnit.owner.AlertMessage("Skill is on cooldown for " +
           ((int)cooldownTimer).ToString() + " seconds");
        else if (castingUnit.mana < manaCost) castingUnit.owner.AlertMessage(
           "Not enough mana");
        else castingUnit.owner.AlertMessage("Not enough resources");
    }
}

public void SpendResources()
{
    castingUnit.mana -= manaCost;
    castingUnit.owner.gold -= goldCost;
    castingUnit.owner.wood -= woodCost;
}

public void ActivateCooldown()
{
    isDisabled = true;
    cooldownTimer = cooldown;
}
```

Budući da svaki efekt može imati vlastite uvijete i kompleksna ponašanja prilikom izvođenja svojih akcija, potrebno je pravilno implementirati Cast() metodu osnovne klase. Primjerice, kod vještine “Charm”, igrač može preuzeti kontrolu nad protivničkom jedinicom, ali samo ako ona trenutno ima životne poene ispod određenog praga kao 40% maksimalnog iznosa. Ovakva ponašanja se moraju definirati nadjačanoj u Cast() metodi.

```
public override void Cast()
{
    if (unit.health <= unit.maxHealth * hpThreshold &&
        !castingUnit.owner.GetAllies().Contains(unit.owner))
    {
        target.owner = castingUnit.owner;
    }
}
```

Neke od vještina stvaraju razne vizualne efekte i puštaju zvuk prilikom korištenja što se također treba izvršiti unutar Cast() metode, a može se vidjeti na “Heal” vještini.

```
public override void Cast()
{
    AudioSource.PlayClipAtPoint(castSound, castingUnit.transform.position);
    Instantiate(healVFX, target.transform.position, Quaternion.identity);
    target.health += amount;
}
```

6.3. Izgradnja zgrada

Neke jedinice imaju mogućnost pokretanja izgradnje novih zgrada što se može aktivirati pritiskom na akcijski gumb za pojedinu zgradu. Nakon klika na vještinu, kursor prelazi u Building stanje koje prikazuje područje na kojemu će se nova zgrada napraviti, ukoliko je taj prostor slobodan. Zgrade imaju određeno vrijeme konstrukcije prije nego postanu iskoristive i dođu u svoje normalno stanje. Za vrijeme izgradnje, igraču neće biti prikazani akcijski gumbi za vještine te hrana koju neke zgrade generiraju, nije dostupna sve dok se konstrukcija u potpunosti ne dovrši.

Slika 7: Prikaz dovršene zgrade pored zgrade u procesu izgradnje

Za vrijeme izgradnje je moguće vidjeti HUD element koji prikazuje koliko je još ostalo do dovršetka zgrade.

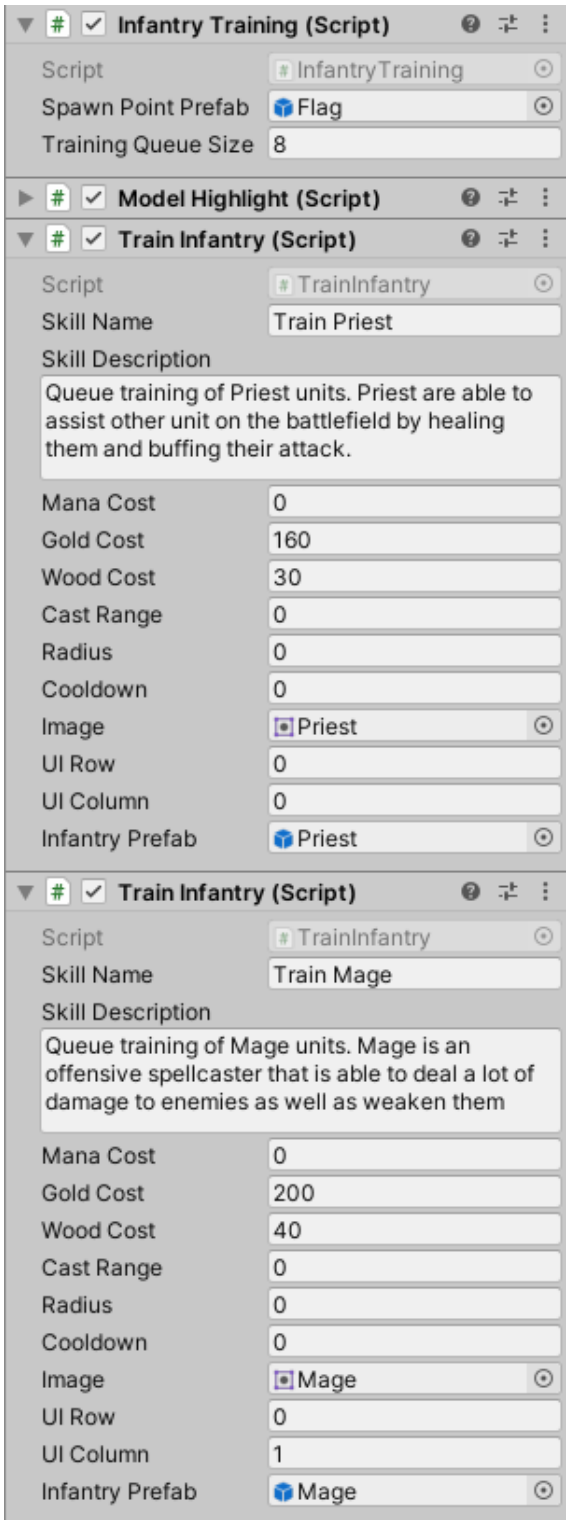
6.4. Trening vojske

Većina zgrada ima mogućnost treninga barem jedne vrste jedinica klase Infantry sa pojedinim iznimkama. Svaka zgrada ima na sebi pridruženu komponentu InfantryTraining koja služi kao red čekanja za treniranje novih jedinica te pruža korisniku mogućnost pregleda kao i prekida treninga odabranih jedinica. Prilikom odabira zgrade, korisnik u svom grafičkom sučelju vidi prikaz svih trenutno poredanih jedinica za trening te grafički element postotka dovršetka izgradnje. Korisnik također može pritisnuti tipku sa ikonicom jedinice u procesu treninga kako bi tu jedinicu uklonio iz reda čekanja, a ukoliko je to jedinica koja je trenutno prva u redu, onda se postotak dovršetka treninga također vraća na početak, dok uklanjanje ostalih jedinica nema efekta na postotak dovršetka trenutno prve po redu jedinice. Korisniku je također omogućeno desnim klikom miša odabrati točku oko zgrade na kojoj će se prilikom dovršetka treninga stvoriti nova jedinica. Ova funkcionalnost može biti korisna ukoliko se zgrada postavljena na mjesto jako blizu drveća ili ostalih zgrada koje bi potencijalno mogle blokirati put novim jedinicama i zatočiti ih u malom prostoru.

```
Public GameObject spawnPointPrefab;
public int trainingQueueSize = 5;
public readonly List<Infantry> infantryTrainingQueue = new List<Infantry>();
public float trainingTime = 0f;
Vector3 spawningPoint;

void Update()
{
    if (infantryTrainingQueue.Count > 0)
    {
        trainingTime += Time.deltaTime;
        if(trainingTime >= infantryTrainingQueue[0].trainingTime)
        {
            Infantry topInfanty = infantryTrainingQueue[0];
            trainingTime -= topInfanty.trainingTime;
            SpawnInfantry(topInfanty);
            infantryTrainingQueue.RemoveAt(0);
            owner.gameUI.UpdateSelectionDisplay(owner);
        }
    }
}

public void QueueTraining(Infantry infantry)
{
    if(infantryTrainingQueue.Count < trainingQueueSize)
    {
        infantryTrainingQueue.Add(infantry);
        owner.gameUI.UpdateSelectionDisplay(owner);
    }
}


public void SetSpawningPoint(Vector3 point)
{
    spawningPoint = gameObject.transform.position + Vector3.Normalize(
       point - gameObject.transform.position) * gameObject.transform.localScale.x;
    Instantiate(spawnPointPrefab, spawningPoint, Quaternion.identity);
```

```
}

public void RemoveFromQueue(int index)
{
    if(infantryTrainingQueue.Count > index)
    {
        if(index == 0) trainingTime = 0;
        infantryTrainingQueue.RemoveAt(index);
        owner.gameUI.UpdateSelectionDisplay(owner);
    }
}

void SpawnInfantry(Infantry infantry)
{
    GameObject newUnit =
       Instantiate(infantry.gameObject, spawningPoint, Quaternion.identity);
    newUnit.GetComponent<Unit>().owner = owner;
    owner.RegisterUnit(newUnit.GetComponent<Unit>());
}
```

Svaka zgrada također mora negdje imati definirano koje sve jedinice ona može trenirati, te poveznicu sa komponentom InfantryTraining kako bi joj mogla dodati nove jedinice u red čekanja. Isto tako je potrebno povezati gumbe sa ikonicama jedinica koji bi klasi govorili kada treba ukloniti jedinicu iz reda. Prilikom generiranja ikonica na grafičkom sučelju, poziva se metoda koja pridružuje index, tj. poziciju u redu čekanja na sam gumb koji bi onda prilikom aktivacije, pozvao metodu RemoveFromQueue u klasi InfantryTraining. Za definiranje popisa jedinica koje zgrada može trenirati se može koristiti mnogo pristupa, no u ovom projektu je trening definiran kao vještina koja se ona može dodati na 18bject zgrade kao komponenta. Unutar komponente je moguće definirati sve parametre za trening pojedinih jedinica 18bjec količinu resursa potrebnu za plaćanje prije same izgradnje. Slika prikazuje prozor inspektora za 18bject “Arcane Tower” koji može trenirati dvije različite vrste jedinica.



Slika 8: Komponenta Building u prozoru inspektora

Red čekanja treninga ima svoju maksimalnu veličinu te neće dozvoliti stavljanje više od odabranog broja jedinica u red čekanja, a isto tako nije moguće započeti trening novih jedinica ukoliko igrač nema dovoljno hrane.

```
public class TrainInfantry : Skill
{
    public GameObject infantryPrefab;
    InfantryTraining infantryTraining;

    void Start()
    {
        targetingTypes.Add(Targeting.None);
        gameObject.GetComponent<Unit>().AddSkill(this);
        castingUnit = gameObject.GetComponent<Unit>();
        mouseController = castingUnit.owner.gameObject.GetComponent<MouseController>();
        infantryTraining = gameObject.GetComponent<InfantryTraining>();
    }

    public override void Cast()
    {
        if (infantryTraining.infantryTrainingQueue.Count <
            infantryTraining.trainingQueueSize)
        {
            if(gameObject.GetComponent<Unit>().owner.upkeep > 0)
            {
                SpendResources();
                infantryTraining.QueueTraining(infantryPrefab.GetComponent<Infantry>());
            }
            else gameObject.GetComponent<Unit>().owner.AlertMessage(
                "Not enough food to train new units");
        }
        else gameObject.GetComponent<Unit>().owner.AlertMessage("Training queue is full");
    }
}
```

6.5. Borba

Kako jedinice nebi trebale koristiti isključivo vještine za međusobnu borbu, potrebno je implementirati neke osnovne napade oružijem koje će jedinice pasivno koristiti. Svaka jedinica ima postavljene vrijednosti Damage Min, Damage Max, Armor, Attack Range, Attack Swing Speed i Guard Range koje služe za definiciju ponašanje prilikom susreta sa neprijateljima. Attack Swing Speed varijabla govori koliko brzo jedinica može napadati, tj. u kojem vremenskom periodu može napraviti jedan napad. Attack range predstavlja udaljenost iz koje jedinica može izvršiti napad što razlikuje strijelce od običnih ratnika sa mačevima. Šteta koju će jedinica napraviti drugoj je nasumično odabrani broj između Damage Min i Damage Max vrijednosti kako bi borba imala neku nepredvidivu komponentu što ju čini zanimljivijom i dinamičnijom. Armor služi za umanjenje primljene štete za određeni postotak i služi kako bi mogli prikazati da je teže naštetiti nekoj zgradi nego primjerice običnom ratniku sa slabijim oklopom. Svaka jedinica u svojoj Unit klasi ima vrijednost za attackTarget koji može biti prazan ukoliko se jedinica trenutno ne bori ili postavljena na svoju metu. Ukoliko je meta pre daleko od jedinice, tj. izvan njenog Attack Range područja, onda se šalje zahtjev za kretanje NavMeshAgent komponenti sve dok se jedinica dovoljno ne približi nakon čega započinje sa napadima. Jedinice također imaju svoj Guard Range, tj. udaljenost sa koje mogu uočiti

neprijateljske jedinice i počet ih napadati svojevoljno bez ikakvog unosa od strane igrača. Ova mehanika služi za stražarenje i smanjenje igračevih odgovornosti.

```
void Update()
{
    if(attackTarget != null && agent.remainingDistance < 0.2)
    {
        if(Vector3.Distance(gameObject.transform.position,
            attackTarget.transform.position) > attackRange)
        {
            agent.SetDestination(attackTarget.transform.position);
        }
        else
        {
            transform.LookAt(attackTarget.transform);
            agent.ResetPath();
            if (attackCooldown <= 0)
            {
                attackCooldown = attackSwingSpeed;
                if (ranged) RangedAttack();
                else MeleeAttack();
            }
            else attackCooldown -= Time.deltaTime;
        }
    }
}

void MeleeAttack()
{
    attackTarget.TakeDamage(this, Random.Range(damageMin, damageMax));
}

void RangedAttack()
{
    GameObject projectile = Instantiate(projectilePrefab, gameObject.transform.position,
        Quaternion.identity);
    projectile.GetComponent<Projectile>().Shoot(this, attackTarget, Random.Range(damageMin,
        damageMax));
}
```

7. HUD

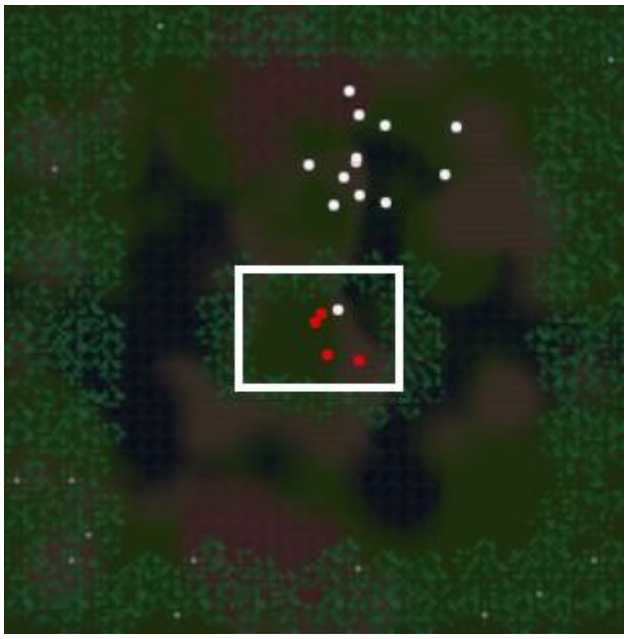
Jedan od većih funkcionalnosti igre je bilo postaviti i napraviti responzivno korisničko sučelje, tj. HUD (*eng. Heads Up Display*). Primarna funkcija ove komponente je pružanje korisnicima veću količinu opcija i informacija za bolje i lakše savladavanje same igre. Korisnika mogu zanimati razne stvari poput mape sa prikazom lokacije vlastitih kao i protivničkih jedinica, trenutno stanje resursa na raspolaganju, prikaz svih odabranih jedinica kao mogućnost promjene između njih, stanje izgradenje građevina ili jedinica i slično. HUD se u ovoj igri sastoji od minimape, prozora sa odabranim jedinicama, portret glavne jedinice, prozor sa akcijskim gumbima za korištenje vještina, prikaza resursa i tipke za glavni izbornik. Dok su prikazi resursa i tipka za glavni izbornik poprilično trivijalne stvari, ostatak kompleksnijih djelova će biti objašnjeni pojedinačno.

Slika 9: HUD igre

7.1. Minimapa

Minimapa prikazuje teren mape na kojoj se igra, lokaciju drveća te jedinice koje su u vlasništvu igrača, igračevih saveznika ili neprijatelja koji su dovoljno blizu da ih vlastite ili savezničke jedinice vide. Minimapa također ima marker prikaza trenutne pozicije kamere na mapi koje se može pomicati za brže prebacivanje kamere sa jednoga dijela mape na drugi.

Glavna svrha minimape je pružanje igraču strateške informacije vezane za pozicioniranje svojih jedinica. Prilikom snimanja svoga terena, kororištena je jedna posebna kamera, slično kao i za portret trenutno odabrane jedinice, koja samo snima sloj na kojem se nalaze teren i resursi te šalje svoju projekciju na RenderTexture pozadine minimape.



Slika 10: Minimapa

Minimapa na sebi sadrži marker kamere koji prikazuje područje koje je trenutno vidljivo na igračevom zaslonu. Kako bi se olakšalo pozicioniranje kamere, moguće je lijevom tipkom miša kliknuti bilo gdje na minimapu kako bi kamera skočila na tu poziciju. Potrebno je skalirati koordinate sa terena na koordinate minimape kako bi pravilno preslikali poziciju na koju trebamo pozicionirati marker ili kameru.

```
void Update()
{
    MinimapFogOfWar();
    cameraMarker.transform.localPosition = new Vector3(
        (mainCamera.transform.position.x - 10) * minimapScaling,
        mainCamera.transform.position.z * minimapScaling, 0);
    if(Input.GetKey(KeyCode.Mouse0))
    {
        if(Input.mousePosition.x/canvasScale < 150 &&
            Input.mousePosition.y/canvasScale < 150)
        {
            mainCamera.transform.position = new Vector3(
            Mathf.Clamp(Input.mousePosition.x/canvasScale/minimapScaling, 0,
            terrain.terrainData.size.x),
            mainCamera.transform.position.y,
            Mathf.Clamp((Input.mousePosition.y - 10)/canvasScale/minimapScaling, 0,
            terrain.terrainData.size.z - 7.5f));
        }
    }
}
```

Minimapa također prikazuje poizicije svih vlastitih kao i savezničkih jedinica, dok neprijateljske jedinice nisu prikazane na mapi sve dok ne dođu dovoljno blizu savezničkih. Kako bi se ova funkcionalnost ostvarila, potrebno je proći kroz sve jedinice na mapi, provjeriti dali su jedinice igrača, saveznika ili neprijatelja te izmjeriti udaljenost do najbližeg saveznika ukoliko su neprijatelji kako bi odredili točno one koje treba prikazati na mapi. Svaka jedinica ima definiranu Vision varijablu koja predstavlja udaljenost iz koje može vidjeti protivničke jedinice.

```
void MinimapFogOfWar()
{
    List<Unit> alliedUnits = new List<Unit>();
    List<Unit> allUnits = new List<Unit>();
    foreach (GameObject unit in GameObject.FindGameObjectsWithTag("Unit"))
    {
        allUnits.Add(unit.GetComponent<Unit>());
    }
    foreach (Player teamPlayer in player.GetAllies())
    {
        foreach (Unit unit in teamPlayer.ownedUnits)
        {
            alliedUnits.Add(unit);
        }
    }
    while (unitMarkers.Count < allUnits.Count)
    {
        GameObject marker = Instantiate(unitPrefab, gameObject.transform, false);
        unitMarkers.Add(marker);
    }
    for(int i = 0; i < unitMarkers.Count; i++)
    {
        if (i < allUnits.Count)
        {
            unitMarkers[i].transform.localPosition = new Vector3(
                allUnits[i].transform.position.x * minimapScaling,
                allUnits[i].transform.position.z * minimapScaling, 0);
            if (alliedUnits.Contains(allUnits[i]))
            {
                unitMarkers[i].SetActive(true);
                unitMarkers[i].GetComponent<Image>().color = Color.white;
            }
            else if (IsVisible(allUnits[i], alliedUnits))
            {
                unitMarkers[i].SetActive(true);
                unitMarkers[i].GetComponent<Image>().color = Color.red;
            }
            else unitMarkers[i].SetActive(false);
        }
        else unitMarkers[i].SetActive(false);
    }
}
```

7.2. Portret

Portret prikazuje glavnu jedinicu od svih trenutno odabranih od strane korisnika, čije će se vještine prikazivati u prozoru za akcijske gumbe, kao i trenutno stanje životnih poena te

mane. Portret također sadrži animirani prikaz modela trenutno odabrane jedinice i njen naziv ispisan odmah iznad portreta. Prikaz animiranog portreta se može izvesti na način da se na element korisničko sučelja poveže RenderTexture umjesto običnih tekstura koji daje mogućnost povezivanja sa odabranom kamerom te prikaz njene projekcije na 2D elementu. Jedna od bitnih stvari za zapamtiti je da se generalno nastoji izbjeći veći broj različitih kamera u sceni jer spadaju među stvari koje puno više mogu usporiti igru u usporedbi sa ostalima, no moguće je podesiti kameru da snima samo određene slojeve igre kao primjerice Units sloj te ignorira sve ostalo što joj ne treba kako nebi nepotrebno renderirala objekte koji nas ne zanimaju. Nakon što je RenderTexture podeđen da prikazuje projekciju kamere, potrebno je staviti sam model jedinice koju želimo prikazati ispred nje što se može postići kopiranjem modela trenutno odabrane jedinice i postavljanjem ga kao dijete elementa postavljenog ispred kamere te uništavanjem modela čim ga više ne zatrebamo. Još jedan mogući način bi bio instanciranje svih postojećih modela u igri ispred kamere odmah na startu te ih po potrebi gasiti i paliti kako bi izbjegli dinamičko instanciranje koje je malo sporije, no po cijeni da svaki puta kada dodajemo nove jedinice moramo ručno podesiti i prikaze modela.

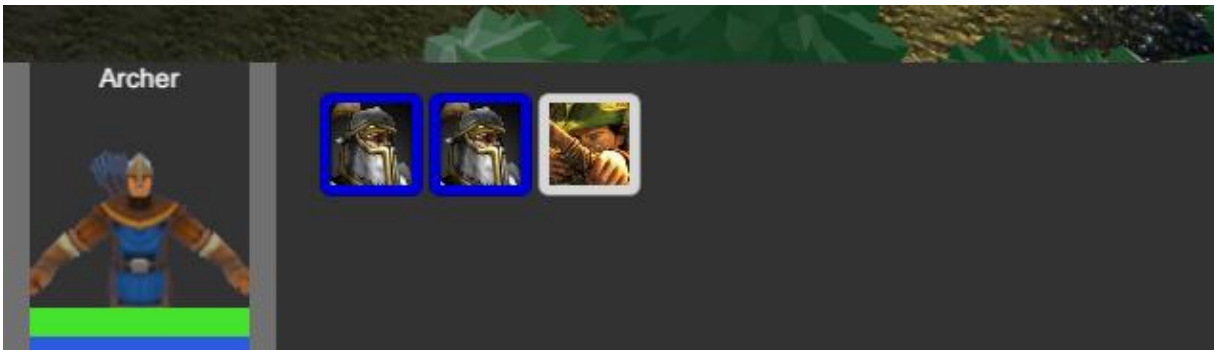
Slika 11: Portret trenutno odabrane jedinice

```
public void UpdateFocusedUnit(Unit unit)
{
    selectionContainer.UpdateFocusedUnit(unit);
    if(unitPortraitModel.transform.childCount != 0)
        Destroy(unitPortraitModel.transform.GetChild(0).gameObject);
    if (unit != null)
    {
        Instantiate(unit.model, unitPortraitModel.transform, false);
        portraitName.text = unit.unitName;
    }
```

```
    else portraitName.text = "";
}
```

7.3. Odabir jedinica

Korisnik može pritisnuti lijevu tipku miša kak bi selektirao jedinicu nad kojom se kursor trenutno nalazi, ukoliko se tipka stisne i drži, stvoriti će se selekcijski prostor koji umjesto jedne, odabire veći broj jedinica. Pošto zgrade imaju dosta velike modele i područja koja Raycast može pogoditi (eng. hitbox), postoji prioritet prilikom višestruke selekcije. Ukoliko selekcijski okvir sadrži u sebi jedinice klase Infantry kao i jedinice klase Building, odabrati će se samo Infantry jedinice kako zgrade nebi smetale za odabir. Nakon uspješnog odabira jedinica, one će se prikazati unutar selekcijskog prozora HUD komponente. Ukoliko je odabrana zgrada, selekcijski prozor će prikazivati jedinice u trenutnom redu čekanja na trening za tu zgradu kao i postotak dovršetka treninga. Odabir jedinice ili većega broja jedinica prikazuje ikone za sve odabrane jedinice do maksimalno 16 te označava trenutnu glavnu jedinicu sa posebnom bojom obruba. Glavna jedinica je ona čije se vještine prikazuju unutar prozora sa akcijskim gumbima. Kao što je vidljivo na slici 12, trenutno odabrana jedinica je označena sa bijelim obrubom te se može mijenjati klikom na jednu od ikonica ili pritiskom Tab tipke na tipkovnici.



Slika 12: Selekcijski prozor Infantry jedinica

Slika 13: Selekcijski prozor Building jedinice sa treningom

Selekcijski prozor na početku stvara 16 ikonica raspoređenih u dva reda te ih postavlja na neaktivno stanje kako bi se sakrile od korisnika, a prilikom selekcije se poziva

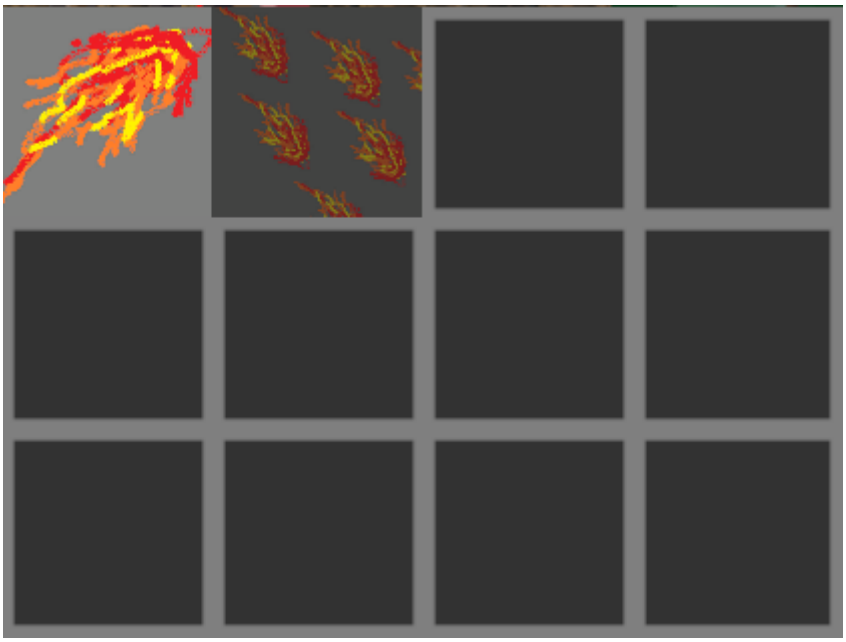
UpdateDisplay() metoda koja onda po potrebi pali određene ikonice te ih puni sa podacima vezanim za jedinicu koju simboliziraju.

```
public void UpdateDisplay(Player player)
{
    if(player.focusedSelectionUnit)
    {
        if (player.focusedSelectionUnit is Building)
        {
            it = player.focusedSelectionUnit.GetComponent<InfantryTraining>();
            if (it.infantryTrainingQueue.Count > 0 ||
                !player.focusedSelectionUnit.GetComponent<Building>().isConstructed)
                buildingProgressBar.SetActive(true);
            for (int i = 0; i < 16; i++)
            {
                if (i < it.infantryTrainingQueue.Count)
                {
                    icons[i].SetActive(true);
                    icons[i].GetComponent<IconButton>().SetLinkedUnit(
                        it.infantryTrainingQueue[i], it);
                }
                else icons[i].SetActive(false);
            }
        }
        else
        {
            it = null;
            buildingProgressBar.SetActive(false);
            for (int i = 0; i < 16; i++)
            {
                if (i < player.selectedUnits.Count && player.selectedUnits[i] is Infantry)
                {
                    icons[i].SetActive(true);
                    icons[i].GetComponent<IconButton>().SetLinkedUnit(
                        (Infantry)player.selectedUnits[i]);
                }
                else icons[i].SetActive(false);
            }
        }
    }
}

public void UpdateFocusedUnit(Unit unit)
{
    for(int i = 0; i < 16; i++)
    {
        IconButton button = icons[i].GetComponent<IconButton>();
        if (button.linkedUnit == unit) button.Highlight(true);
        else button.Highlight(false);
    }
}
```

7.4. Akcijski gumbi

Prozor sa akcijskim gumbima je rešetka sa tri reda i četiri stupca na kojoj se nalaze gumbi za aktivaciju vještina trenutno odabrane glavne jedinice. Gumbi se stvaraju odmah na početku igre te se njihovo stanje postavlja na neaktivno kako bi se sakrili od igrača. Prilikom igračeve selekcije jedinica, šalje se zahtjev za osvježavanje prozora akcijskih gumbi sa listom

vještina koju jedinica sadržava. Druga zadaća akcijskih gumbi je praćenje dali su vještine iz dobivene liste uopće spremne za korištenje budući da vještine zgrada koje trenutno nisu izgrađene nebi trebale biti dostupne jednako kao i vještine koje su trenutno na periodu hlađenja. Vještine koje trenutno nisu dostupne će biti zatamljene i neće se moći koristiti.



Slika 14: Prikaz privremeno nedostupne vještine

Akcijski gumbi također imaju mogućnost prikaza prozora sa nazivom, kratkim opisom vještine te cijene u resursima koju je igrač potreban platiti za njeno korištenje. Prilikom prelaska kursira preko jednog od akcijskih gumbi, prikazuje se prozor sa svim potrebnim informacijama. Ovisno o poziciji kursora na elementima grafičkog sučelja, moguće je prikazivati i sakrivati informacijski prozor vještine.

Slika 15: Prikaz informacijskog prozora akcijskog gumba

Svaka vještina u sebi ima definirane varijable koje označavaju preferiranu poziciju unutar rešetke za pozicioniranje gumbi na kojoj bi se htjela nalaziti. Ukoliko se desi da nekoliko različitih vještina žele biti na istoj poziciji unutar rešetke potrebno je postaviti jednu od njih na tu poziciju dok ostale idu na čekanje sve dok se ne pozicioniraju vještine na ostale preferirane pozicije, a nakon toga se raspodjeljuju preostala slobodna mjesta. Ukoliko jedinica ima više od 12 vještina, neke od njih se jednostavno neće moći prikazati unutar prozora.

```
public void UpdateSkills(List<Skill> skills)
{
    EmptySkills();
    List<Skill> unassignedSkills = new List<Skill>();
    foreach (Skill skill in skills)
    {
        if (actionButtons[skill.UIRow, skill.UIColumn].IsEmpty())
        {
            actionButtons[skill.UIRow, skill.UIColumn].SetButton(skill);
            continue;
        }
        else unassignedSkills.Add(skill);
    }
    foreach (Skill skill in unassignedSkills)
    {
        foreach (ActionButton actionButton in actionButtons)
        {
            if(actionButton.IsEmpty())
            {
                actionButton.SetButton(skill);
                break;
            }
        }
    }
}
```

8. Umjetna inteligencija

Umjetna inteligencija za RTS igre je generalno jako opširna i kompleksna tema za raspravljanje, a pogotovo za samo stvaranje i njenu implementaciju. Razvoj umjetnih inteligencija u svrhe igranja raznih žanrova igara protiv igrača je čak dobio dosta velik interes i unutar akademskog okruženja, kao i među poznatim globalnim kompanijama. Naglasak ovog projekta nije na umjetnoj inteligenciji što znači da se neće provoditi nikakva strojna učenja ili praviti kompleksne neuralne mreže već implementirati vrlo jednostavan sustav prioriteta izgradnje po kojemu će se agent orijentirati te jednostavno snalaženje vojske u prostoru. Stvaranje agenta koji će igrati protiv igrača će biti razdvojeno na glavna dijela: Sustav za izgradnju jedinica i građevina, sustav za upravljanje jedinicama i borbu. Ova dva sustava su u potpunosti odvojena te nisu svjesni jedan drugoga.

8.1. Izgradnja jedinica i građevina

Prvi sustav je sustav za izgradnju jedinica i građevina te služi za praćenje trenutnoga stanja agentove infrastrukture i vojske kako bi odlučio što dalje raditi. Sustav će proći kroz sve jedinice koje trenutno posjeduje, prebrojati koliko radnika ima, koliko obrambenih tornjeva ima, koja je trenutna snaga vojske i slično te nakon toga proći kroz niz uvijeta poredanih po važnosti i uzeti prvi koji se zadovolji. Sljedeća tablica popisuje sve prioritete sa njihovim identifikacijskim brojem.

Broj	Opis prioriteta
0	Treniranje novih jedinica radnika
1	Izgradnja glavne zgrade
2	Izgradnja farmi
3	Izgradnja zgrade Barracks
4	Izgradnja obrambenih tornjeva
5	Izgradnja zgrade Stable
6	Izgradnja zgrade Blacksmith
7	Izgradnja zgrade Arcane Tower
8	Treniranje vojske

Tablica 2: Prioriteti umjetne inteligencije

Ovisno o nekoliko uvijeta, agent si može odrediti prioritet za odabiranje sljedeće radnje. Kako se može vidjeti iz koda ispod, ukoliko agent nema niti jednog radnika onda će prioritet

postati trening novih radnika, ukoliko nema glavnu zgradu onda će ju napraviti, ukoliko mu fali hrane onda će napraviti nove farme, ukoliko ima par osnovnih građevina, a malo vojske, onda će treniratni nove vojnike i tako dalje.

```
if (workersCount <= 0) priority = 0;
else if (!unitNames.Contains("Keep")) priority = 1;
else if (workersCount < 4) priority = 0;
else if (player.TotalUpkeepCost() <= 5) priority = 2;
else if (!unitNames.Contains("Barracks")) priority = 3;
else if (armyStrength < 14) priority = 8;
else if (towerCount < 1) priority = 4;
else if (!unitNames.Contains("Stable")) priority = 5;
else if (armyStrength < 34) priority = 8;
else if (!unitNames.Contains("Blacksmith")) priority = 6;
else if (!unitNames.Contains("Arcane Tower")) priority = 7;
else if (armyStrength < 50) priority = 9;
else if (towerCount < 3) priority = 4;
else priority = 8;
```

Nakon što je prioritetna akcija odabrana, agent će ju pokušati izvršiti ukoliko za to ima dovoljno resursa. Također je bitno napomenuti da ovaj dio sustava upravlja svim radnicima i njihovim zadacima. Radnici će biti raspoređeni na skupljanje resursa tako da se održava balans između zlata i drva, a jedan radnik će biti poslan izgraditi novu građevinu onda kada je to potrebno.

8.2. Borba

Drugi dio sustava služi za slanje borbenih jedinica u rat sa ostalim igračima. Svaka jedinica zasebno zna koristiti svoje vještine i odabrati metu na kojoj će ju iskoristiti u posebnoj skripti koja je aktivna samo ako je vlasnik jedinice umjetna inteligencija. Ove skripte jednostavno provjeravaju stanje mana zaliha jedinice, perioda hlađenja vještine i koriste ju na najbližoj validnoj meti. Sustav za borbu također ima komponentu za određivanje točke interesa za svoju vojsku. Ova točka predstavlja poziciju na mapi prema kojoj će se sve jedinice pod kontrolom početi kretati i započeti borbu sa bilo kojom neprijateljskom jedinicom na koju naiđu. Ova točka je nasumično odabrana sa blagim naklonom prema startnim pozicijama ostalih igrača. Jedinica “Priest“ ima vještine za liječenje životnih poena prijateljskim jedinicama te preuzimanje kontrole nad ranjenim protivničkim jedinicama, primjer njegove skripte za korištenje vještina se može vidjeti u nastavku.

```
public class PriestAI : MonoBehaviour
{
    Heal heal;
    Charm charm;
    Infantry unit;

    void Start()
    {
        heal = gameObject.GetComponent<Heal>();
        charm = gameObject.GetComponent<Charm>();
        unit = gameObject.GetComponent<Infantry>();
    }

    void Update()
    {
        if(unit.mana >= heal.manaCost && !heal.isDisabled)
        {
            Collider[] colliders = Physics.OverlapSphere(gameObject.transform.position,
                heal.castRange, 1 << 7);
            foreach (Collider collider in colliders)
            {
                Unit potentialTarget = collider.gameObject.GetComponent<Unit>();
                if (potentialTarget.owner.teamId == unit.owner.teamId)
                    if (potentialTarget.health + heal.amount <= potentialTarget.maxHealth)
                    {
                        heal.CastAI(Vector3.zero, potentialTarget);
                        break;
                    }
            }
        }

        if (unit.mana >= charm.manaCost && !charm.isDisabled)
        {
            Collider[] colliders = Physics.OverlapSphere(gameObject.transform.position,
                charm.castRange, 1 << 7);
            foreach (Collider collider in colliders)
            {
                Unit potentialTarget = collider.gameObject.GetComponent<Unit>();
                if (potentialTarget.owner.teamId != unit.owner.teamId)
                    if (potentialTarget.health < potentialTarget.maxHealth *
                        charm.hpThreshold)
                    {
                        charm.CastAI(Vector3.zero, potentialTarget);
                        break;
                    }
            }
        }
    }
}
```

8.3. Pomoćne funkcije

Umjetna inteligencija zahtjeva nekoliko različitih manjih akcija i odluka koje inače obavlja stvarni igrač kao primjerice odabir mjesta na kojemu će napraviti novu zgradu, odlučiti koju borbenu jedinicu će napraviti od svih mogućih, provjeriti dali ima resurse za određeni projekt i slično.

```
Vector3 FindBuildPoint()
{
    List<Building> buildings = new List<Building>();
    foreach (Unit unit in player.ownedUnits)
        if (unit is Building) buildings.Add((Building)unit);

    for(int i = 0; i < 100; i++)
    {
        Building neighbourBuilding = buildings[Random.Range(0, buildings.Count-1)];
        Vector3 displacement = new Vector3((Random.Range(0f, 1f) <= 0.5 ? 1 : -1) *
            Random.Range(4, 12) + neighbourBuilding.transform.position.x,
            0,
            (Random.Range(0f, 1f) <= 0.5 ? 1 : -1) * Random.Range(4, 12) +
            neighbourBuilding.transform.position.z);
        if (Physics.OverlapSphere(displacement, 3, 1 << 7 | 1 << 9).Length == 0)
            return displacement;
    }
    return Vector3.zero;
}
```

Pomoćna funkcija za traženje lokacije izgradnje novih građevina funkcionira tako da se odabere jedna od već postojećih zgrada te gleda njena okolina za prazan prostor. Pošto je sve nasumično, uvijek postoji šansa da jednostavno ne postoji niti jedna slobodna lokacija u krugu trenutno postojećih što onda vraća nul-vektor koji će se interpretirati kao nevažeća lokacija. Kako se memorija nebi prepunila, metodi je dozvoljeno maksimalno 100 pokušaja za odabiranje lokacije prije nego iziđe iz petlje i vrati neuspjeh.

Sljedeća pomoćna funkcija koja će biti opisana je funkcija za odabir borbene jedinice koja će sljedeća bit trenirana od svih mogućih. Ova metoda radi na način da se od svih građevina koje sadrže borbene jedinice nasumično odabere jedna i onda još jednom nasumično odabere sama jedinica od svih koje ta zgrada ima za ponuditi.

```
void TrainArmy()
{
    List<Building> buildings = new List<Building>();
    foreach (Unit unit in player.ownedUnits)
    {
        if (unit is Building && unit.unitName != "Keep" && unit.unitName != "Farm" &&
            unit.unitName != "Sentry Tower")
            if ((unit as Building).isConstructed) buildings.Add((Building)unit);
    }
    if(buildings.Count > 0)
    {
        Building chosenBuilding = buildings[Random.Range(0, buildings.Count)];
        float coinflip = Random.Range(0f, 1f);
        switch (chosenBuilding.unitName)
        {
            case "Barracks":
                if (coinflip < 0.5) TryUsingSkill(chosenBuilding, "Train Footman", null,
                    Vector3.zero);
                else TryUsingSkill(chosenBuilding, "Train Archer", null, Vector3.zero);
                break;
            case "Stable":
                TryUsingSkill(chosenBuilding, "Train Knight", null, Vector3.zero);
                break;
            case "Blacksmith":
```

```
                TryUsingSkill(chosenBuilding, "Train Ballista", null, Vector3.zero);
                break;
            case "Arcane Tower":
                if(coinflip < 0.5) TryUsingSkill(chosenBuilding, "Train Priest", null,
                    Vector3.zero);
                else TryUsingSkill(chosenBuilding, "Train Mage", null, Vector3.zero);
                break;
             default:
                 break;
        }
    }
}
```

Kombinacija svih ovih navedenih sustava i pomoćnih funkcija daje jednu vrlo jednostavnu i relativno uvjerljivu umjetnu inteligenciju protiv koje igrač može igrati. Bilo kakve optimizacije skupljanja i trošenja resursa zahtjevaju kompleksnije matematičke modele ili formule koje nisu u opsegu ovoga rada.

9. Fog of War

Fog of War predstavlja mehaniku velikog broja RTS igara koja sakriva udaljene objekte od igrača i njegovih jedinica kako bi borbu učinili što dinamičnijom i nepredvidivom. Svi djelovi mape, osim oni u vidokrugu prijateljskih jedinica, su pod maglom koja blokira igraču pogled. Glavni cilj magle je smanjiti količinu informacija koju igrač prima o protivničkim pokretima. Ova mehanika se može napraviti na veliki broj načina, kako kroz Graph Shader tako i programski kroz skriptne komponente. Sakrivanje protivničkih jedinica se može postići programski kroz izračunavanje udaljenosti protivničkih jedinica od prijateljskih te isključivanja njihovog modela ukoliko su pre daleko. Efekt za tamni teren se može napraviti preko dodavanja Spot Light komponente iznad odabrane jedinice. Drugi pristupi su izrada vlastitog Shader-a za sakrivanje terena ili postavljanje velike geometrije preko čitave mape i mjenjanje stranica njezinog modela. Ovaj projekt koristi jednostavan pristup sa izračunavanjem udaljenosti između jedinica i dodavanjem svjetla na jedinice.

```
List<Unit> alliedUnits = new List<Unit>();
List<Unit> enemyUnits = new List<Unit>();
foreach (GameObject unitObject in GameObject.FindGameObjectsWithTag("Unit"))
{
    Unit unit = unitObject.GetComponent<Unit>();
    if (unit.owner.teamId == player.teamId) alliedUnits.Add(unit);
    else enemyUnits.Add(unit);
}
foreach (Unit enemy in enemyUnits)
{
    bool isVisible = false;
    foreach (Unit ally in alliedUnits)
    {
        if (Vector3.Distance(enemy.transform.position,
            ally.transform.position) <= ally.visionRange)
        {
            isVisible = true;
            break;
        }
    }
    if (isVisible) enemy.hidden = false;
    else enemy.hidden = true;
}
```

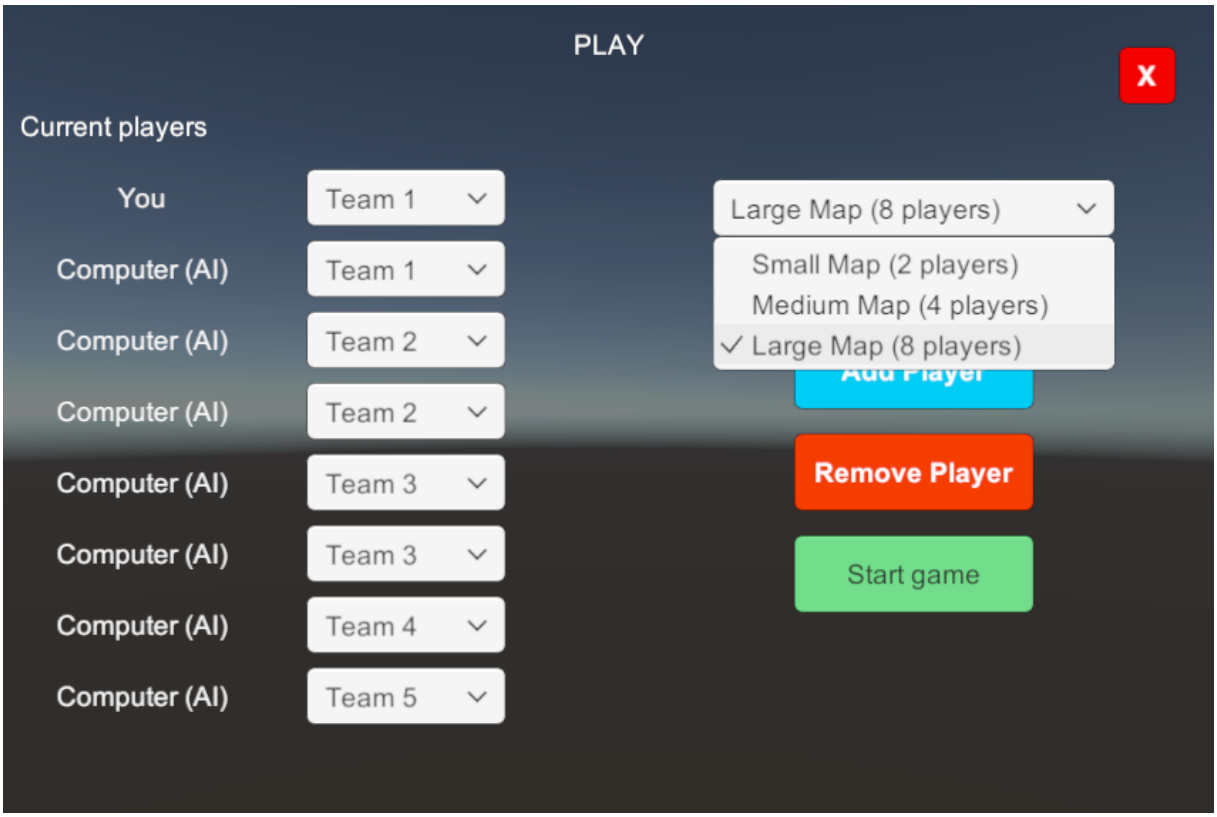
Slika 16: Vidljivi dio mape u krugu jedinice

10. Stanja igre

Osnovna stanja igre su njena inicijalizacija te postavljanje početnih lokacija, objekata i ostalih podataka potrebnih za pravilno igranje, normalno stanje u kojemu se igrači bore jedni sa drugima te završetak koji zaustavlja vrijeme unutar igre te prikazuje pobjednika. Budući da je uvijet poraza igrača eliminacija njegovih radnika i glavne zgrade, svaki puta kada je jedinica uništena provjerava se popis svih jedinica pod kontrolom igrača dali zadovoljava uvijet eleminacije ta ga se uklanja iz igre ukoliko je on ispunjen. Nakon što je ostao samo jedan aktivan tim, igra završava.

10.1. Glavni izbornik

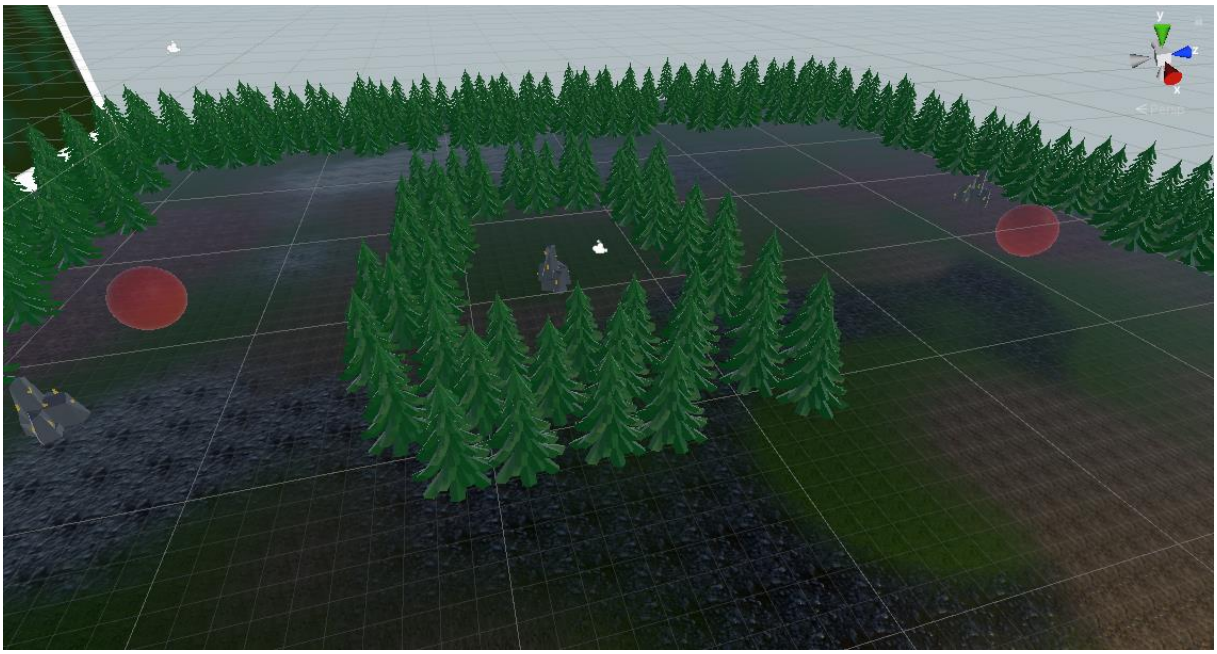
Unutar glavnoga izbornika je moguće odabrati broj igrača koji će se boriti te ih raspodijeliti u timove sa padajućim izbornicima. Isto tako je moguće odabrati mapu na kojoj će se igrači boriti uz uvijet da se ne postavi više igrača nego je to namjenjeno za pojedinu mapu, ovisno o njenoj veličini.



Slika 17: Glavni izbornik za pokretanje igre

10.2. Pokretanje igre

Svaka mapa na sebi mora sadržavati određeni broj objekata startne pozicije koji označavaju mjesta na kojima se igrači mogu stvoriti. Ova mjesta bi trebala biti pravilno raspoređena na način da su dovoljno udaljena jedna od druge te imaju otprilike jednaku količinu početnih resursa u blizini. Prilikom pokretanja igre, igrači će nasumično biti postavljeni na lokacije sa markerima startne pozicije te će im se dodjeliti jedna glavna zgrada i dva radnika nakon čega je ostatak na samome igraču. Markeri startnih pozicija se mogu vidjeti na slici 16 kao crvene kugle.



Slika 18: Markeri startnih pozicija unutar alata

```
void SpawnStartingUnits()
{
    GameObject[] startingPositions =
        GameObject.FindGameObjectsWithTag("StartingPosition");
    Camera.main.transform.position = new Vector3(startingPositions[0].transform.position.x,
        Camera.main.transform.position.y, startingPositions[0].transform.position.z);
    for (int i = 0; i < startingPositions.Length; i++)
    {
        if(i < activePlayers.Count)
        {
            GameObject newUnit;
            newUnit = Instantiate(keepPrefab, startingPositions[i].transform.position,
                Quaternion.identity);
            newUnit.GetComponent<Unit>().owner = activePlayers[i];
            newUnit.GetComponent<Building>().constructionTime = 1;
            activePlayers[i].RegisterUnit(newUnit.GetComponent<Unit>());

            newUnit = Instantiate(workerPrefab, startingPositions[i].transform.position +
                Vector3.forward * 3, Quaternion.identity);
            newUnit.GetComponent<Unit>().owner = activePlayers[i];
            activePlayers[i].RegisterUnit(newUnit.GetComponent<Unit>());
```

```
            newUnit = Instantiate(workerPrefab, startingPositions[i].transform.position +
                Vector3.forward * -3, Quaternion.identity);
            newUnit.GetComponent<Unit>().owner = activePlayers[i];
            activePlayers[i].RegisterUnit(newUnit.GetComponent<Unit>());
            activePlayers[i].startLoc = startingPositions[i].transform.position;
        }
        Destroy(startingPositions[i]);
    }
}
```

10.3. Završetak igre

Igra završava za pojedinoga igrača onda kada ostane bez glavne zgrade i svih radnika, a kada u konačnici ostanu samo igrači koji pripadaju istome timu, oni se proglašavaju pobjednicima, vrijeme u igri se pauzira i otvara se prozor za kraj igre.

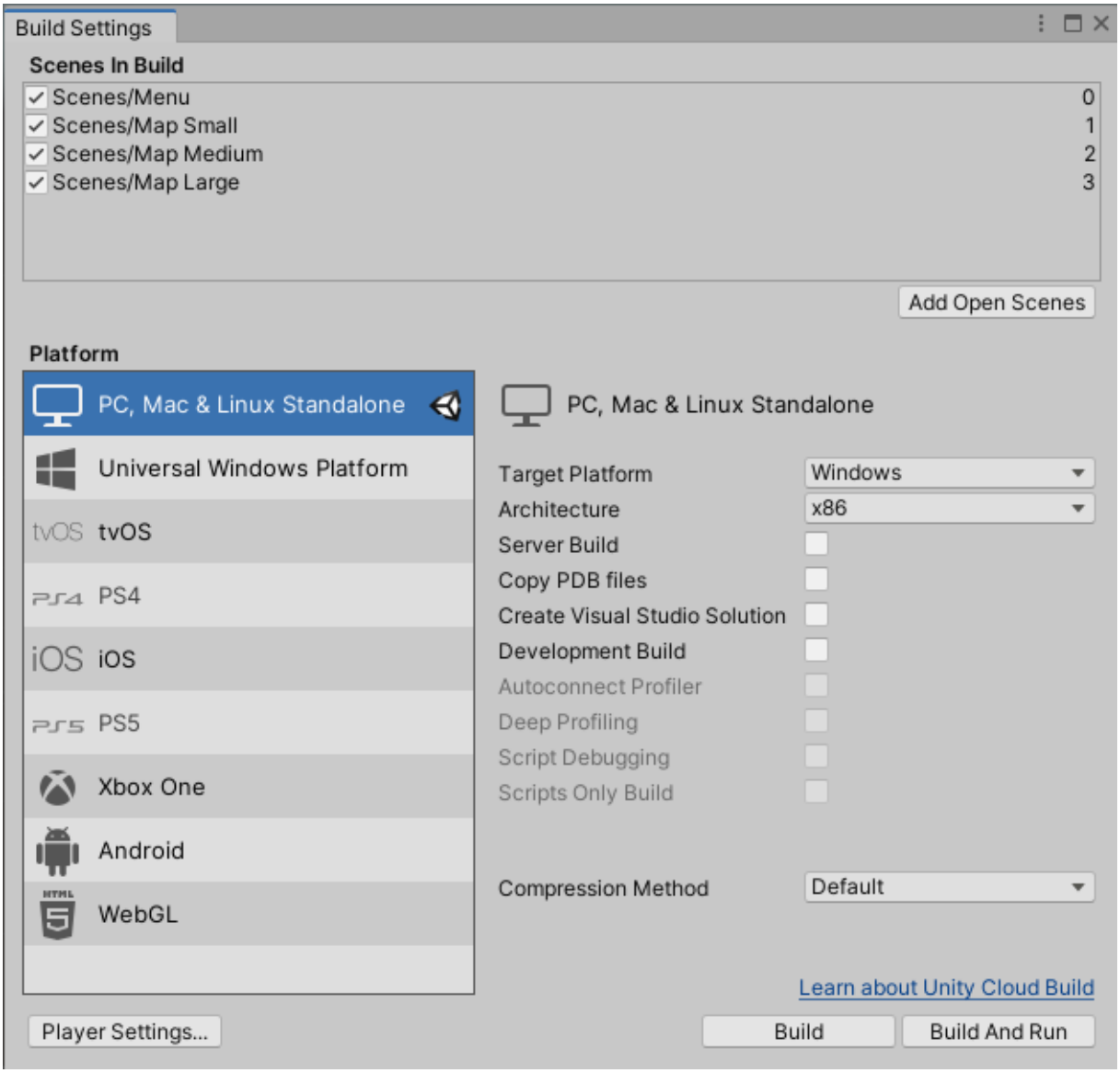
```
void Update()
{
    foreach (Player player in activePlayers)
    {
        bool hasKeep = false;
        bool hasWorker = false;
        foreach (Unit unit in player.ownedUnits)
        {
            if (unit.unitName == "Worker") hasWorker = true;
            if (unit.unitName == "Keep") hasKeep = true;
        }
        if (!hasKeep && !hasWorker) RemovePlayer(player);
    }
    List<int> activeTeams = new List<int>();
    foreach (Player player in activePlayers)
    {
        if (!activeTeams.Contains(player.teamId))
            activeTeams.Add(player.teamId);
    }
    if (activeTeams.Count <= 1)
    {
        gameUI.ShowEndgameWindow(true);
        Time.timeScale = 0;
    }
}

public void RemovePlayer(Player player)
{
    activePlayers.Remove(player);
    foreach (Unit unit in player.ownedUnits)
    {
        Destroy(unit);
    }
}
```

Prilikom eleminacije jednoga igrača, potrebno mu je uništiti sve preostale jedinice koje su možda ostale negdje na mapi kako nebi smetale ili pravile poteškoće u izvođenju koda.

Slika 19: Prozor za kraj igre

11. Distribucija

Nakon što je projekt kompletno razvijen, ispoliran te provjeren od svih potencijalnih grešaka, potrebno ga je pretvoriti u distribucijsku verziju igre koji će korisnici jednostavno moći instalirati i pokrenuti sa svoga računala bez potrebe kompilacije. Ovo omogućava korisniku bez instaliranog Unity alata ili Microsoft Visual Studija da normalno koristi i pokreće napravljeni proizvod. Za distribuciju igre se koristi ugrađeni Unity sustav u kojemu se odabiru parametri poput željenih operacijskih sustava, arhitekture i popisa scena koje konačni proizvod sadrži. Nakon pritiska na Build tipku, Unity će stvoriti konačnu izvršnu datoteku igre.



Slika 20: Distribucija igre

12. Zaključak

Unity dosta olakšava proces razvoja igara te čini mnoge stvari koje su inače dosta naporne zabavnim i zanimljivima. Razvoj video igara je područje sa jako velikim brojem aspekata koji se na prvi pogled čine jednostavnima, no što se dublje ide, to se više vidi koliko su široki i kompleksni. Stvari poput osvjetljenja, modeliranje i animacija, upravljanje zvukom, dizajn levela i samo programiranje su samo neke od vrlo zahtjevnih stvari u koje bi se trebao uložiti veliki broj sati kako bi se pravilno savladale. Industrija video igara upravo iz tog razloga je dominirana od strane velikih kompanija sa timovima koji se svaki bave jednim od ovih aspekata za razliku od individualnih developera koji moraju raditi sve sami. Unity dosta olakšava ovaj proces, no teži dio nije korištenje već postizanje dovoljnog znanja i iskustva za spajanje svega u jednu kvalitetnu cjelinu.

Nakon izrade igre utrkivanja u više igrača za svoj završni rad te par drugih manjih osobnih projekata u Unity alatu, već sam imao dovoljno znanja i iskustva za korištenje mnogih sistema koje Unity pruža svojim korisnicima. Unatoč tome, žanr strateških igara u realnom vremenu me malo iznenadio i uhvatio nespremnim sa svojom količinom i zahtjevnošću sa programske strane. Kako bi se strateška igra pravilno izvela, trebalo bi se postaviti jako dobra arhitektura prije nego se uopće krene u izradu projekta jer vrlo vjerovatno u niti jednom drugom žanru video igara objekti nisu toliko međusobno povezani i isprepleteni koliko u ovome. Svaki objekt ima interakcije sa nekoliko drugih čak i ako se zadaće klasa odvoje što je više moguće. Čak i sa jako dobrim poznavanjem objektno orijentiranog programiranja te njegovih praksi, ovakav projekt može uzeti nekoliko mjeseci za svoju izradu. Objekti HUD elementa imaju ogroman broj interakcija sa različitim objektima unutar same igre, objektima na drugom dijelu HUD-a i obrnuto. Nije samo potrebno sve korektno povezati, nego i postaviti provjere na svim djelovima za razno razne null reference, objekte bez pojedinih komponenata, redosljed stvaranja objekata i slično.

Lakši djelovi ovoga projekta su vjerovatno stvaranje samoga sadržaja igre poput jedinica i njihovih vještina nakon što je sva pozadinska programska logika postavljena. Sve što je bilo potrebno za stvaranje nove jedinice je utipkati njegove atribute u prozor inspektora na Unit komponentu te dodati vještine koje želimo da ona ima. Stvaranje nove mape je također trivijalan posao koji ne zahtjeva više od sat vremena pošto samo treba postaviti.

Smatram da je u konačnici ovaj projekt vrlo dobar izazov za nekoga tko bi se htio baviti razvojem video igara, ne samo u Unity alatu, nego i općenito, pošto je glavni izazov bio stvaranje arhitekture sustava koja bi podržala kasnije lagano stvaranje samoga sadržaja igre.

Popis literature

[1] Unity User Manual. Dostupno na: https://docs.unity3d.com/Manual/index.html. [Pristupljeno: 19. Rujna 2021].

[2] Making a RTS game in Unity (C#), Mina Pêcheux. Dostupno na: https://medium.com/c-sharp-progarmming/making-an-rts-game-in-unity-91a8a0720edc. [Pristupljeno: 18. Rujna 2021].

[3] Unity (game engine), Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Unity_(game_engine). [Pristupljeno: 19. Rujna 2021].

[4] Unity Asset Store, „Low poly styled rocks“. Dostupno na: https://assetstore.unity.com/packages/3d/props/exterior/low-poly-styled-rocks-43486. [Pristupljeno: 19. Rujna 2021].

[5] Unity Asset Store, „Yughues Free Ground Materials“. Dostupno na: https://assetstore.unity.com/packages/2d/textures-materials/floors/yughues-free-ground-materials-13001. [Pristupljeno: 19. Rujna 2021.].

[6] Unity Asset Store, „Awesome Stylized Mage Tower“. Dostupno na: https://assetstore.unity.com/packages/3d/environments/fantasy/awesome-stylized-mage-tower-53793. [Pristupljeno: 19. Rujna 2021].

[7] Unity Asset Store, „FantasyHouse_LowPoly“. Dostupno na: https://assetstore.unity.com/packages/3d/environments/fantasy/fantasyhouse-lowpoly-120429. [Pristupljeno: 19. Rujna 2021.].

[8] Unity Asset Store, „Blacksmith (Stylized)“. Dostupno na: https://assetstore.unity.com/packages/3d/blacksmith-stylized-69249. [Pristupljeno: 19. Rujna 2021.].

[9] Unity Asset Store, „Baker's House“. Dostupno na: https://assetstore.unity.com/packages/3d/environments/fantasy/baker-s-house-26443. [Pristupljeno: 19. Rujna 2021].

[10] Unity Asset Store, „Free Trees“. Dostupno na: https://assetstore.unity.com/packages/3d/vegetation/trees/free-trees-103208. [Pristupljeno: 19. Rujna 2021].

[11] Unity Asset Store, „Toon RTS Units“. Dostupno na: https://assetstore.unity.com/packages/3d/characters/toon-rts-units-67948. [Pristupljeno: 19. Rujna 2021].

[12] Flaticon, „Chinese coin“. Dostupno na: https://www.flaticon.com/free-icon/chinese-coin_3788231. [Pristupljeno: 19. Rujna 2021].

[13] Flaticon, „Chicken Leg“. Dostupno na: https://www.flaticon.com/free-icon/chicken-leg_3143626. [Pristupljeno: 19. Rujna 2021].

[14] Flaticon, „Wood“. Dostupno na: https://www.flaticon.com/free-icon/wood_3275760. [Pristupljeno: 19. Rujna 2021].

[15] Merrick079, „Sword sound 1“. Dostupno na: https://freesound.org/people/Merrick079/sounds/568170/. [Pristupljeno: 19. Rujna 2021].

[16] LiamG_SFX, „Fireball Cast 1“. Dostupno na: https://freesound.org/people/LiamG_SFX/sounds/334234/. [Pristupljeno: 19. Rujna 2021].

[17] Ali_6868, „Arrow Impact 1“. Dostupno na: https://freesound.org/people/Ali_6868/sounds/384914/. [Pristupljeno: 19. Rujna 2021].

[18] colorsCrimsonTears, „Heal - Rpg“. Dostupno na: https://freesound.org/people/Merrick079/sounds/568170/. [Pristupljeno: 19. Rujna 2021].

Prilozi

[1] Strateška igra u realnom vremenu „Diplomski rad“ [Unity Project]. Dostupno na: https://drive.google.com/drive/folders/1Ifi4i5BSlt5x0sx3kGKCob-4CXsdxHWn

Popis slika

Popis tablica